Introduction to Cryptography and Software Security
Problem Set 5: Static Analysis and Symbolic Execution

Due date: 6/6/2019

Please provide your solution by completing the on-line quiz.

Part I: Static Analysis

Answer questions 1–5 by choosing the correct answer for each question.

1. Static analysis
   - analyzes a program's code without running it.
   - is a kind of real analysis for solving numeric equations.
   - analyzes the fixed, or static portions of a program.
   - is always better than testing.
2. Tainted flow analysis aims to discover:
   - A flow from an untrusted source to a trusted sink.
   - A flow from a trusted source to both trusted and untrusted sinks.
   - A flow from an untrusted source to both trusted and untrusted sinks.
   - A flow from a trusted source to an untrusted sink
3. Consider the program below, where the variable `fmt` and the first argument to `printf` are untainted, while the result of `fgets` is tainted. Suppose we analyze this with a tainted flow analysis. This program has no bugs, but which of the following would report a **false alarm**?

```
/* int printf(untainted char *fstr, ...); */
/* tainted char *fgets(...); */

char *chomp(char *s) {
    int i, len = strlen(s);
    for (i=0; i<len; i++)
        if (s[i] == '\n') {
            s[i] = '\0';
            break;
        }
    return s;
}

void foo(FILE *networkFP, untainted char *fmt) {
    char buf[100];
    char *str = fgets(buf, sizeof(buf), networkFP);
    char *str1 = chomp(str);
    char *fmt1 = chomp(fmt);
```

We thank Prof. Michael Hicks (UMD) and Coursera for generously sharing some of their teaching resources with us.

```
    printf(fmt1, str1);
}
```

- Flow-sensitive path-sensitive context-**IN**sensitive analysis.
- Flow-**IN**sensitive path-sensitive context-sensitive analysis.
- Flow-sensitive path-**IN**sensitive context-sensitive analysis.

4. Consider the program below, where the first argument to `printf` is untainted, while the result of `fgets` is tainted. Suppose we analyze this with a tainted flow analysis. This program has no bugs, but which of the following would report a **false alarm**?

```
/* int printf(untainted char *fstr, ...); */
/* tainted char *fgets(...); */

char *chomp(char *s) {
    int i, len = strlen(s);
    for (i=0; i<len; i++)
        if (s[i] == '\n') {
            s[i] = '\0';
            break;
        }
    return s;
}

void bar(FILE *networkFP, int testing) {
    char buf[100];
    char *str = fgets(buf, sizeof(buf), networkFP);
    char *str1 = chomp(str);
    if (testing) {
        str1 = chomp("test format");
        printf(str1, "how did the test string look?\n");
    }
}
```

- Flow-sensitive path-sensitive context-sensitive analysis.
- Flow-**IN**sensitive path-sensitive context-sensitive analysis.
- Flow-sensitive path-**IN**sensitive context-sensitive analysis.

5. Which of the following statements, regarding **implicit** flows, is true?
   - Implicit flows can be prevented by encrypting the flowing data.
   - Implicit flows occur when assigning a tainted value to an untainted variable.
   - Implicit flows may occur when assigning an untainted value to an untainted variable, but conditioned on a tainted value.

Part II: Symbolic Execution

Answer questions 6–8 by choosing the correct answer for each question.

6. What is a key advantage of symbolic execution over static analysis?

- Symbolic executors can consider partial programs (e.g., libraries) while static analyzers cannot.
- As a generalized form of testing, when a symbolic executor finds a bug, we are sure it is not a false alarm.
- Symbolic executors are both sound and complete, while static analyzers can only be one or the other.
- Symbolic executors always consider all possible execution path, while static analyzers cannot.

7. Suppose that `x` and `y` in the following program are symbolic. When the symbolic executor reaches the line that prints "`everywhere`", what will the path condition be?

```
/* assume x and y are both symbolic */
void foo(int x, int y) {
    if (x > 5) {
        if (y > 7) {
            printf("here\n");
        } else {
            if (x < 20)
                printf("everywhere\n");
            else
                printf("nowhere\n");
        }
    }
}
```

- $(x > 5) \wedge \neg(y > 7) \wedge (x < 20)$.
- $(x > 5) \wedge \neg(y > 7) \wedge \neg(x < 20)$.
- $(x > 5) \wedge (y > 7) \wedge (x < 20)$.
- $\neg(y > 7) \wedge (x < 20)$.

8. Suppose that `x` in the following program is symbolic. When the symbolic executor reaches the line that prints "`here`", what will the path condition be?

```
/* assume x is symbolic */
void bar(int x) {
    int z;
    if (x > 5)
        z = 5;
    else
        z = 1;
    if (z > 3)
        printf("here\n");
}
```

- $\neg(x > 5) \wedge (z > 3)$.
- $(x > 5) \wedge (z > 3)$.
- $x > 5$.
- $z > 3$.