SE 421 Fall 2020 Assignment 1 (15 points), Assigned: 8/17/20, due: Wednesday, 8/26/2020

Name (Last, First): **Morellas, Stamatios**

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 HW1-lastname-firstname. Submit the homework through Canvas.
The submission and late policy as described in the syllabus applies.

Problem 1 (2 points): Watch the short introductory NOVA video and write one or two sentence answers to the following questions. Video: https://www.youtube.com/watch?v=LbFCwxANBRc

a) What is the main idea of the video?

The main idea of this video is to describe to the viewer the mechanics by which a network operates, as well as how networks can be configured to be the most efficient. By network, this essentially refers to a graph with a certain amount of "nodes" and the corresponding "edges" or relations between those nodes. In any case, the objective with data structures like these is to maximize the connectivity and to minimize the actual number of connections in order for a graph to be the most efficient. Small world networks are found everywhere, including our brain and even in nature.

b) Describe your own idea of software as a network. Is it a *small world network*?

Software can be considered as a small world network. In an object-oriented architecture, the "nodes" represent the various classes of objects in the software, while the "edges" represent the relationships between those objects and how they interact with different components in the software. A small world network has a set of nodes that act as "hubs" in order to minimize the number of redundant connections, meaning that there will always be at least 1 path or set of relations in which the nodes on the graph can be accessed from.

Problem 2 (13 points): The National Vulnerability Database (NVD) (https://nvd.nist.gov/) is the U.S. government repository of software vulnerabilities. The NVD maintains a list of Common Vulnerabilities and Exposures (CVEs) for widely used software. A **CVE**[1], in simple terms, is a publicly disclosed software vulnerability that an attacker can exploit to adversely impact confidentiality, integrity, or availability. Each CVE is assigned a unique id of the format CVE-YEAR-UniqueNumber. The database has additional information. For example, CVE-2017-11882 (https://nvd.nist.gov/vuln/detail/CVE-2017-11882) for a memory corruption vulnerability in Microsoft Office gives the severity, affected product versions, and a technical description for the vulnerability.

Multiple CVEs could have a common root cause. MITRE Corporation[2] developed a list of Common Weakness Enumerations (**CWEs**)[3] as a classification and categorization scheme of CVEs based on the root cause. For example, the CWE-416[4] 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. MITRE maintains a list of Top-25 CWEs in 2019[5], these are the CWEs with the largest number of associated CVEs found in the year 2019.

The NVD ranks vulnerabilities using the Common Vulnerability Scoring System (CVSS)[6] to enable a decision matrix of risk, remediation, and mitigation specific to particular environment and risk tolerance.

1 https://en.wikipedia.org/wiki/Common_Vulnerabilities_and_Exposures
2 https://cwe.mitre.org/
3 https://cwe.mitre.org/data/
4 https://cwe.mitre.org/data/definitions/416.html
5 https://cwe.mitre.org/data/definitions/1200.html
6 https://en.wikipedia.org/wiki/Common_Vulnerability_Scoring_System

Use the given Excel sheet to answer the following questions. The spread sheet is for Linux kernel vulnerabilities mined from the NVD for the period March 1999 to June 2020.

a) Plot a CWE histogram for the top ten CWE with the highest number of CVEs. Show the number of CVEs for each of the ten CWEs.

b) Plot a detailed CWE histogram for the top ten CWEs. For each of these CWE, show the number of CVEs with "Base Severity" attributes: Critical, High, Low, Medium, and None.

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

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

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_C(int argc, char *argv) {
    char buf[64];
    strcpy(buf, argv[1]);
    return 0;
}
```

Code Snippet D

```
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 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
}
```