Computer Security

06. Exam 1 Review

Paul Krzyzanowski

Rutgers University

Spring 2017

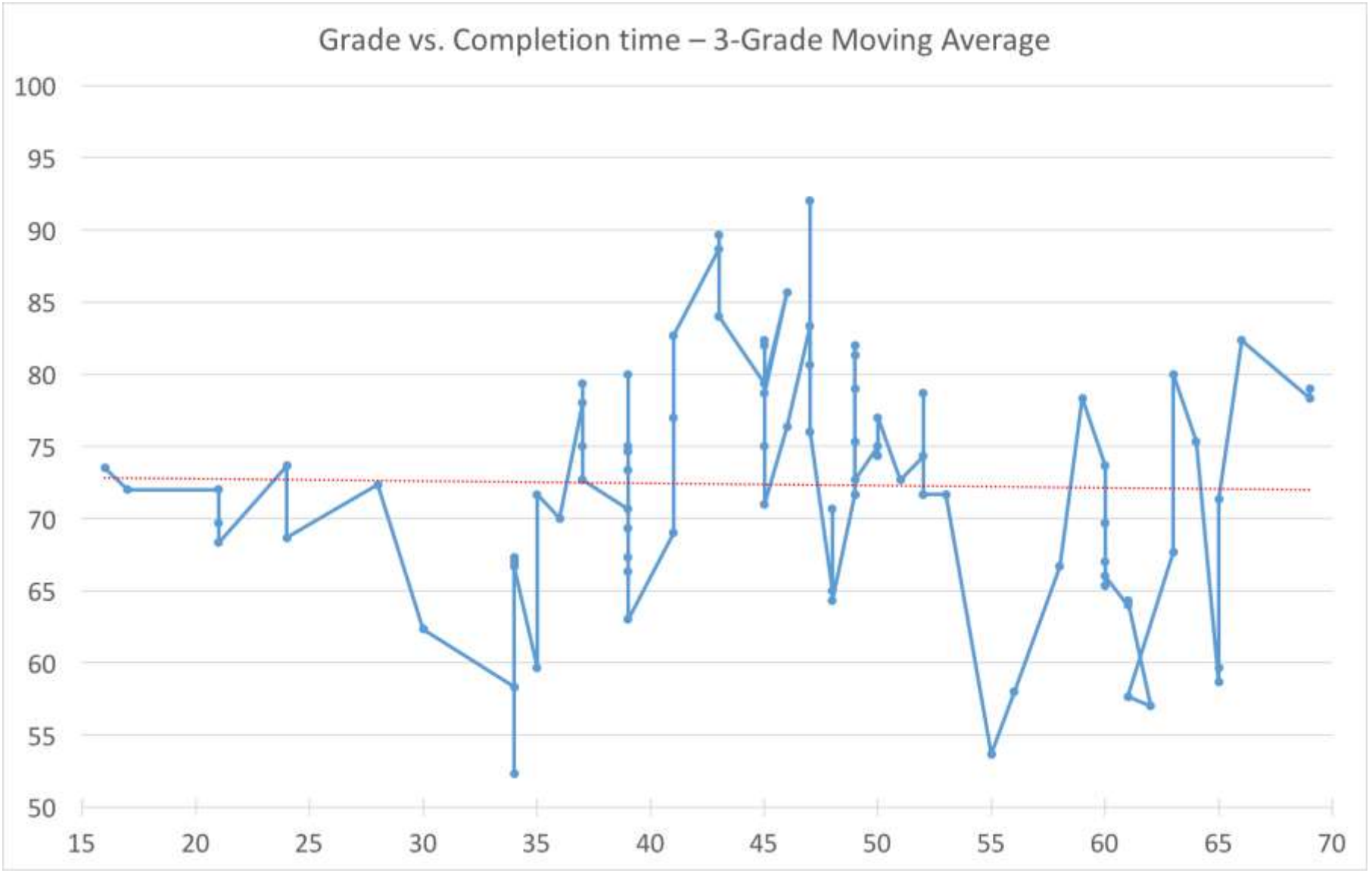
Grades

Average	72.6
σ	14.0
Highest	100
Lowest	41
Top 10%	≥ 90
Top 20%	≥ 83
Bottom 10%	≤ 52
Bottom 20%	≤ 60

Approximate grades

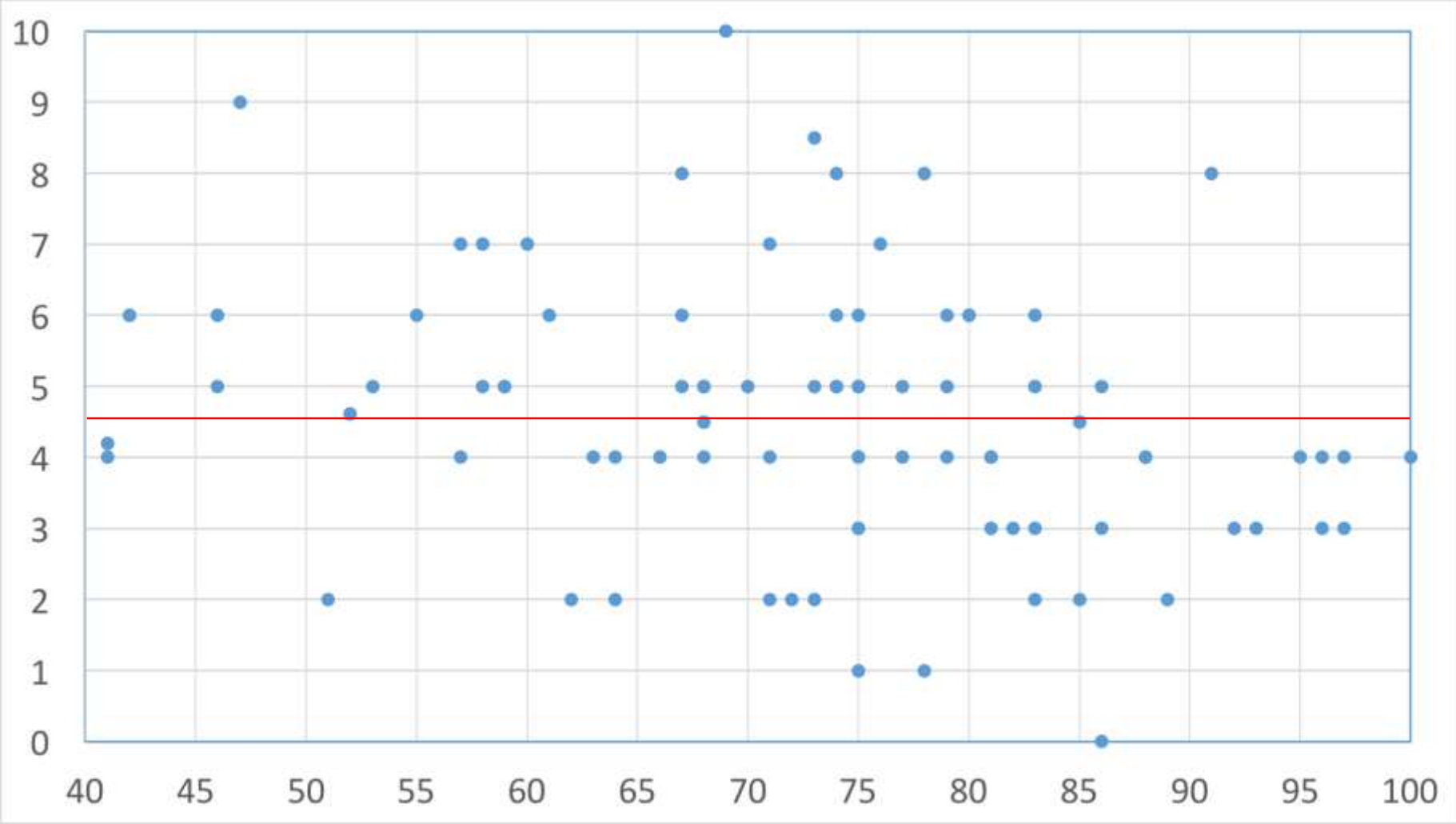
A	80
B+	72
B	64
C+	56
C	48

Grade vs. Completion Time



Grade vs. Pain

Average pain = 4.6



Question 1

Explain the difference between *confidentiality* and *integrity*.

- Confidentiality: restricts ability to read data
- Integrity: refers to the trustworthiness of the system & data – restricts ability to modify data

Question 2

Here is some C code where an application allows a file to be deleted only if it belongs to the group staff. The program works fine. Explain the security flaw in the logic of this program.

```
stat(testfile, &sb);       // get info about the file
g = getgrgid(sb.st_gid);   // look up the group name
if (strcmp(g->gr_name, "staff")
    unlink(testfile);
else
    printf("cannot delete %s, group name = \"%s\"\n",
        testfile, g->gr_name);
```

- `strcmp` and `printf` are safe functions!
- It's not about setting files to group staff and then deleting them – that's what the snippet intends to do
- It's a TOCTTOU problem: we *first* make a decision to delete the file and *then* delete it. That introduces a race condition.

Question 3

How do Linux capabilities help enforce the principle of least privilege?

- Capabilities restrict what a process running as *root* can do.
- Unless a process *needs* specific controls (access to certain system calls), they can be disabled. That way, even if a process gets elevated privileges, it is limited in what it can do.

NOT: file protection

Question 4

Why does the Biba model pose a risk to intellectual property theft?

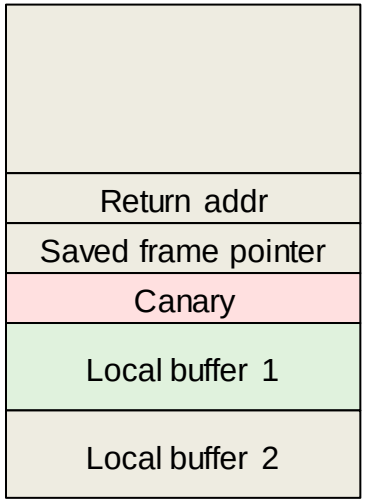
- Biba is not about confidentiality levels but integrity levels.
- A low integrity process can read high integrity data.
- When the same levels are used for confidentiality, a low integrity process can read high-value data even though it cannot modify it
  - Example: Internet Explorer ran as a low integrity process so that malware would not be able to modify user or system files. However, that would not stop malware from reading that data (and possibly uploading it to a server)

Question 5

Under what conditions can stack canaries detect off-by-one overflow attacks?

A canary will only detect an off-by-one overflow for the first allocated buffer – one that is next to the canary.

The question does not ask how stack canaries work



Off-by-one errors in *local buffer 2* will not be detected

Question 6

The Trusted Computing Base (TCB) is:

(a) The set of applications that are security sensitive.

(b) A set of software that adds security to an insecure system.

(c) A computer system that has been configured to be secure.

(d) The components in which vulnerabilities will jeopardize the security of the entire system.

- TCB = underlying system software that you expect to work properly for security to make sense
  - E.g., operating system, compilers
- Policies, applications, user data sit on top of the TCB

Question 7

In POSIX systems without access control lists, a file has:

(a) One owner and one group.
(b) One owner and one or more groups.
(c) One or more owners and one group.
(d) One or more owners and one or more groups.

A user may belong to one or more groups but a file has just one owner and one group:

- fixed set of data (two numbers) in the inode

Question 8

Execute permission for a directory means:
(a) You can create and delete files in that directory.
(b) The directory contains programs that are executable.
(c) You can read the contents of the directory.
(d) You can search for file in a directory but not necessarily see the contents of the directory.

Read = look at what files are in the directory
Write = create & delete files in the directory
Execute = search (e.g., resolve a path)

Question 9

An Access Control List (ACL) is:

(a) A list of files and access permissions for a specific user.
(b) A list of files that a user can access.
(c) A list of user and group access permissions for a file.
(d) A list of users who are authorized to access the system.

An ACL is associated with a file

Set of Access Control Entries:

{user or group}: access permissions

Question 10

What is wrong here?

```
program >secretfile; chmod u=rw,g=,o= secretfile
```

(a) Group and other must be assigned some access permissions; they cannot have none.

(b) There is a race condition that may allow an intruder to read secretfile.

(c) Another user with the same user ID will have access to the file.

(d) A user cannot have both read and write access to the same file.

An intruder may open *secretfile* before the mode is changed.

Question 11

Which activity violates the Principle of Least Privilege?

(a) A mail server has access to all users' mailboxes.
(b) A print server can access a private spool directory.
(c) A web server runs with root privileges to serve pages from user directories.
(d) A user can collaborate with another user by editing the same file.

Principle of Least Privilege: don't give a process access to more than it needs to do its job

Question 12

Which operation is inefficient with *capability lists*?

(a) Check the user's access permissions when opening a file.
(b) Copy file access rights of one user to another user.
(c) Change access rights of a single file for all users.
(d) Delete all access rights for a specific user.

Access Control List:
list of access permissions for different users – associated with a file
Capability List:
list of file access permissions for different files – associated with a user

(a) Relatively easy: you'd expect to have this info cached for an active user

(b) Super easy: copy a capability list from one user to another

(c) A pain: have to search through *all* capability lists

(d) Really easy: delete a user's capability list

Question 13

Mandatory Access Control (MAC) differs from *Discretionary Access Control (DAC)* because:

(a) Users cannot change access permissions for their files.
(b) MAC applies to subjects while DAC applies to objects.
(c) MAC policies apply to a collection of computers while DAC policies apply to only one system.
(d) The kernel enforces MAC permissions while DAC permissions are only advisory.

(a) DAC = users in control; MAC = admin in control

(b) MAC & DAC both deal with how *subjects* are allowed to access *objects*

(c) MAC or DAC don't say anything about a collection of computers – that's up to OS admin controls

(d) The kernel always enforces access permissions – MAC or DAC

Question 14

A risk with the *Bell-LaPadula* model in its basic form is that:
(a) A user with low privileges may overwrite a high-privilege file.
(b) A user with high privileges may overwrite a low-privilege file.
(c) A user with low privileges may read a high-privilege file.
(d) A user with high privileges may read a low-privilege file.

BLP:
Simple Security Property: *no read up*
A subject cannot read from a higher security level
*-property: *no write down*
A subject cannot write to a lower security level

(b) No write down
(c) No read up
(d) Yes, but so what? It's a more trusted user that's reading.

Question 15

Role-based Access Control (RBAC):

(a) Allows file sharing only with users that have the same role.

(b) Assigns hierarchical privilege levels to different classes of users in an organization.

(c) Is a form of discretionary access control.

(d) Is based on defining roles based on job functions.

(a) Having *role* assigned to you does not necessarily give you file access for sharing: you might have the ability to add entries to a database, for example

(b) RBAC does not have a concept of a hierarchy

(c) RBAC is mandatory access control – an admin assigns roles and access rules

(d) Yes. The key point with RBAC is *roles* – a level of indirection between users and object permissions

Question 16

What is the best way to prevent buffer overflow attacks?

(a) Use a language that has run-time checks of array boundaries.
(b) Address Space Layout Randomization.
(c) No-execute stack memory.
(d) Stack canaries.

(a) Run-time checking ensures buffers will not overflow.

(b) Buffers can still overflow – it's just more challenging to find addresses

(c) Buffers can overflow – this led to *return-to-libc* and *return-oriented-programming*

(d) Detects buffer overflow upon function return

- Buffer overflow may be used before the *return* (e.g., other modified variables)
- Exception handlers may be triggered to run injected code prior to the return
- Does not detect buffer overflows on the heap

Question 17

A *landing zone* is:

(a) The current frame pointer, which defines the base for local variables.
(b) A series of no-op instructions preceding injected code.
(c) The buffer containing malicious code.
(d) The location on the stack that contains the target branch address.

If you're not sure of the exact address of a buffer but have a general idea, a series of no-op instructions allow you to create a region into which execution can safely jump – and process no-ops until the useful injected code is reached.

Question 18

What will `printf("%d%n", 123, &x)` do?

(a) Print "123" and write a pointer to the string "123" into x.
(b) Print "123" and write the number 123 into x.
(c) Print "123" and write the number 3 into x.
(d) Print "123" and write the number 1 into x.

%n writes the *number of bytes output thus far* into a specified memory location

`printf("%d", 123)` will print `"123"` – 3 bytes

so `printf(%d%n", 123, &x)` will write the value 3 to the address `x`

Question 19

Fuzzing is the technique of:

(a) Using encrypted return values on the stack so malicious code cannot write meaningful addresses.

(b) Entering easy-to-find patterns to trigger buffer-overflow errors.

(c) Having a compiler generate code to check for buffer overflows.

(d) Exiting a program if a buffer overflow is detected.

Testing technique:

create a buffer overflow

have the program crash

search for the pattern

Question 20

Return-Oriented Programming, ROP, was created to overcome:

(a) Stack canaries.
(b) Data execute protection (DEP).
(c) Address space layout randomization (ASLR).
(d) Buffer overflows.

Data execute protection took away an attacker's ability to inject code

Question 21

With *stack canaries*, a compiler may reorder local variables such that:

(a) Arrays and regular variables are randomly interspersed.

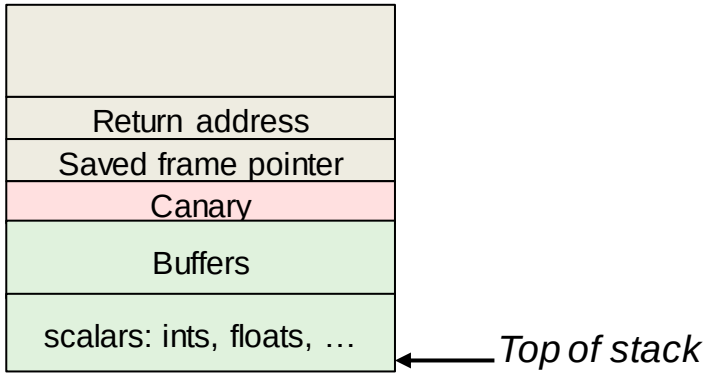
(b) Arrays are allocated onto the heap and not the stack.

(c) Arrays are at the top of the stack, followed by regular variables.

(d) Arrays are at the bottom of the stack, followed by regular variables.

☞☞☞ Unclear answers – you will get credit for (c) or (d) ☜☜☜

A compiler will try to allocate buffers first, followed by regular variables to disallow buffer overflows modifying local variables in the function



Question 22

SQL injection works when:

(a) An attacker uses a buffer overflow exploit to change the query string.
(b) A buffer overflow exploit changes the operation of the SQL interpreter.
(c) User input becomes part of the query string.
(d) Executable code is sent as input instead of a query.

(a), (b): No need for buffer overflows

(d): You're not sending executable code (e.g., opcodes as you would in a buffer overflow attack) – just fragments of text that will change the syntax of the query

Question 23

Setting the LD_PRELOAD shell variable:

(a) Turns off Address Space Layout Randomization (ASLR), enabling attacks.

(b) Preloads user input to a program.

(c) Preloads a different program that will be executed whenever a user tries to run a program.

(d) Allows you to overwrite library functions that a program might use.

LD_PRELOAD will force a library file to be loaded before any other library

When you run your program, it will check the pre-loaded functions first before checking other libraries

Question 24

A *homograph attack* is a form of:

(a) Deception.
(b) Privilege elevation.
(c) Code injection.
(d) Denial of service.

Deception:

A homograph attack creates words (usually domain names) that look the same but use different characters

Question 25

FreeBSD Jails are a big improvement over *chroot* because they:

(a) Do not require root privilege to run.

(b) Can limit the operations that a root user can perform in the jail.

(c) Use a separate memory manager to ensure that jailed processes have their own address space.

(d) Create an isolated file system namespace.

Question 26

Social engineering refers to training individuals to follow proper security policies and be on the lookout for violations.

False.

It's about using deception to manipulate individuals

Question 27

Data integrity means that the data can only be read by authorized users.

False.

It means only authorized users can modify data.

Question 28

A threat is a weakness or error in the security system.

False.

That's a *weakness* or *vulnerability*

A threat is the potential harm from an attack on the system.

Question 29

In POSIX, you can create a file that others can write to but you cannot.

True.

You can disable write access for yourself but enable it for *group* or *other*

Question 30

The *setuid bit* causes a program to run with the user's privileges instead of the program owner's.

False.

The program runs with the program owner's permissions.

Question 31

A capability list is a list of users and the operations they can perform on a specific file.

False.

A capability list is a list of *files* and the operations that a user can perform on each of those files.

Question 32

To avoid problems with pathnames referencing a file outside a base directory, you should reject any files that contain "`../`" substrings.

False.

Not good enough.

You'll end up rejecting valid names, such as "/`whatever../index.html`"

You'll also reject valid traversals within the allowable subtree:

```
/whatever/notes/../index.html
```

Question 33

The *Type Enforcement Model* is essentially an admin-managed access control matrix.

True.

It's an access control matrix (usually really restricted) that is processed before file ACLs

Question 34

The *Chinese Wall Model* relies on saving the state of past accesses.

True.

It has the concept of conflict classes.

Based on what you accessed previously, you may not be allowed access to certain data.

Question 35

A *heap overflow* cannot overwrite a return address.

True.

A return address sits on the stack.

A heap overflow is not able to overwrite it.

The end