Splint: Light-weight program analysis

René Rydhof Hansen

Exercise Sheet
E05-02230

These exercises should be handed in on Friday 9. September 2005 in the mailbox in building 322.

Exercise 1: Getting Started

1. Download the `splint` TGZ-file from `http://www.splint.org`
2. Install `splint` locally (on Linux this is done simply by unpacking the TGZ). Remember to set the `LARCH_PATH` environment variable to point to the `lib/` subdirectory of the `splint` package; similarly the binary can be found in the `bin/` subdirectory.

   Example:

```
$ cd $HOME
$ gtar -xvzf splint-3.1.1.Linux.tgz
  ...
$ export LARCH_PATH=$HOME/splint-3.1.1/lib
$ $HOME/splint-3.1.1/bin/splint
Splint 3.1.1 --- 28 Apr 2003

Source files are .c, .h and .lcl files.  If there is no suffix,
   Splint will look for <file>.c and <file>.lcl.

Use splint -help <topic or flag name> for more information

Topics:

   annotations (describes source-code annotations)
   comments (describes control comments)
   flags (describes flag categories)
   flags <category> (describes flags in category)
   flags all (short description of all flags)
   flags alpha (list all flags alphabetically)
   flags full (full description of all flags)
   mail (information on mailing lists)
   modes (show mode settings)
   parseerrors (help on handling parser errors)
```

```
prefixcodes (character codes in namespace prefixes)
references (sources for more information)
vars (environment variables)
version (information on compilation, maintainer)
```

3. Test your installation by running `splint` on some very small C-programs, e.g., those from the lecture. Remember to use the `+bounds` option if you want to check for buffer overflows.

Exercise 2: Analyse your password program

1. Run `splint` on the password program you wrote in Lab 0. Try running `splint` with different options.
2. You should document your experiments: how many errors were found, what kind of errors, how would you fix the errors.

Exercise 3: `Splint` as a debugger

1. Use `Splint` with option `-exportlocal` on program 1 shown in Figure 1.
2. Document which errors were found and how you could fix them (you do not need to write the corrected C program, just explain how the individual errors could/should be fixed).
3. Which of the errors are security critical?

Exercise 4: Buffer Overflow Annotations

1. What should you write instead of the `???`'s in the `splint` annotation for program 2 (shown in Figure 2) in order for `splint` to accept it? Use options `+bounds -exportlocal`.

Exercise 5: More Buffer Overflow Annotations

1. What should you write instead of the `???`'s in the `splint` annotation for program 3 (shown in Figure 3) in order for `splint` to accept it? Use options `+bounds -exportlocal`.

Exercise 6: Testing someone elses password program

1. Use `splint` on someone elses password program
2. Document your experience and compare to the results from Exercise 1

Exercise 7: Evaluation

1. Is `splint` useful? Why? why not?
2. Would you use it for your next security-related project? Why? Why not?
3. What kind of tool(s) do you think would be useful to you?
4. What kind of errors can `splint` *not* find?
5. Can you construct a program that is accecpted by `splint` but contains a serious security flaw?

```
#include <stdio.h>

#define MAXNAME 15

void say_hello(char *user)
{
  /* Say hi to the user */
  printf("Hello %s!\n", user);
}

int logging(char *user, int count)
{
  /* Print a log message */
  printf("Logging (id: #%d): %s said hello ", user, count);

  /* Logging succeeded */
  return 1;
}

main()
{
  int  counter;
  char *name, *tmpname;

  /* Allocate memory for and get the user name */
  name = (char *) malloc(MAXNAME);
  (void) gets(name);

  /* Logging */
  tmpname = name;
  logging(tmpname,counter);
  free(tmpname);

  /* Greet the user */
  say_hello(name);

  /* Free used memory */
  free(name);
}
```

Figure 1: Program 1

```
#include <string.h>

#define MAXNAME 30

main(int argc, char *argv[])
     /*@requires maxRead(???) >= ???@*/
{
  char progname[MAXNAME];


  strncpy(progname,argv[0],MAXNAME);

  return 0;
}
```

Figure 2: Program 2

```
#include <string.h>

#define MAXNAME 30

/* Copy string and add "-->" */
void copy(/*@unique@*/char *from, /*@out@*/char *to, size_t size)
     /*@requires ??? @*/
{
  strncpy(to,from,size);
  to[size-1] = '<';
  to[size+0] = '-';
  to[size+1] = '-';
  to[size+2] = '\0';
}

main()
{
  char name[MAXNAME+7], name2[MAXNAME] = "antidisestablishmentarianism!";

  copy(name2,name,MAXNAME);
  printf("Input : %s\nOutput: %s\n",name2,name);

  return 0;
}
```

Figure 3: Program 3