English

Universitetet i Bergen
Det matematisk-naturvitenskapelige fakultet

Exam in	:	INF226 Software Security
Semester	:	Autumn 2020
Time	:	09:00 – 12:00, 3rd of December 2020
Number of pages	:	5
Permitted aids	:	Open-book exam

- This exam counts for 60% of your final grade in this course.
- Points on the exercises indicate *approximate* percentage weight on the total final grade.
- Give justifications for your answers, unless otherwise specified.
- Use precise language and make sure to read the exercises carefully.

Exercise 1 (10 points)

Answer the questions below with one or two sentences on each.

a) Why is a buffer overflow in a C program more serious, from a security perspective, than an array out-of-bounds exception in a Java program?
b) How can a buffer over-read, where the attacker can read parts of memory, affect the protection given by a stack canary in the case of a buffer overflow?
c) Why is a session cookie with a random key better than just keeping a cookie with the user name?
d) What must the user do in order to enable an attacker to perform a cross-site request forgery attack?
e) What value would you give to the variables `username` and `password`, in order for the code below to log you in as the user `admin` – without knowing the password?

```
String sql = "FROM User SELECT (username,password)"
           + "WHERE username='" + username +"'"
           + "AND password='" + password + "'";

Statement statement = connection.createStatement();
ResultSet result = statement.executeQuery(sql);
if(result.next()) {
    setLoggedInUser(username);
} else {
    returnToLoginForm();
}
```

Exercise 2 (15 points)

You are working in a company where you have recently been transferred to a group developing a web-application for internal use. The web-application handles a questionnaire sent to clients after their first contact with the company. While reading through the source code which was written before you arrived, you find the following two snippets.

The first piece reads the POST-request and insert the user's answers into the database (along with a random 128 bit identifier which identifies the user and questionnaire):

```
PreparedStatement statement = connection.prepareStatement(insertAnswer);
statement.setString(1,request.getParameter("answerId"));
statement.setString(2,request.getParameter("firstname"));
statement.setString(3,request.getParameter("lastname"));
(...)
statement.execute();
```

The second snippet, is from the admin-module. After checking the admin's session cookie, this code displays the user's answers after getting it from the user:

```
// Get data from DB
PreparedStatement = connection.prepareStatement(getAnswer);
statement.setString(1,currentAnswerId);
statement.execute();
ResultSet rs = statement.getResultSet();
String answerId = rs.getString(1);
String firstname = rs.getString(2);
String lastname = rs.getString(3);
(...)

// Display to admin user:
html.print("<table>");
html.print("<tr><td>Fist name:</td><td>" + firstname +"</td></tr>");
html.print("<tr><td>Last name:</td><td>" + lastname +"</td></tr>");
(...)
html.print("</table>");
```

You immediately recognize that there might be a vulnerability here, related to how the user input is handled and displayed.

a) What kind of vulnerability is there likely to be in this code?

After a quick test, you find out that there is indeed a vulnerability here.

b) How can this vulnerability be exploited by an attacker?
c) Write a short e-mail to your manager reporting the vulnerability, and explain to them what threats such a vulnerability could pose to the company.

The manager assigns someone more experienced with the code-base to fix the vulnerability. When your co-worker has committed their code, you find that their solution consists of a check, in the client-side JavaScript, that there are no "<"-symbols in the data entered in the questionnaire.

d) Write an e-mail to your co-worker explaining them why their solution is not likely to prevent an attack. Outline in the e-mail how you would solve this problem.

e) Write an e-mail to the manager explaining how the group can work more systematically to find these and other kinds of vulnerabilities in the code base.

Exercise 3 (15 points)

An online multiplayer role–playing game is implemented in Java. The game consists of a client and a server program, which communicate objects directly over UDP, using an ObjectOutputStream.

Below is a class from the game which represents a player character. A player character has **stats**: *strength – intelligence – charisma*, which determine how likely the character is to succeed at various tasks in the game.

Experience points are awarded for accomplishments in the game and as the character gain experience, their stats may increase. The relationship between stats and experience is subject to an invariant, which is enforced by the constructor below.

```
public class Character extends Serializable {
    public int experiencePoints = 0;
    public int strength = 0;
    public int intelligence = 0;
    public int charisma = 0;
    private Consumer<WorldState> specialAction;

    // Constructor for Character
    public Character (int experiencePoints,
                      int strength,
                      int intelligence,
                      int charisma,
                      String type) {

        // Check that character is valid
        if((int)(Math.log(experiencePoints)) + 8 < (strength - 10)
                                                 + (intelligence - 10)
                                                 + (charisma - 10))
            throw new IllegalArgumentException("Illegal chacter");

        this.experiencePoints = experiencePoints;
        this.strength = strength;
        this.intelligence = intelligence;
```

```
        this.chrisma = charisma;

        // Set the special action.
        if(type.equals("fighter")
            specialAction = SpecialActions.standGround;
        if(type.equals("wizard"))
            specialAction = SpecialActions.fireball;
        if(type.equals("bard")
            specialAction = SpecialActions.viciousMockery;
    }

    void doSpecialAction(WorldState gameState) {
        // Apply the effect of the special action
        // to the current world state
        specialAction.accept(gameState);
    }
}
```

a) What possible security issues do you see in the above code?
b) How can these be exploited by a dishonest player?
c) How can these be exploited by a network man-in-the-middle?
d) What improvements would you make to the overall security design of the game to address these concerns?
e) Rewrite the above class in particular so that it fits your design, and explain the changes you have made. Make any additional assumptions you need about the rest of the code in order for your class to compile and work.

Exercise 4 (9 points)

In this exercise we will have a closer look at key derivation functions (KDF), such as SCrypt or Argon2.

a) How does a key derivation function make it more difficult for an attacker to guess the password of a user if the hash is leaked?

The Wifi security protocol WPA2 uses the SSID (the network name) as *salt* for the key derivation function for the Wifi-password.

b) Why could this make it easier for an attacker to break into WiFi networks with common SSIDs?
c) What is recommended to use as salt for a KDF?

Exercise 5 (8 points)

As the security engineer of your company Coal Miners United™, you are tasked with implementing the most security-sensitive feature of the company's flagship product (the

Great Coal C Compiler): a stack canary. The stack canary should be 8 bytes long, but you are pretty much free to do whatever you want otherwise.

Propose a design for how the protection would work, how the canary would be generated. For each of the decisions you take, identify the pitfalls that you avoid by taking it. To verify that your design is good, ensure that the following situations are handled properly:

a) the attacker has a buffer overflow of arbitrary length in a program compiled with your protection;
b) your program consists of two functions, *caller* and *callee*. It runs as follows:
    i. caller calls callee
    ii. callee lets the attacker input a string of arbitrary length on the stack using read() (buffer overflow)
    iii. callee prints the string input by the attacker to the standard output
    iv. callee returns
    v. caller lets the attacker input a string of arbitrary length on the stack using read() (buffer overflow)
    vi. caller returns
c) the attacker has a buffer overflow of arbitrary length in a program compiled with your protection and is able to run the program several times.

Exercise 6 (3 points)

From Wikipedia: "Just-in-time (JIT) compilation is a way of executing computer code that involves compilation during execution of a program – at run time – rather than before execution." One use-case for Just-in-time compilation is in browsers, where it is used to speed up execution of JavaScript.

Thinking of the buffer overflow protections you have been taught about (Address space layout randomisation (ASLR), non-executable (NX) stack, position-independent-executables (PIE)...), **explain potential risks of implementing a JIT compiler such as the one in your browser**!

Hint: This is a fairly complicated topic, but the expected answer is fairly simple.