King's College London

This paper is part of an examination of the College counting towards the award of a degree. Examinations are governed by the College Regulations under the authority of the Academic Board.

Examination Period	January 2021 (Period 1)
Module Code	7CCSMSEN (AY2021) - online exam
Module Title	Security Engineering

Format of Examination	Written questions
Start time	TBD GMT
Time Allowed	ONE AND A HALF hours
Instructions	You are permitted to access any materials you wish, but this is not mandated and is not expected. You may use a calculator if you find this helpful.
Rubric	ANSWER THREE OF FOUR QUESTIONS. Each question awards a maximum of 25 marks. All questions carry equal marks. If more than three questions are answered, the three answers with highest marks will count. (Therefore the maximum exam mark is overall 75 over 75.)

The rubric for this paper must be followed and extra answers should not be submitted. For answers that are handwritten, write with blue/black ink on light coloured paper. Include the Module code, question number and student number on every page to be submitted. For answers that are typed, use the template provided.

Submission Deadline	TBD
Submission Process	Work must be submitted to the **level 7** Informatics Assessments KEATS page.

Your work must be submitted as a PDF file. If you have prepared some answers on computer, and some on paper (which have then been digitised), you may upload at most two PDF files – one for computer-prepared answers, one for digitised answers. Do not duplicate answers across the two PDFs – if you do this, the computer-prepared answer will be taken. You should check that your work displays correctly after it has been uploaded.

ACADEMIC HONESTY AND INTEGRITY

Students at King's are part of an academic community that values trust, fairness and respect and actively encourages students to act with honesty and integrity. It is a College policy that students take responsibility for their work and comply with the university's standards and requirements. Online proctoring / invigilation will not be used for our online assessments. By submitting their answers students will be confirming that the work submitted is completely their own. Misconduct regulations remain in place during this period and students can familiarise themselves with the procedures on the College website

Important: Students should copy out the following statement and include it with their submission for each examination:

I agree to abide by the expectations as to my conduct, as described in the academic honesty and integrity statement.

1. Static analysis is a program analysis technique to analyze a program's code without running it.

a. List, and explain the reasons, of 2 of the benefits static analysis provides. [2]

b. List, and explain the reasons, of 3 of the drawbacks static analysis provides. [3]

c. What impact on the software development process does static analysis have? [4]

d. What does it mean when a static analysis is sound? [2]

e. What does it mean when a static analysis is complete? [2]

f. Consider a static taint flow analysis as we have discussed in class. Consider the following code snippet:

```
1 int printf(untainted char *fmt, ...);
2 void read(int, tainted char *input, int);
3
4 char name[10];
5
6 read(0, name, sizeof (name));
7 char *x = name;
8 printf(x);
```

Let us assume we are interested in an analysis that identifies no tainted data flows (where untainted $<$ tainted in a lattice). Given the initial taint source and untainted sink:

i. Create a name for each missing type qualifier, assuming a flow-/path-/context-insensitive analysis. [2]

ii. For each statement in the program, generate constraints on possible solutions, assuming a flow-/path-/context-insensitive analysis.. [3]

iii. Solve the constraints to produce solutions for the type qualifiers identified earlier (1 mark) and state whether the resulting flow is legal or illegal (1 mark). As above, assume a flow-/path-/context-insensitive analysis. [2]

g. Consider a taint flow analysis as we have discussed in class. Consider the following code snippet:

```
1  int printf(untainted char *fmt, ...);
2  void read(int, tainted char *input, int);
3
4  char name[10];
5
6  read(0, name, sizeof (name));
7  char *x;
8  x = name;
9  x = "hello!"
10 printf(x);
```

Let us assume we are interested in an analysis that identifies no tainted data flows (where untainted $<$ tainted in a lattice). Given the initial taint source and untainted sink:

i. Show how the program would be changed if we carried out a flow-sensitive static analysis, assuming a flow-sensitive and path-/context-insensitive analysis. [1]

ii. Create a name for each missing type qualifier, assuming a flow-sensitive analysis, assuming a flow-sensitive and path-/context-insensitive analysis. [1]

iii. For each statement in the program, generate constraints on possible solutions, assuming a flow-sensitive and path-/context-insensitive analysis. [1]

iv. Solve the constraints to produce solutions for the type qualifiers identified earlier (1 mark) and state whether the resulting flow is legal or illegal (1 mark). As above, assume a flow-sensitive and path-/context-insensitive analysis. [2]

2. Consider the following C code fragment:

```
1  int
2  main(int argc, char **argv)
3  {
4
5      int n;
6
7      if (argc == 2)
8          n = foobar(argv[1]);
9      else exit(1);
10
11     printf("Hey! %s, nice to meet ya (%d)\n", argv[1], n);
12
13     exit(n);
14 }
15
16 int
17 foobar(char *arg)
18 {
19
20     char msg[512];
21
22     memset(msg, 0, sizeof(msg));
23
24     strcpy(msg, "Hey! ");
25
26     snprintf(msg + strlen("Hey: "), \
27         sizeof (msg) - strlen("Hey! ") - 1, arg);
28
29     return strlen(msg);
30 }
```

a. Give a thorough description of the program's vulnerability. In particular, name the vulnerability (1 mark) and provide a detailed overview of its exploitation (3 marks). Then, identify and explain thoroughly all the components that are involved in the exploit (4 marks).

[8 marks]

b. How would an attacker exploit the vulnerability? Hint: describe in detail what the injection vector would look like (and what retaddr and retloc the attacker may use). Use symbolic values and addresses when needed (no need to write down the shellcode).

[10 marks]

c. Would bounds checkers mitigate the vulnerability (1 mark)? Explain clearly the reasons (1 mark).

[2 marks]

d. Would StackGuard mitigate the vulnerability (1 mark)? Explain clearly the reasons (2 marks).

[3 marks]

e. How can the program be fixed?

[2 marks]

3. Consider the following C code fragment and assume the program name is fixed to `vuln`, it is invoked as `./vuln` (i.e., `argv[0]`) and cannot be changed by an attacker.

```
1  int
2  main(int argc, char **argv)
3  {
4      if (argv[1])
5          return foo(argv[1]);
6     else
7          return foo(argv[0]);
8  }
9
10 int
11 foo(char *arg)
12 {
13
14     char bar[128];
15
16     if (sizeof(arg) > 128) {
17         strcpy(bar, arg);
18
19         // never return to main
20         exit(0);
21     }
22
23     strcpy(bar, arg);
24
25     return strlen(bar);
26 }
```

a. Does the program suffer from a memory corruption vulnerability? If

not, explain the reasons. If yes, explain the reasons and and how it is possible to *successfully exploit* this vulnerability. In other words, is it possible to provide specific input to such a program to take advantage of its vulnerability and thus execute arbitrary code (for instance, spawning a shell), on x86-32 architectures? If yes, explain how you would exploit it (high-level steps, including what input and size you should provide). If not, explain why and what you would change in the code to make it exploitable.

[5 marks]

b. Assuming that the above code is vulnerable (or can be modified to become vulnerable) and that the vulnerability can be successfully exploited (or can be modified to be exploited), then consider the following x86 assembly code fragment, which may be used to exploit the previous vulnerability:

```
1  int
2  main(void)
3  {
4
5      __asm__(
6          "jmp ahead\n"
7          "back:\n"
8          "    popl %ebx\n"
9          "    movl %ebx, 0x8(%ebx)\n"
10         "    xorl %eax, %eax\n"
11         "    movb %al, 0x7(%ebx)\n"
12         "    movl %eax, 0xc(%ebx)\n"
13         "    movl %eax, %edx\n"
14         "    movl $0xb, %eax\n"
15         "    movl 0xc(%ebx), %ecx\n"
16         "    int $0x80\n"
17         "ahead:\n"
18         "    call back\n"
19         "    .string \"/bin/sh\""
20     );
21 }
```

i. Assuming the above assembly snippet (shown in Question 3 (b)) is placed on the stack, what does the assembly code do? Add comments to each line and explain its semantics in the context of the shellcode. For instance, the instruction `mov $0xb, %eax` copies the constant value `0xb` into the register `%eax`. This represents the index that

refers to the system call `execve`.

[12 marks]

ii. An attacker creates a suitable injection vector to exploit the aforementioned memory error. To this end, he places the shellcode in the injection vector, pads it with his initials so as to create a message long enough to overflow `bar`; then the attacker adds the appropriate address at the right place and terminates the message with a `NULL` ('\0'). In other words, the injection vector looks as follows:

```
+-----------------+--------    ---------+------------+----+
|    nop sled     | shellcode             | 0xbfff1234 | \0 |
+-----------------+--------    ---------+------------+----+
```

Next, he runs the program giving this injection vector to it as its first argument. To his surprise, the attack fails. He asks you for help. State why the attack cannot work.

[2 marks]

iii. Show how the shellcode can be modified to make the attack possible. (Note: don't worry if you do not know the exact syntax of instructions; marks will be awarded for a clear explanation.)

[2 marks]

c. Assume the code shown at the beginning of the question is vulnerable (or can be modified to be so) and the vulnerability can be successfully exploited. State and describe what technique(s) would an attacker use to exploit the vulnerability shown at the beginning of the question, if the kernel enforces a non-executable stack protection (again, assume the small program shown at the beginning of the question is exploitable or can be modified to be exploited successfully)?

[4 marks]

4. A secure software development lifecycle requires security engineering to fit into all the phases of the software development process.

a. In which way does security engineering fit into requirements, design, implementation, and testing/assurance? Motivate your answer. [7]

b. What is a threat model and why is it important? [4]

c. Consider Leslie Lamport's Gold Standard.

i. What is it? [2]

ii. Explain each of Lamport's Gold Standard. [12]