Abstract Interpretation + Impure Catalysts
Our Sparrow Experience

YI Jhee, MS Jin, YB Jung, DH Kim, SH Kong, HJ Lee, HJ Oh, DJ Park,
Kwangkeun Yi
Programming Research Laboratory
Seoul National University
Korea

30 Years of Abstract Interpretation, 01/09/2008 @ San Francisco

What We've Been Doing

Developing the SPARROW system

- an effort to *commercialize* static bug-finders
- *shallow property, full automation, scalable*
    - buffer overrun, memory leak, null dereference, uninitialized access, divide by zero, etc.
- for *non domain-specific* C code

Motivation

- prove by ourselves that static analysis is "useful in real world"
- curious about "extra miles" from academia to industry

What We've Been Doing

Developing the SPARROW system

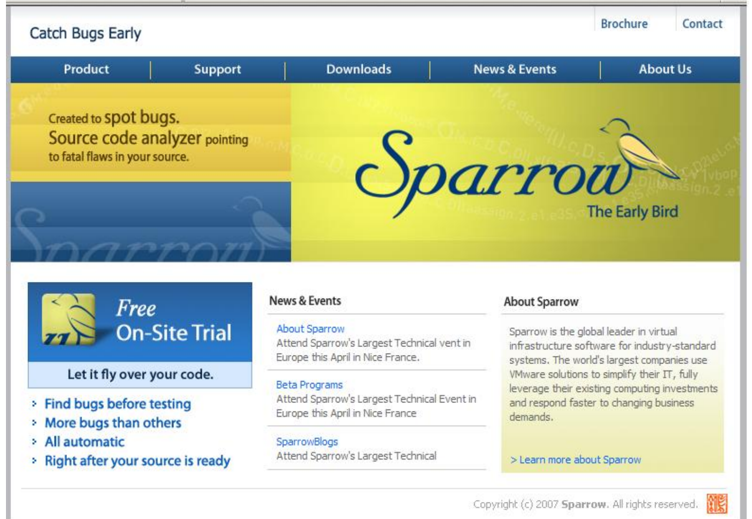
- an effort to *commercialize* static bug-finders
- *shallow property, full automation, scalable*
  - buffer overrun, memory leak, null dereference, uninitialized access, divide by zero, etc.
- for *non domain-specific* C code

Motivation

- prove by ourselves that static analysis is "useful in real world"
- curious about "extra miles" from academia to industry

Of course, the reality

- has been challenging us a lot, and
- we've been struggling to respond to.


Catch Bugs Early
Brochure
Contact
Product
Support
Downloads
News & Events
About Us
Created to spot bugs.
Source code analyzer pointing
to fatal flaws in your source.
Sparrow
The Early Bird
Free
On-Site Trial
Let it fly over your code.
Find bugs before testing
More bugs than others
All automatic
Right after your source is ready
News & Events
About Sparrow
Attend Sparrow's Largest Technical vent in
Europe this April in Nice France.
Beta Programs
Attend Sparrow's Largest Technical Event in
Europe this April in Nice France
SparrowBlogs
Attend Sparrow's Largest Technical
About Sparrow
Sparrow is the global leader in virtual
infrastructure software for industry-standard
systems. The world's largest companies use
VMware solutions to simplify their IT, fully
leverage their existing computing investments
and respond faster to changing business
demands.
> Learn more about Sparrow
Copyright (c) 2007 Sparrow. All rights reserved.

Performance Numbers (1/3)

Memory leak detection (SPEC2000 and open sources) (as of 01/04/2008)

Programs	Size KLOC	Time (sec)	True Alarms	False Alarms
art	1.2	0.68	1	0
equake	1.5	1.03	0	0
mcf	1.9	2.77	0	0
bzip2	4.6	1.52	1	0
gzip	7.7	1.56	1	4
parser	10.9	15.93	0	0
ammp	13.2	9.68	20	0
vpr	16.9	7.85	0	9
crafty	19.4	84.32	0	0
twolf	19.7	68.80	5	0
mesa	50.2	43.15	9	0
vortex	52.6	34.79	0	1
gap	59.4	31.03	0	0
gcc	205.8	1330.33	44	1
gnuchess-5.07	17.8	9.44	4	0
tcl8.4.14	17.9	266.09	4	4
hanterm-3.1.6	25.6	13.66	0	0
sed-4.0.8	26.8	13.68	29	31
tar-1.13	28.3	13.88	5	3
grep-2.5.1a	31.5	22.19	2	3
openssh-3.5p1	36.7	10.75	18	4
bison-2.3	48.4	48.60	4	1
openssh-4.3p2	77.3	177.31	1	7
fftw-3.1.2	184.0	15.20	0	0
httpd-2.2.2	316.4	102.72	6	1
net-snmp-5.4	358.0	201.49	40	20
binutils-2.13.1	909.4	712.09	228	25

Performance Numbers (2/3)

In comparison with other published memory leak detectors

- Number of bugs: SPARROW finds consistently more bugs than others
- Analysis speed: 785LOC/sec, next to the fastest FastCheck.
- False-alarm ratio: 21%
- Efficacy (TrueAlarms/KLOC $\times$ 1/FalseAlarmRatio): biggest

Tool	C size KLOC	Speed LOC/s	True Alarms	False Alarm Ratio(%)		Efficacy
Saturn '05 (Stanford)	6,822	50	455	10%		1/150
Clouseau '03 (Stanford)	1,086	500	409	64%		1/170
FastCheck '07 (Cornell)	671	37,900	63	14%		1/149
Contradiction '06 (Cornell)	321	300	26	56%		1/691
SPARROW	2,543	785	433	21%		1/123

Table: Overall comparison

C program	Tool	True Alarms	False Alarm Count
SPEC2000 benchmark	SPARROW	81	15
	FastCheck '07 (Cornell)	59	8
binutils-2.13.1 & openssh-3.5.p1	SPARROW	246	29
	Saturn '05 (Stanford)	165	5
	Clouseau '03 (Stanford)	84	269

Table: Comparison for the same C programs

Performance Numbers (3/3)

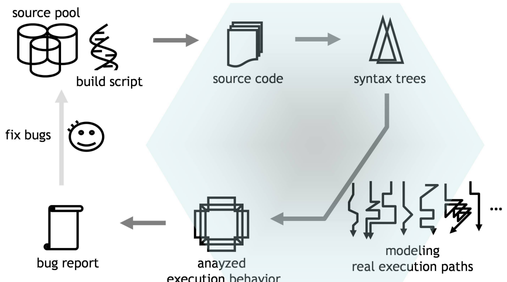
Buffer overrun detection (SPEC2000 and open sources) (as of 01/04/2008)

Programs	Size KLOC	Time (sec)	True Alarms	False Alarms
art	1.2	0.45	0	0
equake	1.5	2.89	0	1
mcf	1.9	0.33	0	0
bzip2	4.6	10.90	23	29
gzip	7.7	3.38	18	24
parser	10.9	260.94	4	13
twolf	19.7	8.59	0	0
ammp	13.2	10.20	6	0
vpr	16.9	11.15	0	3
crafty	19.4	139.80	1	5
mesa	50.2	47.88	2	10
vortex	52.6	40.12	2	0
gap	59.4	28.48	0	2
gzip-1.2.4	9.1	8.55	0	17
gnuchess-5.07	17.8	179.58	1	8
tcl8.4.14/unix	17.9	585.99	1	14
hanterm-3.1.6	25.6	52.25	34	1
sed-4.0.8	26.8	49.34	2	11
tar-1.13	28.3	57.98	1	10
grep-2.5.1a	31.5	47.26	0	1
bison-2.3	48.4	281.84	0	18
openssh-4.3p2	77.3	97.69	0	9
fftw-3.1.2	184.0	102.17	9	4
httpd-2.2.2	316.4	265.43	10	33
net-snmp-5.4	358.0	899.73	3	36

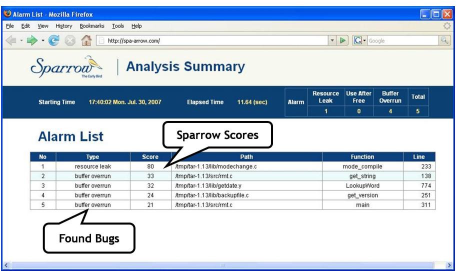
Steps of SPARROW

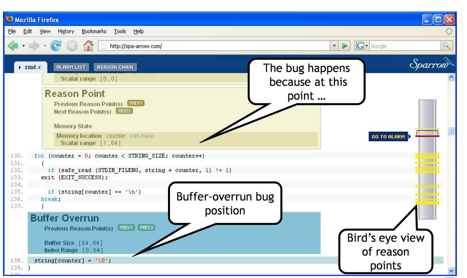
SPARROW is a one-button solution with four steps:

- understanding the code genetics
- parsing and distilling the code
- analyzing the code's run time behaviors
- reporting detected bugs



User Interface: Scored Alarms + Navigating Explanation





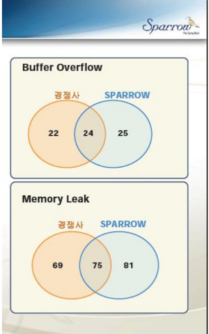
Customers under negotiation

Domestic market at the moment

- Samsung, LG, etc.: personal devices' sw developers
- network switching system sw developers
- other embedded sw developers
- bank system sw developers
- etc.

Complementing others (such as Coverity, GrammaTech, Klockworks, Polyspace).

- BMT at a site (a network device OS, $\sim$ 700KLOC):



Outline

1. SPARROW's Examples
2. Our Approach
3. A Wish

SPARROW's Examples

Note

- Some bugs may look simple (*after a posteriori slicing*), but
  - only few bug paths
    - among the exponential jungle of paths
  - must beat all the paths
    - no prior knowledge possible
  - such prior knowledge? very rough, or a catch-22 situation

Note

- Some bugs may look simple (*after a posteriori slicing*), but
  - only few bug paths
    - among the exponential jungle of paths
  - must beat all the paths
    - no prior knowledge possible
  - such prior knowledge? very rough, or a catch-22 situation
- Pattern-based approach?
  - not tolerant to variations of “patterns”
  - variations should be ample in real code
  - a collection of patterns will always fall short

SPARROW-detected Overrun Errors (1/3)

SPARROW-detected Overrun Errors (1/3)

- in `Linux Kernel 2.6.4`

```
625       for (minor = 0; minor < 32 && acm_table[minor]; minor++);
...       ...
713       acm_table[minor] = acm;
```

Sparrow-detected Overrun Errors (1/3)

- in `Linux Kernel 2.6.4`

```
625       for (minor = 0; minor < 32 && acm_table[minor]; minor++);
...       ...
713       acm_table[minor] = acm;
```

- in a proprietary code

```
if (length >= NET_MAX_LEN)
   return API_SET_ERR_NET_INVALID_LENGTH;
...
buff[length] |= (num << 4);
```

Sparrow-detected Overrun Errors (1/3)

- in `Linux Kernel 2.6.4`

```
625       for (minor = 0; minor < 32 && acm_table[minor]; minor++);
...       ...
713       acm_table[minor] = acm;
```

- in a proprietary code

```
if (length >= NET_MAX_LEN)
   return API_SET_ERR_NET_INVALID_LENGTH;
...
buff[length] |= (num << 4);
```

- in a proprietary code

```
index = memmgr_get_bucket_index(block_size);
...
mem_stats.pool_ptr[index] = prt
```

SPARROW-detected Overrun Errors (1/3)

- in `Linux Kernel 2.6.4`

```
625        for (minor = 0; minor < 32 && acm_table[minor]; minor++);
...        ...
713        acm_table[minor] = acm;
```

- in a proprietary code

```
if (length >= NET_MAX_LEN)
   return API_SET_ERR_NET_INVALID_LENGTH;
...
buff[length] |= (num << 4);
```

- in a proprietary code

```
   index = memmgr_get_bucket_index(block_size);
   ...
   mem_stats.pool_ptr[index] = prt
```

- in a proprietary code

```
imi_send_to_daemon(PM_EAP, CONFIG_MODE, set_str, sizeof(set_str));
...
imi_send_to_daemon(int module, int mode, char *cmd, int len)
{
...
        strncpy(cmd, reply.str, len);
        cmd[len] = 0;
```

SPARROW-detected Leak Errors (2/3)

SPARROW-detected Leak Errors (2/3)

- in sed-4.0.8/regexp_internal.c

```
948:   new_nexts = re_realloc (dfa->nexts, int, dfa->nodes_alloc);
949:   new_indices = re_realloc (dfa->org_indices, int, dfa->nodes_alloc);
950:   new_edests = re_realloc (dfa->edests, re_node_set, dfa->nodes_alloc);
951:   new_eclosures = re_realloc (dfa->eclosures, re_node_set,
952:       dfa->nodes_alloc);
953:   new_inveclosures = re_realloc (dfa->inveclosures, re_node_set,
954:  dfa->nodes_alloc);
955:   if (BE (new_nexts == NULL || new_indices == NULL
956:   || new_edests == NULL || new_eclosures == NULL
957:   || new_inveclosures == NULL, 0))
958:     return -1;
```

Sparrow-detected Leak Errors (2/3)

- in sed-4.0.8/regexp_internal.c

```
948:   new_nexts = re_realloc (dfa->nexts, int, dfa->nodes_alloc);
949:   new_indices = re_realloc (dfa->org_indices, int, dfa->nodes_alloc);
950:   new_edests = re_realloc (dfa->edests, re_node_set, dfa->nodes_alloc);
951:   new_eclosures = re_realloc (dfa->eclosures, re_node_set,
952:       dfa->nodes_alloc);
953:   new_inveclosures = re_realloc (dfa->inveclosures, re_node_set,
954:  dfa->nodes_alloc);
955:   if (BE (new_nexts == NULL || new_indices == NULL
956:   || new_edests == NULL || new_eclosures == NULL
957:   || new_inveclosures == NULL, 0))
958:     return -1;
```

- in proprietary code

```
line = read_config_read_data(ASN_INTEGER, line,
                             &StorageTmp->traceRouteProbeHistoryHAddrType,
                             &tmpint);
...
line = read_config_read_data(ASN_OCTET_STR, line,
                             &StorageTmp->traceRouteProbeHistoryHAddr,
                             &StorageTmp->traceRouteProbeHistoryHAddrLen);
...
if (StorageTmp->traceRouteProbeHistoryHAddr == NULL) {
     config_perror
         (``invalid specification for traceRouteProbeHistoryHAddr'');
     return SNMPERR_GENERR;
}
```

Sparrow-detected Leak Errors (3/3)

- in mesa/osmesa.c(in SPEC 2000)

```
276:   osmesa->gl_ctx = gl_create_context( osmesa->gl_visual );
...
287:     gl_destroy_context( osmesa->gl_ctx );
------------------
1164: GLcontext *gl_create_context( GLvisual *visual,
                                  GLcontext *share_list,
                                  void *driver_ctx )
...
1183:  ctx = (GLcontext *) calloc( 1, sizeof(GLc
...
1211:     ctx->Shared = alloc_shared_state();
---------------------
476: static struct gl_shared_state *alloc_shared
477: {
...
489: ss->Default1D = gl_alloc_texture_object(ss,
490: ss->Default2D = gl_alloc_texture_object(ss,
491: ss->Default3D = gl_alloc_texture_object(ss, 0, 3);
----------------------
1257: void gl_destroy_context( GLcontext *ctx )
1258: {
      ...
1274: free_shared_state( ctx, ctx->Shared );
```

Our Approach

Pure Soup + Impure Catalysts

- pure soup: simple & “sound” abstract interpreter
- impure catalysts: unorthodox & unsound techniques to refine the bug-finding performance

Pure Soup + Impure Catalysts

- pure soup: simple & “sound” abstract interpreter
- impure catalysts: unorthodox & unsound techniques to refine the bug-finding performance

Rationale:

$$(\hat{A} + \hat{B})^3 \quad \text{versus} \quad \hat{A}^3 + \beta^{\theta(\hat{A})}$$

(cf. $(x + y)^3 = x^3 + y^3 + 3x^2y + 3xy^2$)

- sound analysis components: $\hat{A}$ and $\hat{B}$
- their costly composition: $(\hat{A} + \hat{B})^3$
- economical by impure catalysts: $\hat{A}^3 + \beta^{\theta(\hat{A})}$
  - pragmatic approach to find prevalent true cases
  - lose rare true cases, reduce many false alarms

Unsoundness: Necessary Evil

- no complete source, C's flat/linear memory, unknown libraries, dialect extensions, embedded assembly code
- naive soundness $\Rightarrow$ too many alarms of little relevance
- accurate soundness $\Leftarrow$
  - global analysis (impossible), or
  - sound separate analyser and linker (unknown), or
  - domain-dependency (limited code)

The Pure, Underpinning Engine $\hat{A}^3$

A "sound" abstract interpreter

- non-relational, state transition, program-point fixpoint analysis
  - with the interval domain for $2^{\mathbb{Z}}$
  - with the lexical abstractions (malloc, access expr) for locations
- lots of engineering: worklist order, economical widening points, partial join, enlarged program point, context pruning, state localization, inlining, and etc.

The Pure, Underpinning Engine $\hat{A}^3$

A "sound" abstract interpreter

- non-relational, state transition, program-point fixpoint analysis
  - with the interval domain for $2^{\mathbb{Z}}$
  - with the lexical abstractions (malloc, access expr) for locations
- lots of engineering: worklist order, economical widening points, partial join, enlarged program point, context pruning, state localization, inlining, and etc.

What about
relational analysis, context sensitivity, path sensitivity?

The Impure Catalysts $\beta^{\theta(\hat{A})}$

To reduce false alarms and to find more bugs,

The Impure Catalysts $\beta^{\theta(\hat{A})}$

To reduce false alarms and to find more bugs,

- no blind collection & abstraction at program point
  - loop unrolling, if constantly bounded
  - loop unrolling, bounded by $\theta(\hat{A})$: unsound

$$\langle \hat{a}^i, b_j \rangle \sqsubseteq \langle \hat{a}^{i+1}, b_{j+1} \rangle \quad \text{whenever} \quad \hat{a}^i \sqsubseteq \hat{a}^{i+1}$$

  - loop unrolling, always up to k: unsound

The Impure Catalysts $\beta^{\theta(\hat{A})}$

To reduce false alarms and to find more bugs,

- no blind collection & abstraction at program point
  - loop unrolling, if constantly bounded
  - loop unrolling, bounded by $\theta(\hat{A})$: unsound

$$\langle \hat{a}^i, b_j \rangle \sqsubseteq \langle \hat{a}^{i+1}, b_{j+1} \rangle \quad \text{whenever} \quad \hat{a}^i \sqsubseteq \hat{a}^{i+1}$$

  - loop unrolling, always up to k: unsound
- path sensitivity for effect-paths only: unsound
  e.g. paths with malloc/free effects dominates (within procedure boundary)

The Impure Catalysts $\beta^{\theta(\hat{A})}$

To reduce false alarms and to find more bugs,

- no blind collection & abstraction at program point
  - loop unrolling, if constantly bounded
  - loop unrolling, bounded by $\theta(\hat{A})$: unsound

$$\langle \hat{a}^i, b_j \rangle \sqsubseteq \langle \hat{a}^{i+1}, b_{j+1} \rangle \quad \text{whenever} \quad \hat{a}^i \sqsubseteq \hat{a}^{i+1}$$

  - loop unrolling, always up to k: unsound
- path sensitivity for effect-paths only: unsound
  e.g. paths with malloc/free effects dominates (within procedure boundary)
- context sensitivity by parameterized procedural summarization: ai per procedure then from post-state
  e.g. $\lambda\langle \alpha_1, \alpha_2 \rangle.\langle \text{malloc}(\alpha_1\text{->y}), \text{free}(\alpha_2\text{->x}) \rangle$

A Wish

Theory about Unsoundness?

Unsound things are necessary evils in reality.

- Are they just implementation issues, independent of the theory?
- Can there be a theoretical framework to reason about the degree of unsoundness?
- Can there be a systematic way to lessen the unsoundness?

Theory about Unsoundness?

Unsound things are necessary evils in reality.

- Are they just implementation issues, independent of the theory?
- Can there be a theoretical framework to reason about the degree of unsoundness?
- Can there be a systematic way to lessen the unsoundness?

Maybe, Prof. Cousots have already had an answer 30 years ago.

Theory about Unsoundness?

Unsound things are necessary evils in reality.

- Are they just implementation issues, independent of the theory?
- Can there be a theoretical framework to reason about the degree of unsoundness?
- Can there be a systematic way to lessen the unsoundness?

Maybe, Prof. Cousots have already had an answer 30 years ago.

Thank you.