COMP1511 18s2 (webcms)

Code Examples from Lectures on illegal_C

stack_inspect.c

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

void f() {
    int i;
    int x = 9;
    int a[10];

    for (i = 0; i < 16; i++)
        printf("%2d: Address %x contains %p\n", i, &a[10+i], a[10+i]);
}

int main(void) {
    int a = 7;
    printf("function main is at address 0x%x\n", &main);
    printf("function f is at address 0x%x\n", &f);
    f();
    return 0;
}
```

invalid0.c

```
Run at CSE like this


$ gcc-7 invalid0.c -o invalid0
$ ./invalid0
42 42 42 77 77 77 77 77 77 77
```

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

int main(void) {
    int a[10];
    int b[10];
    printf("a[0] is at address %p\n",&a[0]);
    printf("a[9] is at address %p\n", &a[9]);
    printf("b[0] is at address %p\n",&b[0]);
    printf("b[9] is at address %p\n", &b[9]);

    for (int i = 0; i < 10; i++) {
        a[i] = 77;
    }

    // loop writes to b[10] .. b[12] which don't exist -
    // with gcc 7.3 on x86_64/Linux
    // b[12] is stored where a[0] is stored
    // with gcc 7 on CSE lab machines
    // b[10] is stored where a[0] is stored

    for (int i = 0; i <= 12; i++) {
        b[i] = 42;
    }

    // prints 42 77 77 77 77 77 77 77 77 77 on x86_64/Linux
    // prints 42 42 42 77 77 77 77 77 77 77 at CSE
    for (int i = 0; i < 10; i++) {
        printf("%d ", a[i]);
    }
    printf("\n");

    return 0;
}
```

invalid1.c

```
Run at CSE like this


$ gcc-7 invalid1.c -o invalid1
$ ./invalid1
42 42 42 77 77 77 77 77 77 77
```

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

int main(void) {
    int i;
    int a[10];
    printf("i is at address %p\n", &i);
    printf("a[0] is at address %p\n", &a[0]);
    printf("a[9] is at address %p\n", &a[9]);
    printf("a[10] would be stored at address %p\n", &a[10]);

    // loop writes to a[10] .. a[11] which don't exist -
    // but with gcc 7 on x86_64/Linux
    // i would be stored where a[11] is stored

    for (i = 0; i <= 11; i++) {
        a[i] = 0;
    }

    return 0;
}
```

invalid2.c

```
Run at CSE like this


$ gcc-7 invalid2.c -o invalid2
$ ./invalid2
answer=42
```

```c
#include <stdio.h>

void f(int x);

int main(void) {
    int answer = 36;
    printf("answer is stored at address %p\n", &answer);

    f(5);
    printf("answer=%d\n", answer); // prints 42 not 36g

    return 0;
}

void f(int x) {
    int a[10];

    // a[19] doesn't exist
    // with gcc-7 at CSE variable answer in main
    // happens to be where a[21] would be
    // on 64-bit Linux try 19 instead of 21

    printf("a[21] would be stored at address %p\n", &a[21]);

    a[21] = 42;
}
```

invalid3.c

```
Run at CSE like this


$ gcc-7 invalid3.c -o invalid3
$ ./invalid3

I will never be printed.
argc was 1
$
```

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

void f(void);


void f(void);

int main(int argc, char *argv[]) {

    f();

    if (argc > 0) {
        printf("I will always be printed.\n");
    }

    if (argc <= 0) {
        printf("I will never be printed.\n");
    }

    printf("argc was %d\n", argc);
    return 0;
}


void f() {
    int a[10];

    // function f has it return address on the stack
    // the call of function f from main should return to
    // the next statement which is:  if (argc > 0)
    //
    // with gcc7 at CSE  f's return address is stored where a[11] would be
    //
    // so changing a[11] changes where the function returns
    //
    // adding 28 to a[11] happens to cause it to return several statements later
    // at the printf("I will never be printed.\n");
    //
    // on 64 bit linux  machines try instead a[14] += 24

    a[11] += 28;
```

invalid4.c

```
Run at CSE like this


$ gcc-7 invalid4.c -o invalid4
$ ./invalid4
authenticated is at address 0xff94bf44
password is at address 0xff94bf3c

Enter your password: 123456789

Welcome. You are authorized.
$
```

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

int main(int argc, char *argv[]) {
    int authenticated = 0;
    char password[8];

    printf("authenticated is at address %p\n",&authenticated);
    printf("password[8] would be at address %p\n",&password[8]);

    printf("Enter your password: ");
    int i = 0;
    int ch = getchar();
    while (ch != '\n' && ch != EOF) {
        password[i] = ch;
        ch = getchar();
        i = i + 1;
    }
    password[i] = '\0';

    if (strcmp(password, "secret") == 0) {
        authenticated = 1;
    }

    // a password longer than 8 characters will overflow the array password
    // the variable authenticated is at the address where
    // where password[8] would be and gets overwritten
    //
    // This allows access without knowing the correct password

    if (authenticated) {
        printf("Welcome. You are authorized.\n");
    } else {
        printf("Welcome. You are unauthorized.  Your death will now be implemented.\n");
        printf("Welcome. You will experience a tingling sensation and then death. \n");
        printf("Remain calm while your life is extracted.\n");
    }

    return 0;
}
```

tmp.c

```
Run at CSE like this

$ gcc-7 invalid3.c -o invalid3
$ ./invalid3

I will never be printed.
argc was 1
$
```

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

void f(void);


void f(void);

int main(int argc, char *argv[]) {

    f();

    if (argc > 0) {
        printf("I will always be printed.\n");
    }

    if (argc <= 0) {
        printf("I will never be printed.\n");
    }

    printf("argc was %d\n", argc);
    return 0;
}


void f() {
    int a[10];

    // function f has it return address on the stack
    // the call of function f from main should return to
    // the next statement which is:  if (argc > 0)
    //
    // with gcc7 at CSE  f's return address is stored where a[11] would be
    //
    // so changing a[11] changes where the function returns
    //
    // adding 28 to a[11] happens to cause it to return several statements later
    // at the printf("I will never be printed.\n");
    //
    // on 64 bit linux  machines try instead a[16] += 32

    a[16] += 32;
```

invalid3.sh

```
rm -f tmp.$$
for i in `seq  10 16`
do
    for j in `seq 12 32`
    do
        #echo $i $j
        perl -p -e "s/a\[\d+\] \+= \d+/a[$i] += $j/" invalid3.c >tmp.c
        gcc-7 tmp.c -o ./tmp.$$
        #./tmp.$$ 2>&1
        if ./tmp.$$ 2>&1 |egrep always >/dev/null 2>&1
        then
            echo -n .
            continue
        fi
        if ./tmp.$$ 2>&1 |egrep never
        then
            echo "code was a[$i] += $j"
            exit
        else
            echo -n +
        fi
    done
done
rm -f tmp.$$
```

invalid2.sh

```
for i in `seq 11 25`
do
    perl -p -e "s/a\[\d+\] =/a[$i] =/" invalid2.c|
    gcc-7 -x c - -o ./tmp.$$
     ./tmp.$$ 2>&1 >/dev/null|egrep 42 && echo "code was a[$i] = 42"
done
rm -f tmp.$$
```

2.sh

```
for i in `iota 10 20`
do
    for j in `iota 1 20`
    do
        echo -n $i $j ''
        perl -p -e "s/a\[\d+\] \+=.*/a[$i] \+= $j;/" invalid3.c  >tmp.c &&
        gcc tmp.c &&
        a.out 2>/dev/null
    done
done|egrep 42
```