A22652 *No calculator permitted in this examination*

UNIVERSITY OF BIRMINGHAM

School of Computer Science

Third Year – Degree of BSc with Honours
Physics with Particle Physics & Cosmology Physics with Computer Science

Fourth Year – Degree of MEng with Honours
Computer Science/Software Engineering

Fourth Year – Joint Degree of MSci with Honours
Mathematics and Computer Science

Degree of MSc
Advanced Computer Science
Internet Software Systems
Computer Security

Undergraduate Occasional
Computer Science/Software Engineering

06 20010

Secure Programming

Summer Examinations 2011

Time allowed: 1 ½ hours

[Answer ALL Questions]

No calculator

[Answer ALL Questions]

1. (a) Give two code examples using prepareStatement, such that the first example is secure regarding SQL injection, whereas the second is not. [10%]

   (b) The operator UNION in SQL has a low precedence compared to SELECT. That is, if we write a statement without brackets, as in

   SELECT ... WHERE ... = ... UNION SELECT ...

   the statement will be parsed as if it were bracketed as follows:

   (SELECT ... WHERE ... = ...) UNION (SELECT ...)

   Explain why the above is important for SQL command injection attacks. (Hint: it is analogous to the fact that "OR" has a lower precedence than "=".) [10%]

2. (a) Suppose you want to defend against path injection attacks. The allowed filenames consist solely of alphanumeric characters. Give two regular expressions, one that could be used for blacklisting, and the other for whitelisting. [10%]

   (b) Regular expressions are widely used for sanitising input. Explain whether regular expressions can themselves become a security risk. [10%]

No calculator

3. (a) In a denial-of-service attack on software, it is important that a relatively small amount of malicious input supplied by the attacker consumes as many system resources as possible. Choose one such DoS attack and explain what resources are consumed in the attack. [10%]

(b) Suppose you are evaluating a web application in which users can input text. The application tries to be user-friendly by letting a user write abbreviations for pieces of text that are used often. Abbreviations can occur anywhere inside the replacement string. For instance, here are some abbreviations for phrases that occur in form letters to customers:

```
#a = "apologies"
#i = "inconvenience"
#t="#a for the #i"
```

The abbreviation #t is then replaced everywhere by "apologies for the inconvenience".

What is your advice to the developers of this system? Specifically, which possible denial-of-service attack should you warn them about? Give a possible malicious input for this attack.

[10%]

4. (a) Suppose we have some C code in which buf is defined to be a string buffer of size 10, and an attacker provides the following input string

```
"buf[1000] = 0;"
```

Explain whether the above input string represents a command injection attack.

[10%]

Question 4 continues over the page

Question 4 continued

(b) Suppose we are using stack canaries to defend against buffer overflow attacks. The canary value is written between the return address and the vulnerable buffer. If the canary has been overwritten, the function does not return, but raises an exception. The exception handler is itself placed on the call stack. The stack looks like this:

exception handler
parameter 2
parameter 1
return address
canary value
buffer (to be overflown)

other local variables

As usual, the stack grows downward and arrays grow upward.

Suppose the attacker wants to jump to a known address somewhere in memory, using a return-to-libc attack. Explain whether it is possible to perform a successful buffer overflow attack despite the canary defence in this case. [10%]

(c) Consider the following code snippet:

```
main()
{
   int i = 0;
   char b[8];
   int a[2];

   a[0] = PUBLIC;
   a[1] = SECRET;

   gets(b);
   printf("%d", a[i]);
}
```

Two integers PUBLIC and SECRET are stored in an array. Only the value of PUBLIC is printed. Explain how to use a buffer overflow attack to find out the value of SECRET. [10%]

(d) In standard CPU architectures, the call stack grows towards lower addresses, but in principle it could grow towards higher addresses. Explain what the possible consequences for buffer overflow attacks are if the stack were to grow in the opposite direction. Would such attacks become easier or harder? [10%]