Software Security A.A.2018-2019
Individual Project 1

Serena Ferracci 1649134

October 2018

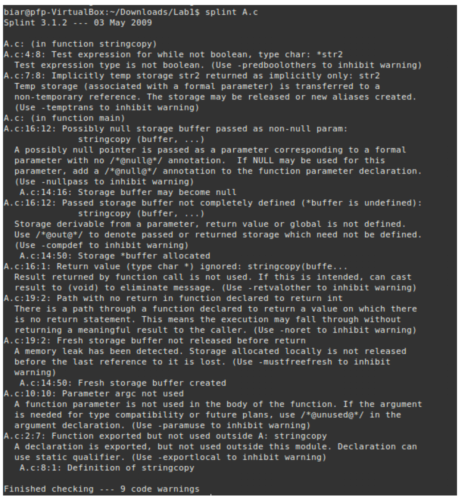
Splint

Splint is a tool for statically checking C programs for security vulnerabilities and programming mistakes. Splint does many of the traditional checks including unused declarations, type inconsistencies, use before definition, unreachable code, ignored return values, execution paths with no return, likely infinite loops, and fall through cases. More powerful checks are made possible by additional information given in source code annotations. Annotations are stylized comments that document assumptions about functions, variables, parameters and types. The program can add these annotations to the code to for a better result and doing that he will understand better the code and what is going on, so he can also spot simpler the possible errors in the code. Moreover, users can define new annotations and an associated check to extend Splint's checking ability. The negative side of these annotations: if a user want to analyze a file or a project that is already written it is not so easy to put annotations, especially when the file is quite big.

Another important down side is that the tool gives a lot of false positives or warnings that might be unimportant, as an example: coding style recommendations. To work around this problem, the tool offers the possibility to customize the showed result selecting what types of errors are reported using command line flags and stylized comments in the code.

In conclusion, Splint is a great tool for finding errors in all areas of an application, but if a programmer is specifically looking for security bugs, Splint can not compete with the other tools. It is recommended that Splint be used along side of another scanner like RATS or Flawfinder, which would re-enforce the search for security vulnerabilities.

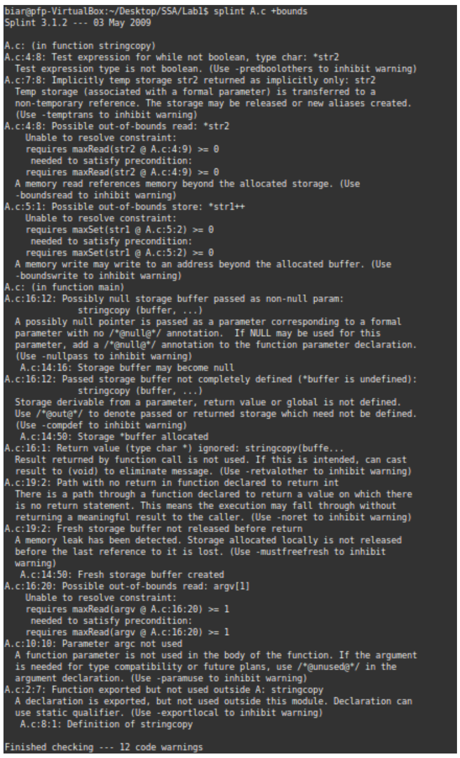
File A



```
biar@pfp-VirtualBox:~/Downloads/Lab1$ splint A.c
Splint 3.1.2 --- 03 May 2009

A.c: (in function stringcopy)
A.c:4:8: Test expression for while not boolean, type char: *str2
  Test expression type is not boolean. (Use -predboolothers to inhibit warning)
A.c:7:8: Implicitly temp storage str2 returned as implicitly only: str2
  Temp storage (associated with a formal parameter) is transferred to a
  non-temporary reference. The storage may be released or new aliases created.
  (Use -temptrans to inhibit warning)
A.c: (in function main)
A.c:16:12: Possibly null storage buffer passed as non-null param:
              stringcopy (buffer, ...)
  A possibly null pointer is passed as a parameter corresponding to a formal
  parameter with no /*@null@*/ annotation.  If NULL may be used for this
  parameter, add a /*@null@*/ annotation to the function parameter declaration.
  (Use -nullpass to inhibit warning)
   A.c:14:16: Storage buffer may become null
A.c:16:12: Passed storage buffer not completely defined (*buffer is undefined):
              stringcopy (buffer, ...)
  Storage derivable from a parameter, return value or global is not defined.
  Use /*@out@*/ to denote passed or returned storage which need not be defined.
  (Use -compdef to inhibit warning)
   A.c:14:50: Storage *buffer allocated
A.c:16:1: Return value (type char *) ignored: stringcopy(buffe...
  Result returned by function call is not used. If this is intended, can cast
  result to (void) to eliminate message. (Use -retvalother to inhibit warning)
A.c:19:2: Path with no return in function declared to return int
  There is a path through a function declared to return a value on which there
  is no return statement. This means the execution may fall through without
  returning a meaningful result to the caller. (Use -noret to inhibit warning)
A.c:19:2: Fresh storage buffer not released before return
  A memory leak has been detected. Storage allocated locally is not released
  before the last reference to it is lost. (Use -mustfreefresh to inhibit
  warning)
   A.c:14:50: Fresh storage buffer created
A.c:10:10: Parameter argc not used
  A function parameter is not used in the body of the function. If the argument
  is needed for type compatibility or future plans, use /*@unused@*/ in the
  argument declaration. (Use -paramuse to inhibit warning)
A.c:2:7: Function exported but not used outside A: stringcopy
  A declaration is exported, but not used outside this module. Declaration can
  use static qualifier. (Use -exportlocal to inhibit warning)
   A.c:8:1: Definition of stringcopy

Finished checking --- 9 code warnings
```



```
biar@pfp-VirtualBox:~/Desktop/SSA/Lab1$ splint A.c +bounds
Splint 3.1.2 --- 03 May 2009

A.c: (in function stringcopy)
A.c:4:8: Test expression for while not boolean, type char: *str2
  Test expression type is not boolean. (Use -predboolothers to inhibit warning)
A.c:7:8: Implicitly temp storage str2 returned as implicitly only: str2
  Temp storage (associated with a formal parameter) is transferred to a
  non-temporary reference. The storage may be released or new aliases created.
  (Use -temptrans to inhibit warning)
A.c:4:8: Possible out-of-bounds read: *str2
    Unable to resolve constraint:
    requires maxRead(str2 @ A.c:4:9) >= 0
     needed to satisfy precondition:
    requires maxRead(str2 @ A.c:4:9) >= 0
  A memory read references memory beyond the allocated storage. (Use
  -boundsread to inhibit warning)
A.c:5:1: Possible out-of-bounds store: *str1++
    Unable to resolve constraint:
    requires maxSet(str1 @ A.c:5:2) >= 0
     needed to satisfy precondition:
    requires maxSet(str1 @ A.c:5:2) >= 0
  A memory write may write to an address beyond the allocated buffer. (Use
  -boundswrite to inhibit warning)
A.c: (in function main)
A.c:16:12: Possibly null storage buffer passed as non-null param:
              stringcopy (buffer, ...)
  A possibly null pointer is passed as a parameter corresponding to a formal
  parameter with no /*@null@*/ annotation.  If NULL may be used for this
  parameter, add a /*@null@*/ annotation to the function parameter declaration.
  (Use -nullpass to inhibit warning)
   A.c:14:16: Storage buffer may become null
A.c:16:12: Passed storage buffer not completely defined (*buffer is undefined):
              stringcopy (buffer, ...)
  Storage derivable from a parameter, return value or global is not defined.
  Use /*@out@*/ to denote passed or returned storage which need not be defined.
  (Use -compdef to inhibit warning)
   A.c:14:50: Storage *buffer allocated
A.c:16:1: Return value (type char *) ignored: stringcopy(buffe...
  Result returned by function call is not used. If this is intended, can cast
  result to (void) to eliminate message. (Use -retvalother to inhibit warning)
A.c:19:2: Path with no return in function declared to return int
  There is a path through a function declared to return a value on which there
  is no return statement. This means the execution may fall through without
  returning a meaningful result to the caller. (Use -noret to inhibit warning)
A.c:19:2: Fresh storage buffer not released before return
  A memory leak has been detected. Storage allocated locally is not released
  before the last reference to it is lost. (Use -mustfreefresh to inhibit
  warning)
   A.c:14:50: Fresh storage buffer created
A.c:16:20: Possible out-of-bounds read: argv[1]
    Unable to resolve constraint:
    requires maxRead(argv @ A.c:16:20) >= 1
     needed to satisfy precondition:
    requires maxRead(argv @ A.c:16:20) >= 1
A.c:10:10: Parameter argc not used
  A function parameter is not used in the body of the function. If the argument
  is needed for type compatibility or future plans, use /*@unused@*/ in the
  argument declaration. (Use -paramuse to inhibit warning)
A.c:2:7: Function exported but not used outside A: stringcopy
  A declaration is exported, but not used outside this module. Declaration can
  use static qualifier. (Use -exportlocal to inhibit warning)
   A.c:8:1: Definition of stringcopy

Finished checking --- 12 code warnings
```

Figure 1: Splint A.c without and with bounds warnings

The purpose of the function is to retrieve the string passed to the function through `argc`, copy the string into a buffer and then print the buffer to the user.

The warnings showed by Splint are 9 and some of them are real vulnerabilities, others are not so important. The warnings are:

- `Test expression for while not boolean`. This warning is not important, Splint does not recognize implicit test expressions as valid ones. To solve the warning, it is added an explicit test. The function checks is the analyzed character is a string terminator.
- `Implicitly temp storage str2 returned as implicitly only`. The auxiliary function return `str2` that was passed as parameter. Splint sees the variable as a temporal one and so informs the user that the storage may be released or new aliases created for str2. There are two possible solution:
  - mark the str2 variable as `/*@returned@*/`. Splint assumes the result of `stringcopy` is the same storage as its second parameter. No error is reported, since the only storage is then transferred through the return value.
  - since the returned value is not used in the main function, it is possible to change the return value to void and, in this way, it does not return anything. The logics of the function does not change, it still perform the copy.
- `Possibly null storage buffer passed as non-null param`. The parameter `buffer` passed to the `stringcopy` function can be NULL. In fact after the allocation, there is not a check to verify the correct execution of the `malloc` function. To solve the warning, it is check on the return value of the `malloc` function.
- `Passed storage buffer not completely defined`. The first parameter passed to to `stringcopy` is just allocated in main, so it must be marked as `/*@out@*/` to denote passed storage which need not be defined.
- `Return value (type char *) ignored`: The resolution of this warning depends on how the warning on str2 has been salved. If the function is declared as void, this warning it is solved implicitly. Otherwise, the return value of `stringcopy` is stored in a variable. The presence of this variable will cause another warning, the storage should be released before the return statement. Since the variable is another pointer to the memory area pointed by str2, that is `argv[1]`, there is no need to free the area. So, it can be considered a false positive.
- Path with no return. In the main function is missing the return statement. To solve the warning it is added `return 0` at the end of the main function.
- `Fresh storage buffer not released before return`. The warning refers to the variable `buffer` that is allocated using malloc, but it is not released before the return statement. For this, `free(buffer)` is added in the main function before the return.
- `Parameter argc not used`. The parameter argc is passed to the main function, but it is not used. The parameter contains the number of input is passed by the user, so it is useful to check if argv contains at least a value, that will be the one used by the function.
- `Function exported but not used outside`. Since the function is not used outside, it is declared as static. The access to static functions is restricted to the file where they are declared.

Using the flag `+bounds` the warnings are 12. The 3 additional warnings are:

- `Possible out-of-bounds read:  argv[1]`. The warning is salved adding the annotation: `/*@requires maxRead(argv) >= 1` $\wedge$ `maxRead(argv[1]) >= 0 @*/`. A requires clause specifies a predicate that must be true at a call site; when checking a function implementation Splint assumes the constraints given in its requires clauses are true at function entry.
- `Possible out-of-bounds read:  *str2` and `Possible out-of-bounds store:  *str1++`. The annotation used to salve the two warnings is `/*@requires maxRead(str2) >= 0` $\wedge$ `maxSet(str1) >= 0@*/`. So as to ensure that the two variables have at least a dimension greater or equal than 0.

File B

To the original file, a basic main function is added in order to compile and execute the file. In this way it is possible to understand better what the executable do and what are the vulnerabilities. The main function is only used to open an existing text file, defined to text the executable, and to call the original function, `func`, passing as parameter the file descriptor of the file just opened. After the call, the main function close the text file and return.

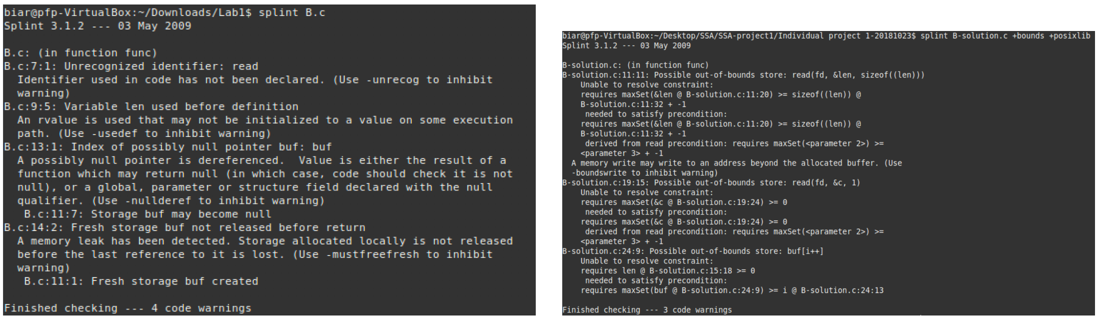


Figure 2: Splint B.c with and without flags

The purpose of the function `func` is to read on a file an integer, that will be used to define the size of a buffer, and read

File C

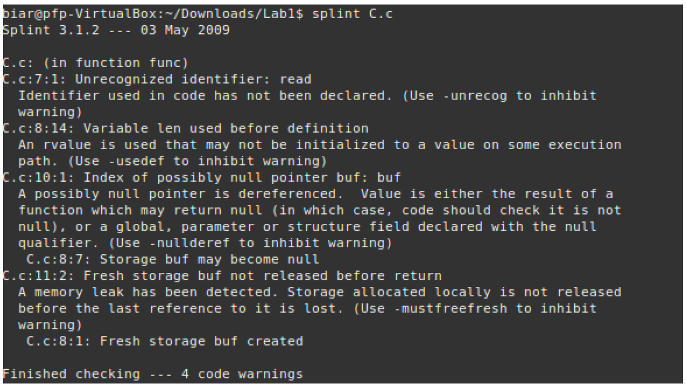


Figure 3: Splint B.c

FlawFinder

Flawfinder is a simple yet efficient ad quick tool that scans your C/C++ source code for calls to typical vulnerable library functions. It was developed by David Wheeler external link, a renowned security expert. It is run from the command line. Its output can easily be customized using specific command-line options. It is possible to call Flawfinder on a folder and it will analyze all the files present in it, but this analysis should be done only to have a general idea on the folder. The result of the analysis is a list of possible vulnerabilities divided by levels, the tool show first the riskiest ones. After the fist analysis, the user should call Flawfinder on each single file, possibly starting from the one with the riskier vulnerabilities.

Flawfinder has some pros and cons. Flawfinder works by doing simple lexical tokenization (skipping comments and correctly tokenizing strings), looking for token matches to the database, so the tool is base on a black list. Flawfinder then examines the text of the function parameters to estimate risk. Unlike tools such as splint, gcc's warning flags, and clang, flawfinder does not use or have access to information about control flow, data flow, or data types when searching for potential vulnerabilities or estimating the level of risk. Thus, flawfinder will necessarily produce many false positives for vulnerabilities and fail to report many vulnerabilities. On the other hand, flawfinder can find vulnerabilities in programs that cannot be built or cannot be linked. It can often work with programs that cannot even be compiled (at least by the reviewer's tools). Flawfinder also doesn't get as confused by macro definitions and other oddities that more sophisticated tools have trouble with. Flawfinder can also be useful as a simple introduction to static analysis tools in general, since it is easy to start using and easy to understand.

File A

```
biar@pfp-VirtualBox:~/Downloads/Lab1$ flawfinder A.c
Flawfinder version 1.27, (C) 2001-2004 David A. Wheeler.
Number of dangerous functions in C/C++ ruleset: 160
Examining A.c

No hits found.
Lines analyzed = 18 in 0.51 seconds (2683 lines/second)
Physical Source Lines of Code (SLOC) = 19
Hits@level = [0]   0 [1]   0 [2]   0 [3]   0 [4]   0 [5]   0
Hits@level+ = [0+]   0 [1+]   0 [2+]   0 [3+]   0 [4+]   0 [5+]   0
Hits/KSLOC@level+ = [0+]   0 [1+]   0 [2+]   0 [3+]   0 [4+]   0 [5+]   0
Minimum risk level = 1
There may be other security vulnerabilities; review your code!
```

```
biar@pfp-VirtualBox:~/Desktop/SSA/Lab1$ flawfinder --minlevel=0 A.c
Flawfinder version 1.27, (C) 2001-2004 David A. Wheeler.
Number of dangerous functions in C/C++ ruleset: 160
Examining A.c
A.c:18:  [0] (format) printf:
  If format strings can be influenced by an attacker, they can be
  exploited. Use a constant for the format specification. Constant format
  string, so not considered very risky (there's some residual risk, especially
  in a loop).

Hits = 1
Lines analyzed = 19 in 0.51 seconds (3536 lines/second)
Physical Source Lines of Code (SLOC) = 19
Hits@level = [0]   1 [1]   0 [2]   0 [3]   0 [4]   0 [5]   0
Hits@level+ = [0+]   1 [1+]   0 [2+]   0 [3+]   0 [4+]   0 [5+]   0
Hits/KSLOC@level+ = [0+] 52.6316 [1+]   0 [2+]   0 [3+]   0 [4+]   0 [5+]   0
Minimum risk level = 0
Not every hit is necessarily a security vulnerability.
There may be other security vulnerabilities; review your code!
```

Figure 4: FlawFinder A.c minlevel=1 and minlevel=0

There is only one hit that suggest to use a constant for the format specification, otherwise an attacker could influence the format string. This can be considered a false positive since or just a suggestion derived from the simple presence of `printf`, since it is already used a constant format.

```
biar@pfp-VirtualBox:~/Downloads/Lab1$ flawfinder B.c
Flawfinder version 1.27, (C) 2001-2004 David A. Wheeler.
Number of dangerous functions in C/C++ ruleset: 160
Examining B.c
B.c:7:  [1] (buffer) read:
  Check buffer boundaries if used in a loop.
B.c:12:  [1] (buffer) read:
  Check buffer boundaries if used in a loop.

Hits = 2
Lines analyzed = 14 in 0.51 seconds (1940 lines/second)
Physical Source Lines of Code (SLOC) = 14
Hits@level = [0]   0 [1]   2 [2]   0 [3]   0 [4]   0 [5]   0
Hits@level+ = [0+]   2 [1+]   2 [2+]   0 [3+]   0 [4+]   0 [5+]   0
Hits/KSLOC@level+ = [0+] 142.857 [1+] 142.857 [2+]   0 [3+]   0 [4+]   0 [5
+]   0
Minimum risk level = 1
Not every hit is necessarily a security vulnerability.
There may be other security vulnerabilities; review your code!
```

Figure 5: FlawFinder B.c

```
biar@pfp-VirtualBox:~/Downloads/Lab1$ flawfinder C.c
Flawfinder version 1.27, (C) 2001-2004 David A. Wheeler.
Number of dangerous functions in C/C++ ruleset: 160
Examining C.c
C.c:7:  [1] (buffer) read:
  Check buffer boundaries if used in a loop.
C.c:9:  [1] (buffer) read:
  Check buffer boundaries if used in a loop.

Hits = 2
Lines analyzed = 11 in 0.51 seconds (1699 lines/second)
Physical Source Lines of Code (SLOC) = 11
Hits@level = [0]   0 [1]   2 [2]   0 [3]   0 [4]   0 [5]   0
Hits@level+ = [0+]   2 [1+]   2 [2+]   0 [3+]   0 [4+]   0 [5+]   0
Hits/KSLOC@level+ = [0+] 181.818 [1+] 181.818 [2+]   0 [3+]   0 [4+]   0 [5
+]   0
Minimum risk level = 1
Not every hit is necessarily a security vulnerability.
There may be other security vulnerabilities; review your code!
```

Figure 6: FlawFinder B.c