Software & System Security

IMPORTANT NOTES: Study lecture materials at least 1 hour and prepare Question 1-4 prior to the tutorial session. Prepared questions will be discussed in the tutorial session.

1. What is a *buffer overflow* and why can it be potentially used to attack a system? Explain in terms of abstract concepts (Input, Storage, Memory content, Memory addresses, etc.). What needs to be done at development time to prevent buffer overflow attacks? What can be done at run-time (or compile time).

2. Give an example of a violation of the principle of least privilege, and explain how the system design could be modified to fix the violation.

3. What is a command injection vulnerability? Give an example and explain how to defend against such vulnerabilities.

4. Pick two of the following software design principles, and for each one, explain what it means, give an example of what could go wrong if the principle is not followed, and explain which security goal(s) (confidentiality, integrity, authenticity and availability) would be breached (It can be more than one goal(s)).

   (a) Use secure defaults
   (b) Defense in depth
   (c) Separation of privilege
   (d) Assume external systems/entities are insecure
   (e) Authorize after authentication
   (f) Only receive control instructions from trusted sources

5. Consider the following C program. What type of software vulnerability is present in it, and how could it potentially be exploited by an attacker?

```
1:      #include <stdio.h>
2:      #include <string.h>

3:      int main(int argc, char *argv[]){

4:              char buffer[96];
5:              unsigned short s;
6:              int i;

7:              if(argc < 3){
8:                      return -1; /* error if less than 2 arguments */
9:                                 /* (3 arguments including prog. name) */
10:             }

11:             i = atoi(argv[1]); /* convert first arg string to an int i */
12:             s = i;             /* convert int i to a short s */

13:             if(s >= 96){            /* (a) */
14:                     return -1;
15:             }

16:             printf("s = %d\n", s);

17:             memcpy(buffer, argv[2], i); /* (b) */
18:             buf[i] = '\0';
```

```
19:                 printf("%s\n", buffer);

20:                 return 0;
21:         }
```