Week 4 Quiz

1. What are formal methods for secure and robust programs?

- Using some rigorous method like math to prove the source code is correct
- Making a convincing argument that the source code is correct
- Testing the executable by ensuring every control path is taken
- Having the source code certified correct by a trusted party

Correct This is the only formal method among the answers.

2. Which of the following should you check for? (Select all that apply.)

- When computing x / y, check that y is not O.

Correct This states something that if false will cause an exception.

- In When computing array[++i], check that i is not the index of the last element of the array.

Correct This will index beyond the end of the array.

- When computing x % y, check that x and y are both positive.

*Correct This gives an undefined result (but note that both / and % are defined so that (x/y)*x + x%y = x).*

- When computing x " y, check that x and y are both signed.

3. Which of the following is true for informal methods?

- An informal method gives a hand waving description of why the program works.
- An informal method gives a proof of correctness assuming any assumptions are true.
- An informal method gives a strong argument for correctness of a program.
- An informal method gives a proof of correctness of a program.

Correct Informal is a method of reaching a goal that does not involve proof, but does allow a strong, rigorously reasoned argument for correctness.

4. Which of the following is NOT true for ad hoc methods?

- Ad hoc methods give weak arguments of correctness.
- Ad hoc methods do not give proofs of correctness.
- Ad hoc methods may say a program is correct when, in reality, it is not.
- Ad hoc methods test the program so that all paths of control are exercised.

Correct Ad hoc methods fall far short of this statement. The reasoning goes, "Okay, the code looks right. It passes our tests, so we'll just go with it." The problem with this is that there no proof of correctness, there's not even a good argument.

5. Why are checklists for secure programming helpful?

    - Using a checklist requires some understanding of programming and systems, but not of secure coding.
    - A checklist is helpful for writing secure code, but not for validating that a program is a secure one.
    - Checklists are a good reminder of what needs to be done.
    - Anyone can use a checklist to write secure code.

    Correct You need to know quite a bit about secure programming in order to be able to understand a checklist.

6. For a function, the precondition states _______ ; the postcondition states ________

    - what the programmer wants the function to do...what the programmer believes the function will do
    - any constraints on the parameters of the function...whether the arguments actually passed to the function satisfy the precondition
    - what is assumed when the function is called...what the function will do, assuming the preconditions are satisfied
    - what is relevant to the function and is true...what the function actually does

Correct The precondition states what is expected to be true when the function is called, and the postcondition what the state is assuming the preconditions are met. What the programmer believes is not a precondition or postcondition.

7. Why do you log enough information about a login to reconstruct the login actions? i /1 point

    - Because you want to be able to determine when the most login activity occurs
    - Because you want to be able to audit the login attempt and verify the cleartext password is the correct one
    - Because you want to be able to figure out whether an attempt to login is rejected
    - Because you want to be able to determine what happened should an unauthorized login succeed

Correct If someone impersonates a user, you want to figure out when he or she logged in, and what happened to enable them to log in.

8. Which of the following indicates a poorly structured program?

    - Making the connections between security-related modules and other security-related modules (and non-security related modules) difficult for an analyst, and hence an attacker, to figure out.
    - Making the security-related parts of the program as simple as possible
    - Modularizing security-relevant elements so each module performs exactly one security-related function
    - Separating the security-related and non-security-related parts of the program into different sets of modules.

Correct Although it is tempting to do this, a good analyst or attacker can figure out what those connections are. Instead it's best to focus on modularity and simplicity, all points to help structure the program soundly.