Program Assertions in Security (with SPLINT)

- C. A. R. Hoare, Turing award winner
- Edsger Dijkstra , Turing award winner
- David Evans, David Larochelle, SPLINT group
  - CS Dept, U of Virginia
  - http://www.splint.org/

1

SPLINT – Static Checking

- Splint is a tool for statically checking C programs for programming "mistakes"
- Splint does all of the traditional checks including unused declarations, type cast errors, execution path without returns
- It also detects errors pertaining to type mismatch, inconsistent memory management, null dereference

2

SPLINT Design Goals

- To be used by typical programmers as part of the development process
  - Fast, Easy to Use
- can be used to check legacy code
  - Handles typical C programs
- Encourage a proactive security methodology
  - Document key assumptions

3

SPLINT

- Lightweight static analysis tool
- Simple dataflow analyses
- Unsound and Incomplete
- not many of thousands of users adding annotations to code: gradual learning curve
- Detects inconsistencies between code and specifications
- Examples: memory management (leaks, dead references), null dereferences, information hiding, undocumented modifications, etc.

4

Approach

- Programmers add "annotations"
  - Simple and precise.
  - Describe programmers intent:
    - Types, memory management, data hiding, aliasing, modification, nullity, buffer sizes, security, etc.
- SPLINT detects inconsistencies between annotations and code.
  - Fast dataflow analyses.

5

SPLINT approach

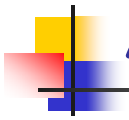
- Document assumptions about buffer sizes
  - Semantic comments
  - Provide annotated standard library
  - Allow user's to annotate their code
- Find inconsistencies between code and assumptions
- Make compromises to get useful checking
  - Use simplifying assumptions to improve efficiency
  - Use heuristics to analyze common loop idioms
  - Accept some false positives and false negatives (unsound and incomplete analysis)

6

Splint Annotations

- Annotations are stylized comments that document the assumptions made about function formal parameters, global variables, memory references etc
- Splint can perform powerful checks based on user-specified annotations
- Splint annotations are represented as: /*@…. @*/

7



Annotations

- requires, ensures
- maxSet
  - highest index that can be safely written to
- maxRead
  - highest index that can be safely read
- char buffer[100];
  - ensures maxSet(buffer) == 99

8

SPLINT Annotation Example

```
char *strncat (char *d, char *s, size_t n)

/*@
     requires
       maxSet(d) >= maxRead(s) + n
@*/
```

9

SPLINT Annotation Example

```
char *strcpy (char *s1, const char *s2)

/*@  requires maxSet(s1) >= maxRead(s2) @*/
/*@  ensures maxRead(s1) == maxRead(s2)
      /\ result == s1    @*/;
```

10

SPLINT: Buffer Overflow Example

```
void func(char *str) {
   char buffer[256];
   strncat(buffer, str, sizeof(buffer)- 1);
}
```

```
strncat.c:4:21: Possible out-of-bounds store:
      strncat(buffer, str, sizeof((buffer)) - 1);
  Unable to resolve constraint:
    requires maxRead (buffer @ strncat.c:4:29) <= 0
  needed to satisfy precondition:
    requires maxSet (buffer @ strncat.c:4:29)
      >= maxRead (buffer @ strncat.c:4:29) + 255
   ...
```

11

Warning Reported

```
strncat.c:4:21: Possible out-of-bounds store:
      strncat(buffer, str, sizeof((buffer)) - 1);
  Unable to resolve constraint:
    requires maxRead (buffer @strncat.c:4:29)<= 0
  needed to satisfy precondition:
    requires maxSet (buffer @ strncat.c:4:29)
      >= maxRead (buffer @strncat.c:4:29) + 255
  derived from strncat precondition:
    requires maxSet (<parameter 1>)
           >=  maxRead (<parameter1>) + <parameter 3>
```

12

SPLINT generates preconditions

- `strcpy(ls_short,entry->arg[0]);`
- strcpy (s1, s2)
  - requires maxSet(s1) >= maxRead(s2)
- substituting the actual parameters:

```
maxSet(ls_short @ ftpd.c:1112:14) >=
   maxRead(entry->arg[0] @ ftpd.c:1112:23)
```

13

Overview of SPLINT checking

- Intraprocedural
  - But use annotations on called procedures and global variables to check calls, entry, exit points
- Expressions generate constraints
  - C semantics, annotations
- Axiomatic semantics propagates constraints
- Simplifying rules
  - e.g. maxRead(str+i) ==> maxRead(str) - i
- Produce warnings for unresolved constraints

14

SPLINT constraints

1. `t++;`
2. `*t = 'x';`
3. `t++;`

- leads (after simplifications) to the constraints:
  1. `requires maxSet(t @ 1:1) >= 1,`
  2. `ensures maxRead(t @ 3:4) >= -1`
  3. `ensures (t @ 3:4) = (t @ 1:1) + 2.`

15

Checking

- Simple dataflow analysis
- Intraprocedural – except uses annotations to alter state around procedure calls
- Integrates with other LCLint analyses (e.g., nullness, aliases, ownership, etc.)
- SPLINT checks
  - type abstractions, modifications
  - globals, memory leaks, dead storage,
  - naming conventions,
  - undefined behavior, incomplete definition…

16

Error Detection

- Errors detected by splint includes:
  - Type mismatch
  - Memory leaks
  - Null dereference
  - Use of un-initialized formal parameters, returning undefined storage
  - Undocumented use of global variables
  - Empty if statements, missing breaks, unreachable code

17

Type Mismatch

- Common occurrence of type mismatch:
  - Assignment Statements
  - Boolean expressions
  - Mathematical expressions
  - printf() statements
  - Array index

18

Loop Heuristics

- Recognize common loop idioms
- Use heuristics to guess number of iterations
- Analyze first and last iterations
- Example:
  - for (init; *buf; buf++)
  - Assume maxRead(buf) iterations
  - Model first and last iterations

19

Case studies

- wu-ftpd 2.5 and BIND 8.2.2p7
  - Detected known buffer overflows
  - Unknown buffer overflows exploitable with write access to config files
- Performance
  - wu-ftpd: 7 seconds/ 20,000 lines of code
  - BIND: 33 seconds / 40,000 lines
  - Athlon 1200 MHz

20

Results

	Instances in wu-ftpd (grep)	LCLint warnings with no annotations added	LCLint warning with annotations
strcat	27	19	12
strcpy	97	40	21
strncpy	55	4	4
Other Warnings	-	132 writes 220 reads	95 writes 166 reads

21

SPLINT analysis of wu-ftp-2.5.0

```
ftpd.c:1112:2:Possible out-of-bounds store.  Unable to
   resolve constraint:
      maxRead ((entry->arg[0] @ ftpd.c:1112:23))
       <= (1023)
   needed to satisfy precondition:
      requires maxSet ((ls_short @ ftpd.c:1112:14))
                  >= maxRead ((entry->arg[0] @
                          ftpd.c:1112:23))
   derived from strcpy precondition:
      requires
              maxSet (<param 1>) >= maxRead (<param 2>)
```

22

wu-ftpd vulnerablity

```
int acl_getlimit(char *class, char *msgpathbuf)
{
  struct aclmember *entry = NULL;

  while (getaclentry("limit", &entry)) {
      …
      strcpy(msgpathbuf, entry->arg[3]);

  }
}
```


Type Mismatch Example

```
#include <stdlib.h>

int main(){
  long typelong = 65555;
  unsigned short typeshort;

  typeshort = typelong;              // line  7
  if(typelong)                      // line  8
    printf("the value %d %ld \n", typeshort, typelong);
  return 1;
}
```

- Output: the value 19 65555

Type Mismatch Example *(contd.)*

```
Splint 3.0.1.6 --- 11 Feb 2002

type.c: (in function main)
type.c:7:3: Assignment of long int to unsigned short int: typeshort =
    typelong
  To ignore signs in type comparisons use +ignoresigns
type.c:8:6: Test expression for if not boolean, type long int: typelong
  Test expression type is not boolean or int. (Use -predboolint to inhibit
  warning)
type.c:9:35: Format argument 1 to printf (%d) expects int gets unsigned
    short
               int: typeshort
  type.c:9:24: Corresponding format code

Finished checking --- 3 code warnings
```

25

I/O Streams Challenge

- Many properties can be described in terms of state attributes
  - A file is *open* or *closed*
    - fopen: returns an *open* file
    - fclose: *open* → *closed*
    - fgets, etc. require open files
  - Reading/writing – must reset between certain operations

26

Defining Openness

```
attribute openness
  context reference FILE *
  oneof closed, open
  annotations
     open ==> open  closed ==> closed
  transfers
    open as closed ==> error
    closed as open ==> error
  merge open + closed ==> error
  losereference
    open ==> error "file not closed"
  defaults
    reference ==> open
end
```

Object cannot be open on one path, closed on another

Cannot abandon FILE in open state

27

Specifying I/O Functions

```
/*@ open @*/ FILE *fopen
    (const char *filename,
     const char *mode);

int fclose (/*@ open @*/ FILE *stream)
  /*@ensures closed stream@*/ ;

char *fgets (char *s, int n,
             /*@ open @*/ FILE *stream);
```

28

IO Stream Results on …

- wu-ftpd 2.6.1 (20K lines, ~4 seconds)
- No annotations: 7 warnings
- After adding ensures clause for ftpd_pclose
  - 4 spurious warnings
    - 1 used function pointer to close FILE
    - 1 reference table
    - 2 convoluted logic involving function static variables
  - 2 real bugs (failure to close ftpservers file on two paths)

29

SPLINT Implementation

- Extended LCLint
  - Open source checking tool [FSE '94] [PLDI '96]
  - Uses annotations
  - Detects null dereferences, memory leaks, etc.
- Integrated to take advantage of existing checking and annotations (e.g., modifies)
- Added new annotations and checking for buffer sizes

30

SPLINT performance

- Can check >100K line programs
- checks about 1K lines per second
- Detects real bugs in real programs
  - including itself, of course
  - Wu-ftpd
  - Several buffer overflow vulnerabilities

31

SPLINT SUMMARY

- Detecting Buffer Overflows: Annotations express constraints on buffer sizes
  - e.g., maxSet is the highest index that can safely be written to
- Checking uses axiomatic semantics with simplification rules
- Heuristics for analyzing common loop idioms
- Detected known and unknown vulnerabilities in wu-ftpd and BIND

32