Static Analysis for Buffer Overrun Detection

Vinod Ganapathy, Somesh Jha
University of Wisconsin-Madison

Joint work with
David Chandler, David Melski, David Vitek
Grammatech Inc.

The Problem

- Buffer Overflows:
  - Highly exploited class of vulnerabilities
    - CERT database ~40% of advisories
    - C programs are highly vulnerable
- Goal:
  - Build a tool that detects buffer overflows

Our Solution

- Static Analysis:
  - Analyze code and prevent bugs before deployment
- As opposed to:
  - Runtime attack prevention
    - Buffer Overflow can be transformed to DoS
    - Performance Penalty

Milestones

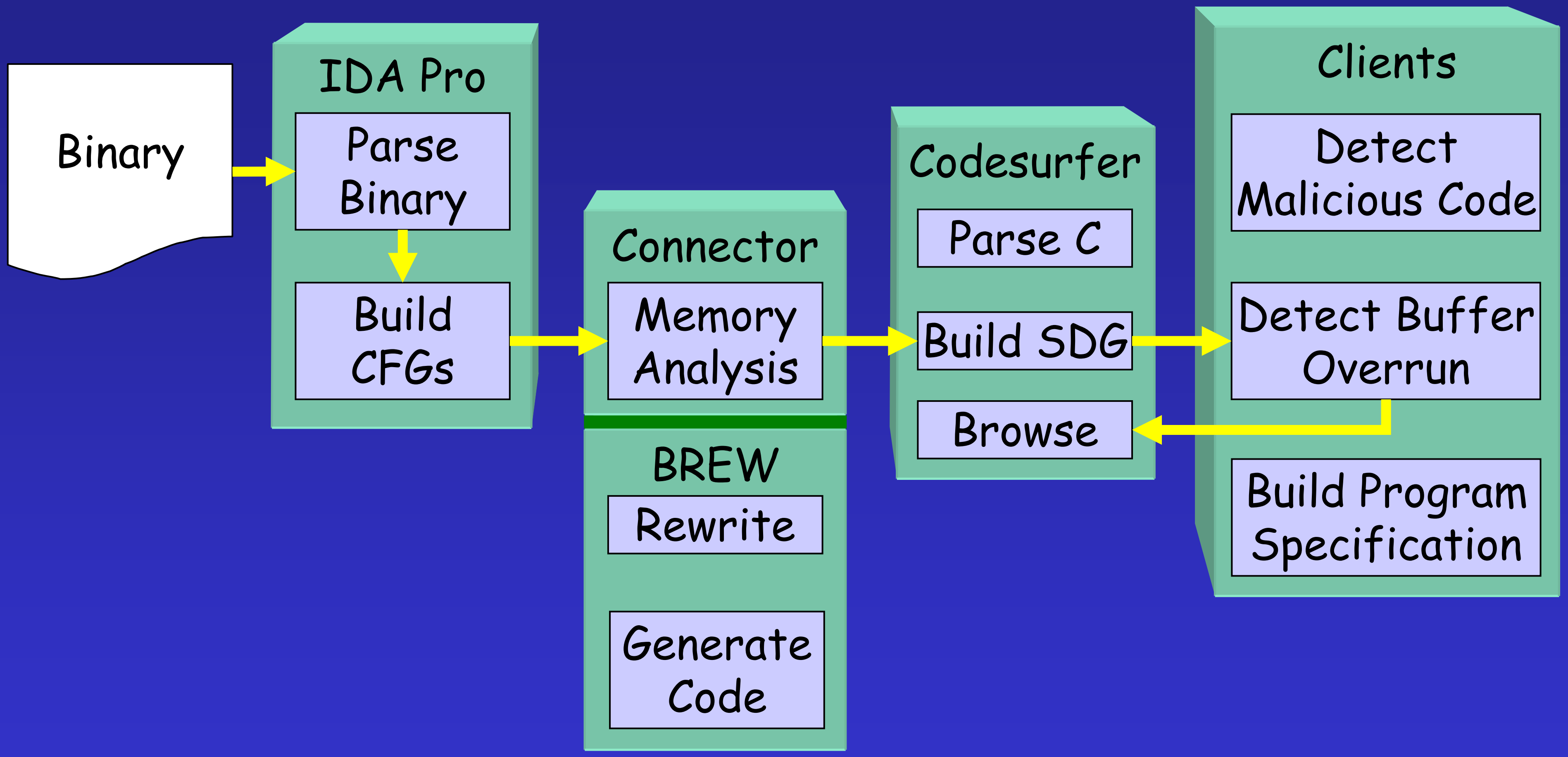
- ☑ Built tool to identify overruns in C
  - Made analysis Context Sensitive

    (respect call-return semantics)
- ➔ Flow sensitive analysis (respect program order)
- ➔ Buffer Overrun tool for object code

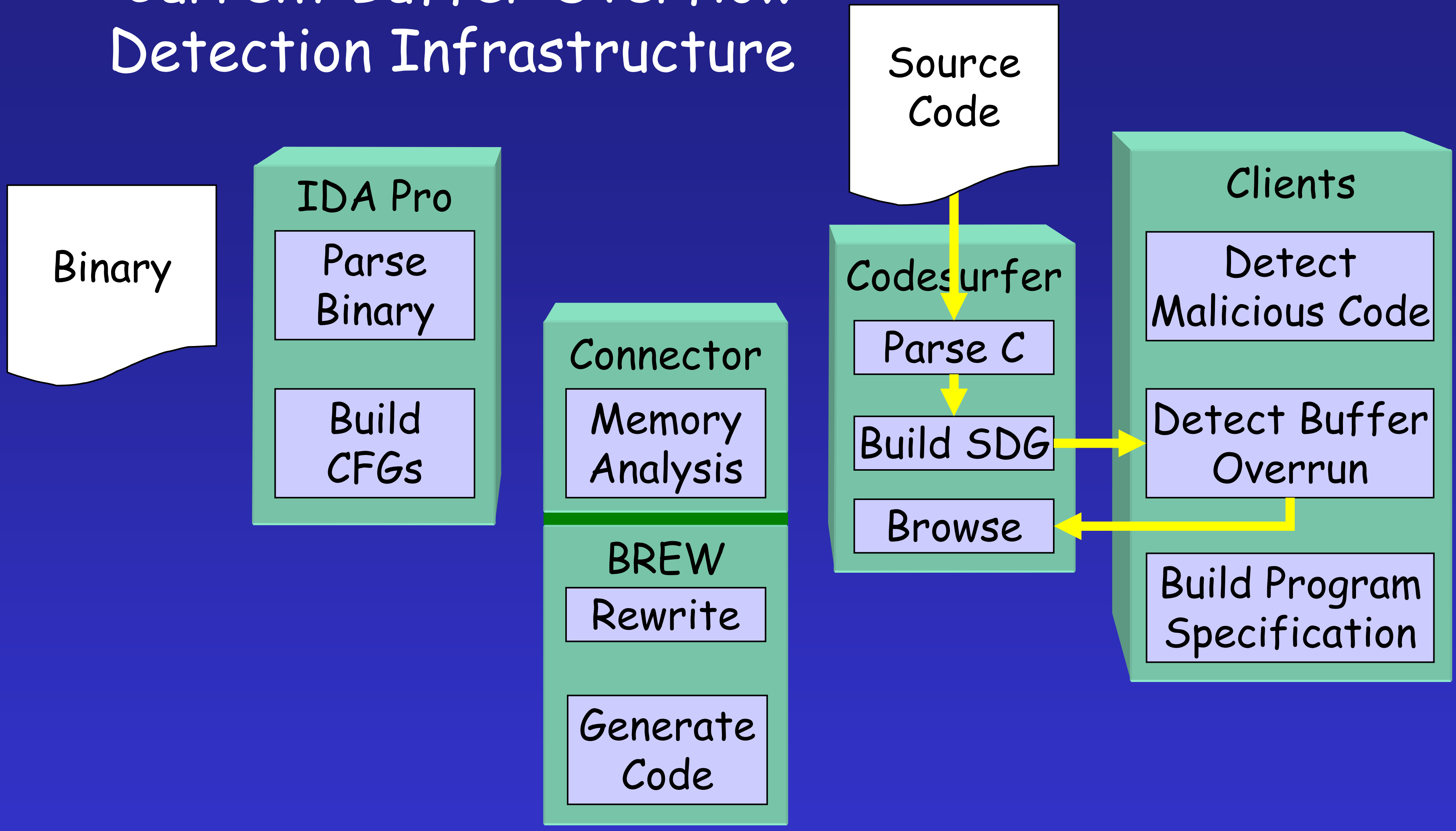
Overview of Talk

- Tool Architecture
- Constraint Generation & Resolution
- Adding Context Sensitivity
- Results
- Current efforts and future work

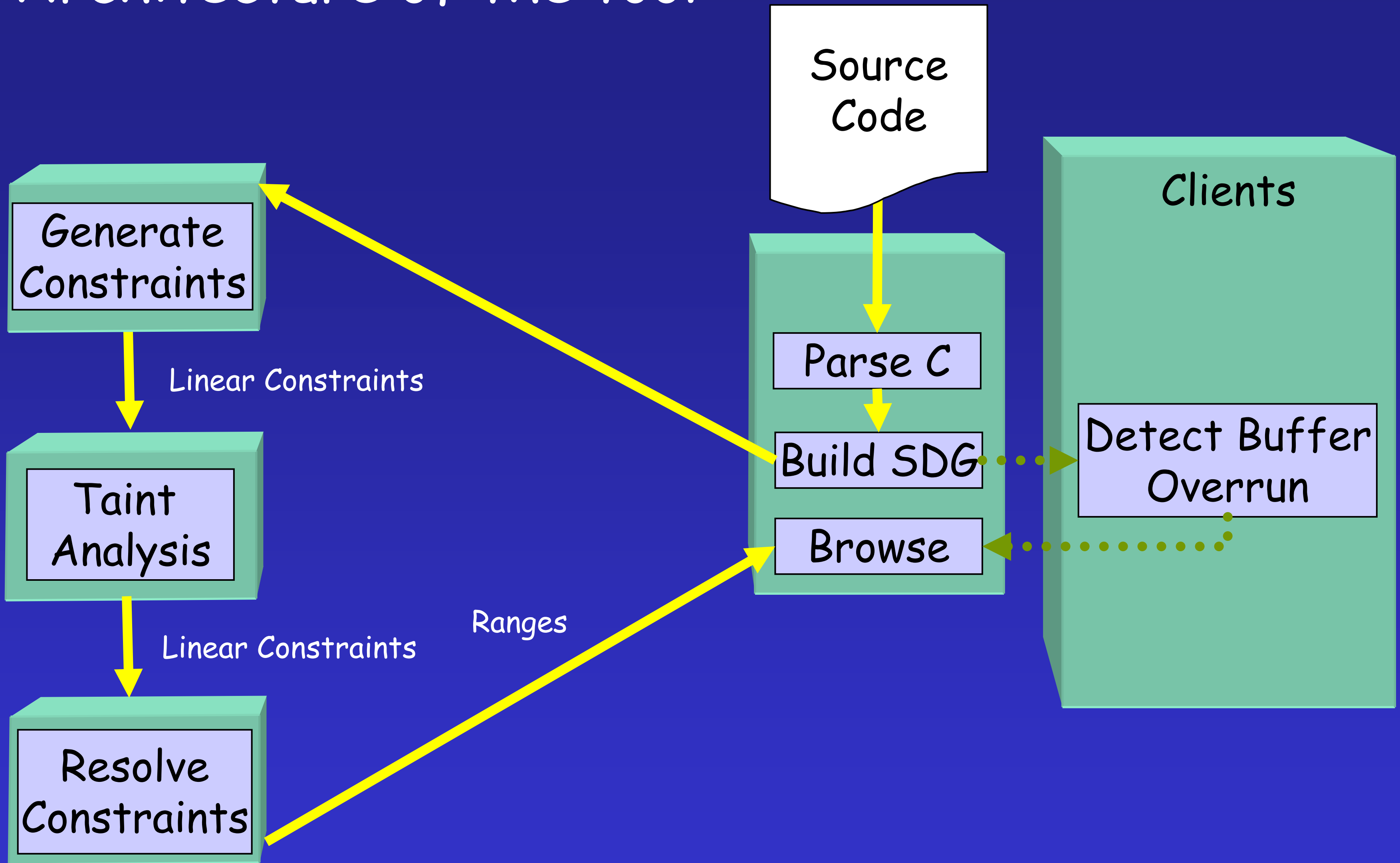
Buffer Overflow Detection Infrastructure



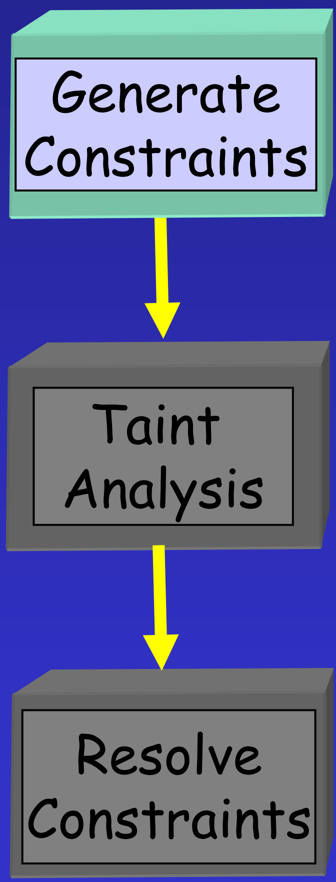
Current Buffer Overflow Detection Infrastructure



Architecture of the tool



Constraint Generation



- Input: SDG
- Output: Linear Constraints
- Basic Idea:
  - Treat buffers as abstract data types
  - Reflect changes in buffers by changing associated buffer variables

Constraint Generation

- Four variables for each string buffer

```
buf_len_max, buf_len_min
buf_alloc_max, buf_alloc_min
```

- Operations on a buffer

```
strcpy(target, source)
target_len_max >= source_len_max
target_len_min <= source_len_min
```

Constraint Generation

- Options Available:
  - *Flow-Sensitive* Analysis:
    - Respects program order
  - *Flow-Insensitive* Analysis:
    - Does not respect program order
  - *Context-Sensitive* modeling of functions:
    - Respects the call-return semantics
  - *Context-Insensitive* modeling of functions:
    - Ignores call-return semantics => imprecise

Context-Insensitive Analysis

```
foo () {
    int x;
    x = foobar(5);
}
```

```
bar () {
  int y;
  y = foobar(30);
}
```

```
int foobar (int z) {
    int i;
    i = z + 1;
    return i;
}
```

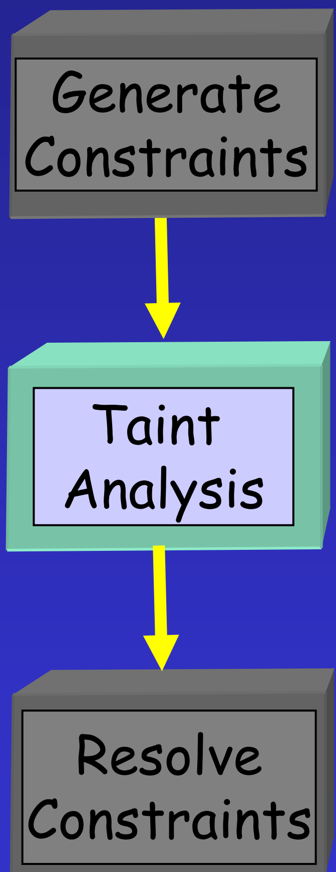
False Path

Result: x = y = [6..31]

Constraint Generation

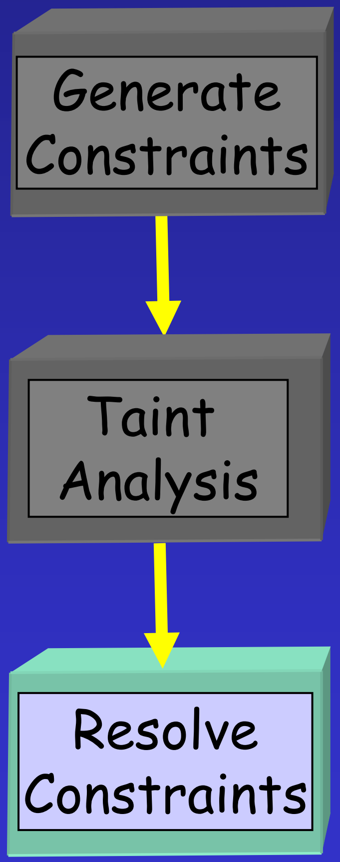
- First half of the talk:
  - Flow-insensitive
  - Context-insensitive for user-defined functions
  - Context-sensitive for some library functions
- Second half of the talk:
  - Context-sensitive for user-defined functions
  - Summary constraints [Sharir & Pnueli 1981]

Taint Analysis



- Remove un-initialized constraint variables
  - e.g. due to incomplete modeling of libraries
- Remove potentially infinite variables
  - e.g. those entered by the user
- Required for solver to function correctly

Constraint Resolution



- Abstract Problem:
  - Given a set of constraints on `min` and `max` variables
  - Get tightest possible fit satisfying the constraints
- Our approach:
  - Model and solve as a linear program

Linear Programming

- A set of constraints C
- Subject to: An objective function F
- Example:

```
Maximize: x
Subject to:
   X <= 3
```

Linear Program Solver

- In our case:
  - Constraints are available
  - Goal: Obtain values for buffer bounds
- Modeling as a Linear Program

Tightest possible fit

```
Minimize: max variable
Subject to:
      Set of Constraints
```

Least Upper Bound

And

```
Maximize: min variable
Subject to:
      Set of Constraints
```

Greatest Lower Bound

Linear Program Solver

- *However,* it can be shown that:

  `Min:` Σ (max vars) - Σ(min vars)

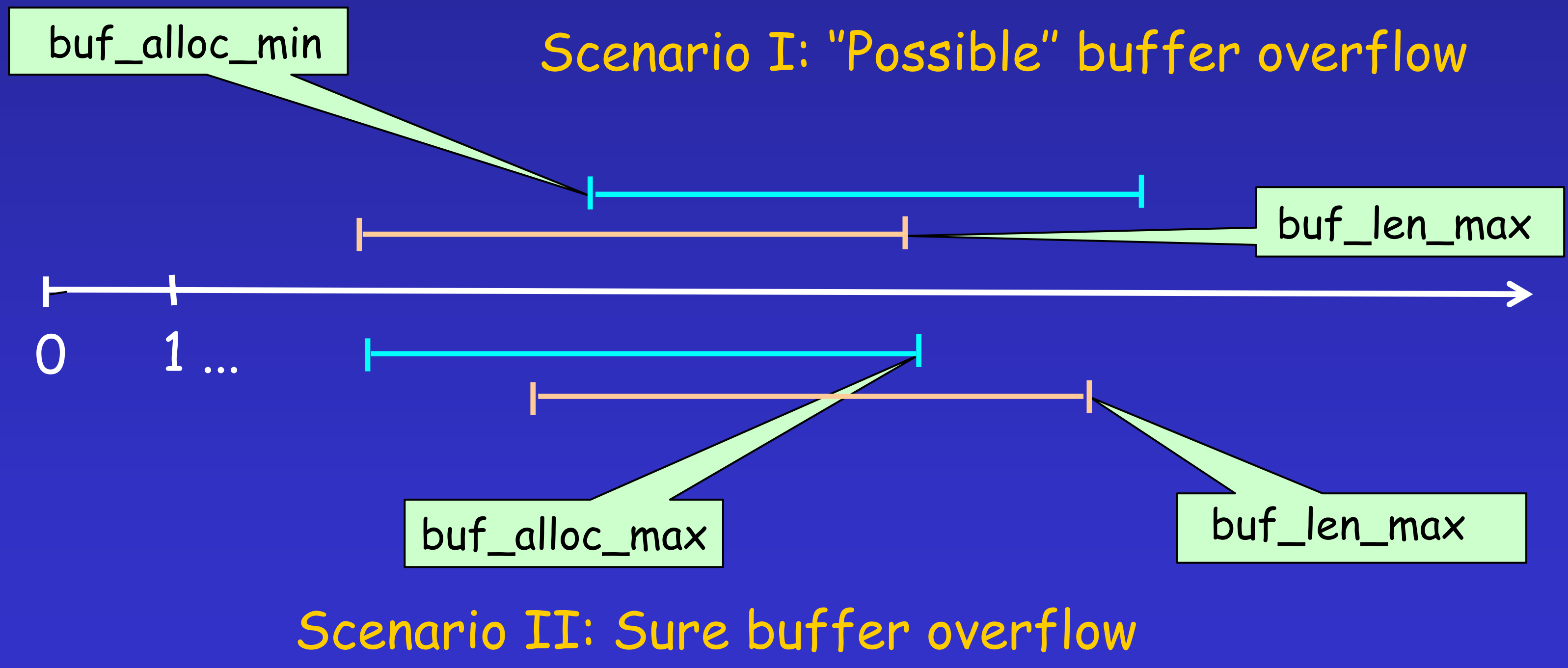
  `Subject to:` Set of Constraints

  yields the same solution for each variable

- Solve just *one LP* and get values for all variables!
  - Joint work with Michael Ferris

Detector: Basic Idea

- Takes values from the LP solver
- Detects overruns based on the values



Overview of Talk

- Tool Architecture
- Constraint Generation & Resolution
- Adding Context Sensitivity
- Results
- Current efforts and future work

Adding Context Sensitivity

- Basic Idea:
  - Summarize the called function.
  - Insert the summary at the call-site in the caller
  - Remove false paths
- Advantages:
  - User functions context sensitivized
  - e.g. wrappers around library functions

Adding Context Sensitivity

```
foo () {
    int x;
    x = foobar(5);
}
```

```
bar () {
   int y;
   y = foobar(30);
}
```

```
int foobar (int z) {
    int i;
    i = z + 1;
    return i;
}
```

Adding Context Sensitivity

```
foo () {
    int x;
    x = foobar(5);
}
```

x = 5 + 1

```
bar () {
   int y;
   y = foobar(30);
}
```

y = 30 + 1

```
int foobar (int z) {
    int i;
    i = z + 1;
    return i;
}
```

Summary: i = z + 1

Adding Context Sensitivity

```
foo () {
    int x;
    x = foobar(5);
}
```

x = 5 + 1

```
bar () {
  int y;
  y = foobar(30);
}
```

y = 30 + 1

```
int foobar (int z) {
    int i;
    i = z + 1;
    return i;
 }
```

Jump Functions

Adding Context Sensitivity

```
foo () {
    int x;
    x = foobar(5);
}
```

x = 5 + 1

```
bar () {
   int y;
   y = foobar(30);
}
```

y = 30 + 1

```
int foobar (int z) {
    int i;
    i = z + 1;
    return i;
}
```

No false paths ☺

x = [6..6]

y = [31..31]

i = [6..31]

Overview of Talk

- Tool Architecture
- Constraint Generation & Resolution
- Adding Context Sensitivity
- Results
- Current efforts and future work

Results

Application	SLOC	Vulnerability	Detected?
WU-FTPD-2.5.0	16000	CA-1999-13	Yes
WU-FTPD-2.6.2	18000	None	4 New*
Sendmail-8.7.6	38000	Identified by BOON	Yes
Sendmail-8.11.6	68000	CA-2003-07	Yes, but...

* Acknowledged by WU-FTPD development team

rdserversc

File Edit Functions Queries Go Window Help

```
while (fgets(buffer, BUFSIZ, svrfp) != NULL) {

    /* Find first non-whitespace character */
    for (bcp = buffer; ((*bcp == '\t') || (*bcp == ' ')); bcp++);

    /* Get rid of comments */
    if ((ecp = strchr(buffer, '#')) != NULL)
        *ecp = '\0';

    /* Skip empty lines */
    if ((bcp == ecp) || (*bcp == '\n'))
        continue;

    /* separate parts */

    hcp = bcp;
    for (acp = hcp;
         (*acp && !isspace(*acp)); acp++);

    /* better have something in access path or skip the line */
    if (!*acp)
        continue;

    *acp++ = '\0';

    while (*acp && isspace(*acp))
        acp++;

    /* again better have something in access path or skip the line */
    if (!*acp)
        continue;

    ecp = acp;

    while (*ecp && (!isspace(*ecp)) && *ecp != '\n')
        ++ecp;

    *ecp = '\0';

    if ((hp = gethostbyname(hcp)) != NULL) {
        struct in_addr in;
        memmove(&in, hp->h_addr, sizeof(in));
        strcpy(hostaddress, inet_ntoa(in));
    }
    else
        strcpy(hostaddress, hcp);

    strcpy(accesspath, acp);
```

Overview of Talk

- Tool Architecture
- Constraint Generation & Resolution
- Adding Context Sensitivity
- Results
- Current efforts and future work

Current efforts and future work

- Flow sensitivity
  - How? Variants of SSA.
  - What is the correct constraint generation model?
  - How to support it using LP?
- Analysis of binaries

Analysis of binaries

- Abstraction to generate constraints on?
  - Source code: variables.
- Type information?
- Identifying arguments to calls
- Will *have* to use flow- and context-sensitive analysis to get good results

The end.