Exam INF226 – 2021 Autumn
Candidate 116

Exercise 1

a)
The call stack of a C program is located on the upper part of the memory layout. The stack is built up around a last in first out logic, which is a popular queue algorithm.

b)
ASLR or Adress Space Layout Randomization is a good technique commonly used to prevent memory exploitations such as buffer overflow attacks. This type of mitigation randomly moves the address space locations around, making it hard for an attacker to predict values for attack.

c)
Cookies consist of multiple parameters deciding how the information in the cookie can be used and transferred. The Secure Flag parameter tells us that the cookie can only be passed on used a secure connection, namely HTTPS or SSL.

d)
Untrusted data should not be inserted directly into SQL queries seeing as you could end up with some unwanted SQL injections if an attacker plays around with strings to create his own queries. Attackers can do things to the application or database that they originally was not intended to, and a data breach of sensitive and private user data can be leaked.

e)

We add salt to our key derivation function when implementing password authentication to make sure the data that was sent to the key derivation function comes out as a completely random and unique output, namely a salted hash value. Salting our passwords is effective against brute-force attacks such as premade dictionaries containing hash values for passwords and rainbow table attacks.

f)
Static program analysis is when you analyse the program and its code without running the program. Dynamic program analysis is when you run the application and scan it for security faults while it is running and test many different parts of the application, something that often can end up with unexpected outcomes.

Exercise 2

a)

Cross-site request forgery is a common type of attack where an attacker performs unwanted actions on websites using requests made from third party sites. A CSRF attack might be possible if the browser included the session cookie in its requests. In our example, websites such as GitLab and GitHub could be subject to a CSRF attack where a third-party site could create their own form containing the attackers public keys, and attempt to inject this form into the targeted users list of trusted keys. The result could be fatal, and the attacker can perform malicious actions on the targeted users behalf.

For this attack to succeed the targeted user must be logged into the application and enter the attackers third party site.

b)

Seeing as the HTML code presented in this task only uses POST and not GET, a CSRF attack would be prevented if the SameSite flag in the cookie was set to “Lax”. Setting a SameSite flag guarantees that the browser does not pass the session cookie on when and attacker tries to send request from a third-party site. The “Lax” attribute does not prevent CSRF if the application uses GET in their form, but “Strict” prevents it in all cases.

c)

As mentioned above, “Lax” will prevent CSRF as long as the form does not use GET. Thus, the security of the form would be lowered if we used GET instead of POST. “Strict” prevents CSRF in both cases.

Exercise 3

a)

When we evaluate the code in this example, we can see multiple locations where user data is inserted into the HTML forms. There is no sign of escaping HTML, and is not being encoded, which means this code will have a XSS or cross-site scripting vulnerability. A successful attack could lead to the attacker's own code, for example a script containing JavaScript, to be added to the already existing HTML and the code would be executed.

b)

To fix this vulnerability I would create an escape HTML/encode method that would take every user data as input, escape all code to make sure it is interpreted as String and not code, and return the result. This would mean that if a user inserted <script>alert("Hey")</script>, it would be interpreted as String and presented as it is instead of executing it as code. HTML sanitizing could also be mentioned here, which is used to transform untrusted data to trusted data using sanitizing, before inserting them into the document.

c)

To exploit this code the attacker would want to use a script that extracts the element that handles the input, as well as the button for submitting the data. This script would have to check if a user is currently looking at our posted message, and then post our malicious message on behalf of the logged in user.

The code below is not 100% correct, I saved this task till last and ran out of time. But the explanation above a long with the code gives an ok explanation on how one would do it. I would on loading the page or message run a function that sends a message on behalf of the user.

```
<script>

document.getElementById("messageInput").onload = function()
{scaryFunc()};

let button = document.querySelector("input");


function scaryFunc() {

  document.getElementById("messageInput").innerHTML =  «XYZZY»;

  button.click()

}

</script>
```

Exercise 4

a)

To make sure users pick good passwords I would follow the NIST requirements for strong and safe passwords. I would force the users to make the passwords at least 8 characters long, but I would not set an upper limit. We could also force the user to use numbers and special characters in their passwords to make them more complex, but length of the password is the most important factor in having a secure password. The user needs to fill in two password forms before registering, making sure that he inserts the correct password. A check for common passwords would also be in place.

b)

XAuth SHA1 hash: SHA1 just like MD5, is a very old password authentication method and might not be secure anymore. SHA1 and other old methods should be avoided, and newer ones should take its place to stay on top. It is unlikely that SHA1 would prevent brute-forcing and rainbow table attacks seeing as it is very old and outdated.

XAuth Plaintext: Not very secure, the password is stored in plaintext and as 4-6 digit numeric code. This type of method can be easy to brute-force. Since it is stored in plaintext which would lead in easy extraction of the data if the database were breached.

ZAuth Argon2: Argon2 is very similar to SCrypt and blocks a lot of attacks. Brute-forcing would take time and be very hard for the attacker. Salting is in place which means rainbow tables would not be effective.

WAuth SHA256 + salt: Can be very easily brute-forced. Hashing is in place and rainbow tables will not be effective.

c)

SMS two factor authentication is in most cases secure, but not always. The codes sent to the user is stored in plaintext. This means that if attack get a hold of the users' messages, he can easily just grab the codes and use them for malicious intent. An attacker can also forge a fake website and trick the user into trusting it, making him insert private and sensitive data into the application. The more information the attacker has about the user and the application, the easier it is for him to trick the user into making mistakes. If the attacker has extracted enough personal information and data, he can use this to take over the targeted users accounts. Typical phishing attack.

As we see in the CVE database there are ways to get a hold of SMS content:

CVE-2018-14066: SQL injection in com.android.provider.telephony allows access to SMS messages by unauthorised apps (Limited to some specific phones)

d)

Public key based two-factor authentication is a modern way of authenticating users. The way this works is that the application trust that you are not getting attacked the first time you enter the application, which practically means when you register. Trust upon first use. When registering, a public and private key is assigned to you. The public key is sent out to the system that controls the login functionality. The private key never leaves the authenticator (you). When entering the system you will notify the system, that has your public key, that you indeed possess the private key, and you will be granted access. There are many typical crypto wallet scams where users give away their private key/seed phrase, and their funds are stolen. Private keys always stays with the authenticator.

Exercise 5

a)
This access control method is called ACL or Access Control List.

b)
This problem is a privilege escalation issue (confused deputy could also be mentioned here). Since this is a user who has had his privileges stripped, he should not be able to perform actions such as "write". But there is an error in the program, more specific with the bot, that lets muted or banned users send messages they should not be able to. The program is being exploited by the "attacker" in order to get privileges he is not supposed to have.

c)
In a capability based system we have a set of capabilities connected to the user, which then points at the object/channel. In a access control list system it is the channel that holds the permissions and points back at the user.

Alice (join, read, write, moderate) → SuperChannel etc.

The types of capabilities we can have is join, read, write and moderate. In this table I have chosen that Maleroy is currently muted in SuperChannel

User	**Capabilities**	**Channel**
Alice	join, read, write, moderate	SuperChannel
Bob	join, read, write	SuperChannel
Maleroy	Join, read	SuperChannel

d)
The users without moderation permissions should not be able to give themselves or other users permissions (other than join, read and write that they get upon joining). An overseer or admin should be in control of this, and give capabilities where need be. He would simply overwrite the current capability in the table and replace them with the updated one or remove them completely.

e)

Users or systems with moderation powers would be able to remove capabilities from users if need be. The set of capabilities for that specific channel would then be updated in the table.

For both d) and e) the tldr; is that authorized users should be able to delegate and revoke capabilities from users in the system.

f)
To fix this problem you will now have the bot check the users' capabilities when a user tries to post messages with the bot. If all the capabilities are met when the bot message is posted, there should be no issues. If a user is banned, he won't be able to send any messages using the bot in the respective channel. If a user gets banned after he has scheduled a message with the bot, it should not be posted.

A concrete example: Maleroy schedules a message saying "Alice sucks." In 1 hour from now, but just got banned from SuperChannel. Before the message is sent (or just instantly as Maleroy is banned) the bot should check if Maleroy’s capabilities has changed. If write permissions has been removed, the message should never post.