ECM2426

(with Answers)

UNIVERSITY OF EXETER

COLLEGE OF ENGINEERING, MATHEMATICS AND PHYSICAL SCIENCES

COMPUTER SCIENCE

Examination, January 2021

Network and Computer Security

Module Leader: Prof Achim Brucker

Duration: TWO HOURS + 30 MINUTES UPLOAD TIME

Answer ALL questions.

Question 1 and Question 2 are worth 20 marks each. Question 3 and Question 4 are worth 30 marks each.

Write clearly in the sense of logic, language, and readability. The clarity of your arguments and explanations affects your grade.

The marks for this module are calculated from 70% of the percentage mark for this paper plus 30% of the percentage mark for associated coursework.

This is an OPEN BOOK examination.

Question 1

Security Fundamentals & Access Control

(a) Many modern websites allow users to log-in using external services such as Google, Facebook, or Github. This mechanism is called *single sign-on*.

Briefly explain two threats to the core information security goals (confidentiality, integrity, and availability) of a user using such a single sign-on service. Name which goal is violated by each of your threats.

(4 marks)

Example threats are:

- **Confidentiality:** The single sign-on provider can learn which other websites/services a user is using.
- **Availability:** If the single sign-on service is not available, the user is also not able to log into the third party sites using this service.

Marking Scheme:

- *[2]* for each correctly identified and classified threat (total: **[4]**).
- *[1]* for a correct threat that is classified (wrong security goal named).

(b) Consider two configurations of a desktop computer using Linux as operating system:

- Scenario A uses a separate user account with standard privileges and a root account with a different password. The main user knows both passwords.
- Scenario B uses sudo (e.g., a "run as administrator" feature) that allows the regular user to run programs, *after confirmation* as root. The user does not know the password for the root account.

Considering the scenarios A and B, does the use of sudo (or a similar system) improve the overall system security. Briefly argue either way.

(4 marks)

If, as given in the description, assume strong passwords, then

- Scenario A has, e.g., the *advantage* that a local attacker (i.e., an attacker also having a local system account) compromising the users' account, has gained no advantage wrt compromising the system administrator account. It has, e.g., the *disadvantage*, that the user needs to know two passwords, which might increase the likelihood for choosing weak passwords (or using the same).

- Scenario B has, e.g., the *advantage* that the user does not need to remember a second password. It also has the *advantage* that sudo allows for a fine-grained delegation of system administrator privileges. It has, e.g, *disadvantage* that it turns the user account itself into a more privileged account compared to a user account that has no additional system administrator privileges enabled via sudo.

Based on the example given, one can argue either way. In particular when full system administrator privileges are transferred via sudo, the system can be considered less secure.

Marking Scheme:

- **[2]** for a correct example (e.g., one advantage/disadvantage of A and B)
- **[2]** for a conclusive argument (can go either way).

(c) Consider a procurement system in a company that allows member of staff to order items. The system has the following organisational roles and tasks:

- members of *staff* can *order* goods
- *manager*s can *order* goods
- orders from regular staff needs to be *approve*d by *budget* holders or *managers* (i.e., managers can approve their own orders).

In the following, we only focus on the access control aspect of this business process. In particular, you can assume that the system ensures that for all ordered goods, an approval is requested.

Consider the following Role-Based Access Control (RBAC) configuration:

$$
\begin{aligned}
\mathit{USERS} &= \{\text{alice}, \text{bob}, \text{charlie}, \text{dave}, \text{eve}\}\\
\mathit{ROLES} &= \{\text{staff}, \text{manager}, \text{budget}\}\\
\mathit{PERMISSION} &= \{\text{order}, \text{approve}\}\\
\mathit{UA} &= \{\\
&\quad(\text{alice}, \text{staff}),\\
&\quad(\text{bob}, \text{staff})\\
&\quad(\text{charlie}, \text{budget}),\\
&\quad(\text{dave}, \text{budget}),\\
&\quad(\text{dave}, \text{manager}),\\
&\quad(\text{eve}, \text{manager})\\
&\}\\
\mathit{PA} &= \{\\
&\quad(\text{staff}, \text{order}), (\text{manager}, \text{order})\\
&\quad(\text{approve}, \text{manager}),\\
&\quad(\text{budget}, \text{approve})\\
&\}
\end{aligned}
$$

(i) Formally show that Dave can approve his own orders.

(4 marks)

(ii) Briefly argue that the given RBAC configuration ensures that orders from regular staff need to be approved by a different subject (which has to be either in role budget or staff).

(4 marks)

(iii) Formally define the invariant that needs to be maintained when new users are added, to ensure that the orders from regular staff always need to be approved by a different subject (either a manager or a budget holder)

(4 marks)

(i) Dave's permissions can be computed as follows:

$$\begin{aligned} AC(\{\text{dave}\}) &= (PA \circ UA)(\{\text{dave}\}) \\ &= PA(\{\text{ budget}, \text{manager}\}) \\ &= \{\text{order}, \text{approve}\} \end{aligned}$$

Marking Scheme:

- **[4]** for a correct formal solution
- at most *[2]* for an informal argument

(ii) We can observe that

- only charlie, dave, and eve can approve orders and
- neither charlie, dave, nor eve are in the role staff.

Thus, we can conclude that no member of staff is able to approve orders.

Marking Scheme:

- full marks (**[4]**) for any conclusive informal argument
- at most *[2]* for a mostly correct solution.

(iii) We need to ensure that no member of the role staff is also a member of the roles budget or manager:

$$\forall u \in \mathit{USERS}.\ \mathit{UA}(u) \cup \{\text{budget}, \text{manager}\} = \emptyset$$

Marking Scheme:

- full marks (**[4]**) for any correct invariant that is expressed formally
- at most *[2]* for a formally incorrect invariant that "captures the correct intent"
- at most *[2]* for a informal statement of the invariant

(Total 20 marks)

Question 2

PKIs & Cryptography

(a) Briefly explain two advantages of hybrid encryption systems, i.e., implementations that use both symmetric and asymmetric encryption schemes for encrypting data.

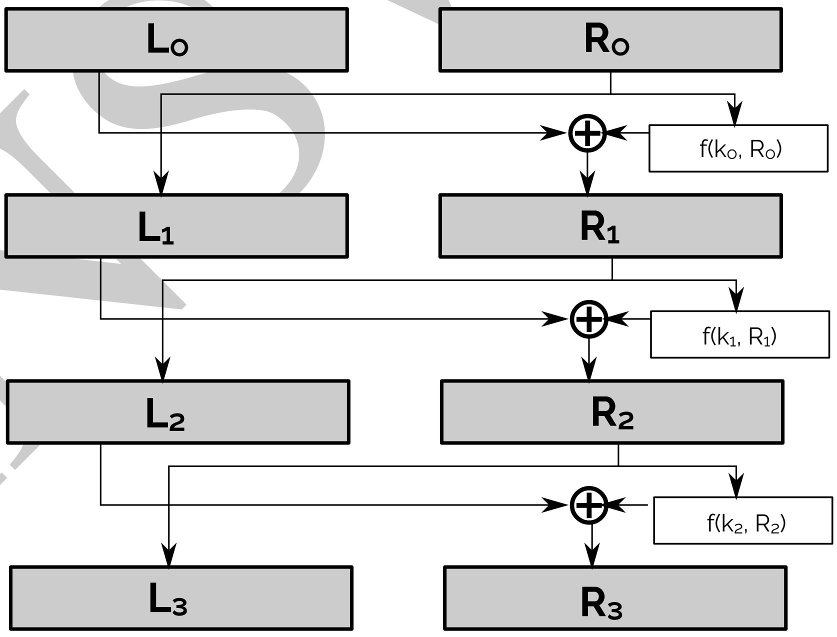
(4 marks)

- allows to use a long-term asymmetric key-pair that is highly security critical but only used for negotiating a short-lived symmetric session key.
- In general, symmetric encryption is significantly faster. Hybrid schemes allow using symmetric schemes for encrypting the data while asymmetric encryption is only used for encryption the (small) symmetric session key.

Marking Scheme:

- *[2]* for each correctly explained advantage (total: **[4]**).

(b) Consider the following DES-like symmetric encryption scheme:



with

$$f(k_i, R_i) = (k_i + R_i)$$

where _ $\oplus$ _ denotes the bit-wise *exclusive or* (xor) and _ $+$ _ denotes the bit-wise *or*. In the following, we use binary numbers.

In the following, we consider a configuration using the key

$$k = \underbrace{011}_{k_2}\underbrace{111}_{k_1}\underbrace{101}_{k_0}$$

(i) Encrypt the plain text:

$$m = \underbrace{000}_{L_0}\underbrace{111}_{R_0}$$

(3 marks)

A) Round 0: $L_0 = 000$, $R_0 = 111$, $f(101, 111) = 111$

B) Round 1: $L_1 = 111$, $R_1 = 111$, $f(111, 111) = 111$

C) Round 3: $L_2 = 111$, $R_2 = 000$, $f(011, 000) = 011$

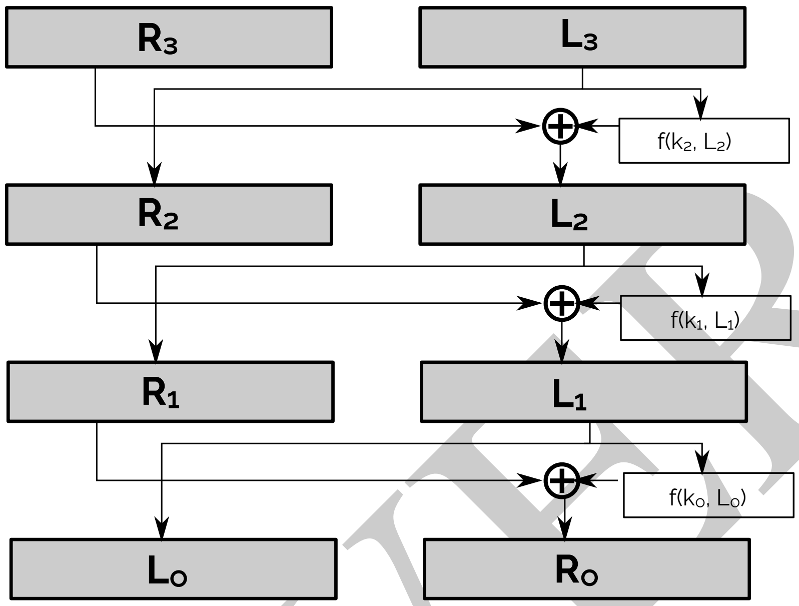
D) Result: $L_3 = 000$, $R_3 = 100$

Marking Scheme:

- *[1]* for each correct round (total: **[3]**)

(ii) Sketch the corresponding decryption scheme.

(3 marks)



Marking Scheme:

- *[1]* for each correct round (total: **[3]**)

(iii) Decrypt the cipher text:

$$c = \underbrace{000}_{L_3}\ \underbrace{100}_{R_3}$$

(3 marks)

A) Round 0: $R_3 = 100$, $L_3 = 000$, $f(011, 000) = 011$

B) Round 1: $R_2 = 000$, $L_2 = 111$, $f(111, 111) = 111$

C) Round 3: $R_1 = 111$, $L_1 = 111$, $f(101, 111) = 111$

D) Result: $R_0 = 111$, $L_0 = 000$

Marking Scheme:

- *[1]* for each correct round (total: **[3]**)

(c) Consider the textbook version of RSA. Prove mathematically that RSA is a reversible crypto scheme.

(7 marks)

Given the clear text m, the private key (e, n) and the public key (d, n), we need to show that:

$$(m^d \pmod n)^e \bmod n = (m^e \pmod n)^d \bmod n$$

This can be shown as follows:

$$\begin{aligned}(m^d \pmod n)^e \bmod n &= (m^d)^e \bmod n \\ &= m^{d \cdot e} \bmod n \\ &= m^{e \cdot m} \bmod n \\ &= (m^e)^d \bmod n \\ &= (m^e \bmod n)^d \bmod n\end{aligned}$$

Marking Scheme:

- **[3]** for stating the prove goal mathematically (at most *[1]* for an informal statement).
- **[4]** for the mathematical proof, at most *[2]* for a correct semi-formal argument.

(Total 20 marks)

Question 3

Security Protocols

(a) Do you consider the Dolev-Yao attacker model appropriate for security protocols that require a physical *proximity* between the agents (e.g., Bluetooth).

Briefly argue either way.

(4 marks)

The Dolev-Yao model does not include any notion of distance/proximity:

- One can argue that the Dolev-Yao model *is suitable*, as it over-approximates possible attacks: a protocol that is secure wrt the Dolev-Yao model is also secure under a weaker attacker model requiring physical proximity of the attacker.
- One can argue that the Dolev-Yao model *is not suitable*, as it rejects a lot of protocols that would be secure in practice.

Marking Scheme:

- **[1]** for stating that the Dolev-Yao model does not include any notion of proximity.
- **[3]** for a conclusive argument.

(b) Consider the following protocol:

Step 1: $A \longrightarrow B: \ \{N_A, A\}_{\mathrm{pk}(B)}$
Step 2: $B \longrightarrow A: \ \{\{N_A\}_{\mathrm{pk}(A)}, \{N_B\}_{\mathrm{pk}(A)}\}_{\mathrm{inv}(\mathrm{pk}(B))}$
Step 3: $A \longrightarrow B: \ \{N_B\}_{\mathrm{pk}(B)}$

(i) Briefly explain whether this protocol is secure (i.e., establishes a mutually authenticated and secure channel) or not.

- If it is secure, give an informal justification why it is secure.
- If an attack is possible, explain the attack, i.e., how it works and why it is possible.

(6 marks)

The protocol is not secure. The attacker can modify the second message to $B \longrightarrow A : \ \{N_A, B\}_{\text{pk}(A)}, \{N_A, B\}_{\text{pk}(A)}$, resulting in violating the authentication over N_B.

Marking Scheme:

- **[3]** for stating that the protocol is not secure.
- **[3]** for a correct attack

(ii) What is the *minimal* initial knowledge set for the agents A and B that ensures that the protocol is executable?

(5 marks)

- A: A, B, $\text{pk}(\_)$, $\text{inv}(\text{pk}(A))$
- B: B, $\text{pk}(\_)$, $\text{inv}(\text{pk}(A))$

Marking Scheme:

- **[2]** for knowledge of A
- **[3]** for knowledge of B

(c) Consider the following intruder knowledge:

$$\begin{aligned} M = \Big\{ & \{\!|m|\!\}_{g(n_1)}, \{\!|\{\!|h(m)|\!\}_{n_2}|\!\}_{g(n_1)}, \{n_1\}_{\text{inv}(\text{pk}(a))}, \{\!|m|\!\}_{n_2}, \\ & n_1, n_2, \text{pk}(a), \text{pk}(b), \text{pk}(i), \text{inv}(\text{pk}(i)), \\ & \{secret\}_{\text{pk}(b)}, \{\{\!|secret|\!\}_{h(\{\!|m|\!\}_{n_2})}\}_{\text{inv}(\text{pk}(a))} \Big\} \end{aligned}$$

where g and h are functions (i.e., $g, h \in \Sigma$ and, in particular, g is a public function: $h \in \Sigma_P$).
Prove formally that the intruder can learn the message "$secret$".

(15 marks)

$$\dfrac{\dfrac{\dfrac{}{\{\{\!|secret|\!\}_{h(\{\!|m|\!\}_{n_2})}\}_{\mathrm{inv}(\mathrm{pk}(a))} \in \mathcal{DY}(M)}\,\textsc{Axiom}}{\{\!|secret|\!\}_{h(\{\!|m|\!\}_{n_2})} \in \mathcal{DY}(M)}\,\textsc{OpenSig} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\{\!|m|\!\}_{n_2} \in \mathcal{DY}(M)}\,\textsc{Axiom} \quad \dfrac{}{n_2 \in \mathcal{DY}(M)}\,\textsc{Axiom}}{\{\!|m|\!\}_{n_2} \in \mathcal{DY}(M)}\,\textsc{DecSym}}{h(\{\!|m|\!\}_{n_2}) \in \mathcal{DY}(M)}\,\textsc{Composition}}{secret \in \mathcal{DY}(M)}\,\textsc{DecSym}$$

Marking Scheme:

- **[15]** FOR A CORRECT PROOF.
- UP TO *[10]* DEDUCTION FOR A CORRECT INFORMAL PROOF.
- UP TO *[10]* FOR A WRONG BUT FORMAL PROOF ATTEMPTS THAT PARTIALLY USE THE CORRECT REASONING (I.E., CORRECT SUB-PROOFS).
- NO DEDUCTION FOR SMALL MISTAKES (NO RULE NAMES, OBVIOUS TYPOS, ETC.)

(Total 30 marks)

Question 4

Application Security

(a) Imagine you are the security expert in a development team building a small web application for advertising internships to students of the University of Exeter. In particular:

- authenticated representatives from registered companies should be able to
  - post new internships,
  - review applications from students, and
  - close an internship, if the position has been filled.
- authenticated students from the University of Exeter should be able to
  - browse open internships,
  - apply for internships (including uploading of supporting documents),
  - share open internships with their friends (e.g., via email).

For each threat category of the STRIDE threat modelling approach,

- identify one concrete threat/attack and
- propose one countermeasure to mitigate the identified threat.

(12 marks)

- **S**poofing Identity:
  - *Threat:* A student could try to spoof the identity of a company representative.
  - *Countermeasure:* Use second-factor authentication to avoid password guessing attacks.
- **T**ampering with Data:
  - *Threat:* A student might try to exploit a SQL injection vulnerability to change their application after the deadline or modify applications of other users.

  - *Countermeasure:* Ensure that database access is always done via prepared statements. This can be checked by using a SAST tool.
- **R**epudiation:
  - *Threat:* A company representative might want to deny that they uploaded a description (e.g., if a extraordinarily high salary was offered).
  - *Countermeasure:* Implement an confirmation mechanism (e.g., a two-step email confirmation) for critical processes.
- **I**nformation Disclosure:
  - *Threat:* Can users learn the version of the web server used?
  - *Countermeasure:* Ensure a system configuration (validated, e.g., by a penetration test) that does not reveal details about the software components/versions used.
- **D**enial of Service:
  - *Threat:* Can an attacker bring down the system by issuing a large number of requests.
  - *Countermeasure:* Use horizontal scaling provided by the web hoster (and accept the remaining risk).
- **E**levation of Privilege:
  - *Threat:* Can an unauthenticated user hijack a session of an authenticated users by exploiting a XSS vulnerability.
  - *Countermeasure:* Ensure that all XSS mitigation technologies of the used web framework are enabled and implement client-side allow-lists.

Marking Scheme:

- In principle *[1]* (**[6]** in total) for each identified threat and *[1]* (**[6]** in total) for each correctly (matching) countermeasure.

(b) Consider the following Python program

Python

```
1  # dbuser and dbpwd are obtained from a
2  # configuration file
3  def role(uid):
4    if uid == "achim":
5      return "academic"
6    else
7      return "student"
8
9  con = pymysql.connect('localhost', dbuser, dbpwd,
10                   'database')
11 # get parameters from web request
12 uid = getRequestParameter("uid")
13 with con:
14   cur = con.cursor()
15   cur.execute("SELECT␣*␣FROM␣users␣WHERE␣uid="
16           +uid)
17   firstname, lastname = cur.fetchone()
18   print("Welcome␣" + firstname + "␣" + lastname)
19
20   cur.execute("SELECT␣*␣FROM␣description␣WHERE␣role="
21           + role(uid))
22   msg = cur.fetchone()
23   print("You␣are␣an␣" + msg)
```

(i) Briefly argue whether this program is secure or not.
- If it is secure, explain why.
- If it is not secure,
  - name the vulnerability,
  - state which line of code is vulnerable,
  - explain how an attacker can exploit the vulnerability, and
  - propose a fix and briefly explain, how it protects the application.

(10 marks)

The program is vulnerable to *SQL Injection* in line 15 and 16. Thus, an attacker can pass SQL commands as parameters (e.g., `uid`) that are then executed by the database (and, e.g., return additional data or modify data). This can be mitigated by using a proper prepared statement:

```
cur.execute("SELECT * FROM description WHERE
role=?") ur.setString(1,uid)
```

This call is secure, as this type of execute-call strictly separates executable parts from data.

Marking Scheme:

- **[2]** for the correct name (SQL Injection).
- **[1]** for the correct line of code.
- **[3]** for the explanation of a concrete attack
- **[4]** for a correct fix (including explanation) using a prepared statement.

(c) Consider the following Python programme.

Python

```
1 def f(x, y):
2   result = 0;
3   if (hash(x) == 1234):
4     return (1/(5-y))
5   else:
6     return result
```

Assume you are asked to recommend an automated security tool for finding a potential crash (exception) in the program.

Would you recommend a static application security testing tool (SAST) or a dynamic application security testing tool.

(8 marks)

The problem that you want to detect in this function is the potential division by zero in line 4. To find this bug, a dynamic testing tool, first, would need to generate an input x for which hash(x)==1234 holds and, second, for y it needs to generate the value 5. In particular for the first condition, it is very unlikely that a dynamic tool generates a suitable input (recall that reversing good hash functions is a very hard problem). In contrast, a static analysis tool usually treats the call to the hash functions as a black box (i.e., assuming that there exists and x for which hash(x)==1234 holds)and just assumes that the code may enter the if-statement. A good static analysis tool will even be able to infer that y needs to have the value 5 to trigger the bug. A not so good (precise) static analysis tool might just assume that the value of 5-y can be zero. Thus, we can conclude that this type of bugs is hard to detect using a dynamic testing tool, while a static tool should detect this issue. Thus, choosing a static tool is the right solution here.

Marking Scheme:

- **[2]** for recommending a SAST tool.
- **[6]** for a conclusive explanation.

(Total 30 marks)