**INF226 - Eksamen Vår 2021**

**Exercise 1:**

1. If an attacker manages to abuse a program by injecting shell code on the stack where buffer overflow is being performed, the damages can be fatal. One possible way for a non-executable stack to prevent this type of attack is to make the return pointer point to the buffer where the shell code was to be injected.
2. CRSF is a type of attack where an attacker gets access of someone else’s session cookie and can perform actions on behalf of them. If the SameSite flag is sat to either “Lax” or “Strict” the browser will not include the cookie in the request sent from an attacker, and CSRF will be prevented.
3. Untrusted data should not be directly inserted into the HTML seeing as an XSS attacker with malicious intent could have his own script containing JavaScript inserted into the site. XSS or cross-site-scripting is a type of attack where someone injects their own code into existing code in a program and can have some other user unknowingly execute the attackers’ script on the site.
4. We add salt to our key derivation functions to increase the security and complexity of the password authentication. Adding salt will prevent the attacker from looking up passwords using premade dictionaries containing hash-password values, more specific a rainbow table attack could be prevented.
5. To prevent XSS when inserting data into an HTML content these characters needs to be deal with:

& escaped to &amp

< escaped to &lt

> escaped to &gt

“ escaped to &quot

‘ escaped to &#39

1. Having proper access control is critical in developing safe apps with low risk of being breached. If capabilities can be forged anyone can give themselves permissions or powers to performs actions they were not originally intended to.

**Exercise 2:**

1. Access control list model has a subject associated with a permission, the same way the company you are hired to help currently has. With a lot of people in a company and many different permissions, the list can get extremely long and be hard to navigate and update. With a role-based access control model you have a more systematic collection of permissions. The permissions are now assigned to roles instead of directly to employees. The respective roles are then assigned to the employees. For a low-level business with only one or few employees ACL might suffice, but for bigger companies with a lot of different positions and permissions a RBAC model would be suitable.
2. Ayla, Espen, Chaitanya and Xian has access to the machine supply room but not the office.

|  |  |
| --- | --- |
| **Roles** | **Employee** |
| Manager | Susan |
| Machinist | Ayla, Chaitanya, Espen, Xing, Lene |
| Cleaner | Matthew, Steve |
| Trainee | Alice, Bob |
| Office worker | Mark, Lene |

|  |  |
| --- | --- |
| **Roles** | **Permission** |
| Manager | Enter factory out of hours, enter factory in hours, enter office, access machine supply room. |
| Machinist | Enter factory in hours, use machine A, use machine B, use machine C, access machine supply room. |
| Cleaner | Enter factory out of hours |
| Trainee | Enter factory in hours, use machine C |
| Office worker | Enter office |



Adding a new regular worker: If you have RBAC adding a new worker only means you must give the person the “Machinist” role. If you have the ACL model you would have to specify each Machinist permission over again and assign it to the person.

Giving all workers access to a new machine: In a RBAC model you only have to add the permission “use machine X” for the Machinist role, and every machinist would then be able to use machine X. In a ACL model you would have to add that permission to every existing machinist yourself.

Upgrading a trainee to a regular worker: In a RBAC model you only have to remove the old trainee role, and assign the new machinist role. In a ACL model you would have to remove multiple permission, and then add multiple permission.

These simplifications can improve security. Having a easy to use, systematic collection of roles, permission and people, lowers the chance of making mistakes. We are only human and we do mistakes all the time. With a RBAC model, if a mistake do occur, it will be easier to locate and fix the issue, as opposed to ACL. With a RBAC model it will be easier for the administrator to keep track of permissions, give new permissions with low risk of errors, and correction mistakes if any.

**Exercise 3:**

1. The gets(p.species) in the code is what causes the problem. The gets does not limit the length of p.species in any way, and data larger than 1024 byte could be sent to the program, resulting in a buffer overflow.
2. The attacker would have some code to perform his malicious actions, namely shell code, and a part where the function return pointer is overwritten and directed back into the buffer to have his shell code executed.
3. Back into the buffer, to his own shell code.
4. NOOP stands for no-operation. A NOOP sled is a sequence of no-operation data which only purpose is to slide the execution flow of the program to the next memory address. The goal is to have the return address land on the start of the shell code, so the malicious code can be executed.
5. A stack canary is a value we can insert between the buffer and the rest of the stack, including the return pointer. This will serve as a protection when someone tries to overflow the buffer, and a stack smashing will be detected. Adding this to the stack can make it harder for the attacker to perform buffer overflow since the canary value changes every time you run the program, which means it can be hard to predict the value.

**Exercise 4:**

1. The user input is placed directly into the SQL query, meaning the SQL injection might be possible. Using SQL injection, an attacker can display private and sensitive data, from the database, on the user-side of the app.
2. By manually testing the code with common SQL injection inputs such as:

" or ""="

10 OR 1=1

‘hey”)’

or other inputs containing typical SQL query characters (‘ “ = () \*) to create your own queries.

1. According to the GDPR the business owners have to notify authorities such as Datatilsynet within 72h of discovery of the breach. The owners also has to inform its users what has happened, and that security standards has not been met.
2. The new version is not secure either seeing as the queries are not being parameterized, as they should when using prepared statements, and user input is still just being concatenated right into the query.

PreparedStatement stmt = connection.prepareStatement("INSERT INTO order (userid,itemid,address) VALUES(?,?,?); SELECT orderid FROM order WHERE userid=?;");

stmt.setObject(1, currentUser);

stmt.setObject(2, itemID);

stmt.setObject(3, deliveryAddress);

stmt.setObject(4, currentUser);

stmt.executeUpdate();

ResultSet rs = stmt.getResult();

(...) // Code which displays all the current orders to the user

**Exercise 5:**

1. The CVE database is a collection of any reported security faults that can occur. The information stored there tells us what type of fault it is, how it can be abused, what systems are vulnerable, how it can be prevented. It can be useful for a developer to create security for a program while checking the CVE for related faults, seeing as we are only human and can forget one or two issues. It can then be nice to have a list where thousands of developers have reported in their issues.
2. OWASP ZAP and SonarQube are other resources often mentioned and used when dealing with security issues.
3. This is a fault where buffer overflow can result in privilege escalation to root. Attackers can get admin powers.