**Question 1 - multiple choice, shuffle**

Why is waiting to think about security until after the software is built a bad idea?

A: You might miss important security requirements that necessitate a re-design

Feedback: Yes, but not just this

B: You might make critical mistakes in the software's design

Feedback: Yes, but not just this

C: Fixing problems once the software is built is more difficult and more expensive

Feedback: Yes, but not just this

\*D: All of the above

**Question 2 - multiple choice, shuffle**

What is an **abuse case**?

A: A scenario that illustrates a system's functional requirements

Feedback: Functional requirements are what the system is supposed to achieve, e.g., word processing or accounting; they are not normally security relevant.

\*B: A scenario that illustrates a potential failure in security under relevant circumstances

C: An official report made by MITRE Corp that describes a discovered software vulnerability and possible abuse of it

Feedback: This is a CVE, or *common vulnerability and exposure*, not an abuse case.

D: An example of a heated disagreement between the security team and the development team

Feedback: An abuse case speaks about an interaction with a system of interest, not an interaction among the team members that built it

**Question 3 - multiple choice, no shuffle**

Which of the following is a reason to make an explicit threat model when designing a system?

A: So that you avoid an incoherent defense

Feedback: Yes, but not just this

B: So you can defend against the most likely/costly/important attacks

Feedback: Yes, but not just this

C: So you can explicitly list and challenge assumptions that underlie your design

Feedback: Yes, but not just this

\*D: All of the above

**Question 4 - multiple choice, shuffle**

Suppose you design software for a bank and the bank's customers may remotely log into its site using commodity PCs. These PCs might have malware on them, which could log keystrokes or read files stored on the machine. Which threat model (using terms defined in the lectures) makes the most sense for you to consider, when designing the bank's site?

A: Network user

Feedback: Network users can only interact with a site via its normal network interface. As such, they have no direct view of other users, but malware does have such a view.

B: Snooping user

Feedback: Snooping users can view the message traffic of other users, but they cannot view their files or keystrokes directly, as malware could

\*C: Co-located user

Feedback: The malware is potentially co-located with the client user's software on the same machine

D: Malicious user

Feedback: This is not a distinct kind of threat model -- one shoudl assume that all attackers are potentially malicious

**Question 5 - multiple choice, shuffle**

What is a good defense against powers that are particular to a *snooping user*?

\*A: Using encryption

Feedback: Snooping users can view the network message traffic of others interacting with a site, so encrypting that traffic limits the negative effects of snooping

B: Using a type-safe language

Feedback: Using a type-safe language is useful under all threat models we have considered; here we are interested in snooping users in particular

C: Using passwords to authenticate users

Feedback: Using passwords for authentication is useful for all threat models; it does not defend against snooping users in particular, e.g., if passwords are not encrypted on transmission

D: Using a firewall

Feedback: Firewalls filter or alter potentially harmful traffic into an enterprise. But they cannot affect snooping users on either side of the firewall

**Question 6 - multiple choice, shuffle**

A **denial of service attack** violates what security policy/goal?

\*A: Availability

Feedback: Denying service makes that service unavailable to users that depend on it

B: Authentication

Feedback: Authentication is orthogonal to denial of service -- denial of service can take place even when authentication is not used

C: Authorization

Feedback: Authorization is orthogonal to denial of service -- denial of service can take place even when authorization is not used

D: Integrity

Feedback: Denying service prevents access to resources, but does not affect the resources' integrity

**Question 7 - multiple choice, shuffle**

When talking about computer security, what do we mean by the term, **principal**?

\*A: An actor, or role, that is the subject of a security policy

Feedback: Principals can be people, computer programs, or some other entity acting in a particular role, like *manager* or *client*

B: A rule of thumb for secure coding

Feedback: This is a *principle* or *rule*

C: A foundational observation

Feedback: This is an *insight*, not a principal

D: A method for delegation

Feedback: Principals can delegate authority to other principals, but principals are not a method for doing so

**Question 8 - multiple choice, shuffle**

Passwords, biometrics, and user-owned SMS-receiving mobile phones are useful for what security mechanism?

\*A: Authentication

Feedback: These are all methods by which a principal proves his identity to a system he is interacting with

B: Authorization

Feedback: Authorization is about deciding whether a principal may carry out a particular action, but passwords etc. are used to establish the identity of a principal, independent of any particular actions.

C: Audit

Feedback: Audit is about detecting and/or diagnosing an attack, not about establishing identity

D: Small trusted computing base (TCB)

Feedback: The goal of a small trusted computing base is not a security mechanism, it is a principle of secure design

**Question 9 - multiple choice, no shuffle**

We identified several categories of secure design principles, with respect to how they deal with attacks. Running each browser tab in a separate OS process (as done by the Chrome browser) is an example of which category?

A: Prevention (of an attack)

Feedback: Implementing a tab in a separate process does not prevent an exploit or breach of that tab (compared to a single process model) but does limit what such a breach can accomplish, because only that process's resources are accessible

\*B: Mitigation (of the damage from an attack)

Feedback: Using a separate tab mitigates an attack because it reduces the resources/targets available to a successful exploit (Recovery is also a correct answer)

C: Recovery (from a successful attack)

Feedback: You could argue that isolating a tab makes it easier to recover from a breach: You can easily kill the tab's process with less effect on the rest of the system. But actually this is just another way of saying the attacked was mitigated.

D: None of the above

**Question 10 - multiple choice, shuffle**

Suppose you are implementing a graphical user interface for interacting with an implementation of the RSA cryptosystem, and you want to give users a way to generate new keys. Which of the following designs most takes security into account?

A: Use a text box to ask the user to fill in how many bits they want their key to be

Feedback: This design is non-optimal because it does not pick a good default if the user does not understand the security ramifications of his choice

B: Don't ask the user about key size at all -- always use 256 bits

Feedback: Magic constants are often not a good idea, because circumstances change. In this particular case, RSA keys of 2048 bits are now in common use, so 256 bits is probably too few

C: Ask the user, but set the default response to be 2048 bits, which is chosen based on the assumption of a strong adversary

Feedback: This is not a bad design, because it picks a safe default choice, but it could be better

\*D: Allow the user to use a slider to choose the number of bits, setting slider initially to point at 2048 bits. As the user moves the slider to larger or smaller values, visualize the difference in relative protective power, e.g., using a meter.

Feedback: This is the best choice because it picks a safe default value (the starting point of the slider), but also allows the user to explore the security ramifications of alternative choices

**Question 11 - multiple choice, shuffle**

Suppose you are implementing an extensible data management system. You want to accommodate plug-ins that can implement storage rules and query processing functionality for different data formats (e.g., relational data, object data, XML data, etc.). Which of the following designs most takes security into account?

A: The plug-ins are linked directly in the address space of the data management software, ensuring high performance

Feedback: A vulnerability in the plugin exposes the entire application, because the plug-in is not compartmented

\*B: The plug-ins are implemented as separate OS processes; these processes communicate to/from the main process to handle queries/updates for the data formats they support

Feedback: This is the best choice: a vulnerability in a plug-in will affect that plug-in but will have limited impact (only what it can effect via the communication API) on the rest of the application

C: The plug-ins and the main data management software are linked into the operating system kernel as a special kind of device driver, to give them direct access to stable storage and the network stack, while the OS can enforce their security

Feedback: A vulnerability in the plug-in exposes the entire application and OS because the plug-in is not compartmented

D: The plug-ins are implemented as separate OS processes but which share memory with the main process (and may access its memory as well), for better efficiency. Queries/updates occur via inter-process communication.

Feedback: This is a good choice, but sharing memory exposes more of the main application (i.e., parts of its memory) to a buggy plugin that could be exploited. Limiting interactions to be only via interprocess communication to handle queries/updates reduces the space of malicious actions.

**Question 12 - multiple choice, shuffle**

**Promoting privacy** is a goal that follows from which category of secure design principle?

\*A: It is an example of *trusting with reluctance* because promoting privacy means sharing private information with as few software components as possible, meaning that fewer need to be trusted to protect the information

B: It is an example of *favoring simplicity* because privacy is quite simply the right thing to do

Feedback: That's a bit dogmatic

C: It is an example of *monitoring and recovery* because failure to promote privacy could be discovered by monitoring

Feedback: Performing monitoring in and of itself does not promote privacy: A poor design could constantly leak private information, regardless of whether such leaks are monitored

D: It is an example of *defense in depth* because privacy is a deep topic that is often debated.

Feedback: In-depth debate is great, but it's not a secure software design principle

**Question 13 - multiple choice, shuffle**

Encrypting a password database is an example of what category of design principle?

\*A: It is an example of *defense in depth*

Feedback: You could argue that it is defense in depth because while a system likely has defenses in place to prevent an adversary from directly accessing the database, encrypting the database protects that database even if these other defenses are breached

B: It is an example of *favoring simplicity*

Feedback: Encrypting the database is not simpler than not encrypting it

C: It is an example of *monitoring and recovery*

Feedback: An encrypted database is not a means of recovering from a breach; rather it is a defense against certain negative consequences that could arise following a breach (compromised user accounts)

**Question 13 - multiple choice, variation 1, shuffle**

Encrypting a password database is an example of what category of design principle?

\*A: It is an example of *trusting with reluctance*

Feedback: You could argue that encrypting the database is reducing our trust in the rest of the system's defenses which, if breached, might allow the database to be exfiltrated

B: It is an example of *favoring simplicity*

Feedback: Encrypting the database is not simpler than not encrypting it

C: It is an example of *monitoring and recovery*

Feedback: An encrypted database is not a means of recovering from a breach; rather it is a defense against certain negative consequences that could arise following a breach (compromised user accounts)

**Question 14 - checkbox, shuffle, partial credit**

Which of the following vulnerabilities can VSFTPD's secure string library help protect against?

\*A: Integer overflow

Feedback: Recall the code for copying a string checks to make sure that accounting for the null terminator will not overflow the integer containing the string's length

B: SQL injection

Feedback: The secure string library pays no attention to the contents of strings, so it will happily construct SQL-injecting strings if instructed to do so

C: Privilege escalation

Feedback: Privilege escalation is orthogonal to string construction

**Question 14 - checkbox, variation 1, shuffle, partial credit**

Which of the following vulnerabilities can VSFTPD's secure string library help protect against?

\*A: Buffer overflow

Feedback: Strings are coupled with their allocated size and current length, so string operations -- like copying or concatenation -- can be checked to ensure they do not overflow a buffer

\*B: Integer overflow

Feedback: Recall the code for copying a string checks to make sure that accounting for the null terminator will not overflow the integer containing the string's length

C: Format string attack

Feedback: The secure string library deals with strings of type struct mystr, which are separate from the char\* strings used for format strings

**Question 14 - checkbox, variation 2, shuffle, partial credit**

Which of the following vulnerabilities can VSFTPD's secure string library help protect against?

\*A: Integer overflow

Feedback: Recall the code for copying a string checks to make sure that accounting for the null terminator will not overflow the integer containing the string's length

\*B: Buffer overflow

Feedback: Strings are coupled with their allocated size and current length, so string operations -- like copying or concatenation -- can be checked to ensure they do not overflow a buffer

C: Privilege escalation

Feedback: Privilege escalation is orthogonal to string construction

**Question 15 - multiple choice, shuffle**

VSFTPD forks a new process to handle each client connection. It could have, instead, spawned a thread within the main process to handle each connection, as is done in many servers. How would this alternative design compare to the original?

A: It would be equally secure and would perform better because threads are cheaper to manage than processes

Feedback: It might perform better, but will not be as secure as more attack surface is exposed

\*B: It would be less secure because a compromise by a malicious client in one thread could (more easily) access data used by another client's thread, since they share the same address space

Feedback: This fact is due to threads sharing the same address space as their host process

C: It would be more secure because threads are not subject to denial of service attacks but processes are

Feedback: Threads can also be manipulated to deny service, depending on the design. For example, if a new thread is created for each connection attempt then many connection attempts could create many threads, which will eventually overwhelm the system

D: It would be more secure because we could apply the SecComp system call to these threads, but could not do so for processes

Feedback: This is a false statement, as SecComp can be applied to processes

**Question 16 - multiple choice, shuffle**

FTP servers can be asked to list a directory of files. VSFTPD could do this by calling the system's **ls** (or **dir**) command, displaying the result to a client. But VSFTPD does not do this, and implements directory listings using the relevant system calls directly. Why might you argue that VSFTPD's design makes sense from a security perspective?

\*A: **ls** does more than is needed, and thus unnecessarily expands the TCB

Feedback: An example of this sort of thing is the ShellShock bug: the bash shell is very powerful, but little of its functionality is used in many applications. The ShellShock bug exposed these applications to serious attack unnecessarily.

B: Using ls provides less control over the output, which leaves users open to XSS-style attacks

Feedback: This statement is not really true: the server could reformat the output received from ls before displaying it, if desiredI. In any case there's no way to upload scripts so that ls would redisplay them

C: Calling ls involves forking a new process, which is less secure than running within the same process

Feedback: Forking a new process, particularly at the same privilege level as the parent process, does not reduce security

D: Calling ls doesn't give us any way to employ fail-safe defaults

Feedback: This statement is not really true, as it's a question for the FTP server itself; by default it could call ls with few parameters unless directed by other measures to do otherwise.