**SE 421 Fall 2021 Assignment 2 (15 points), Assigned: 9/1, due: Wednesday, 9/8**

**Name (Last, First): Ogbondah, Chimzim**

**Submission**: (a) The answers should be typed. (b) The first page should include the top two lines with your last and the first name. (c) The question should be included for every answer. (d) The file should be named HW2-lastname-firstname. Submit the homework through Canvas.

Similar examples are in the lecture notes. The examples were further elaborated during lectures. Read the notes carefully. In case you missed any lecture, listen to it. Studying notes and attending lectures would be very helpful for the homework. The submission and late policy are as described in the syllabus.

**Problem 1 (2 points):** In less than 250 words, summarize the attack described by Ken Thompson in the paper *Reflections of Trusting Trust*. Include an answer to why can the attack not be detected by recompiling.

**Answer**: In *Reflections of Trusting* by Ken Thompson, he gives background on himself and how he saw that its more efficient to program in a team than alone. He then goes into his college days and self-replicating programs. He breaks it up into 3 stages. Stage 1, replicating any extra baggage along with the main algorithm and writing feasibility. Stage 2, which talks about printable and not printable characters with the C compiler and altering for new ones. Finally, Stage 3, which goes more into trojan horses or ‘bugs’ and how they can go undetected by the compiler. These bugs are errors in the program which don’t necessarily cause syntax errors but cause the program to break on certain inputs. Ken Thompson talks about how you shouldn’t 100% trust code that you didn’t write because compiling or recompiling the program won’t show where the ‘bugs’ or errors are in the algorithm/program.

**Problem 2 (8 points):** Each code snippet below (A, B, C, D, and E) contains a security vulnerability. Understand and assign each snippet one of the CWEs from the following Table. A given code snippet may be associated with multiple CWEs.

**Background:** Multiple CVEs could have a common root cause. MITRE Corporation[[1]](#footnote-1) developed a list of Common Weakness Enumerations (CWEs)[[2]](#footnote-2) as a classification and categorization scheme of CVEs based on the root cause. For example, the CWE-416[[3]](#footnote-3) is a category of vulnerabilities where the root cause is referencing a memory after it has been freed. A CWE corresponds to multiple CVEs. Occasionally, a CVE could map to multiple CWEs because of multiple root causes.

|  |  |  |
| --- | --- | --- |
| **CWE** | **CWE URL** | **Associated Code Snippets** |
| CWE-416: Use After Free | https://cwe.mitre.org/data/definitions/416.html | Code Snippet A |
| CWE-121: Stack-based Buffer Overflow | https://cwe.mitre.org/data/definitions/121.html | Code Snippet D |
| CWE-20: Improper Input Validation | https://cwe.mitre.org/data/definitions/20.html | Code Snippet B |
| CWE-401: Missing Release of Memory after Effective Lifetime | https://cwe.mitre.org/data/definitions/401.html | **Code Snippet C** |
| CWE-772: Missing Release of Resource after Effective Lifetime | https://cwe.mitre.org/data/definitions/772.html | Code Snippet E |

**Code Snippet A**

|  |
| --- |
| #define BUFFER\_SIZE 512    int code\_snippet\_A() {  int error\_code = 0; // success (no failure)  char\* buffer = (char\*) malloc(BUFFER\_SIZE);  if(buffer == NULL) {  error\_code = 1; // an error occured  logError("operation aborted due failure to allocate memory", buffer);  return error\_code;  }  ...  ...  error\_code = 0; // success  free(buffer);  logInfo("operation was successfull, pointer has been freed", buffer);  return error\_code;  } |

**Code Snippet B**

|  |
| --- |
| void code\_snippet\_B(bool c1, bool c2, bool c3, int a1, int a2) {  int x = a1 + a2;  int d = a1;  if(c1){  x = a1;  }else{  x = a2 - 1;  }  if(c2){  if(!c3){  d = d - a1;  }  }else{  d = d + 1;  }  int z = x / d;  } |

**Code Snippet C**

|  |
| --- |
| int code\_snippet\_D(int argc, char \*\*argv) {  if (argc < 2) {  return 1;  }  int val = atoi(argv[1]);  if (val > 0) {  char\* buffer = malloc(val \* sizeof(char \*));  if (buffer == NULL) {  return 2;  }  for (int i = 0; i < val; i++) {  buffer[i] = 'A';  }  }  return 0;  } |

**Code Snippet D**

int code\_snippet\_C(int argc, char \*argv) {

char buf[64];

strcpy(buf, argv[1]);

return 0;

}

**Code Snippet E**

|  |
| --- |
| int code\_snippet\_E() {  // reads the content of the configuration file in system  FILE\* file = fopen("database.config", "r");  if (!file) {  printf("cannot open %s\n", fName);  return -1; // error  } else {  // reads the content of "file" into "buffer"  char\* buffer = readFile(file);  if(buffer != NULL) {  if(checkChecksum(buf)) {  return -1; // error  } else {  decodeBuffer(buffer);  }  }  }  fclose(f);  return 0; // success  } |

**Problem 3 (5 points):** Answer the following questions using the given code.

****

1. Give the number of behaviors produced by the software.
   1. 6
2. List all the behaviors using the line numbers 3, 4, 5, 6, 8, 9, 10, 12, 15, 17 for program statements.
   1. 3 4 5 8 15 17
   2. 3 4 5 8 9 10 17
   3. 3 4 5 6 8 15 17
   4. 3 4 5 6 8 9 12 17
   5. 3 4 5 8 9 12 17
   6. 3 4 5 6 8 9 10 17
3. List the subset of behaviors that can result in the DBZ vulnerability.
   1. All subsets can result in a DBZ
4. Give the input values that can cause division by zero.
   1. { main(anyNumber,0, T/F, T, T):, main(a1=a2, F/T, T, F), main(anyNumber, -1, T/F, F, T/F) }

1. https://cwe.mitre.org/ [↑](#footnote-ref-1)
2. https://cwe.mitre.org/data/ [↑](#footnote-ref-2)
3. https://cwe.mitre.org/data/definitions/416.html [↑](#footnote-ref-3)