# Our SPARROW Experience with Abstract Interpretation + Impure Catalysts

Kwangkeun Yi

ropas.snu.ac.kr/~kwang
Research on Software Analysis for Error-free Computing Center
School of Computer Science and Engineering
Seoul National University

10/01/2010 @ SATE 2010, NIST, U.S.A.

# 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

From the elusive three (deep property, scalability, automation)

– deep property

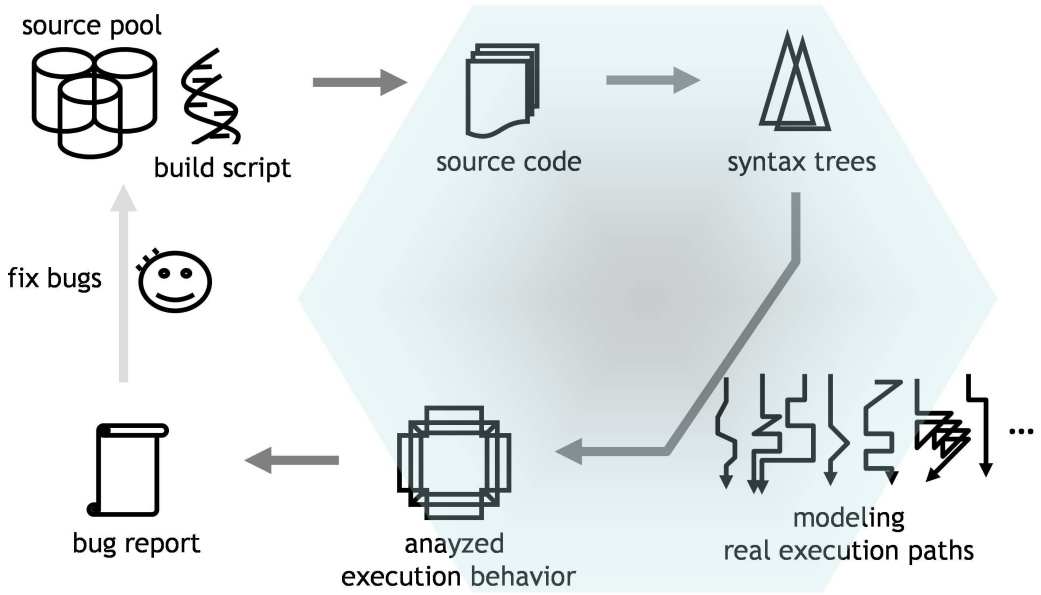
\+ domain-independence

# Outline

1. Introducing SPARROW: performance and technology
2. Discussing SPARROWresult for SATE 2010 benchmarks
3. Concluding with a thesis/view

# Steps of Sparrow

Sparrow is a one-button solution with four steps:



# Performance Numbers (1/3)

Memory leak detection (SPEC2000 and open sources)

| 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)

| 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 |

# 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

```
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)

- 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;
```

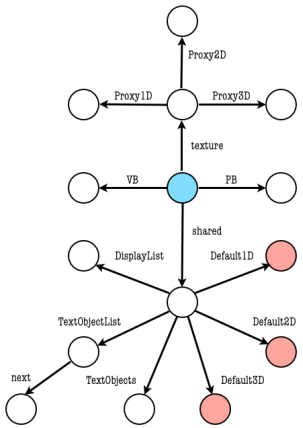
- in proprietary code

```
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 );
```



# SparrowApproach: Pure Soup + Impure Catalysts

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

# 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

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

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 a finite component: unsound
  - loop unrolling, 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. $\lambda\langle\alpha_1, \alpha_2\rangle.\langle \mathsf{malloc}(\alpha_1\texttt{->y}), \mathsf{free}(\alpha_2\texttt{->x})\rangle$

# SPARROW Result for SATE'10 Benchmarks

For buffer-overrun errors

| pgm | loc | time | alarms | true | false |
|---|---|---|---|---|---|
| dovecot-2.0 | 222K | 4m | 6 | 0 | 6 |
| wireshark-1.2.0 | 2,271K | 23h 31m | 57 | 27 | 30 |

# dovecot-2.0 alarms

All 6 alarms are false alarms

- 4 false alarms

```
unsigned int x;
... arr[x % 4]
```

  - src/lib-impag/impa-utf7.c: L139, L123

- 2 false alarms

```
dst = malloc(T);
src = malloc(T);
... memcpy(dst+off, src, T-off);
```

  - src/lib/hmac-md5.c: L30, L31

# wireshark-1.2.0 alarms (1/4)

27 true alarms and 30 false alarms

- 2 true alarms

```
for (rd=0, wr=0; wr < outbuf_size; rd++, wr++) {
  ...
  switch(c) {
    ...
     out[++wr] = c;      // overrun
     break;
    ...
  }
}
```

  - print.c: L927, epan/dissectors/packet-ncp-sss.c: L539

- 1 true alarm

```
if (csub <= 8) iax2_info->callState = tap_cmd_voip_state[cssub];
```

  - epan/dissectors/packet-iax2.c: L1597

# wireshark-1.2.0 alarms (2/4)

27 true alarms and 30 false alarms

- 2 true alarms

```
hidden_item = proto_tree_add_item(...); // can return -1
...
... = buf[hidden_item];
```

  - epan/dissectors/packet-ospf.c: L1293, L2666

- 18 true alarms

```
int mpa_layers[] = {-1, ...};
...
return(mpa->padding ? mpa_padding_data[mpa_layers[mpa->layer]] : 0);
// mpa->layer is a parameter, hence can be -1
```

  - wsutil/mpeg-audio.c: L77-L95

# wireshark-1.2.0 alarms (3/4)

27 true alarms and 30 false alarms

- 3 true alarms

```
// key is a parameter
... memcpy(k_ipad, key, key_len); // overrun: need to check the key variable
... memcpy(k_opad, key, key_len); // overrun: need to check the key variable
```

  - epan/crypt/crypt-md5.c: L340, L341, epan/dissectors/packet-evrc.c: L188

- 1 true alarm

```
static void lshift(char *d, int count, int n) {
   char out[64];
   int i;
   for (i=0; i<n; i++)
      out[i] = d[(i+count)%n]; // overrun, i and count is int type
   ...
}
```

  - epan/crypt/crypt-des.c: L169

# wireshark-1.2.0 alarms (4/4)

27 true alarms and 30 false alarms

- 21 false alarms

```
for (inh = inr->list; inh; inh=inh->next) {...}
```

  index boundary depends on list structure/string content

- 4 false alarms

```
if (octet == 0x0f) return;
... = dgt->out[octet & 0x0f] // Sparrow fails to prune
```

- 5 false alarms

```
unsigned int x;
... arr[x % 4] // Sparrow fails to recognize unsigned int
```

# Our Position: Towards Domain-Specific SPARROW's

If we specialize SPARROWfor a particular SW, we can achieve

- zero-false-alarm
- automatic
- scalable analyzer

Why useful?

- specializations for programming idoms
- programming idioms are preserved across versions
- for important/long-lasting SW's

Thank you

QnA