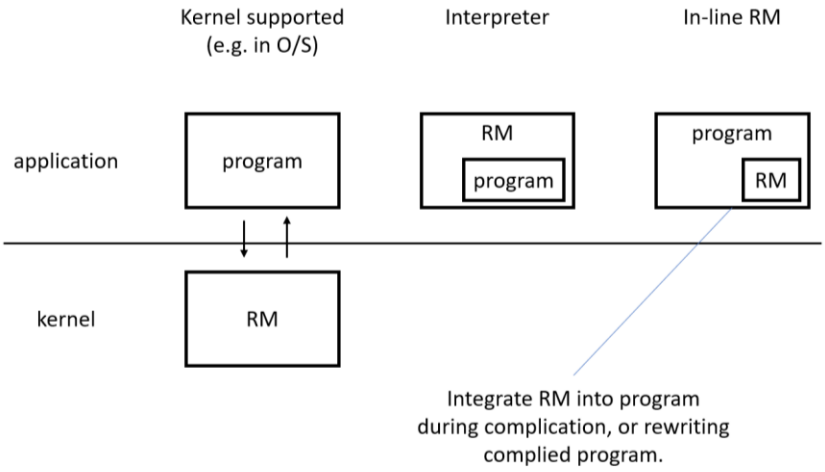
1)

a) 1. Place in OS – program needs to go through RM when it needs to access resources
2. Run program in RM – RM acts as interpreter and mediate all access request by program
3. Integrate RM into program during compilation, or rewriting compiled program

b)



c) Number of possible characters = 26 + 10 = 36
Number of possible passwords = 36^{10}
Note: Since passwords are case insensitive, uppercase and lowercase alphabets are considered only once, e.g. A and a are considered the same, hence there is no need to consider the alphabets twice.

d) $\log_2 no.of\ possible\ passwords = 40$
$No.of\ possible\ passwords = 2^{40}$

e)
e.1) D (*mode consists of the 9 bits ACL of the file)*
e.2) B
e.3) A (*when threshold is increased, increase rejection of legitimate user, increase false negative)*
e.4) B
e.5) C
e.6) C

2)

a) *Note: Include subjects as objects in access control matrix*

	Manual.doc	Code.c	Code.exe	Evaluation.xls	Persent.ppt	Alice	Bob	Cathy	Dan
Alice	O, R, W	R	R, E						
Bob	R	O, R, W	O, R, E						
Cathy				O, R, W					
Dan					O, R, W				

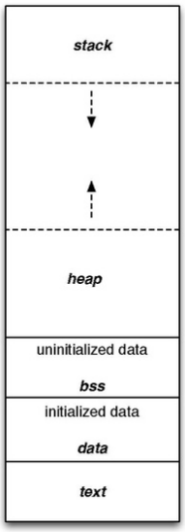
b)

	Manual.doc	Code.c	Code.exe	Evaluation.xls	Persent.ppt	Alice	Bob	Cathy	Dan
Alice	O, R, W	R	R, E						
Bob	R	O, R, W	O, R, E						
Cathy	R	R	R, E	O, R, W					
Dan				R	O, R, W				

c) 110 100 000

d) 1. Password stored in encrypted form.
2. Salt is added to password before encryption.
3. Password stored in /etc/shadow file which is not accessible to all users.
4. Password protected by access control to the shadow file – only authorized users can access

e) *Note: not sure of the answer*



Text segment (Code segment) contains executable segments and is usually read only to prevent programs from accidentally modifying its instructions.

Stack segment – Write, Execute

Text segment – Read, Execute

Figure 1: Typical arrangement of memory

f) Unix Command: umask ; umask 027

3)

a) Software is secure if it can handle intentionally malformed input (attacker picks the inputs). Secure software would protect the integrity of the runtime system. To make software more secure, it is tested against untypical usage patterns where hostile parties can and will provide malicious inputs.

b) *Solver Opinion: As the question is only 4 marks, it is not necessary to give detailed code as long as you explain the idea behind each of them.*

1. Integer Overflow

```
char buf[128];
combine(char *s1, size_t len1,
        char *s2, size_t len2)
{
  if (len1 + len2 + 1 <= sizeof(buf)) {
   strncpy(buf, s1, len1);
   strncat(buf, s2, len2);
  }
}
```

`len1 < sizeof(buf)`

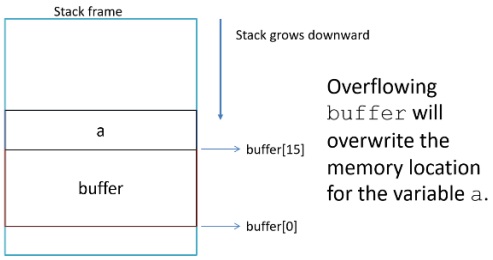
`len2 = 0xffffffff`

`len2 + 1 = 2^32-1 + 1 = 0 mod 2^32`

`strncat` will be executed

2. Buffer Overflow

```
int main(int argc, char *argv[])
{
  char a = 'a';
  char buffer[16];
  if(argc < 2){
    printf("Usage: %s <string>\n", argv[0]);
    exit(0);
  }
  strcpy(buffer, argv[1]);
  printf("Value of a: %c\n", a);
  return 0;
}
```



3. strcpy

No checks on whether either or both arguments are NULL.

No checks on the length of the destination string.

```
char str[5];
strcpy(str, "Hello, World!");
```

There would be buffer overflow as strcpy would copy everything until "/0"is found.

4. printf vulnerability

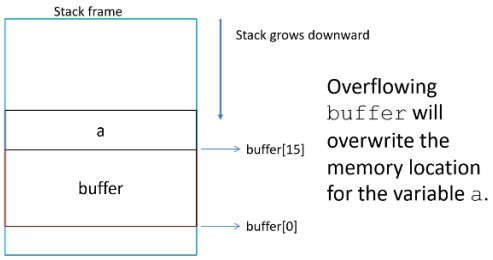
The printf function takes variable-length arguments. The number of arguments depends on the number of escape sequences in the format string. C compiler does not check that the number of escape sequences matches the number of arguments.

```
int x=1, y=2, z=3;
printf("Value of x: %d, value of y: %d, value of z: %d",
        x, y);
```

The program is accepted by compiler. If arguments in printf do not match the escape sequences, at runtime, printf retrieves values from the stack. This allows one to retrieve content of the stack via printf.

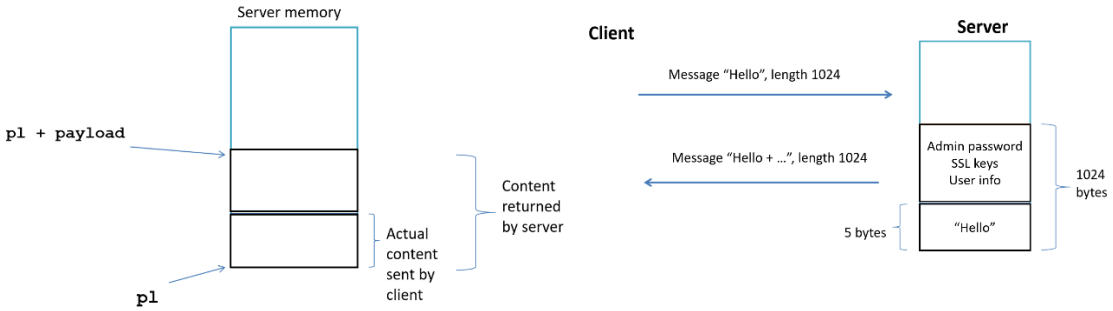
c) Buffer: concrete implementation of a variable. If the value assigned to a variable exceeds the size of the allocated buffer, memory locations not allocated to this variable are overwritten.

```
int main(int argc, char *argv[])
{
  char a = 'a';
  char buffer[16];
  if(argc < 2){
    printf("Usage: %s <string>\n", argv[0]);
    exit(0);
  }
  strcpy(buffer, argv[1]);
  printf("Value of a: %c\n", a);
  return 0;
}
```



d) This bug is related to the 'heartbeat' extension of the SSL protocol. A client may send an 'echo' message to the server to keep the connection alive. The server replies to the client by playing back the echo message. The echo message contains length of the message and the content of the message. Both variables controlled by the client.

The issue lies with this line of code: `memcpy(bp, pl, payload)`, where it copes payload bytes from memory location pointed to by pl to the buffer bp. Since the client controls both the length and the message content of the heartbeat message, the message content may be less than the specified length. Memory content outside those allocated for pl gets copied to bp and sent to client.



e) *Note: not sure about answer*
1. Leakage of confidential information such as admin password, SSL keys and user info
2. Stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet.

4)

a) The attacker controls the values of len1 and len2, thus it could be set such that there would be integer overflow by adding up len1 + len2, causing the code below to be executed when it should not have.

b) Check that each length is within the bounds of buffer. In this case, since we know size_t is an unsigned integer, we do not need to check the lower bound, as the values of len1 and len2 will always be zero or positive numbers. Subtract len1 (or equivalently len2) from sizeof(buf), and compare it with len2. A possible fix would be as follows:

```
char buf[128];
void combine(char *s1, size_t len1, char *s2, size_t len2) {
if(len1 > sizeof(buf) || len2 > sizeof(buf))
      return;
if(len2 < sizeof(buf) - len1) {
      strncpy(buf, s1, len1);
      strncat(buf, s2, len2);
      }
}
```

c) Assuming unsigned 32 bit integer, max value = $2^{32} - 1$.

k	i_k
1	1
2	3
3	3^2
4	3^3

As we can see, the value of i_k follows the following pattern, 3^{k-1}. We need to find the value where $i_{k+1} < i_k$. This happens when there is an integer overflow. Hence, we find the value of k when 3^{k-1} is closest to the max value.

$$3^{k-1} = 2^{32} - 1$$
$$3^{k-1} = 2^{32} \ (-1 \ is \ negligible)$$
$$(k-1)log3 = 32log2$$
$$k - 1 = \frac{32log2}{log3}$$
$$k - 1 = 20.18$$
$$k \approx 21$$

We can check our answer by confirming that 3^{20} < max value, and 3^{21} exceeds the max value. Thus the value of i_k is 3^{20}.

Note: Not very sure of answer but this is how I approached the question. Since the question did not state whether the int is 16 or 32 bit or signed or unsigned, I assumed that it was 32 bit unsigned. You could also assume it was 16 bit signed as long as you state your assumptions clearly.

d) The partial ordering in VSTa microkernel is not a lattice. For example, consider the lists .1.2 and .1.3. No list can have both .1.2 and .1.3 as its prefixes, so they have no l.u.b. To make it into a lattice, we can add a "top" element, let's call it O. Then define the ordering for O as follows: for every s, $s \leqslant O$, The l.u.b. of any two elements is defined as follows:

$$l.u.b(s,t) = \begin{cases} s & \text{if } t \text{ is a prefix of } s \\ t & \text{if } s \text{ is a prefix of } t \\ O & otherwise. \end{cases}$$

The greatest lowest bound for any two lists s and t is just their longest common prefix, and it always exists. So by adding the element O as above, we turn the partial order into a lattice.

e) At runtime (Android 6.0): dangerous permissions require explicit approval from user. If an app requests a dangerous permission X at runtime:

- If the app does not currently have any permissions in the permission group of X, the system shows a dialog box to ask user approval.
- If the app already has another dangerous permission in the same permission group as X, the system immediately grants the permission without any interaction with the user.

For example, if an app had previously requested and been granted the READ_CONTACTS permission, and it then requests WRITE_CONTACTS, the system immediately grants that permission.

--End of Answers--