A First Step Towards Automated Detection of Buffer Overrun Vulnerabilities

Wagner, Foster, Brewer, Aiken

UC Berkeley

NDSS ’00

CS 711 29 Sep 2005

Motivation

- C is unsafe
  - Buffer overruns a major security problem
  - up to 50% of CERT-reported vulnerabilities (up to 1999)
  - yadda yadda yadda
- Want: automatic static detection of overruns
  - Dynamic testing doesn't test all the cases
  - Static testing provides assurance before deployment
  - Automatic to deal with large legacy code bases

Design Philosophy

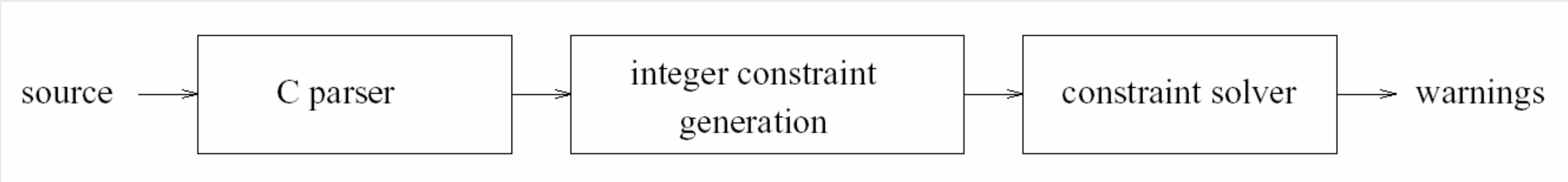
- Practical
  - ⇒Scalable
    - ⇒ Flow insensitive
    - ⇒ Context insensitive
      - ⇒ Imprecise
  - ⇒ Useful
    - ⇒ Few false positives/false negatives
      - ⇒ Precise
- Trade off between precision and scalability
  - Have they found the sweet spot?

Design

- Treat C strings as abstract data type
  - Gloss over pointer arithmetic, layout in memory, …

- Model buffers as pairs of integer ranges
  - For each string variable track:
    - allocated size of buffer
    - length (number of bytes in use)
  - Reduces overrun problem to tracking integer ranges
    - Buffer overrun if max length of v > allocated size of v

- Architecture



Constraint Language

- $\mathbb{Z}^{\infty} = \mathbb{Z} \cup \{-\infty, \infty\}$
- Range: $[m,n] = \{ i \in \mathbb{Z}^{\infty} : m \leq i \leq n \}$
  - $S+T = \{ s + t : s \in S, t \in T\}$
  - $S-T = \{ s - t : s \in S, t \in T\}$
  - $S \times T = \{ s \times t : s \in S, t \in T\}$
  - $\min(S,T) = \{ \min(s, t) : s \in S, t \in T\}$
  - $\max(S,T) = \{ \min(s, t) : s \in S, t \in T\}$

- Range closure of S = [inf S, sup S]
  - Take range closures for all operations
- e.g.

$$[2,2] \times [1,4] = [2,8]$$
$$\min([1,4], [3,6]) = [1,4]$$

Constraint Language

- Integer range expression

  $$
  \begin{aligned}
  e ::= {} & v \in \mathit{Vars} \\
  & \mid\ n \in \mathbb{Z} \\
  & \mid\ n \times v \ \mid\ e + e \ \mid\ e - e \\
  & \mid\ \max(e, \ldots, e) \ \mid\ \min(e, \ldots, e)
  \end{aligned}
  $$

- Integer range constraint

  $e \subseteq v$

- Assignment α: Vars $\rightarrow \mathbb{Z}^{\infty}$
  - "satisfies constraints" with obvious definition

Constraint Generation

- Parse source code, and then…
- For each integer program variable **`v`**
  - Have range variable v
- For each string variable **`s`**
  - Have two variables: alloc(s) and len(s)
  - Note: len(s) includes the '\0' terminator
- For each function **`f(a1, …, an)`**
  - Have a variable for each formal param
  - Have a variable for return value, f_return
  - Note: functions monomorphic (context insensitive)
- For each statement
  - Generate a constraint…

Constraint Generation for Statements

- Integer expressions and integer variables modeled by appropriate range operations

- **`v = e`** produces constraint $e \subseteq v$
  - e.g. **`i = i + j`** produces $i + j \subseteq i$
- Model string library by pattern matching:

`char s[n];`	$n \subseteq \text{alloc}(s)$
`s = "foo"`	$\{4\} \subseteq \text{alloc}(s)$ $\{4\} \subseteq \text{len}(s)$
`strcpy(src, dst)`	$\text{len}(src) \subseteq \text{len}(dst)$
`strcat(s, sfx)`	$\text{len}(s)+\text{len}(sfx)-1 \subseteq \text{len}(s)$
`p[n] = '\0'`	$\min(\text{len}(p), n+1) \subseteq \text{len}(p)$
…	…

Constraint System

- Now have a constraint system
- Solve it [2 slides away]
  - get a satisfying assignment α
- For each string variable s
  - α(len(s)) = [a, b]
  - α(alloc(s)) = [c, d]
  - if b ≤ c then the buffer never overruns
  - if a > d then buffer always overruns!
  - if [a,b] and [c,d] overlap then there may be an overrun

Imprecision from Pointers

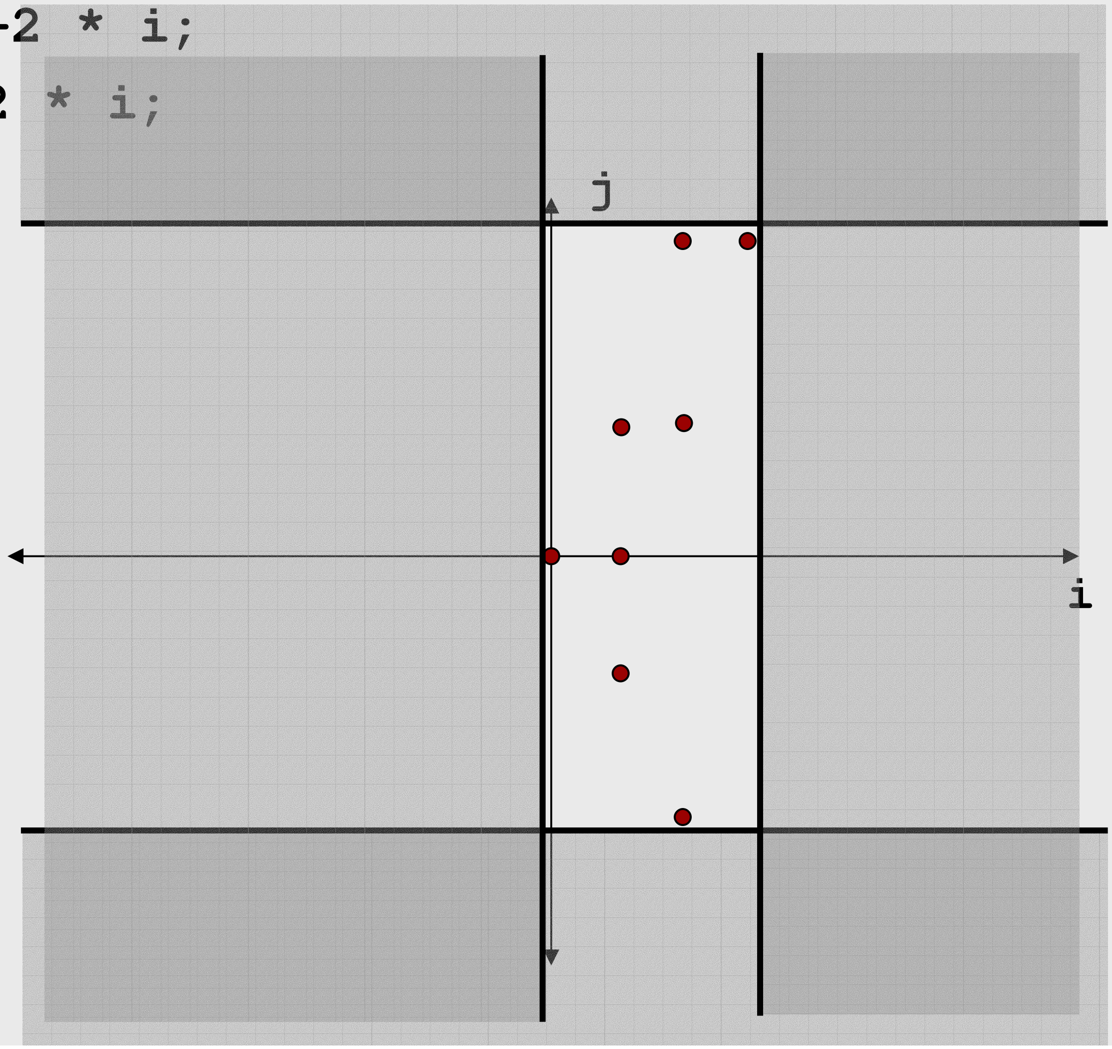
- Ideally, should have soundness:
  - α(v) ⊇ {values that v may take during execution}
  - Don't, due to aliasing, double indirect pointers, structs, unions, …

`char s[20], *p, t[10]`	20 ⊆ alloc(s) 10 ⊆ alloc(t)
`strcpy(s, "Hello");`	6 ⊆ len(s)
`p = s + 5;`	alloc(s)-5 ⊆ alloc(p)
	len(s)-5 ⊆ len(p)
`strcpy(p, " world!");`	8 ⊆ len(p)
`strcpy(t, s)`	len(s) ⊆ len(t)

- All structures assumed to be potentially aliased, only one variable for each field of structure

Solving Range Constraints: Bounding Box

```
int i=0, j=0;
for (i = 1; i < 3; i++) {
    j = -2 * i;
    j = 2 * i;
}
```



Solving Integer Range Constraints

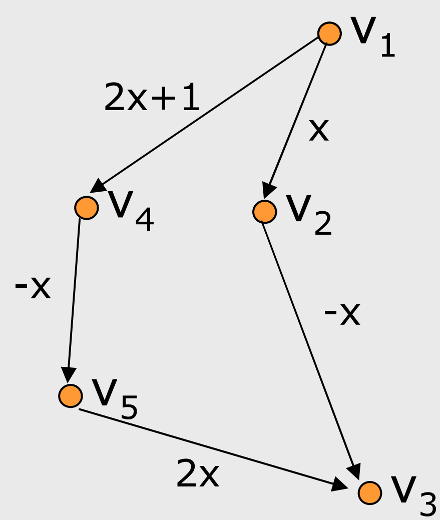
- Assume all constraints of the form

$$n \subseteq v_i \qquad \text{or} \qquad f(v_i) \subseteq v_j$$

for affine functions f

$5 \subseteq v_1$

$7 \subseteq v_4$



$\alpha(v_1) = [5,5]$

$\alpha(v_2) = [5,5]$

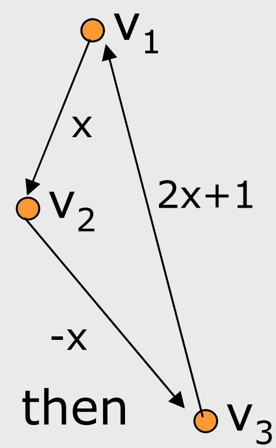
$\alpha(v_3) = [-22,-5]$

$\alpha(v_4) = [7,11]$

$\alpha(v_5) = [-11,-7]$

Solving Integer Range Constraints

- What about cycles?
  - Can handle precisely without infinite ascending chains
- Cycle $f = f_n \circ \ldots \circ f_1$
  - composition of affine functions will be affine function
  - e.g. $f(x) = -2x + 1$



- Compare $f(\alpha(v))$ to $\alpha(v)$
  - if $f(\alpha(v)) \subseteq \alpha(v)$ then least solution is $\alpha(v)$
  - if $\sup(f(\alpha(v))) > \sup(\alpha(v))$ and $\inf(f(\alpha(v))) < \inf(\alpha(v))$ then least solution is $[-\infty, \infty]$
  - if $\sup(f(\alpha(v))) > \sup(\alpha(v))$ then set $\alpha(v)$ to $[\inf(\alpha(v)), \infty]$ and try again
  - if $\inf(f(\alpha(v))) < \inf(\alpha(v))$ then set $\alpha(v)$ to $[-\infty, \sup(\alpha(v))]$ and try again
- e.g. $f(x) = -2x + 1$, $f([0,5]) = [-9, 1]$
  - Least solution is $[-\infty, 5]$

Experiments

- Linux Net Tools
  - 3.5 kloc
  - Previously hand audited in 1996
  - Tool found new buffer overrun bugs (probably exploitable)
- Sendmail 8.9.3
  - 32 kloc
  - Previously hand audited
  - Found some minor bugs (probably not exploitable), including complex off-by-one error
- Sendmail 8.7.5
  - 32 kloc
  - Prior to Sendmail hand audit, to test false negatives

Limitations/comparison

- Large number of false positives
  - Requires human to check them
  - e.g. sendmail 8.9.3, of 44 warnings, 4 were bugs
  - Reduce with improved analysis?

Improved analysis	False alarms that could be eliminated
flow-sensitive	$19/40 \approx 48\%$
flow-sens. with pointer analysis	$25/40 \approx 63\%$
flow- and context-sens., with linear invariants	$28/40 \approx 70\%$
flow- and context-sens., with pointer analysis and inv.	$38/40 \approx 95\%$

  - What's the alternative?
    - 695 call sites to potentially unsafe string functions, all to be checked by hand…

Discussion

- How to improve soundness while maintaining scalability?
  - Add context sensitivity, pointer analysis
    - Ganapathy, Jha, Chandler, Melski, Vitek "Buffer Overrun Detection using Linear Programming and Static Anaylsis" (CCS03)
  - Add limited forms of flow sensitivity
    - [GJCMV 03] suggest SSA form for some flow-sensitivity
    - Different constraints vars for different lexical scopes?
      - e.g. **`int x; … while (x < 10) { … }; …`**
        $x_{while} \subseteq [-\infty, 9]$ $\quad\quad x_{while} \subseteq x$
  - Other forms of solutions than ranges? Linear relations?
  - Other ways?

Discussion II of II

- Approach to false negatives interesting…
  - How else to measure false negatives?
- Advantages/disadvantages of constraint-based approach?
- Usefulness
  - What does it take to get an analysis used?
  - Downloadable as an extension to eclipse?