Solver: Nurhayati & Wang Bi Jun

Email Address: NURH0048@e.ntu.edu.sg & WANG1198@e.ntu.edu.sg

1. (a) Prevention, detection, reaction, recover.
   (b) Covert channel.
   (c.1)
   Alphanumeric characters + 6 special characters + case-sensitive = 26 * 2 + 6 = 58
   Entropy = 58ˆ8 = 128,063,081,718,016
   (c.2)
   Alphanumeric characters = 26 * 2 (if case sensitive) = 52
   Entropy = 52ˆ12 = 390,877,006,486,250,192,896
   Yes, due to password combinations.
   (d.1) C
   (d.2) A
   (d.3) B
   (d.4) C
   (d.5) C
   (d.6) A

2. (a)

	report.doc	**code.c**	**code.exe**	**present.ppt**	**present.pdf**	**results.txt**
Alice	O,R,W		R,E	O,R,W	O.R	
Bob	R,W	O,R,W	O,R,E		R	
Cathy						R,W
Dan						O,R,W

(b)

	report.doc	**code.c**	**code.exe**	**present.ppt**	**present.pdf**	**results.txt**
Alice	O,R,W	R	R,E	O,R,W	O.R	
Bob	R,W	O,R,W	O,R,E	R	R	
Cathy		R,E			R	R,W
Dan	R		R			O,R,W

code.c | { Alice: {R} ; Bob:{O,R,W} ; Cathy:{R,E} }

(c)
UNIX password authentication is a simple authentication because claimant proves its identity to the verifier by giving up the secret. The use of OTP tokens make the authentication protocol a strong authentication because it involves mutual authentication, where both parties take both the roles of claimant and verifier. It includes challenge-response protocols, which is a sequence of steps to prove knowledge of shared secrets, without giving up the secret.

(e)
Salt is random data that is used as an additional input to a one-way function that "hashes" a password. Password encrypted together with a 12-bit random "salt" that is stored in the clear form.

- A salt is just a random string.
- Each password has its own salt.
- The salt value is stored along with the hash of password+salt.
  - So the salt is not secret.
- For a salt of n-bit, the attacker needs to pre-compute 2 of hashes for the same password. n

(f)
The system is secure if $R \subseteq P$.
The system is precise(not overprotective) if $R = P$.

3. (a)
   - Corruption of program data
   - Memory access violation
   - Execution of code chosen by attacker

   (b)
   - Use Safe Programming Languages i.e Java and Python which does not have direct access to memory.
   - Stackguard. This checks for stack integrity by embedding canaries in stack frames and verify their integrity prior to function return.
   - Address Space Layout Randomization (ASLR): make it hards for attacher to guess the return address.

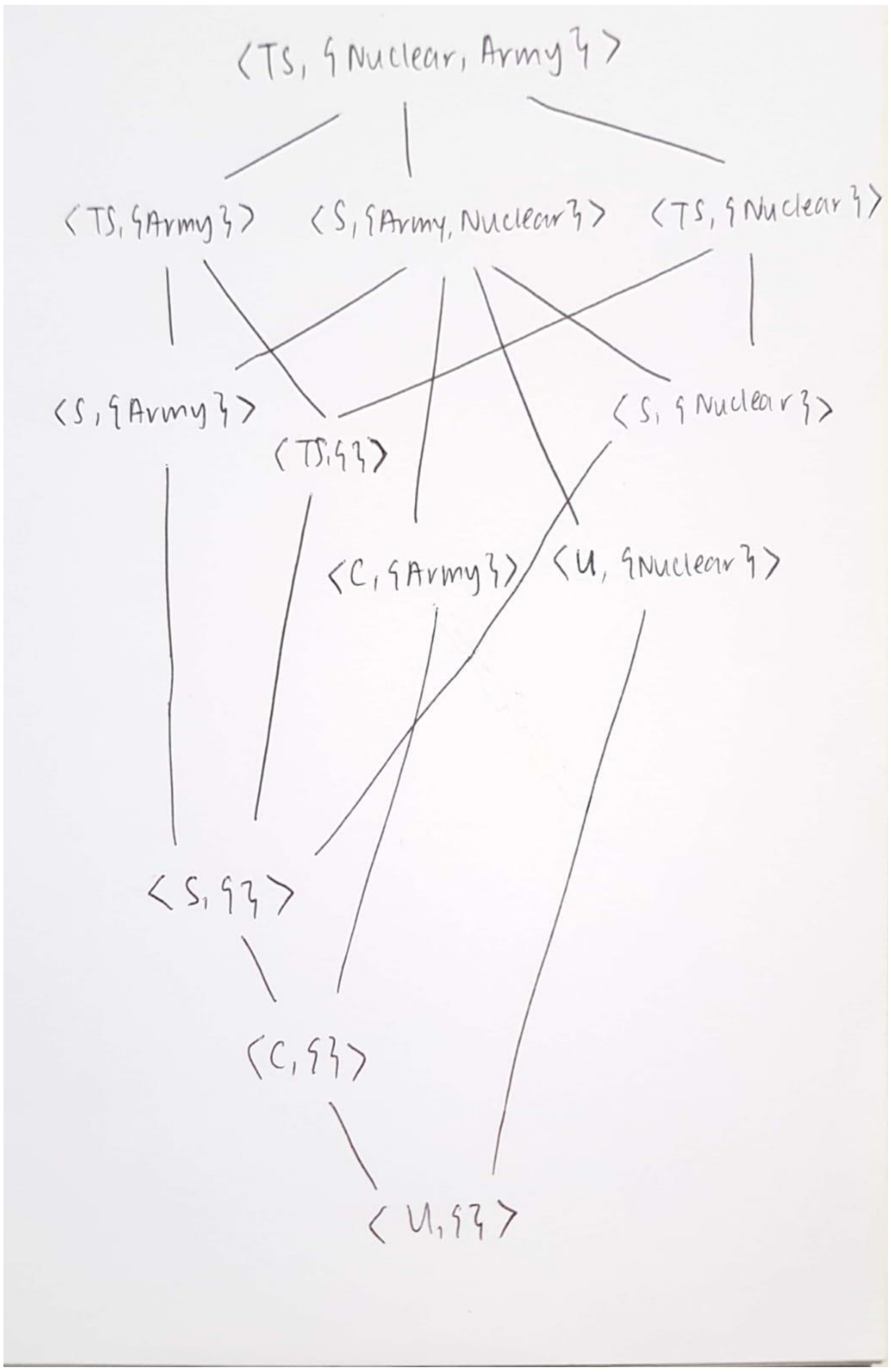
   (c)
   (i)
   An attacker can still take control of Bob's server, and in particular, remove files, by using a "return-to-libc" attack, where the return address is overflowed with the address of the unlink function in libc. The attacker must also arrange for the stack to contain proper arguments for unlink at the right location on the stack.

(ii)
The return address from thestrcpyfunction is on the stack following the buf array. If the attacker provides an input longer than 256 bytes, the subsequent bytes can overwrite the return address from strcpy, vectoring the execution of the program to an arbitrary address when strcpy returns.

4. (a) (i)
TOP SECRET > SECRET > CONFIDENTIAL > UNCLASSIFIED

Subject	**Security Level**
president	(TOP SECRET, {Nuclear}, {Army})
colonel	(SECRET, {Army}, {Nuclear})
major	(CONFIDENTIAL, {Army})
soldier	(UNCLASSIFIED, {Nuclear})

Object	**Security Level**
army_position	(SECRET, {Army})
number_of_army_units	(CONFIDENTIAL, {Army})
number_of_nuclear_units	(CONFIDENTIAL, {Nuclear})
costs_of_nuclear	(UNCLASSIFIED, {Nuclear})
costs_of_army	(UNCLASSIFIED, {Army})
nuclear_code	(TOP SECRET, {Nuclear})



(ii)

- Yes.
  - (president,Nuclear,Observe) is permitted because
    fs(president = (TOP SECRET, {Nuclear}, {Army}) >=
    (UNCLASSIFIED, {Nuclear}) = fo(costs_of_nuclear)
  - (president,Army,Observe) is permitted because
    fs(president = (TOP SECRET, {Nuclear}, {Army}) >=
    (UNCLASSIFIED, {Army}) = fo(costs_of_army)

- No.
  - (Major,Nuclear,Observe) is not permitted because
    fs(Major) = (CONFIDENTIAL, {Army}) <= (CONFIDENTIAL, {Nuclear}) = fo(Nuclear)
  - (Major, Army, Observe) is permitted because
    fs(Major) = (CONFIDENTIAL, {Army}) >= (CONFIDENTIAL, {Army}) = fo(Army)

- Yes,
  - (Colonel,Army,Observe) is permitted because
    fs(colonel = (SECRET, {Army}, {Nuclear}) >= (CONFIDENTIAL, {Army}) = fo(number_of_army_units)
  - (Colonel,Nuclear,Observe) is permitted because
    fs(colonel = (SECRET, {Army}, {Nuclear}) >= (CONFIDENTIAL, {Nuclear}) = fo(number_of_nuclear_units)

- No
  - (Colonel,Army,Alter) is not permitted because
    fs(colonel = (SECRET, {Army}, {Nuclear}) >= (SECRET, {Army})= fo(army_position)

- No
  - (Major,Nuclear,Alter) is not permitted because
    fs(major =(CONFIDENTIAL, {Army}) is not <= (TOP SECRET, {Nuclear}) = fo(nuclear_code)

- Yes
  - (Soldier,Nuclear,Alter) is permitted because
    fs(soldier =(UNCLASSIFIED, {Nuclear}) <= (TOP SECRET, {Nuclear}) = fo(nuclear_code)

(b) To address that, we can propose KARMA, an adaptive live patching system for Android kernels. KARMA features a multi-level adaptive patching model to protect kernel vulnerabilities from exploits. Specifically, patches in KARMA can be placed at multiple levels in the kernel to filter malicious inputs, and they can be automatically adapted to thousands of Android devices. In addition, KARMA's patches are written in a high-level memory-safe language, making them secure and easy to vet, and their run-time behaviors are strictly confined to prevent them from being misused.

All the best for your exams!