Takehome Midterm Exam 2

Mudathir Mahgoub
0119655301

May 5, 2018

1. Explain the differences between cross-site scripting (XSS) and cross-site request forgery (CSRF) attacks in your own words. Describe a fictitious scenario in which CSRF can be used to leak confidential information about the victim user. Finally, explain the confused deputy problem in your own words. Give an example of a web attack that occurs as the browser acts as the confused deputy.

2. Explain the "Spectre attack" in your own words and identify the goal of the adversary for this attack. What is the fundamental weakness of the system that the attackers exploit in the Spectre attack? What is the impact of the Spectre attack? In your own words, give a scenario in which an attacker can exploit Spectre attack to achieve something malicious.

3. How does the different cryptographic currency schemes (e.g., Bitcoin) ensure that an arbitrary adversary cannot bundle up some arbitrary transactions to make a block and add it to the blockchain? Explain the process in your own words. Why is it the case that an attacker can double spend only if he possesses 51% of the computing power of the whole network? Explain how is it possible.

4. Explain the buffer overflow attack. Your description of the attack should include the weakness of the system that the adversary exploits, the objective of the adversary in this attack, and the steps the adversary should take to carry out this attack. Describe the different defenses (e.g., Canary, ASLR) that can thwart the buffer overflow attack, and discuss their relative advantages and disadvantages. Can you have buffer overflow attacks in a program written in Java? Please explain your answer.

5. Suppose you are given the following C function $function\_to\_test(.,.)$ that takes as input two integers x and y, and returns another integer z. The function has a bug denoted by the statement "assert(0)".

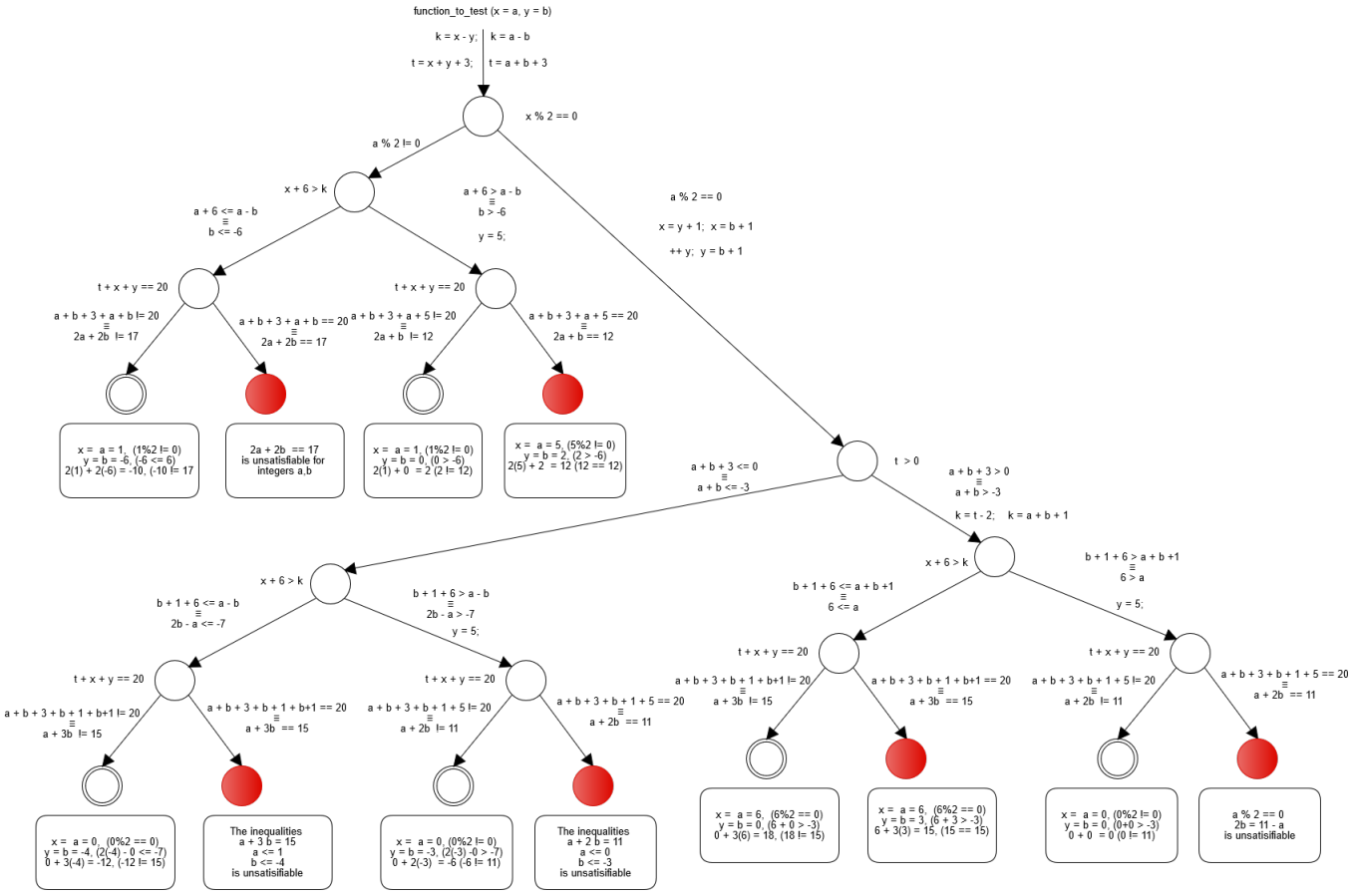
```c
int function_to_test(int x, int y) {
  /*Suppose inputs x and y are set to have symbolic values */

  int k = x - y;
  int t = x + y + 3;
  if(x % 2 == 0) /* check whether x is even; remainder is 0 when x is divisible by 2 */
  {
    x = y + 1;
    ++y; /* Increment y value by 1 */
    if(t > 0){
      k = t - 2;
    }
  }
  if(x+6 > k) {
    y = 5;
  }
  if(t+x+y == 20){ /* Check whether t+x+y is equal to 20*/
    assert(0); /* Bug */
  }
```

```
20
21    int z = (t + x + y) / x;
22    return z;
23  }
```

(a) Your goal is to run symbolic execution on the following program and determine which concrete inputs for the variables x and y trigger the bug.

As shown in the following figure, the inputs that trigger the bug are $x = 5, y = 2$ and $x = 6, y = 3$.



(b) Please show all the path constraints resulting due to the symbolic execution on the following program.

The paths are shown in the figure above:

i. $(a\%2 \neq 0) \wedge (b \leq -6) \wedge (2a + 2b \neq 17)$
ii. $(a\%2 \neq 0) \wedge (b \leq -6) \wedge (2a + 2b = 17)$
iii. $(a\%2 \neq 0) \wedge (b > -6) \wedge (2a + b \neq 12)$
iv. $(a\%2 \neq 0) \wedge (b > -6) \wedge (2a + b = 12)$
v. $(a\%2 = 0) \wedge (a + b \leq -3) \wedge (2b - a \leq -7) \wedge (a + 3b \neq 15)$
vi. $(a\%2 = 0) \wedge (a + b \leq -3) \wedge (2b - a \leq -7) \wedge (a + 3b = 15)$
vii. $(a\%2 = 0) \wedge (a + b \leq -3) \wedge (2b - a > -7) \wedge (a + 2b \neq 11)$
viii. $(a\%2 = 0) \wedge (a + b \leq -3) \wedge (2b - a > -7) \wedge (a + 2b = 11)$
ix. $(a\%2 = 0) \wedge (a + b > -3) \wedge (6 \leq a) \wedge (a + 3b \neq 15)$
x. $(a\%2 = 0) \wedge (a + b > -3) \wedge (6 \leq a) \wedge (a + 3b = 15)$
xi. $(a\%2 = 0) \wedge (a + b > -3) \wedge (6 > a) \wedge (a + 2b \neq 11)$

xii. $(a\%2 = 0) \wedge (a + b > -3) \wedge (6 > a) \wedge (a + 2b = 11)$

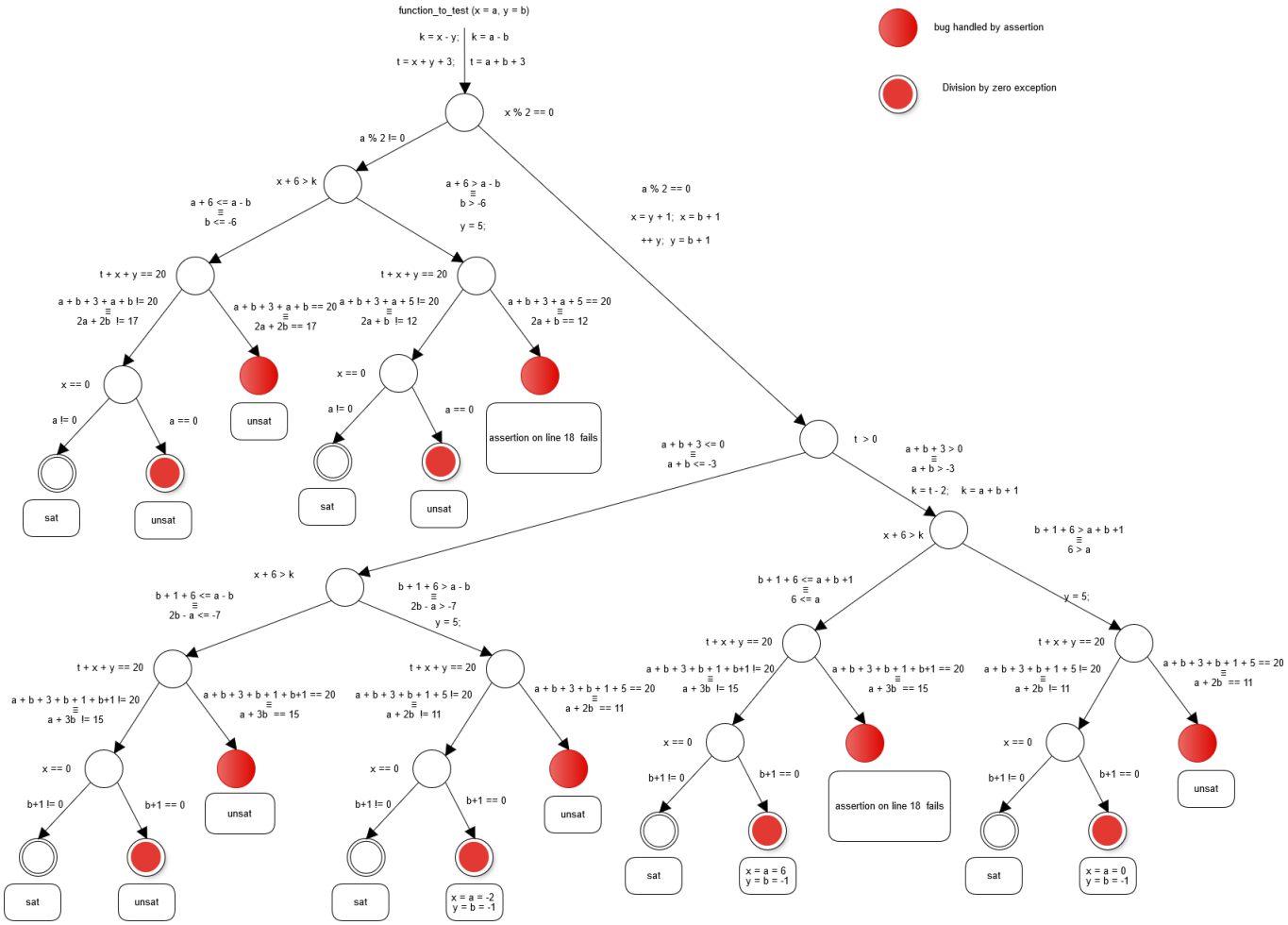
(c) Additionally, assume that you would like the symbolic execution to identify whether the function has any "divide by zero" exceptions. How would you modify the symbolic execution to identify such cases?

Only line 20 uses integer division, therefore we need to ensure the denominator x in $(t + x + y)/x$ to be non zero. To achieve that we can imagine we have an assertion $assert(x\ != 0)$ before line 21. Assuming the path constraint until line 21 is C and the symbolic value of x is x', then if the formula $C \wedge (x' == 0)$ is sat, then there is a "divide by zero" exception. If the formula is unsat, then this exception wouldn't be thrown.

Paths that already proved to be unsat or fail some assertions could be excluded.

(d) Provide the modified path constraints due to the additional divide by zero checking.

The following picture shows the paths with division by zero. Note that paths already proved to be unsat or fail the assertion in line 18 are excluded because the execution would not reach line 21. As shown in the figure, the inputs $x = -2, y = -1$, $x = 6, y = -1$, or $x = 0, y = -1$ throw "divide by zero exception".



Below are the constraints that checks when $x = 0$:

i. $(a\%2 \neq 0) \wedge (b \leq -6) \wedge (2a + 2b \neq 17) \wedge (a = 0)$

ii. $(a\%2 \neq 0) \wedge (b > -6) \wedge (2a + b \neq 12) \wedge (a = 0)$

iii. $(a\%2 = 0) \wedge (a + b \leq -3) \wedge (2b - a \leq -7) \wedge (a + 3b \neq 15) \wedge (b + 1 = 0)$

iv. $(a\%2 = 0) \wedge (a + b \leq -3) \wedge (2b - a > -7) \wedge (a + 2b \neq 11) \wedge (b + 1 = 0)$

v. $(a\%2 = 0) \wedge (a + b > -3) \wedge (6 \leq a) \wedge (a + 3b \neq 15) \wedge (b + 1 = 0)$

vi. $(a\%2 = 0) \wedge (a + b > -3) \wedge (6 > a) \wedge (a + 2b \neq 11) \wedge (b + 1 = 0)$

(e) Explain what would change if you were to use concolic execution instead of symbolic execution while answering sub-question (a) described above.

Concolic execution assigns random values to the input variables x and y. Initially concolic execution starts with an empty path. Then based on the concrete random inputs, the execution traverses the code, and whenever it encounters a branch, it adds the corresponding condition to the path as a conjunction. When it finishes, the path obtained is the path of the random input. To explore other paths, the last condition in the path can be negated and the path formula can be passed to the SMT solver to get an input for that path. To explore parent branches, the last condition can be removed and the process can be repeated by negating the last condition in the parent path constraint.

For example, if the random input is $x = 6, y = 0$, then the concolic execution would get the following constraint:

$$(a\%2 = 0) \wedge (a + b > -3) \wedge (6 \leq a) \wedge (a + 3b \neq 15)$$

Negating the last condition would generate this formula

$$(a\%2 = 0) \wedge (a + b > -3) \wedge (6 \leq a) \wedge (a + 3b = 15)$$

The SMT solver would return the input $x = 6, y = 3$ which hits the bug.