NTNU

Department of (Computer and Information Science)

Examination paper for (TDT4237) (Software Security)

Academic contact during examination: Jingyue Li

Phone: 9189 7446

Examination date: 13-May-2019

Examination time (from-to): 9.00

Permitted examination support material: D

Other information:

Language: English

Number of pages (front page excluded): 7

Number of pages enclosed: 1

Introduction

In this course, the written exam will count 70% of the final grade, and the remaining 30% of the final grade comes from the compulsory exercises.
So, your final grade of this course will be:
(Points you get from this written exam) * 70% + your grade of compulsory exercises.

If you feel that any of the problems require information that you do not find in the text, then you should

- Document the necessary assumptions
- Explain why you need them

Your answers should be brief and to the point.

Problem 1 – (40 points)

1) (2 points) Explain why prepared statement and bind variables can defend against SQL injection attacks?

The root cause of SQL injection attack is that the users' inputs are interpreted as control. Prepared statements and bind variables can defend against SQL injection because only the prepared statements will be compiled as commands. The users' input in the bind variables will always be interpreted as data rather than commands.

2) (3 points) Explain the three possible ways to store session tokens and compare their advantages and disadvantages.

- (1 point) Embed in all URL links, e.g., https://site.com/checkout? sessionToken= 1234
  - Advantage: easy to implement
  - Disadvantage: Referer can leaks URL session token to 3rd parties, and users may publish URL (with token info.) in blogs

- (1 point) In hidden form field, e.g., <input type= "hidden" name = "sessionToken" value = "1234">
  - Advantage: Can be used to defend against CSRF attack
  - Disadvantage: Do not work for long-lived sessions and every protected web page must embed this hidden token

- (1 point) Browser cookie, e.g., setcookie: sessionToken = 1234
  - Advantage: The browser will manage the cookie and is easy to be implemented for long-lived sessions than using a hidden form field.

- Disadvantage: Browser sends cookies with every request, even when it should not (e.g., vulnerable to CSRF attack)

3) (4 points) Explain how to identify if a web site is vulnerable to CSRF attacks.

CSRF is an attack that forces an end user to execute unwanted actions on a web application in which he/she is currently authenticated. A test case for identifying vulnerable to CSRF attacts can be constructed as follows:
Blackbox testing (4 points)

1. Identify a URL on your site where a CSRF attack could have a negative effect on your site. For this example, lets say a GET request to http://mysite.com/account/del will delete the account you are logged in as
2. Next, create a basic HTML page that is totally separate from the site you are testing. On this HTML page include the following <img src="http://mysite.com/account/del" width="0" height="0">
3. Next, create a dummy account on the site you want to test, and log into that account.
4. With the session still active open the basic HTML page you created in the same browser.
5. If the account gets deleted, you have a CSRF vulnerability

Or read the source of the code to see if any of the following defense is implemented (4 points)

- Authentication again
  - E.g., require Authentication again before the money transfer
    - Password
    - BankID
- Validation via action token, i.e., combine tokes in the cookie and in the hidden form field

4) (4 points) Explain what a clickjacking attack is and how to defend against the clickjacking attack.

(2 points) A clickjacking attack is when an attacker uses transparent layers to trick a user into clicking a button or link on the top level page (which is transparent and malicious) when they intended to click on the button or link below the top level page (which looks to be a benign page).

To prevent clickjacking, you can implement any of the following defense (You will get 2 points if you can list any of the option below)

- Preventing other web pages from framing the site you want to defend (e.g., Defending with X-Frame-Options Response Headers)
- Employing defensive code in the UI to ensure that the current frame is the most top-level window

5) (4 points) Explain why using the ECB model in block cipher is insecure and how to deal with it.

(2 points) When using the ECB model in a block cipher, all blocks use the same key sets generated from identical key and key expansion function. It does not hide data patterns well.

(2 points) To deal with the insecure issue of ECB, the right mode of operation (e.g., Cipher Block Chaining (CBC)) needs to be used. In CBC mode, the ciphertext of a previous block will be used as the key for encrypting/decrypting the next block. In addition, an initialization vector (i.e., a random number) is used as the encryption/decryption key of the first block.

6) (4 points) Explain digital signature, Certification Authority (CA), and how they are related to the SSL/TLS handshake process.

(1 point) When Bob signs a document, he will first hash the content of the document and then uses his secret key SK_{Bob} and F^{-1} to generate the signature. Bob will send his public key PK_{Bob}, the document m, and the signature to Alice, who wants to read m and verify if m is signed by Bob. At Alice side, she will use the F function, the $PK_{Bob,}$ and the Signature of Bob to generate a value the compares the value with the hash of m. If the hash of m matches the generated value, Alice will believe that the document is digitally signed by Bob.

(1 point) A certification authority (CA) is an entity that issues digital certificates. The main task of CA is to certify the ownership of a public key by the named subject of the certificate.

(2 points) Preparation for the handshaking process: First, Public Keys (PK) of Certification Authority (CA), i.e., PK-CA, is embedded in all browsers and web servers when browser and servers are shipped. The server needs to generate its Secret Key (SK-Server) and Public key (PK-Server). The PK-Server needs to be digitally signed by CA using CA's private key (SK-CA) to generate the certificate of the server. So, the certificate contains information about the server's Public Key, i.e., PK-Server.

Handshaking process:

- Step 1: the web browser sends a request for handshaking to the server.
- Step 2: the server replies the request by sending back its certificate. From the certificate, the web browser can get information about the PK-Server because the web browser has PK-CA and PK-Server is signed by SK-CA, i.e., the Secret Key of CA.
- Step 3: the web browser generates the "shared key." The "shared key" is encrypted using PK-Server, and is sent back to the server.
- Step 4: the server decrypts the "shared key" using its SK-Server.

The handshaking process finishes. Both the web browser and the server have the "shared key."

7) (4 points) Explain the Biba model and why it can help improve integrity.

The Biba model describes a set of access control rules in order to ensure data integrity.

The policies of the Biba model are:

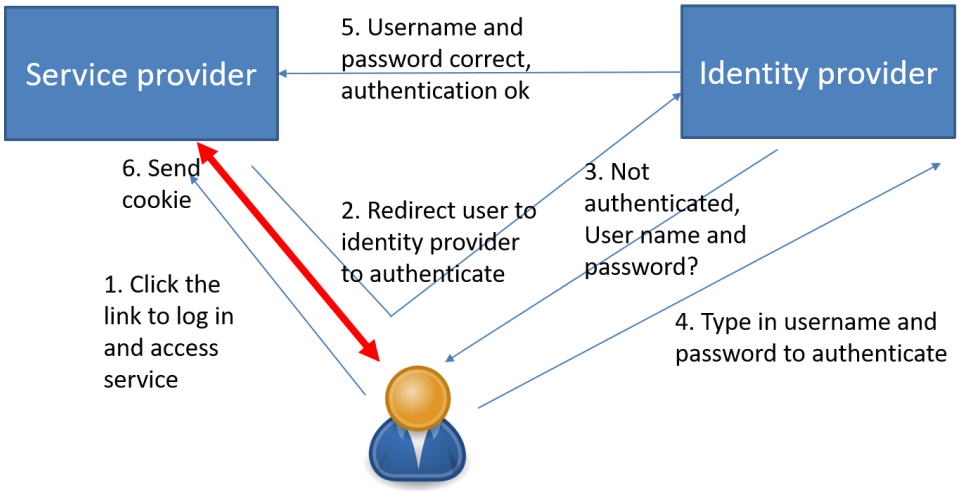
(2 points) (No write up) meaning low classified subject could not modify the high classified object. For example, software downloaded from the web cannot write to OS.

(2 points) (No read down) meaning high classified subject will not read and use data of the low classified object. For example, the signaling sys. does not use data from passenger info. sys.

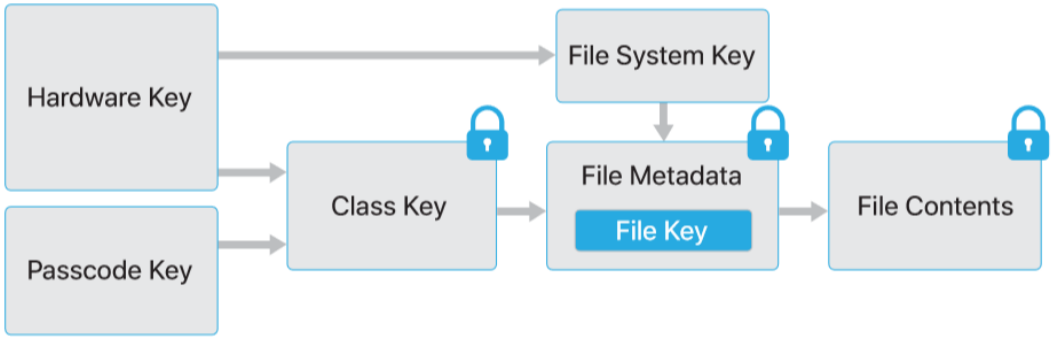
8) (3 points) Explain the components and process of Single Sign-On (SSO).

Single sign-on is an authentication process that allows a user to sign on in multiple applications with one set of credentials (authentication across domains). The main components are the identity provider (a centralized authentication server which validates user identities and issues access tokens), the service provider (which requests from the identity provider to authenticate a specific user), and the users.

(2 points) The process is below:



9) (4 points) Explain how files are encrypted in iOS.



(1 point) Every time a file on the data partition is created, Data Protection creates a new 256-bit key (the "per-file" key) and gives it to the hardware AES engine, which uses the key to encrypt the file as it is written to flash memory.

(1 point) The per-file (or per-extent) key is wrapped with one of several class keys, depending on the circumstances under which the file should be accessible.

(1 point) The metadata of all files in the file system is encrypted with a random key, which is created when iOS is first installed or when the device is wiped by a user.

(1 point) The class key is protected with the hardware UID and, for some classes, the user's passcode.

10) (4 points) Explain the proof-of-work consensus model of the blockchain, what security attack such a model can defend against, and the disadvantage of the model.

(2 points) In the proof of work (PoW) model, a user publishes the next block by being the first to solve a computationally intensive puzzle. The solution to this puzzle is the "proof" they have performed work.

(1 point) The use of a computationally difficult puzzle helps to combat the "Sybil Attack" – a computer security attack (not limited to blockchain networks) where an attacker can create many nodes (i.e., creating multiple identities) to gain influence and exert control.

(1 point) The disadvantage is that it requires a lot of computation.

11) (4 points) Explain what web application firewall of Azure is and why it can only partially defend against the session fixation attack.

(2 points) Web application firewall for Azure is a feature that can be configured to filter input/output traffic of the web application deployed in the cloud. The web application firewall can be used to help filter out malicious inputs by setting up the blacklisting or whitelisting rules.

(2 points) However, if the session is not managed by the application firewall, the application firewall cannot defend against the session fixation attack. The web application must make sure that the session expires when the lifetime of the token issued by Azure expires.

Problem 2 – (30 points in total)

For each of the code snippets listed below, your task is to:

- Identify all security vulnerability and their locations in the code (Note: you may find more than one vulnerabilities in one code snippet. You need to list and identify all of them.)

- Explain why these are security vulnerabilities/issues and how to exploit the vulnerabilities to attack the app
- Fix the code (You may use pseudo-code for this. Remember to explain your solution).

To present your answsers in a structured manner, you may format your answers like:
Code Snippet 1: vulnerability 1: ... (in Line ...); It is vulnerable because ... and the method to exploit the vulnerability is ...; The fix could be ....
...
Code snippet 2: ...

Code snippet 1

Source: guyrutenberg.com

```
1. PREFIX = '/home/user/files/'
2. full_path = os.path.join(PREFIX, filepath)
3. read(full_path, 'rb')
```

Vulnerability: line 2 directory path traversal attack. This can be exploited by inputting a filepath that starts with "../", which would give an attacker access to higher levels of the folder hierarchy than what is intended.

Fix: Sanitize the filepath variable to remove possible malicious inputs, such as “../”

Code snippet 2

Source: lets-be-bad-guys project

```
1.  users = {
2.
3.      '1': {
4.          'name': 'Foo',
5.          'email': 'foo@example.com',
6.          'admin': False,
7.      },
8.
9.      '2': {
10.         'name': 'Bar',
11.         'email': 'bar@example.com',
12.         'admin': True,
13.     }
14. }
15.
16. def user_profile(request, userid=None):
17.
18.     env = {}
19.
20.     user_data = users.get(userid)
21.
```

```
    if request.method == 'POST':

        user_data['name'] = request.POST.get('name') or user_data['name']

        user_data['email'] = request.POST.get('email') or user_data['email']

        env['updated'] = True

    env['user_data'] = user_data

    env['user_id'] = userid

    return render (request, '/profile.html', env)

```

/profile.html

```
{% block content %}

{% if updated %}

<p>Updated your user profile, thanks!</p>

{% endif %}

<form action="{% url ' code-profile' user_id %}" method="POST">

{% if user_data.admin %}

<p>You are an admin! Use it wisely.</p>

{% endif %}

<p><label for="name">Name:</label>

    <input type="text" name="name" id="name" value="{{ user_data.name }}"></p>

<p><label for="email">Email:</label>

    <input type="email" name="email" id="email" value="{{ user_data.email }}"></p>

<p><label for="password">Password:</label>

    <input type="password" name="password" id="password"></p>

<p><input type="submit"></p>

</form>

{% endblock %}
```

/urls.py

```
1. url(r'^folder/users/(?P<userid>\d+)$',
2. exercises.user_profile, name='code-profile'),
```

Note: "**folder" is the folder which profile.html locates**

Vulnerability: There is an "Insecure Direct Object References" vulnerability in the def user_profile function. An attacker can manipulate the user id in the URL to see profiles of other users.

Fix: Proper access control must be implemented so that users can only see the information they are allowed to get access.

Code snippet 3

Source: wiki.sei.cmu.edu

```
1. import java.io.*;
2.
3. class DeserializeObj {
4.
5.   public static Object deserialize(byte[] buffer) throws IOException,
   ClassNotFoundException {
6.
7.     Object ret = null;
8.
9.     try (ByteArrayInputStream bais = new ByteArrayInputStream(buffer)) {
10.
11.      try (ObjectInputStream ois = new ObjectInputStream(bais)) {
12.
13.        ret = ois.readObject();
14.
15.      }
16.
17.    }
18.
19.    return ret;
20.
21.  }
22.
23. }
```

Vulnerability: insecure deserialization in method "deserialize." An attacker could compromise the data stream by inserting malicious code into it. When the data stream is deserialized, the malicious code can be exploited.

Vulnerability fix: Sanitize the deserialized data before using them

Code snippet 4

Source: CWE-643

XML doc

```
<users>
  <user>
    <login>john</login>
    <password>abracadabra</password>
    <home_dir>/home/john</home_dir>
  </user>
  <user>
    <login>cbc</login>
    <password>1mgr8</password>
    <home_dir>/home/cbc</home_dir>
  </user>
</users>
```

Java code used to retrieve the home directory based on the provided credentials

```
1. XPath xpath = XPathFactory.newInstance().newXPath();

2. XPathExpression xlogin = xpath.compile("//users/user[login/text()='" +
   login.getUserName() + "' and password/text() = '" + login.getPassword() +
   "']/home_dir/text()");

3. Document d =
   DocumentBuilderFactory.newInstance().newDocumentBuilder().parse(new
   File("db.xml"));
4. String homedir = xlogin.evaluate(d);
```

Vulnerability: line 2 in Java code, which is a XPath injection vulnerability. Attackers can type in malicious input to bypass the check of username and password, i.e., bypassing the authentication. Assume that user "john" wishes to leverage XPath Injection and login without a valid password. By providing a username "john" and password "' or '='" the XPath expression now becomes `//users/user[login/text()='john' or ''='' and password/text() = '' or ''='']/home_dir/text()`

which, of course, lets user "john" login without a valid password, thus bypassing authentication.

Fix: Using blacklisting and whitelisting to sanitize the inputs from users or Use parameterized XPath queries (e.g. using XQuery). This will help ensure separation between the data plane and control plane.

Code snippet 5

Source CWE-601

```
1. public class RedirectServlet extends HttpServlet {
```

```
2.      protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

3.          String query = request.getQueryString();

4.          if (query.contains("url")) {

5.              String url = request.getParameter("url");

6.              response.sendRedirect(url);

7.           }

8.      }

9.  }
```

Vulnerability: Line 5, which is a URL redirection vulnerability. An attacker could use the RedirectServlet as part of an e-mail phishing scam to redirect users to a malicious site.

Fix: Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications.

Problem 3 – (30 points in total)

Case description:

Company COOL wants to make a SocialUber app to facilitate sharing of the private autonomous car as a taxi and to mitigate possible safety risks. The app will have both web and mobile versions.

Three kinds of stakeholders will be the users of the app, namely the autonomous car owners, the autonomous car users, and friends/relatives of the autonomous car users.

- The autonomous car owners want to earn money by renting out their autonomous car as a taxi when they do not use the car. They will use the app to register their information and the information and availability of the car. They also want to use the app to receive payment from the car user.
- The autonomous car users want to order and use an autonomous car as a taxi. They will use the app to order the autonomous car, to

see the location of the ordered car, and to make payment after using it.
- When start using the autonomous car, a car user can use the app to inform his/her friends/relatives about the location of the car in the whole journey, so that his/her friends can trace where the car is and can help inform police or ambulance, if an incident happens.

To use the app, the autonomous car owner needs to register and fill in the following information:
- Username and password (mandatory)
- Real name (mandatory)
- Email address (mandatory)
- Phone number (mandatory)
- Home address (mandatory)
- Information of the car (e.g., registration ID, size of the car, and price to be used as a taxi) (mandatory)
- A bank account to receive the payment (mandatory)

The car owner needs to log in the system to change the information and to update the availability of the car.

The autonomous car user needs to register and fill in the following information:
- Username and password (mandatory)
- Real name (mandatory)
- Email address to receive a receipt (optional)
- Phone number (mandatory)
- Credit card information, if the user wants to use one-click payment (optional)
- The username of the friend/relative he/she wants to link to (optional). When filling such information, an SMS will be sent to the phone of the friend/relative and for approving the link. The information will be saved to the app, only if the friend/relative approves the link.

The autonomous car user needs to log in the system to order the car.

A friend/relative of the car user needs to register and fill in the following information:

- Username and password (mandatory)
- Phone number (mandatory)

The friend/relative of the car user needs to log in the system to see the location of the car in use.

All filled-in information by car owners, car users, and their friends will be encrypted and saved at the server side of the app.

A simplified usage scenario is as follows:

- A car user opens the app and types in the starting and ending address of a journey.
- The app searches for available autonomous cars nearby and returns a list of their possible arrival time and prices.
- The user chooses an autonomous car from the list.
- The app sends SMS to the car owner and asks the car owner to approve the request of using the autonomous car.
- The autonomous car owner approves the request.
- The app sends a confirmation code to the car user.
- The autonomous car leaves home and arrives at the starting address of the journey.
- The car user opens the door of the autonomous car using the confirmation code he/she receives from the app.
- The car user sits in the car.
- The care user can open the app to choose from the list in the app one friend/relative to monitor the journey.
- In the whole journey, the app will keep updating the location of the car to the friend/relative of the car user. The friend/relative can see the car location after logging in.
- The car arrives at the destination.
- The car user leaves the car, closes the door, and clicks the button “journey completed” in the app. The friend/relative of the car user will get informed in the app that the journey is complete if he or she is observing the journey.
- The app will calculate and show the price to the user. The user can then click the “one-click payment” button to pay the journey if the

credit card information is saved in advance. The user can also pay the journey through filling in credit card information. If the user does not make the payment, he or she could not use the app to make a new order. After the payment, 0.1% of the payment will be transferred to Company COOL as the cost of using the SocialUber app.

- After the payment, the app notices car owner about the payment.
- The app will send a receipt to the user using email if his/her email address to receive a receipt is saved.

Your task is to make a risk-based assessment of this application based on the RMF (Risk Management Framework).

Your tasks include:

- Task 1: Identify business goals, business assets, and business risks (5 points).

  Business assets: user identities, company's reputation, emails, and their contents, admin credentials, payment transactions, personal information, encryption keys
  Business goals: gain reputation, gain more users, increase revenues, deliver reliable application, reduce costs for development
  Business risks: user data is stolen by attackers, application unavailable, the application is hijacked by the attacker

- Task 2: Identify at least 10 technical risks related to the SocialUber app using threat modeling and explain how threat modeling is performed (12 points). (Note: You do not need to draw the threat modeling graphs. However, you need to explain briefly how the threat modeling, e.g., misuse cases and attack trees, are applied to help you identify the technical risks.)

  There are many possible technical risks. Some examples are below. Students will get one point by listing every proper and relevant technical risk (max 10 points for 10 technical risks).

  - TR1: SQL- injection with known SQL-injections like 'admin'--' leading to an attacker getting admin access to the application to steal credit card information of the user.

- TR2: DDOS attack by an attacker sending to many packets to the application and the system goes down, and no car can be booked
- TR3: Weak password policies leading to users creating bad passwords which can be exploited by an attacker to impersonate others to order and use the taxi without payment.

Students must be able to show that the technical risks are derived from misuse case analysis or attack tree analysis. If students just list the high-level terms of risks (e.g., SQL injection) without being able to show the links between the risks and the case, 2 points will be deducted.

- Task 3: Derive security requirements from each technical risk identified, and design and describe black-box penetration test cases (including test steps and expected results of each step) to verify each derived security requirement (10 points)

For each listed technical risk, one corresponding security requirement and at least one test case must be listed. The security requirement is basically to define how the risk should be mitigated. The test case must have test steps and expected results of each step. Students will get one point if a security requirement and its corresponding test case are listed.

- Task 4: This SocialUber app must be compliant with the General Data Protection Regulation (GDPR). Discuss how to address the privacy issue of this app (3 points)

(1 point) List of some privacy data, such as user Phone number, email address, Credit card information, and so on.

(2 points) Explain how to deal with any two of the five principles of privacy (i.e., Transparency, Purpose, Rights, Fair use, and Minimalisation) in this case study.