Extensible Lightweight Static Checking

On the I/O Streams Challenge Problem

David Evans
evans@cs.virginia.edu
http://lclint.cs.virginia.edu

LCLint

University of Virginia
Computer Science

Everyone Likes Types

- Easy to Understand
- Easy to Use
- Quickly Detect Many Programming Errors
- Useful Documentation
- **…even though they are lots of work!**
  - 1/4 of text of typical C program is for types

Limitations of Standard Types

Type of reference never changes

Language defines checking rules

One type per reference

Limitations of Standard Types / Attributes

Limitations of Standard Types	Attributes
Type of reference never changes	State changes along program paths
Language defines checking rules	Programmer defines checking rules
One type per reference	Many attributes per reference

Similar to Vault, linear types, typestates, etc.

LCLint

- Lightweight static analysis tool [FSE'94, PLDI'96] – "quick and dirty"
- Simple dataflow analyses
- Unsound and Incomplete
- Several thousand users…perhaps ¼ adding annotations to code: gradual learning curve
- Detects inconsistencies between code and ~~specifications~~ ~~annotations~~ documented assumptions
- Examples: memory management (leaks, dead references), null dereferences, information hiding, undocumented modifications, etc.

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* $\rightarrow$ *closed*
    - fgets, etc. require open files
  - Reading/writing – must reset between certain operations

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

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);
```

Reading, ‘Riting, ‘Rithmetic

```
attribute rwness
  context reference FILE *
  oneof rwnone, rwread, rwwrite, rweither
  annotations
    read ==> rwread   write ==> rwwrite
    rweither ==> rweither    rwnone ==> rwnone
  merge
    rwread + rwwrite ==> rwnone   rwnone + * ==> rwnone
    rweither + rwread ==> rwread  rweither + rwwrite ==> rwwrite
  transfers
    rwread as rwwrite ==> error "Must reset file between read and write."
    rwwrite as rwread ==> error "Must reset file between write and read."
    rwnone as rwread ==> error "File in unreadable state."
    rwnone as rwwrite ==> error "File in unwritable state."
    rweither as rwwrite ==> rwwrite     rweither as rwread ==> rwread
  defaults
    reference ==> rweither
end
```

Reading, ‘Righting

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

int fgetc (/*@read@*/ FILE *f) ;
int fputc (int, /*@write@*/ FILE *f) ;

/* fseek resets the rw state of a stream */
int fseek (/*@rweither@*/ FILE *stream,
           long int offset, int whence)
  /*@ensures rweither stream@*/ ;
```

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.)

Example

```
FILE *f = fopen (fname, “rw”);
int i = fgetc (f);
if (i != EOF) {
  fputc (i, f);
  fclose (f);
}
```

f:openness = open
f:rwness = rweither

Possibly null reference f passed where non-null expected

f:openness = open, f:rwness = rwread

Attribute mismatch – passed read where write FILE * expected.

f:openness = closed, f:rwness = rwnone

Branches join in incompatible states: f is closed on true branch,open on false branch

Results

- On my code…works great
  - Checked LCLint sources (178K lines, takes 240 seconds on Athlon 1.2GHz)
  - No annotations: 2 errors
  - Added 1 ensures clause

    static void loadrc (FILE *p_rcfile, cstringSList *)
        /*@ensures closed p_rcfile@*/ ;
  - No more warnings

Results – "Real" Code

- 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)

Taintedness

```
attribute taintedness
  context reference char *
  oneof untainted, tainted
  annotations
    tainted reference ==> tainted
    untainted reference ==> untainted
    anytainted parameter ==> tainted
  transfers
    tainted as untainted ==> error
  merge
    tainted + untainted ==> tainted
  defaults
    reference ==> tainted
    literal ==> untainted
    null ==> untainted
end
```

tainted.xh

```
int fprintf (FILE *stream, /*@untainted@*/ char
   *format, ...) ;

/*@tainted@*/ char *fgets (char *s, int n, FILE *)
  /*@ensures tainted s@*/ ;

char *strcpy (/*@returned@*/ /*@anytainted@*/ char *s1,
              /*@anytainted@*/ char *s2)
  /*@ensures s1:taintedness = s2:taintedness@*/ ;

char *strcat (/*@returned@*/ /*@anytainted@*/ char *s1,
              /*@anytainted@*/ char *s2)
  /*@ensures s1:taintedness
             = s1:taintedness | s2:taintedness@*/ ;
```

Buffer Overflows

- Most commonly exploited security vulnerability
  - 1988 Internet Worm
  - Still the most common attack
    - Code Red exploited buffer overflow in IIS
    - >50% of CERT advisories, 23% of CVE entries in 2001
- Finite-state attributes not good enough
  - Need to know about lengths of allocated buffers

Detecting Buffer Overflows

- More expressive annotations
  - 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
- Paper (with David Larochelle) in USENIX Security 2001

Will Programmers Add Annotations?

- C in 1974: char *strcpy ();
- C in 1978: char *strcpy (char *s1, char *s2);
- C in 1989: char *strcpy (char *s1, const char *s2);
- C in 1999: char *strcpy (char * restrict s1,
  const char * restrict s2);
- C in 20??:

```
nullterminated char *strcpy
   (returned char *restrict s1,
    nullterminated const char *restrict s2)
     requires maxSet(s1) >= maxRead (s2)
     ensures s1:taintedness = s2:taintedness
     ensures maxRead(s1) = maxRead (s2);
```