English

Universitetet i Bergen
Det matematisk-naturvitenskapelige fakultet

Exam in	:	INF226 Software Security
Semester	:	Autumn 2021
Time	:	09:00 – 12:00, 2nd of December 2021
Number of pages	:	7
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 (12 points)

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

a) Where in the memory layout of a C program does the call stack lie?
b) How can address space layout randomization (ASLR) make it more difficult for an attacker to perform return oriented programming?
c) What does the *Secure* flag on a cookie indicate?
d) Why should untrusted data not be inserted directly into a SQL query using string concatenation?
e) Why do we add salt to a key derivation function when implementing password based authentication?
f) What is the difference between static and dynamic program analysis?

Exercice 2 (12 points)

Services such as GitLab and GitHub provide a web interface for managing git-repositories. When pushing changes to the repositories, users must authenticate to the server hosting the repository. GitLab and GitHub provide two alternative authentication methods, authentication tokens and SSH public/private keys. We will here focus on the **SSH key pair method of authentication**.

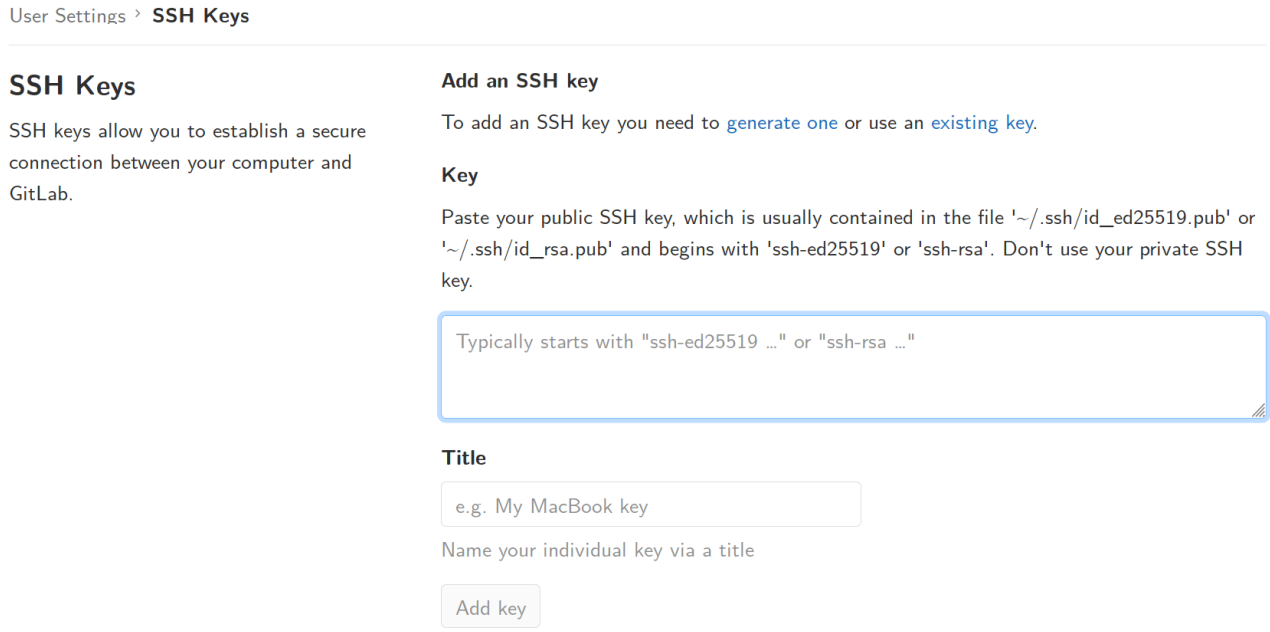

Figure 1: A screenshot of the key entering form on GitLab

The way SSH key pair authentication works is that the user generates a key pair on their local machine. Then they visit their GitLab settings page where **there is a form to enter their public key**. After the public key has been posted, the corresponding private key can be used to push changes to the repository.

Let us now imagine that the key-submission form has not been protected from cross-site request forgery (CSRF). For instance, the HTML code for this form (if simplified a bit) could look like:

```
<form id="new_key" action="/profile/keys" method="POST">
  <p>Paste your public SSH key.</p>
  <textarea name="key[key]" id="key_key"></textarea>

  <label class="label-bold" for="key_title">Title</label>
  <input name="key[title]" id="key_title">

  <input type="submit" name="commit" value="Add key">
</form>
```

a) Explain how an attacker could use CSRF to insert their own public key into a victim's list of trusted keys. What would the victim need to do in order for the attack to begin?
b) Would the CSRF be prevented if the SameSite flag on the session cookie was set to 'Lax'? Why/why not?
c) How would the security of the form be affected if method was set to "GET" instead of "POST"?

Exercice 3 (12 points)

Here is a simplified excerpt from the original InChat source code, which generates HTML code for showing an event in a channel.

```
// BEGIN CODE LISTING
/**
 * Render an event as HTML.
 */
private Consumer<Stored<Channel.Event>>
           printEvent(PrintWriter out, Stored<Channel> channel) {
    return (e -> {
        switch(e.value.type) {
            case message:
                out.println("<div class=\"entry\">");
                out.println("    <div class=\"user\">" + e.value.sender + "</div>");
                out.println("    <div class=\"text\">" + e.value.message);
                out.println("    </div>");
                out.println("    <div class=\"messagecontrols\">");
                out.println("        <form action=\"/channel/"
                          + channel.value.name + "\" method=\"POST\">");
                out.println("        <input type=\"hidden\" name=\"message\" value=\""
                          + e.identity + "\">");
                out.println("        <input type=\"submit\"
                                       name=\"deletemessage\"
                                       value=\"Delete\">");
                out.println("        </form><form style=\"grid-area: edit;\"");
                out.println("                     action=\"/editMessage\"");
                out.println("                     method=\"POST\">");

                out.println("        <input type=\"hidden\"
                                       name=\"message\"
                                       value=\"" + e.identity + "\">");
                out.println("        <input type=\"hidden\"
                                       name=\"channelname\"
                                       value=\"" + channel.value.name + "\">");
                out.println("        <input type=\"hidden\"
                                       name=\"originalcontent\"
                                       value=\"" + e.value.message + "\">");
```

```
                out.println("        <input type=\"submit\"
                                        name=\"editmessage\" value=\"Edit\">");
                out.println("        </form>");
                out.println("    </div>");
                out.println("</div>");
                return;
            case join:
                out.println("<p>" + formatter.format(e.value.time) + " "
                                  + e.value.sender + " has joined!</p>");
                return;
        }
    });
}
// END CODE LISTING
```

One of the issues in this code is that it is vulnerable to cross-site scripting (XSS).

a) Explain why the above code is vulnerable to cross-site scripting.
b) How would you fix this vulnerability?

An attacker could exploit this vulnerability by crafting a special message with JavaScript code which will execute when a user views the message in the chat.

c) What message would an attacker send in order to cause a user viewing the message to send a message to a channel? To be specific, assume that the channel is called "SuperChannel", and that the attacker wants the user to post the message "XYZZY". For reference, the source code for the message sending form is below.

```
<form class="entry" action="/channel/SuperChannel" method="post">
  <div class="user">You</div>
  <input type="hidden"
         name="newmessage"
         value="Send">
  <textarea id="messageInput"
            name="message"></textarea>
  <div class="controls\">
  <input type="submit"
         name="send"
         value="Send"></div>
</form>
```

Exercice 4 (10 points)

A local dance club and a local music studio has an agreement that, for a small fee, visiting musicians playing in the club can use studio while visiting – provided it is not already booked. In order to minimise the organisational overhead, the music studio decides to put a booking system online for the musicians.

You are tasked with getting the **authentication mechanisms** up and running for this booking site, so that the musicians can register and log in. Part of the specification is that it should use password authentication, so that is what you will implement.

a) How would you ensure that the users are likely to pick good passwords? What requirements and other measures would you put in place?

For the platform you are building the site on, you find four plugins for storing passwords. Their details are specified below.

Name	Storage	Password format
XAuth	SHA1 hash	ASCII
YAuth	Plaintext	4–6 digit numeric code
ZAuth	Argon2	Unicode
WAuth	SHA256 + salt	Unicode

b) Compare the four mechanisms above, from the perspective of a potential breach of the database. How would the various mechanisms fare against brute force, dictionary and rainbow table attacks?

After some deliberation, you decide to add a second factor to your authentication mechanism. At first you consider SMS verification codes.

c) How can SMS verification codes be vulnerable to phishing attacks, such as tricking the user to visiting a proxy site?

You then read about WebAuthn, which is two-factor authentication based on public key cryptography.

d) How could public key based two-factor authentication protect against a malicious proxy?

Exercice 5 (14 points)

In this exercice we will consider a hypothetical chat system, called *SafeChat*. This system resembles InChat which was in the mandatory assignment, but we imagine that the issues we found in the assignments have been fixed and that the system now has an access control system. There has also been added a notion of *presence* where you can see which users in the channel are currently logged in.

The system has the following permissions:

Permission :
Join
Read
Write
Moderate

The Join permission gives access to join the channel and see the list of other joined users. Read lets a user read the messages in the channel, and write lets a user post messages to the channel. The Moderate permission gives access to delete messages and set permissions for other users.

The access control is then organised in a table according to the schema:

User	Channel	Permission
...	...	...

`User` is a foreign key to the identity field of a user, `Channel` a foreign key to the identity field of a channel, and `Permission` a foreign key to the above permission table.

Example: In a channel called SuperChannel, with Alice, Bob and Mallory all chatting, Alice being the moderator, the table would look like:

User	Channel	Permission
Alice	SuperChannel	Join
Alice	SuperChannel	Read
Alice	SuperChannel	Write
Alice	SuperChannel	Moderate
Bob	SuperChannel	Join
Bob	SuperChannel	Read
Bob	SuperChannel	Write
Maleroy	SuperChannel	Join
Maleroy	SuperChannel	Read
Maleroy	SuperChannel	Write

Sometimes Maleroy misbehaves, and Alice will mute her, taking away her writing privileges:

User	Channel	Permission
Alice	SuperChannel	Join
Alice	SuperChannel	Read
Alice	SuperChannel	Write
Alice	SuperChannel	Moderate
Bob	SuperChannel	Join
Bob	SuperChannel	Read
Bob	SuperChannel	Write
Mallory	SuperChannel	Join
Mallory	SuperChannel	Read

a) Which kind of access control system is this? (Of the kinds we have discussed in class.)

The developers of SafeChat then implemented a bot – a program which joins the chat

as a client and can perform various actions at the command of the users. For instance, if Bob is not present in the chat at the moment, Alice can issue a command to the bot to write a message when Bob returns:

```
@BOT say "Hi Bob!" in SuperChannel when Bob is present
```

The messages can also be timed:

```
@BOT say "Happy New Year!" in SuperChannel at 2022-01-01T00:00
```

The bot was super useful – but a problem was soon discovered: By design, the bot had `Write` access to all channels. It needed this to perform its function. So, Mallory, who had been banned from SuperChat for bad behaviour, started giving the bot commands to post messages there for her:

```
@BOT say "Alice is STUPID!" in SuperChannel when Alice is present
```

At first the developers tried to go around this issue by implementing checks in the bot code: When a user issues a say-command, the bot checks that they are in the joined list of that channel. But it was still not certain that the user had actual write access – so this only worked when Mallory was banned, not muted.

Another problem with this approach was that Mallory would join a channel, fill the bot with obscene messages for the future, and then leave. Banning Mallory after the fact would then not stop the flood of messages from the bot.

b) What general class of security problems does the bot issue belong to? Explain in some detail why the problem belongs to this class. *Hint: this class of problems is inherent to this kind of access control system.*

It then occurred to the developers that perhaps a completely different model of access control would be needed to solve this problem.

After some research, they decided to go with a *capability based access control system*. **The idea would be that the bot would use the command issuer's capabilities when performing actions.**

c) Suggest a table schema for a capability based access control system for SafeChat, and explain your design. The system should have centrally controlled capability ownership stored in the database **or** unguessable tokens – choose what you find most appropriate.

An important feature in a capability based access control system is the ability to transfer capabilities from one user to another. And in SafeChat it would also be important to be able to revoke a capability.

d) How would transferring capabilities between users be done in the database schema you outlined?
e) How would revoking capabilities between users be done in this system?
f) How should the capabilities be organised for the bot to prevent the issues SafeChat has been struggling with? Give a concrete example, for instance using the situation with SuperChannel.