Solver: Bay Si Yao & Soh Wei Lin

1)	a)	The salt have to be stored in plaintext because when checking for a password entered by a user against the stored password (hash(plaintext_salt + plaintext_password)), the authentication system will combine the entered plaintext password with the salt in plaintext for the particular user, hash them together and compare the resulting hash to the stored hash to see if authentication succeeds. If the salt is not stored in plaintext but rather stored in the form of a hashed salt (hash(plaintext_salt)), to pass the authentication system, this hashed salt needs to be reversed first before appending to the entered plaintext password, before it is against hashed to compare with the stored hash in the system. However, hash function is a one-way function. It is computationally infeasible to reverse the hashed salt. Therefore, the salt has to be stored in plaintext.
	b)	False Match Rate (FMR) = 0.02% = 0.0002 Failure To Acquire (FTA) = 0% = 0 n = 1000 samples Probability that system would confirm that none of the stored biometrics match Alice's biometric = $(1 - FMR)^n$ $= (1 - 0.0002)^{1000}$ = 0.818714376 = 81.9% OR = 1 – FPIR $= 1 - [(1 - FTA) * (1 - (1 - FMR)^n)]$ $= 1 - [(1 - 0) * (1 - (1 - 0.0002)^{1000})]$ = 0.818714376 = 81.9%
	c)	A subject may not give rights it does not possess to another. The only exception is when the subject is the owner of an object. The owner of an object can give rights over the object to other subjects, regardless of whether those rights are enabled or not as he can grant himself the rights for the object. This is the case in Unix/Linux systems.
2)	a)	(SECRET, {})
	b)	(PUBLIC, {})
	c)	Ordering on security labels: $(x, A) \leq (y, B)$ if and only if $x \leq y$ and $B \subseteq A$. **Case 1: Subject (x, A) has access to fewer objects than subject (y, B).** Suppose subject labelled with (x, A) = (CONFIDENTIAL, {LECTURERS, ADMINISTRATORS}) and subject labelled with (y, B) = (SECRET, {ADMINISTRATORS}) so that (CONFIDENTIAL, {LECTURERS, ADMINISTRATOR}) ≤ (SECRET, {ADMINISTRATORS}) As CONFIDENTIAL ≤ SECRET & {ADMINISTRATOR} $\subseteq$ {LECTURERS, ADMINISTRATORS}.

		Based on the access control policy given earlier in the question (access to an object o, labelled with (x, A), by a subject s, labelled with (y, B), is allowed if $x \leq y$ and $A \cap B = \{\}$) Subject with label **(CONFIDENTIAL, {LECTURER, ADMINISTRATORS})** can access objects with labels (CONFIDENTIAL, {}), (CONFIDENTIAL, {STUDENTS}), (PUBLIC, {}) and (PUBLIC, {STUDENTS}). ➔Number of objects that can be accessed = 4 Subject with label **(SECRET, {ADMINISTRATORS})** can access objects with labels *(SECRET, {}), (SECRET, {STUDENTS}), (SECRET, {LECTURERS}), (SECRET, {STUDENTS, LECTURERS}), (CONFIDENTIAL, {}), (CONFIDENTIAL, {STUDENTS}), (CONFIDENTIAL, {LECTURERS}), (CONFIDENTIAL, {STUDENTS, LECTURERS}), (PUBLIC, {}), (PUBLIC, {STUDENTS}), (PUBLIC, {LECTURERS}), (PUBLIC, {STUDENTS, LECTURERS}).* ➔ Number of objects that can be accessed = 12 Since 4 < 12, subject labelled with (CONFIDENTIAL, {LECTURERS, ADMINISTRATORS}) has access to fewer objects than subject labelled with (SECRET, {ADMINISTRATORS}). **Case 2: Subject (x, A) has access to same number of objects as Subject (y, B)** Suppose subject labelled with **(x, A) = (CONFIDENTIAL, {LECTURERS, ADMINISTRATORS})** and subject labelled with **(y, B) = (CONFIDENTIAL, {LECTURERS, ADMINISTRATORS})** so that (CONFIDENTIAL, {LECTURERS, ADMINISTRATORS}) $\leq$ (CONFIDENTIAL, {LECTURERS, ADMINISTRATORS}) As CONFIDENTIAL = CONFIDENTIAL & {LECTURERS, ADMINISTRATORS} $\subseteq$ {LECTURERS, ADMINISTRATORS}. Since both subjects have the same security label, both can access objects with security labels (CONFIDENTIAL, {}), (CONFIDENTIAL, {STUDENTS}), (PUBLIC, {}), and (PUBLIC, {STUDENTS}). ➔ Number of objects that can be accessed for both subjects = 4 Since both subjects can access 4 objects, both subjects have access to same number of objects.
3)	a)	IF non-root user is allowed to change the ownership of any file that he or she does not have read or write access to, the non-root user would be able to access any file on the system as if he or she does not have read or write access to that file, he or she just need to change the ownership of that file to him or herself, then grant the desired access rights for him or herself on that file. A potential security problem could also be that a user could potentially use that privilege to plant a "trojan horse" in the system to access another user's file. For example, suppose a non-root user, say Alice writes a program that prints the password hash file (e.g. /etc/shadow in Linus systems), and sets the SUID bit. So the program will execute under the effective user id of the owner, which in this case is Alice. So no harm is done,

		as Alice, as a non-root user, will not be able to access the password hash file. But suppose at some point the system changes the ownership of the file to the root, then the program will assume the effective user id of root when it gets executed, so anyone executing will be able to read the password hashes. In practice this kind of situation rarely arises, but if changes of ownership of files are ever triggered then there will be a potential security problem.	
	b)	i)	TRUE, by setting the OBJECT_INHERIT_ACE flag.
		ii)	FALSE, local ACEs (either negative or positive) will be placed on top of the ACL of the child object (above the inherited negative or positive ACEs).
		iii)	TRUE, by setting the NO_PROPAGATE_INHERIT flag.
	c)	Leaking of information via logs, Android provides a centralised logging facility via the Log API. In early versions of Android, apps with READ_LOGS permission can read the content of the logs. Sensitive information has been known to be leaked through logs. Thus, in later versions of Android (since Jelly Bean), apps can only read their own logs.	
4)	a)	The program can be exploited to display the first line of the shadow password file /etc/shadow by providing the user input of "../../../../../etc/shadow". This exploit works because "../" is a shorthand notation to traverse back to the parent directory. Using "../" 5 times followed by "/etc/shadow" will result in the effective path of "/etc/shadow". In addition, the owner of the program is root and its SUID bit is set, thus the program will be executed with root privilege and shadow password file can be accessed by root to display the first line inside.	
	b)	A simple fix could be to filter out the shorthand notation of "../" to traverse back to the parent directory in the user input. Without the use of the shorthand notation, attacker will not be able to traverse all the way back to root directory. Another fix could be to change the ownership of the program to a "dummy" user that is specially created just for the sole purpose, thus even if the attacker uses the shorthand notation of "../" to traverse back to root and to /etc/shadow, the program with SUID bit set will execute with the privilege of that "dummy" user instead of root. As shadow file can only be accessed by root, the exploit will not work.	
	c)	This mechanism prevent Stack Smashing Attack, as the main goal of such attack is to overwrite the stack area, so that malicious codes can be injected into a buffer in the stack, and modify the return pointer stored in the stack frame to redirect it to location where the malicious codes are stored.	
5)	a)	For BLP Model, "observe" operation must comply with the ss-property and "alter" operation must comply with the *-property. (also implicitly the Discretionary Security (ds) – property) **Simple Security (ss) – property (no read-up):** If $(s, o, a) \in b$, and a is in "observe" mode, then $f_s(s) \geq f_o(o)$.	

***-property (star property) (<u>no write-down</u>):**
For every $(s, o, a) \in b$, where a is in "alter" mode, we need to check two conditions:

1. $f_c(s) \leq f_o(o)$ and
2. For all object o' that s has access to in "observe" mode, we have $f_o(o') \leq f_o(o)$.

*With assumption that $f_c = f_s$, we do not need to check the second condition in the *-property. Given (s, o, "alter"), if it passes the first condition check, we would know that:*

$$f_s(s) = f_c(s) \leq f_o(o)$$

Now suppose that (s, o', "observe") ∈ b. By the ss-property, this means that $f_s(s) \geq f_o(o')$.
Combine this with the first inequality above, we get that:

$$f_o(o') \leq f_s(s) \leq f_o(o)$$

*So whenever the first condition of the *-property is satisfied, the second condition will always be true, so there is no need to check the second condition explicitly.*

Access Control Matrix

	O1	O2	O3
S1	Observe	Observe	
S2			Alter
S3	Observe		Observe

b) No, subject S1 cannot alter object O1.
It violates the *-property (*no write down as it may leak secrets*) of BLP access control rules as

$$f_s(S1) = (1, \{A, B\}) > (0, \{A\}) = f_o(O1)$$

If S1 is permitted to alter O1, "Secret" can be leaked in the following sequence:

1. S1 observes O2 (S1 retrieves "Secret" from O2)
2. S1 alters O1 (S1 writes "Secret" to O1)
3. S3 observes O1 (Notice that S3 has no access rights to O2, but now S3 is able to indirectly "observe" O2 because S1 leaked "Secret" of O2 via the "alter" operation to O1)
4. S3 obtains "Secret" of O2

--End of Answers--