Front Page

UNIVERSITY OF OSLO
The Faculty of Mathematics and Natural Sciences
Written examination TEK5510/9510
2022 2nd SEMESTER
Duration: December 14. at 15:00 to 19:00 (4 hours)
Permitted aids: None
It is important that you read this front page before you start.

Rember to explain your answers.

If you need to zoom in on screenshots, press 'Ctrl' and '+' at the same time. Use the '+' in numpad (the right hand part of the keyboard). Remember to enable numpad with the 'numpad' key if it is not enabled. To zoom out, use '-' instead of '+'. It may also work to hold 'Ctrl' while scrolling.

1 Security principles and malware

a) Define the three security principles Confidentiality, Integrity, and Availability. (3p)

b) For each security principle, give an example of a malware type that can attack the principle and how such an attack would work. (3p)

1 a) 1p for each correct definition, written in own words and shows understanding of the concept.

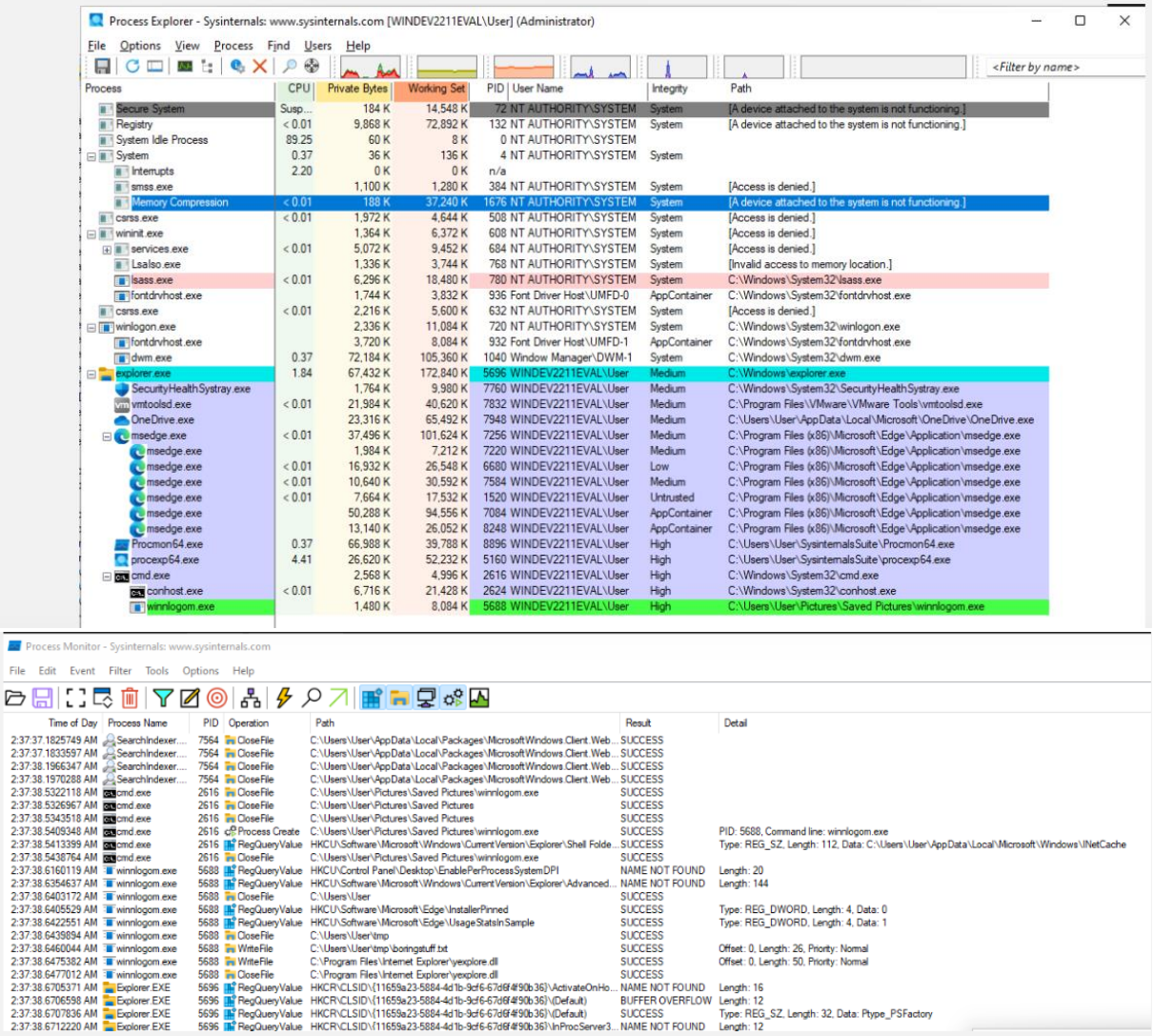
- Confidentiality is about preventing unauthorized disclosure of information (unauthorized reading)
- Integrity is about preventing unauthorized modification of information (unauthorized writing)
- Availability is the property of being accessible and usable upon demand by an authorized entity

1 b) 1p for each correct example with reasonable explanation. It is important to show understanding of the difference between the security principles. An answer like “Arbitrary code exection attacks all of them” does not explain how the attack would work and does not receive full score.

Possible examples:

- Confidentiality: A **keylogger** is installed on a victim computer and logs all key strokes. The attacker can then gain access to this information, including passwords.
- Integrity: A **backdoor** can be used to gain access to a victim computer and change the contents of important files, without the legitimate user of the computer knowing. This will break the integrity of the system.
- Availability: A **ransomware** can be installed on a victim computer to encrypt important files. This attack makes the important files unavailable to the legitimate user.

2 Analyzing an executable



You are given screenshots from Process explorer and Process monitor.

a) What can you tell about the process with PID 5688 from the screenshots? Does this process appear to be malicious? What signs do you see that suggest it being malicious/non-malicious? (6p)

b) Based on the information in the screenshots, describe a possible purpose of the process with PID 5688. (2p)

c) If you were to analyze the executable further, what would be your next steps? (2p)

2 a) 1p for each of: (max 6)

- It was started by cmd.exe, as it is below cmd in the process hierarchy.
- Recently started, since it is highlighted in green.
- It runs as administrator (with high integrity), as cmd.exe.
- It reads from the registry, seen from the RegQueryValue events in Process Monitor. Two registry values that are read are related to Edge.
- It writes two files yexplore.dll in the c:\Program Files\Internet Explorer folder and boringstuff.txt in C:\Users\User\tmp.

- The dll yexplore.dll is written to the Internet Explorer folder, and the name yexplore.dll is similar to iexplore.dll, possibly an attempt not to draw attention.
- The name winnlogom.exe is suspiciously similar to the legitimate winlogon.exe, and it is located under C:\Users\User\Pictures\Saved Pictures, which is an unusual location for a legitimate application.

2b) 2p for something like: The application appears to be related to web browsers. The registry values that are read are related to Edge. The gathered information may be written to the file boringstuff.txt. The file yexplore.dll may be a malicious dll that it wants to inject into Internet Explorer.

2c) 1p for each suggestion like: (max 2p)

- Check the contents of the written files yexplore.dll and boringstuff.txt
- Analyze yexplore.dll and winnlogom.exe in IDA
- Check the filters in Process Monitor to see if it reads interesting files, writes to or reads more from the registry etc.
- Use Regshot before and after running the application to look for changes.

3 Integer computation

Do the following calculations. For each exercise, show the computations in binary numbers **and** explain the result. All numbers below are in decimal. (2p for each)

a) 125+12 signed 8-bit
b) 125+12 unsigned 8-bit
c) 15-19 signed 8-bit
d) 15-19 unsigned 8-bit

3

a) 125 + 12 = 0111 1101_2 + 1100_2 = 1000 1001_2 = -119, overflow. The first bit is the sign bit.

b) 125 + 12 = 0111 1101_2 + 1100_2 = 1000 1001_2 = 137, no problem.

c) 15-19 = 1111_2 – 1 0011_2 = 1111 1100_2 = -4, no problem

d) 15-19 = 1111_2 – 1 0011_2 = 1111 1100_2 = 252, underflow. No sign bit.

4 A vulnerable function

This is the source code of application.c:

```
void copyBuf (char *str) {
        char buf[12];
        strcpy(buf, str);
}
int main (int argc, char **argv) {
        copyBuf(argv[1]);
        return 0;
}
```

a) This application is vulnerable. What type of vulnerability is it? (1p)
b) What is a stack cookie? (1p)
c) Draw the stack frame of the function copyBuf with and without a stack cookie. (Draw two figures) (4p)
d) What would happen if we run the application with input: **AAAABBBBabcd1234efgh5678ijkl**
   i) without a stack cookie (2p)
   ii) with a stack cookie (2p)
For each case, explain the steps, not only the end result.

4

a) 1p for: Stack buffer overflow

b) 1p for: Stack cookie is an unpredictable value that is placed on the stack below the local buffer and it is checked before the function returns. It would not prevent an overflow, but it would prevent the function from returning to the address that overwrote EIP.

c) Stack frame of copy buf without stack cookie:

__ __ __ __ // Local buffer, size 12
__ __ __ __
__ __ __ __
__ __ __ __ // Saved frame pointer (EBP)
__ __ __ __ // Saved instruction pointer (EIP)
__ __ __ __ // Argument (argv[1]/str)

Stack frame of copy buf with stack cookie: AAAABBBBabcd1234efgh5678ijkl

__ __ __ __ // Local buffer, size 12
__ __ __ __
__ __ __ __
__ __ __ __ // Stack cookie
__ __ __ __ // Saved frame pointer (EBP)
__ __ __ __ // Saved instruction pointer (EIP)
__ __ __ __ // Argument (argv[1]/str)

1p for correct position of EBP,
1p for correct position of EIP,
0.5p for correct position and length of local buffer
and 0.5p for correct position of argument
1p for correct difference with and without stack cookie.

d)

i)Without stack cookie:
AAAABBBBabcd fills the buffer,
1234 overwrites the saved frame pointer
efgh overwrites the saved EIP, which makes the function return to hgfe (0x68676665) because of endianness. It is not necessary to write the address in hexadesimal, but it is necessary to have the correct endianness for full score.

-0.5p for wrong endianness in the address it returns to
-1p for forgotten EBP

ii) With stack cookie:
AAAABBBBabcd fills the buffer,
1234 overwrites the stack cookie
efgh overwrites the saved frame pointer
5678 overwrites the saved EIP
When the function is about to return, it detects the changed stack cookie, which prevents the function from returning to the address that overwrote the saved EIP.

-1p for wrong value for the stack cookie
-1p for stating that the overwrite is prevented or that it returns to the overwritten address.

5 Hashing

What is a hash function? (2p)
How are hash functions relevant for storing passwords? Are there differences between Linux and Windows? (4p)
Explain how a password is checked when a user logs in and how password cracking works. Are there differences between Linux and Windows? (4p)

1.

2p for: A hash function is a one-way function that is easy to compute for a given input, but it is difficult to determine what the input was given the output.

2.

2p for: Instead of storing passwords in cleartext, the hashed passwords are stored. This is good since it is difficult to determine what the input (password) was if someone gets hold of the password hashes.

2p for: In Linux passwords are salted before hashing. This means that a salt (string) is prepended to the password before hashing it. The salt is stored together with the hash. In Windows passwords are not salted, and the hash is equal for everyone with the same password.

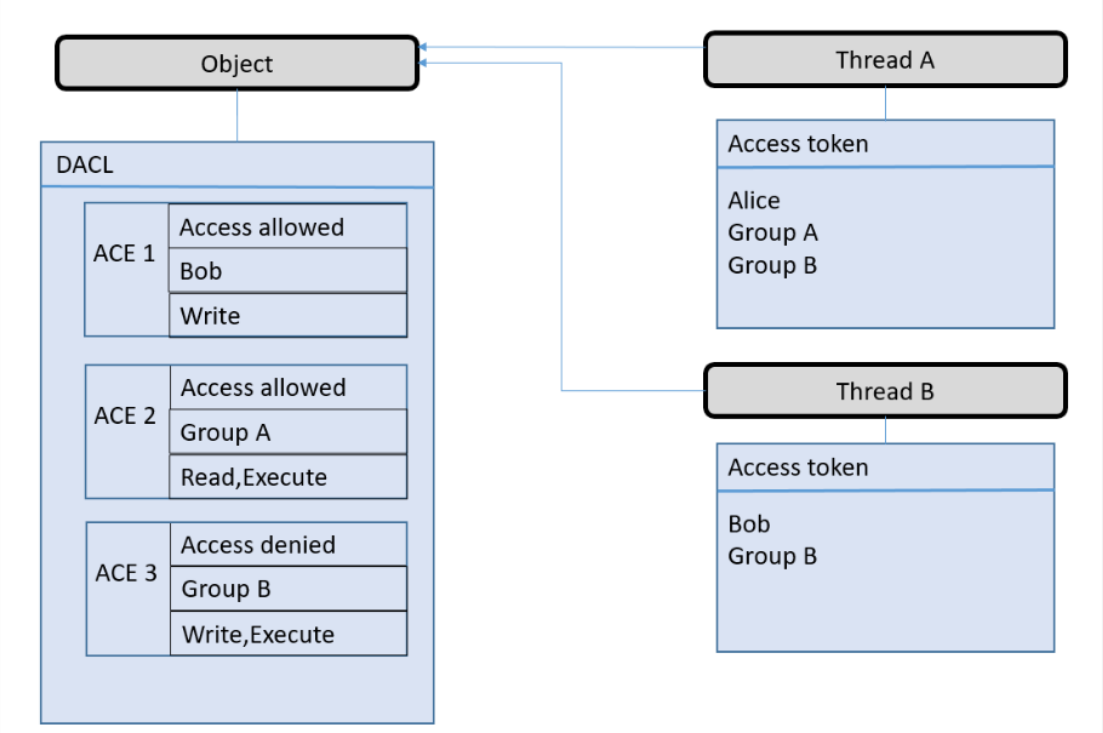
3.

1p for: When a user logs in, the password given by the user is hashed, and then this hash is compared with the password hash stored for that user. In Linux the salt is prepended before the password is hashed. (If many **correct** details are given about the logon process, the role of SAM, LSA, /etc/shadow is given, an additional 1p can be given if full score is not reached on the question.)

1p for: Password cracking is done by guessing many passwords, hashing them and comparing the resulting hash with the hash of the password you want to crack.

2p for: In Windows all users with the same password have the same password hash. Therefore it can be useful to make large tables of hashes corresponding to passwords, and look up the hashes in the large tables. In Linux, because of the salt, each password must be cracked individually.

6 Windows



For each case below, what is the result of the request? Explain how each Access Control Entry influences the result.

a) Thread A requests write and execute access to the object. (2p)

b) Thread B requests write and execute access to the object. (2p)

c) Thread A requests execute access to the object. (2p)

d) Thread B requests read access to the object. (2p)

e) You change the order of ACE2 and ACE3, making the order of the ACEs: ACE1, ACE3, ACE2. Give an example of a request that would give different results before and after the reordering of the ACEs. Explain why. (2p)

6

The answer must be explained to get full score. 2p for each.

a)

ACE 1 does not apply since it only applies to Bob. ACE 2 applies, since it concerns Group A, and Execute access is granted, while write access is still missing. ACE 3 applies, since it concerns Group B, and Write access is denied. Since not all access that was asked for is granted, no access is granted.

b)

ACE 1 applies since it concerns Bob, and Write access is granted. ACE2 does not apply, since it concerns Group A. ACE3 applies, and Write and Execute access are denied. No access is given.

c)

ACE 1 does not apply since it only applies to Bob. ACE 2 applies, since it concerns Group A, and Execute access is granted. This was all that is asked for, and access is granted.

d)

ACE 1 concerns Bob, but is not relevant for Read access. ACE 2 does not apply since it concerns group A. ACE 3 is also not relevant for Read access. The end of the list is reached without the Read access granted, and no access is given.

e)

The result of c would change:

ACE 1 does not apply since it only applies to Bob. ACE 3 applies, since it concerns Group B, and Execute access is denied. ACE 2, which granted Execute access is not reached. No access is granted.

Other requests with different outcomes are also accepted

7 **Linux**

```
-rw-r--r-x 1 Doffen student 1617 Oct 28 11:01 tent.make
drwxr----- 2 Doffen student  512 Oct 25 17:44 tricks
```

Ole and Doffen are members of the group student. Donald is not member of student.

a) Which of Ole, Doffen and Donald can read tent.make? Explain (2p)
b) Which of Ole, Doffen and Donald can execute tent.make? Explain (2p)
c) Doffen wants both Ole and Donald to have write access to tricks. How must the file permissions of tricks be to achieve this? What must be changed? (2p)
d) Doffen wants Donald to have read access to tricks, but not himself and Ole. How must the file permissions of tricks be to achieve this? What must be changed? (2p)

7.

a) 2p for: Ole, Doffen and Donald can all read tent.make. Doffen because of the first r, for owner, Ole because of the second r, for group, and Donald because of the third r, for others.
b) 1p for: Only Donald can execute tent.make, because of the x for others. 1p for: Doffen cannot execute the file now, but can give himself execute permission, since he is the owner.
c) 2p for: He must add the write permission to both group and others to give both write access. It is not sufficient to give write permission only to others. (These permissions will work: drwxrw- -w-.)
d) 2p for: He must add read permission for others and remove read permission for owner and group. (These permissions will work: d-wx- - - r- -.)

8 Hardware security

a Give a short overview of the Cold Boot attack. What does it attack and what are the main steps? (2p)

b Give a short overview of the Meltdown attack. What does it attack and what are the main steps? (2p)

c) How can Meltdown be mitigated? (2p)

d) Cold Boot and Meltdown exploit hardware vulnerabilities, not software vulnerabilities. Use these two attacks as examples to explain differences between hardware and software vulnerabilities. (2p)

8

a) 2p for: A running system using full disk encryption has the encryption keys loaded into RAM. If the system is powered off, the data in RAM is lost, but not instantly. DRAM loses their content gradually over a period of time. The time depends on the temperature. The Cold Boot attack exploits this by freezing the RAM when powering off the system. Then the RAM can be moved to another system and read.

b) The main steps of meltdown:

1. Flush/evict an array out of memory, to prepare for the side channel attack
2. Attempts to read a secret without having access. The secret is used as an index into the array, such that which element that is read depends on the secret
    i. The instructions are executed tentatively (out-of-order) before it is discovered that necessary read access is missing
    ii. An exception occurs, which must be handled or suppressed
3. Read every element of the array. The element that takes the shortest time to read, is the element that was loaded into memory based on the secret. This leaks what the secret was.

1p is given for explaining the side channel attack based on timing (Flush&Reload or Evict&Reload).

1p for the rest.

c) 2p for: Meltdown can be mitigated by making sure that the contents of the internal storage buffers in the processor are overwritten/deleted before another process can read from it. New processors can be made with instructions for this, but in old processors, this must be done with sequences of existing instructions that overwrite the internal storage.

d) 1p for: Software vulnerabilities can be mitigated by rewriting code and releasing a new version.
1p for: Hardware vulnerabilities may require new hardware to be shipped. Sometimes, like in the case with meltdown, the existing functionality in the hardware could be used in new ways to prevent the attack without replacing the physical component.

9 Reversing

a) Give five examples of which types of information about an executable you can find when analyzing it in IDA? (5p)

b) When starting analysis of a new executable in IDA, several strategies can be used to identify interesting parts of the executable. Give examples of two different strategies that can be used. (4p)

9

a) 1p for each type of information like: (max 5)
   - Strings,
   - Imported functions,
   - Exported functions,
   - The (assembly) code of the executable,
   - Functions in the executable

b) 2p for each of:
   - Starting from exported functions or the entry point (depending on whether it is a dll or an exe.)
   - Look for interesting strings, like file names, error messages, paths, and find cross references to the code that is using them.

   Other reasonable strategies are accepted, but these are the two we have focused on.

10 Threat modeling

a) What is threat modeling, and how is it related to securing software? (2p)

b) The threat modeling process consists of four high level steps. Describe the threat modeling process. (4p)

10

a) 1p for: Threat modeling is a process for identifying and reasoning about potential threats to an application.
   1p for: Threat modeling is done as part of a development process in order to find and mitigate potential threats and vulnerabilities before the application is finished, to end up with a more secure application.

b) The four high level steps are:

   - Decomposing the application: What are you building?
   - Identifying and ranking threats: What can go wrong?
   - Finding countermeasures and mitigations: What are you going to do about it?
   - Validating and acting upon previous steps: Did you do an acceptable job at the first three steps?

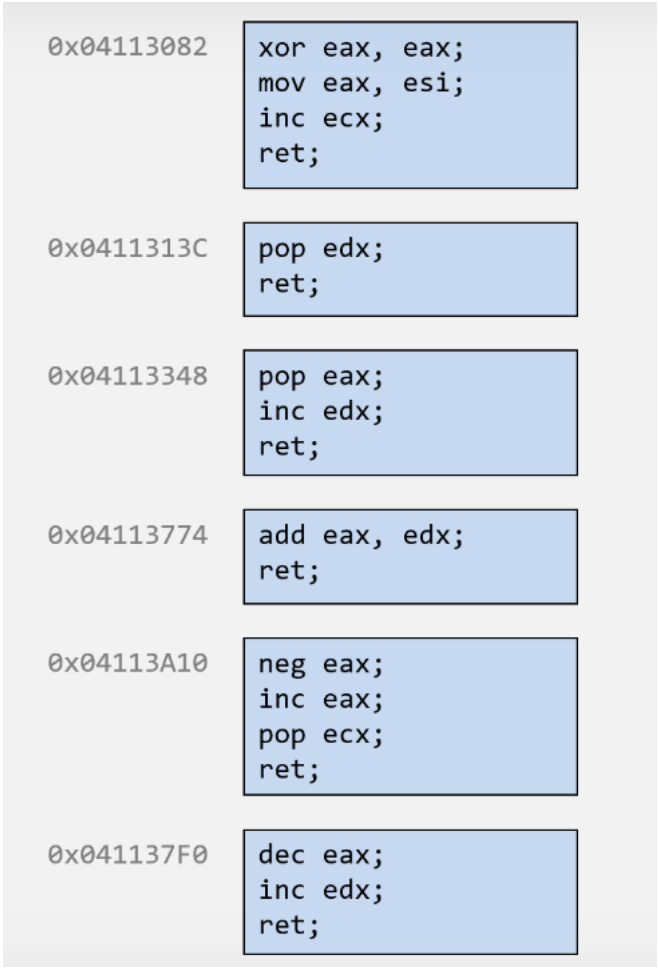
1p for each step that is described reasonably. The name of each step must not be identical to this list to get full score, but the descriptions must fit the step.

11 Return oriented programming (ROP)

Make a stack layout for a ROP-chain that ends with the **0x04113774** gadget calculating **0x4 + 0x41424344**, putting the answer into **eax**. Show the values of each (relevant) register after each gadget.

The following restriction applies: The primitive that lets you write the ROP-chain into the stack does not handle 0-bytes.

(6p)



There are several possibilities. Here are two:

Example 1:

0x0411313C //pop edx; ret; Result: edx =0x7777777B

0x7777777B //value for edx

0x04113348 //pop eax; inc edx; ret; Result: edx =0x7777777C, eax= 0x88888888

0x88888888 //value for eax

0x04113774 //add eax, edx; Result: edx =0x7777777C, eax= 0x00000004

0x0411313C //pop edx; ret; Result: edx =0x41424344, eax= 0x00000004

0x41424344 //value for edx

0x04113774 //add eax, edx; Result: edx =0x41424344, eax= 0x41424348

Example 2:

0411313C # pop edx; ret; Result: edx = 0x41424342

41424342 # value for edx

04113348 # pop eax; inc edx; ret Result: edx = 0x41424343, eax = 0xFFFFFFFC

FFFFFFFC # value for eax

04113A10 # neg eax; inc eax; pop ecx; ret R: edx = 0x41424343,eax=0x00000005, ecx = 0xDEADBEEF

DEADBEEF # dummy value for ecx

041137F0 # dec eax; inc edx; ret Result:edx=0x41424344, eax = 0x00000004, ecx=0xDEADBEEF

04113774 # add eax, edx; ret Result:edx=0x41424344, eax = 0x41424348, ecx=0xDEADBEEF

6p for all steps correct and explained.

4p if the concept is understood and well explained, but computations are wrong.

-1p if only one wrong computation.

12 **Android**

a) How are **permissions** used in Android? Explain the permission types **Normal**, **Dangerous**, and **Signature**. (3p)

b) What is a **Trusted Execution Environment**? What is it used for in Android? (4p)

12

a) 0,5p for: Permissions are used to control what resources/functionality each app has access to.
0,5p for: Normal permissions are not shown to the user and are considered safe, like access to INTERNET, BLUETOOTH, NFC, SET_ALARM.
1p for: Dangerous permissions can be used to get access to personal information and must be granted by the user. Dangerous permissions can be CALENDAR, CONTACTS, SMS, LOCATION, PHONE.
1p for: Signature permissions are declared by apps, and these can only be granted to apps with the same key as the one declaring the permission. These can be INSTALL_PACKAGES, MOUNT_FORMAT_FILE_SYSTEMS

b) 2p for: Trusted Execution Environment is an isolated environment for executing security critical code. Memory and resources are not shared with the Rich Execution Environment (i.e., Android). It can be a separate processor or dedicated processor mode.

In Android it us used for: (1p for each of these, max 2)

- Lock screen passcode verification,

- Android KeyStore,
- Fingerprint matching