Raccoon:
버퍼 오버런 취약점 분석기

김유일 <youil.kim@arcs>
전진성 <jinseong.jeon@arcs>
한환수 <hwansoo.han@arcs>

Motivation 1

- 버퍼 오버런 (Buffer Overrun)
  - 소프트웨어 보안 취약점의 대표적인 원인
  - 예1: Morris 웜 (1988)
  - 예2: Code Red 바이러스 (2001)
- 프로그램 정적 분석 기법의 가능성
  - 버퍼 오버런 취약점 정적 분석기

Motivation 2

- 사용자의 관점에서,
  기존의 프로그램 분석기는
  - 충분히 정확하지 않거나,
  - 충분히 효율적이지 못하다.

BOON

Splint

Coverity

CSSV

Airac

최종 목표: 정확하고 빠른 분석기

An Example

```
125: static void
126: get_string (char *string)
127: {
128:   int counter;
129:
130:   for (counter = 0; counter < STRING_SIZE; counter++)
131:     {
132:       if (safe_read(STDIN_FILENO, string+counter, 1) != 1)
133:         exit (EXIT_SUCCESS);
134:
135:       if (string[counter] == '\n')
136:         break;
137:     }
138:   string[counter] = '\0';
139: }
```

Line 125~139 of rmt.c from GNU tar 1.13

What We Need?

Integer Value Analysis

+

Pointer Analysis

+

Buffer Analysis

What We Need?

Integer Value Analysis

- Constant Propagation [Kildall73]
- Intervals [CC76]
- Zones [Mine01]
- Octagons [Mine01]
- Polyhedra [CH78]

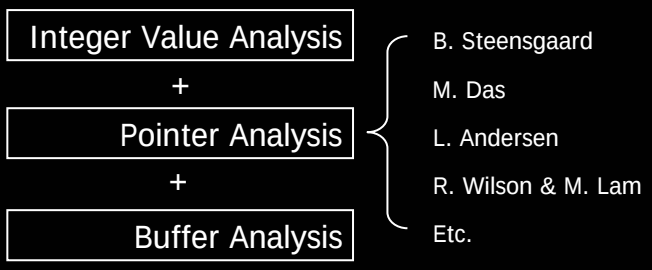
+

Pointer Analysis

+

Buffer Analysis

What We Need?



Observation 1

```
void f(void)
{
  int x[100], i=0, j=0;
  while (i < 99) {
    i++; j++;
  }
  x[i] = 0;
  x[j] = 0;
}
```

FALSE ALARM

Relational analysis
may be useful in some cases.

Our Claim:
Interval analysis
is usually enough.

Observation 2

```
void g(void)
{
  int x[100], i=0, j=100;
  int *p;
  p = &i;
  x[*p] = 0;
  p = &j;
  x[*p] = 0;
}
```

FALSE ALARM

Flow-sensitive pointer analysis may be useful in some cases.

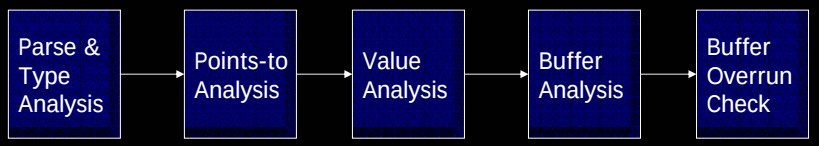
Our Claim:
Flow-insensitive pointer analysis is usually enough.

기본 방침: 도끼로 연필 깎지 말자.

Other Factors

- How to handle aggregates?
  - Structures
  - Arrays
- How to handle complicated features?
  - Type castings
  - **setjmp** and **longjmp**
  - Signal handlers
  - Inline assembly code, etc.

Our Approach



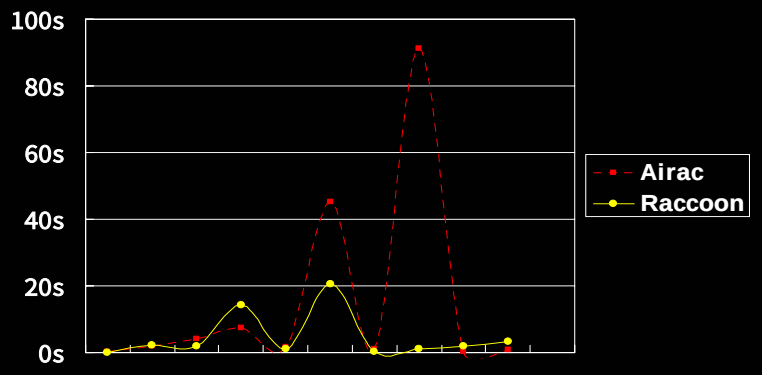
- 정확하고 빠른 분석기
- 분석의 정확도를 스스로 조절
- 분석의 정확도를 조절할 수 있는 구조

Our Position

- A Prototype Analyzer
  - 각 분석의 정확도를 조절할 수 있는 구조
- Preliminary Experiments
  - Parts of Linux kernel 2.6.4
  - 3.0 GHz Intel Pentium 4 with 2.5 GB RAM

Experimental Results: Overview

- Linux Kernel 2.6.4



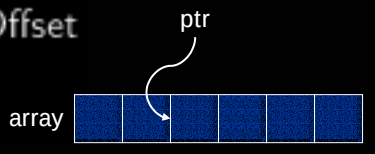
Design

- Value Analysis

$$\mathsf{Mem}_v : \mathsf{AbsLoc} \mapsto \mathsf{Interval}$$

- Buffer Analysis

$$\begin{aligned}
\mathsf{Mem}_p &: \mathsf{AbsLoc} \mapsto \mathsf{Buffers} \\
\mathsf{Buffers} &: 2^{\mathsf{Buffer}} + \{\top\} \\
\mathsf{Buffer} &: \mathsf{Id} \times \mathsf{Size} \times \mathsf{Offset} \\
\mathsf{Size} &= \mathsf{Interval} \\
\mathsf{Offset} &= \mathsf{Interval}
\end{aligned}$$



Implementation

- OCaml + CIL 1.3.3
- Points-to Analysis
  - One level flow algorithm [Das00]
- Interval Analysis
  - Flow sensitive
  - Context sensitive (customizable)
  - Widening and narrowing operations
  - Backward abstract interpretation

Limitation

- Impreciseness
  - Combine all fields in a structure
  - Treat return values from library functions in a conservative way (as a top)
- Unsoundness
  - Ignore insides of library functions

Experimental Results: Details

Filename	#Lines	Airac Time	Raccoon Time	Airac Alarms	Raccoon Alarms	Real Bugs
vmax301.c	245	0.28s	0.05s	1	1	1
xfrm_user.c	1201	45.07s	20.58s	2	2	1
usb-midi.c	2206	91.32s	1.22s	10	4	2
atkbd.c	811	1.99s	2.16s	2	2	2
af_inet.c	1273	1.17s	0.27s	1	2	1
eata_pio.c	984	7.50s	14.24s	3	7	1
cdc-acm.c	849	3.98s	2.03s	3	2	2
lp6_output.c	1110	1.53s	1.21s	0	0	0
mptbase.c	6158	0.79s	3.29s	1	1	1
aty128fb.c	2466	0.32s	1.88s	1	1	1

Case Study: **eata_pio.c**

```
$ raccoon –start_function eata_pio_detect eata_pio.cil.c
...
eata_pio.c:725: reg_IRQL[gc->IRQ]
Warning: expression 'gc->IRQ' may cause overrun
Size : [16,16]
Index: (top)

eata_pio.c:730: reg_IRQ[gc->IRQ]
Warning: expression 'gc->IRQ' may cause overrun
Size : [16,16]
Index: (top)

eata_pio.c:945: reg_IRQ[i]
Warning: expression 'i' may cause overrun
Size : [16,16]
Index: [0,16]
```

Case Study: **af_inet.c**

```
$ raccoon –start_function inet_register_protosw af_inet.cil.c
af_inet.c:994: inetsw[p->type]
Warning: expression 'p->type' may cause overrun
Size : [11,11]
Index: (top)

af_inet.c:995: inetsw[p->type]
Warning: expression 'p->type' may cause overrun
Size : [11,11]
Index: (top)
```


Case Study: **af_inet.c**

```
980: void inet_register_protosw(struct inet_protosw *p)
981: {
982:         struct list_head *lh;
983:         struct inet_protosw *answer;
984:         int protocol = p->protocol;
985:         struct list_head *last_perm;
986:
987:         spin_lock_bh(&inetsw_lock);
988:
989:         if (p->type > SOCK_MAX)
990:                 goto out_illegal;
991:
992:
993:         answer = NULL;
994:         last_perm = &inetsw[p->type];
995:         last_for_each(lh, &inetsw[p->type]) {
```

Our Plan

- 최적화
- 보다 세밀한 조절이 가능하도록 개선
  - Separating fields for some structures
  - Weakly Relational Domains [Mine00]
  - Trace Partitioning [Mauborgne05]
- 새로운 아이디어 적용
  - 경고 메시지의 원인을 분석하고,
  - 분석의 정확도를 스스로 조절하는 분석기