Secure programming
Homework 2
Due: Friday, Nov. 11, 2016

The practice of static analysis for C code.

(1) Please use the following tools to analyze the code in Fig.2. Note that you should review all issues and explain the important ones to write down your comments. If possible, please try to modify this program to pass the security checking.

(2) Try the same thing as part (1) by finding another program(s)/project(s) (from Internet or your homework in the past) with the length more than 300 lines (at least 5 security problems).

Splint is a tool for statically checking C programs for security vulnerabilities and coding mistakes. With minimal effort, Splint can be used as a better lint. If additional effort is invested adding annotations to programs, Splint can perform stronger checking than can be done by any standard lint. (see: http://www.splint.org/)

VisualCodeGrepper V2.0.0 (http://sourceforge.net/projects/visualcodegrepp/)
Code security review tool for C/C++, C#, VB, PHP, Java and PL/SQL.

On-line or IDE Tools (such as PVS-Studio, Kiuwan, CPPCheck or others)
See Static source code analysis tools for C http://spinroot.com/static/
(WIKI) List of tools for static code analysis
https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis

---

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_WORDS 50
#define WORD_LEN 20
#define BUF_SIZE (1024)

int read_line(char str[], int n);
void quicksort(char **low, char **high);
char **split(char **low, char **high);
int winner(void);

int main(void)
{
  char *words[MAX_WORDS], word[WORD_LEN+1];
  int i, cho, num_words = 0;

  printf("Enter Choice: 1:sort or 2:game -> ");
  scanf("%d", &cho);
  if (cho==1){
       getchar();
  }
  else if (cho==2) {
       winner();
       return 0;
  }
  else {
    printf("Error!!");
    return 0;
  }

  for (;;) {
    if (num_words == MAX_WORDS) {
      printf("-- No space left --\n");
      break;
    }

    printf("Enter word: ");
    read_line(word, WORD_LEN);
    if (strlen(word) == 0)
      break;

    words[num_words] = malloc(strlen(word) + 1);
    if (words[num_words] == NULL) {
      printf("-- No space left --\n");
      break;
    }

    strcpy(words[num_words], word);
    num_words++;
  }
```

```
    quicksort(words, words + num_words - 1);

    printf("\nIn sorted order:");
    for (i = 0; i < num_words; i++)
        printf(" %s", words[i]);
    printf("\n");

    return 0;
}

int read_line(char str[], int n)
{
    int ch, i = 0;

    while ((ch = getchar()) != '\n')
        if (i < n)
            str[i++] = ch;
    str[i] = '\0';
    return i;
}

void quicksort(char **low, char **high)
{
    char **middle;

    if (low >= high) return;
    middle = split(low, high);
    quicksort(low, middle - 1);
    quicksort(middle + 1, high);
}

char **split(char **low, char **high)
{
    char *part_element = *low;

    for (;;) {
        while (low < high && strcmp(part_element, *high) <= 0)
            high--;
        if (low >= high) break;
        *low++ = *high;

        while (low < high && strcmp(*low, part_element) <= 0)
            low++;
        if (low >= high) break;
        *high-- = *low;
    }

    *high = part_element;
    return high;
}

int winner() {
    char* inBuf;
    char* outBuf;
    char* fmt = "the winner is: %s";

    printf("Enter name: \n");
    inBuf = (char*) malloc(BUF_SIZE);
    if (inBuf == NULL) {
        return -1;
    }
    read(0, inBuf, BUF_SIZE);
    outBuf = (char*) malloc(BUF_SIZE);
    if (outBuf == NULL) {
        return -1;
    }
    sprintf(outBuf, fmt, inBuf);
    fprintf(stdout, "%s\n", outBuf);
    fprintf(stderr, "%s\n", outBuf);
    free(inBuf);
    free(outBuf);
      }
```

Fig. 2 The C code to test the static analysis tools