UNIVERSITY OF BIRMINGHAM

School of Computer Science

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

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

Degree of MSc
Advanced Computer Science
Computer Security
Natural Computation

Undergraduate Occasional
Electronic and Electrical Engineering

06 20010
Secure Programming

Summer Examinations 2008
Time Allowed: 1 ½ hours

[Answer THREE out of Four Questions]

[Marks indicated on this paper add up to 99%
The final 1% is a bonus mark reserved for outstanding answers.]
1. (a) Why is it important to identify trust boundaries in software, and what should happen at such boundaries? [8%]

(b) Describe a situation where you would use an AND-node in an attack tree, giving a concrete example. [8%]

(c) Explain whether automatic program analysis can help with the technique of Candidate Points in software security auditing. As an example, consider a tool that reports all usages of the C function strcpy() in the code. [8%]

(d) Discuss whether it is in principle possible to:

(i) Write useful software that is 100% free from buffer overflows.

(ii) Write a software tool that checks for any given application whether that application has any buffer overflows, such that the tool is 100% accurate. [9%]

2. Suppose you are learning a new programming language. You notice that this language has some complex rules about how certain characters in strings are handled. For example, a double backslash "\\" changes the way the rest of the string is processed.

(a) Based on that information alone, why can you already see potential security vulnerabilities for software written in this language? [8%]

(b) You notice that programmers in this language often write loops that iterate over a string that has been read from IO and check for the presence of "\\", raising an exception if it is found. What are the programmers probably trying to do in terms of software security? (There are several technical terms for it.) Explain why this technique is unlikely to be very effective. [8%]

(c) Suggest a more effective technique than the one in part (b). You can assume that the language has some general-purpose libraries much like Java.lang. [8%]

(d) Explain whether the same defence as under part (c) is also applicable against attacks that overflow the stack of an XML parser. [9%]
3.  (a) Suppose you are reviewing some code that accesses a database. You notice that strings containing words like SELECT are concatenated with string variables to form queries. Explain why there is a potential vulnerability here. [8%]

(b) Explain a more secure way to construct the queries. [8%]

(c) Explain whether the following code snippet is good practice in terms of security.

```java
PreparedStatement stmt =
    con.prepareStatement("SELECT * FROM Table WHERE name = " + username);
ResultSet rs = stmt.executeQuery();
```

[8%]

(d) Explain the relevance of parse trees to this kind of attack. [9%]
4. Suppose some C code contains a data structure like this, with an array of characters (called name) and a function pointer (called e_handler):

```c
struct
{
    char name[60];
    int count;
    int (*e_handler());
} s;
```

(A struct in C is much like an object in Java; containing several pieces of data next to each other.)

(a) To what extent could the above structure contribute to a vulnerability? [8%]

(b) To see whether a potential vulnerability in code can actually be exploited, one may need to analyze some data flow. Explain how that applies to the code above. [8%]

(c) Explain whether the following techniques would be effective as a defence in this case:

- address space randomization

- a non-executable stack [8%]

(d) For both defence techniques in part (c), discuss whether they are costly in terms of run-time overhead, particularly compared to the technique of stack "canaries" or "cookies". [9%]