Holistic Software Security (ECE 695) – Assignment 0

Aravind Machiry
amachiry@purdue.edu

The goal of this assignment is to assess your understanding of software security concepts. Answer each of the following questions. *When in doubt, always give more details.*

Problem 1

Lets see your C (and C++). Try to find all security issues (if any) in each of the following code snippets.

a) Baby steps!

```
int main(int argc, char **argv) {
   char buf[10];
   strcpy(buf, argv[0]);
   ....
}
```

b) Lets dance!

```
size_t s;
char *p;
scanf("%lu", &s);
p = (char*)malloc(s + 4);
if (p) {
  strcpy(p, "HDR");
  fgets(p+3, s, stdin);
} else {
  printf("Out of memory!\n");
  return -1;
}
...
int main(int argc, char **argv) {
   char buf[10];
   strcpy(buf, argv[0]);
   ....
}
```

c) I am fancy!

In the following code, `dfsize` is the size of buffer pointed by `dfstr`.

```
static void webize( char* str, char* dfstr, int dfsize ) {
char* cp1;
char* cp2;

for ( cp1 = str, cp2 = dfstr;
     *cp1 != '\0' && cp2 - dfstr < dfsize - 1;
     ++cp1, ++cp2 ) {
switch ( *cp1 ) {
  case '<':
  *cp2++ = '&';
  *cp2++ = 'l';
  *cp2++ = 't';
  *cp2 = ';';
  break;
  case '>':
   *cp2++ = '&';
   *cp2++ = 'g';
   *cp2++ = 't';
   *cp2 = ';';
   break;
  default:
   *cp2 = *cp1;
         break;
  }
 }
 *cp2 = '\0';
}
```

d) Sizing!!

```
int *p;
int q[20];
unsigned s;
...
memset(q, 0, sizeof(q));
...
p = malloc(s);
if (p != NULL) {
   memset(p, 'A', sizeof(p));
} else {
  return -1;
}
```

e) Lets print!

```
char format[20];
// Read format to display the log string.
scanf("%19s", format);
...
// Print the log_str in required format.
printf(format, log_str);
...
```

f) The amazing destructor!

```
class base {
  public:
  base() {

  }

  ~base() {
  }
}
class sub: public base {
  public:
```

```
  sub() {

  }

  ~sub() {
  }
}
int main() {
   base *b = new sub();
   ....
   delete b;
}
```

g) The amazing check!

```
char fl;
....
int ret = sscanf(buf, %s, &fl);
if (ret != 1) {
   printf("Read Error\n");
   return -1;
}
```

Problem 2

Lets check your understanding of runtime internals!! Make sure that you justify your answer.

a) [Yes/No] If we avoid storing return address on runtime stack then stack-based buffer overflows do not cause any security issues (especially, control-flow hijacking).

b) [Yes/No] We can always prove that a given program does not have any security vulnerabilities.

c) [Yes/No] Exhaustive testing proves that the a given program does not have any bugs.

Problem 3

Operating Systems (OS) security concepts.

a) [Yes/No] A process can know physical addresses of its virtual addresses. Justify your answer in either case.

b) [Yes/No] A process can read and write memory that belong to the operating system kernel. Justify your answer in either case.

c) Operating system should always sanitize (i.e., verify) addresses given by a user process. Why? E.g., Destination address provided for read/write `syscall`.

d) [Yes/No] Is there any security issue in the following code? Justify your answer in either case.

```
unsigned gl;
char flag_buf[4];
...
unsigned i;
if (!copy_from_user(&i, buf, sizeof(i)) {
  if (i<4) {
    if (!copy_from_user(&gl, buf, sizeof(gl)) {
            flag_buf[gl] = 0;
    }
  }
}
...
```