Highly Scalable Distributed Dataflow Analysis

Joseph L. Greathouse

Chelsea LeBlanc       Todd Austin       Valeria Bertacco

Advanced Computer Architecture Laboratory
University of Michigan
Software Errors Abound

- NIST: SW errors cost U.S. ~$60 billion/year as of 2002
A problem has been detected and Windows has been shut down to prevent damage to your computer.

The problem seems to be caused by the following file: SPCMDCON.SYS

PAGE_FAULT_IN_NONPAGED_AREA

If this is the first time you've seen this Stop error screen, restart your computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical information:

*** STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)

*** SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c
Software Errors Abound

- **NIST:** SW errors cost U.S. ~$60 billion/year as of 2002
Software Errors Abound

- NIST: SW errors cost U.S. ~$60 billion/year as of 2002
- FBI CCS: Security Issues $67 billion/year as of 2005
  - >⅓ from viruses, network intrusion, etc.

A problem has been detected and Windows has been shut down to prevent damage to your computer.

The problem seems to be caused by the following file: SPOMDCON.SYS

PAGE_FAULT_IN_NONPAGED_AREA

If this is the first time you've seen this stop error screen, restart your computer. If this screen appears again, follow these steps:

- Check to make sure any new hardware or software is properly installed.
- If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need.
- If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing.
- If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical Information:

```
** STOP: 0x00000050 (0xFD3094C2, 0x00000001, 0xFBFE7617, 0x00000000)  
**  SPOMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3e6dd67c
```
Goals of this Work

- High quality dynamic software analysis
  - Find difficult bugs that other analyses miss

- Distribute Tests to Large Populations
  - Low overhead or users get angry

- Accomplished by sampling the analyses
  - Each user only tests part of the program
Dynamic Dataflow Analysis

- **Associate** meta-data with program values
- **Propagate/Clear** meta-data while executing
- **Check** meta-data for safety & correctness
- **Forms** dataflows of meta/shadow information
Example Dynamic Dataflow Analysis

Data

Meta-data

Input
Example Dynamic Dataflow Analysis

\[ x = \text{read}_\text{input}() \]
Example Dynamic Dataflow Analysis

\[
x = \text{read_input}()
\]
Example Dynamic Dataflow Analysis

\[ y = x \times 1024 \]

\[ x = \text{read}_\text{input}() \]

\[ y = x \times 1024 \]
Example Dynamic Dataflow Analysis

\[
\begin{align*}
   a & \mathbin{+}= y \\
   z & = y \times 75 \\
   y & = x \times 1024
\end{align*}
\]

\[
\begin{align*}
   x & = \text{read\_input}() \\
   y & = x \times 1024 \\
   a & \mathbin{+}= y \\
   z & = y \times 75
\end{align*}
\]
Example Dynamic Dataflow Analysis

\[
a += y \\
z = y \times 75 \\
y = x \times 1024
\]

Input

\[
x = \text{read_input}()
\]

\[
y = x \times 1024
\]

\[
a += y \\
z = y \times 75
\]

validate(x)

Clear
Example Dynamic Dataflow Analysis

\[ a += y \]
\[ z = y \times 75 \]
\[ y = x \times 1024 \]
\[ w = x + 42 \]

**Input**
\[ x = \text{read\_input()} \]

**Validate**
\[ \text{validate}(x) \]

**Input Data**

**Meta-data**
Example Dynamic Dataflow Analysis

```
a += y
z = y * 75
y = x * 1024
w = x + 42
Check w
```

```
validate(x)
```

```
x = read_input()
```

```
y = x * 1024
```

```
a += y
z = y * 75
```

```
Input
```

```
validate(x)
```

```
w = x + 42
```

```
Check w
```
Example Dynamic Dataflow Analysis

\[
a \leftarrow y
\]
\[
z = y \times 75
\]
\[
y = x \times 1024
\]
\[
\text{validate}(x)
\]
\[
w = x + 42
\]
\[
\text{Check } w
\]
\[
\text{Check } a
\]
\[
\text{Check } z
\]
Distributed Dynamic Dataflow Analysis

- Split analysis across large populations
  - Observe more runtime states
  - Report problems developer never thought to test
Distributed Dynamic Dataflow Analysis

- Split analysis across large populations
  - Observe more runtime states
  - Report problems developer never thought to test
Distributed Dynamic Dataflow Analysis

- Split analysis across large populations
  - Observe more runtime states
  - Report problems developer never thought to test
Distributed Dynamic Dataflow Analysis

- Split analysis across large populations
  - Observe more runtime states
  - Report problems developer never thought to test
Distributed Dynamic Dataflow Analysis

- Split analysis across large populations
  - Observe more runtime states
  - Report problems developer never thought to test
Distributed Dynamic Dataflow Analysis

- Split analysis across large populations
  - Observe more runtime states
  - Report problems developer never thought to test
Distributed Dynamic Dataflow Analysis

- Split analysis across large populations
  - Observe more runtime states
  - Report problems developer never thought to test
Problem: DDAs are Slow

- Symbolic Execution (e.g. TaintCheck) 10-200x
- Data Race Detection (e.g. Helgrind) 2-300x
- Memory Checking (e.g. Dr. Memory) 5-50x
- Taint Analysis (e.g. TaintCheck) 2-200x
- Dynamic Bounds Checking 10-80x
- FP Accuracy Verification 100-500x
Our Solution: Sampling

- Lower overheads by skipping some analyses

![Graph showing ideal detection accuracy vs. overhead with two labels: No Analysis and Complete Analysis. The graph shows a linear relationship with higher overhead leading to lower ideal detection accuracy.]
Our Solution: Sampling

- Lower overheads by skipping some analyses

![Graph showing ideal detection accuracy vs. overhead%

Ideal Detection Accuracy (%)

Overhead

SPEED ACCURACY

SPEED ACCURACY

SPEED ACCURACY

SPEED ACCURACY

SPEED ACCURACY

SPEED ACCURACY
Sampling Allows Distribution

![Graph showing the relationship between ideal detection accuracy (%) and overhead. The accuracy increases linearly with overhead.](image)
Sampling Allows Distribution

- Ideal Detection Accuracy (%)
- Overhead

Beta Testers

Developer

University of Michigan
Sampling Allows Distribution

- Ideal Detection Accuracy (%)
- Overhead

- End Users
- Beta Testers
- Developer
Sampling Allows Distribution

Many users testing at little overhead see more errors than one user at high overhead.
Cannot Naïvely Sample Code

Input
Cannot Naïvely Sample Code

\[ a += y \]
\[ y = x \times 1024 \]
\[ x = \text{read\_input}() \]
Cannot Naïvely Sample Code

\[
a += y \\
\]
\[
z = y \times 75 \\
\]
\[
y = x \times 1024 \\
\]
\[
x = \text{read}\_\text{input}() \\
\]

Skip Instr.
Cannot Naïvely Sample Code

\[
a += y \\
z = y * 75 \\
y = x * 1024
\]
Cannot Naïvely Sample Code

```
a += y
z = y * 75

y = x * 1024
w = x + 42
validate(x)
```

Input

\[ x = \text{read\_input()} \]

\[ y = x \times 1024 \]

\[ w = x + 42 \]

\[ \text{validate}(x) \]

\[ z = y \times 75 \]
Cannot Naïvely Sample Code

\[ a += y \]
\[ z = y \times 75 \]
\[ y = x \times 1024 \]
\[ w = x + 42 \]

\[ x = \text{read\_input}() \]
\[ \text{validate}(x) \]

Check \( w \)
Check \( z \)
Check \( a \)
Cannot Naïvely Sample Code

\[ a += y \]
\[ z = y \times 75 \]
\[ y = x \times 1024 \]
\[ w = x + 42 \]

1. Validate(x)
   - Input
   - \( x = \text{read}_\text{input}() \)
   - \( y = x \times 1024 \)
   - \( w = x + 42 \)
   - False Positive
   - False Negative

Check w

Check z

Check a
Our Solution: Sample **Data**, not Code

- Sampling must be aware of meta-data

- Remove meta-data from skipped dataflows
  - Prevents false positives
Our Solution: Sample Data, not Code

- Sampling must be aware of meta-data

- Remove meta-data from skipped dataflows
  - Prevents false positives
Dataflow Sampling Example

Input
Dataflow Sampling Example

\[
\begin{align*}
    x &= \text{read\_input}() \\
y &= x \times 1024 \\
a &= a + y
\end{align*}
\]
Dataflow Sampling Example

\[
\begin{align*}
\text{Input} & \\
\text{x = read_input()} & \\
y & = \text{x} \times 1024 & \\
a & += y & \\
z & = y \times 75 & \\
\text{Skip Dataflow} & \\
\end{align*}
\]
Dataflow Sampling Example

\[ a += y \]
\[ z = y \times 75 \]
\[ y = x \times 1024 \]
\[ x = \text{read\_input()} \]
Dataflow Sampling Example

\[
\begin{align*}
\text{Input} & \quad x = \text{read\_input()} \\
& \quad y = x \times 1024 \\
& \quad a += y \\
& \quad z = y \times 75 \\
& \quad \text{validate}(x)
\end{align*}
\]
Dataflow Sampling Example

\[ a += y \]
\[ z = y \times 75 \]
\[ y = x \times 1024 \]
\[ w = x + 42 \]

\[ x = \text{read\_input()} \]

\[ \text{validate}(x) \]
Dataflow Sampling Example

$\begin{align*}
    &x = \text{read\_input}() \\
    &y = x \times 1024 \\
    &z = y \times 75 \\
    &a += y \\
    &w = x + 42
\end{align*}$

Check $w$
Check $z$
Check $a$
Dataflow Sampling Example

```
// Input
x = read_input()

// Intermediate Calculations
y = x * 1024
z = y * 75

// Intermediate Variables
w = x + 42
a += y

// Checks
Check a
Check z
Check w
```

**False Negative**
Mechanisms for Dataflow Sampling (1)

- Start with demand analysis

**Demand Analysis Tool**

- Native Application
- Instrumented Application (e.g. Valgrind)

**Operating System**
Mechanisms for Dataflow Sampling (1)

- Start with demand analysis

Demand Analysis Tool

Native Application

Instrumented Application (e.g. Valgrind)

Operating System
Mechanisms for Dataflow Sampling (1)

- Start with demand analysis

[Diagram]

Demand Analysis Tool

- Native Application
- Instrumented Application (e.g. Valgrind)

Meta-data
Mechanisms for Dataflow Sampling (1)

- Start with demand analysis

Diagram:
- Demand Analysis Tool
  - Native Application
  - Instrumented Application (e.g. Valgrind)
- Operating System
Mechanisms for Dataflow Sampling (1)

- Start with demand analysis

**Demand Analysis Tool**

- Native Application
- Instrumented Application (e.g., Valgrind)

**Operating System**
Mechanisms for Dataflow Sampling (1)

- Start with demand analysis

**Demand Analysis Tool**

- Native Application
- Instrumented Application (e.g. Valgrind)

**Operating System**

No meta-data
Mechanisms for Dataflow Sampling (1)

- Start with demand analysis

**Demand Analysis Tool**

- Native Application
- Instrumented Application (e.g. Valgrind)

**Operating System**
Mechanisms for Dataflow Sampling (2)

- **Remove** dataflows if execution is too slow
Mechanisms for Dataflow Sampling (2)

- **Remove** dataflows if execution is too slow
Mechanisms for Dataflow Sampling (2)

- **Remove** dataflows if execution is too slow

![Sampling Analysis Tool Diagram]

- Native Application
- Instrumented Application
- Meta-data
Mechanisms for Dataflow Sampling (2)

- **Remove** dataflows if execution is too slow
Mechanisms for Dataflow Sampling (2)

- **Remove** dataflows if execution is too slow
Mechanisms for Dataflow Sampling (2)

- **Remove** dataflows if execution is too slow

![Sampling Analysis Tool](image)

- Native Application
- Instrumented Application
- Operating System
Mechanisms for Dataflow Sampling (2)

- **Remove** dataflows if execution is too slow

```
Sampling Analysis Tool

Native Application

Instrumented Application

Clear meta-data

Operating System
```
Mechanisms for Dataflow Sampling (2)

- **Remove** dataflows if execution is too slow

![Sampling Analysis Tool Diagram]

- Native Application
- Instrumented Application
- Operating System
Prototype Setup

- Taint analysis sampling system
  - Network packets untrusted
- Xen-based demand analysis
  - Whole-system analysis with modified QEMU
- Overhead Manager (OHM) is user-controlled
Benchmarks

- Performance – Network Throughput
  - Example: `ssh_receive`
- Accuracy of Sampling Analysis
  - Real-world Security Exploits

<table>
<thead>
<tr>
<th>Name</th>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>Stack overflow in Apache Tomcat JK Connector</td>
</tr>
<tr>
<td>Eggdrop</td>
<td>Stack overflow in Eggdrop IRC bot</td>
</tr>
<tr>
<td>Lynx</td>
<td>Stack overflow in Lynx web browser</td>
</tr>
<tr>
<td>ProFTPD</td>
<td>Heap smashing attack on ProFTPD Server</td>
</tr>
<tr>
<td>Squid</td>
<td>Heap smashing attack on Squid proxy server</td>
</tr>
</tbody>
</table>
Performance of Dataflow Sampling

ssh_receive

Throughput (MB/s)

Maximum % Time in Analysis

Throughput with no analysis
Accuracy at Very Low Overhead

- Max time in analysis: 1% every 10 seconds
- Always stop analysis after threshold
  - Lowest probability of detecting exploits

<table>
<thead>
<tr>
<th>Name</th>
<th>Chance of Detecting Exploit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>100%</td>
</tr>
<tr>
<td>Eggdrop</td>
<td>100%</td>
</tr>
<tr>
<td>Lynx</td>
<td>100%</td>
</tr>
<tr>
<td>ProFTPD</td>
<td>100%</td>
</tr>
<tr>
<td>Squid</td>
<td>100%</td>
</tr>
</tbody>
</table>
Accuracy with Background Tasks

*ssh_receive* running in background

% Chance of Detecting Exploit

- Apache
- Eggdrop
- Lynx
- ProFTPD
- Squid

Maximum % Time in Analysis

- 10%
- 25%
- 50%
- 75%
- 90%
Accuracy with Background Tasks

ssh_receive running in background

% Chance of Detecting Exploit

Maximum % Time in Analysis

- Apache
- Eggdrop
- Lynx
- ProFTPD
- Squid
Conclusion & Future Work

Dynamic dataflow sampling gives users a knob to control accuracy vs. performance

- Better methods of sample choices
- Combine static information
- New types of sampling analysis
Conclusion & Future Work

Dynamic dataflow sampling gives users a knob to control accuracy vs. performance

- Better methods of sample choices
- Combine static information
- New types of sampling analysis
Outline

- Software Errors and Security
- Dynamic Dataflow Analysis
- Sampling and Distributed Analysis
- Prototype System
- Performance and Accuracy
Detecting Security Errors

- **Static Analysis**
  - Analyze source, formal reasoning
    + Find all reachable, defined errors
    - Intractable, requires expert input, no system state

- **Dynamic Analysis**
  - Observe and test runtime state
    + Find deep errors as they happen
    - Only along traversed path, very slow
Security Vulnerability Example

- Buffer overflows a large class of security vulnerabilities

```c
void foo()
{
    int local_variables;
    int buffer[256];
    ...
    buffer = read_input();
    ...
    return;
}
```
Security Vulnerability Example

- Buffer overflows a large class of security vulnerabilities

```c
void foo()
{
    int local_variables;
    int buffer[256];
    ...
    buffer = read_input();
    ...
    return;
}
```

If `read_input()` reads 200 ints
Security Vulnerability Example

- Buffer overflows a large class of security vulnerabilities

```c
void foo()
{
    int local_variables;
    int buffer[256];
    ...
    buffer = read_input();
    ...
    return;
}
```

If `read_input()` reads >256 ints
Performance of Dataflow Sampling (2)

Throughput of netcat_receive

Throughput with no analysis

Throughput (MB/s)

Maximum % Time in Analysis
Performance of Dataflow Sampling (3)

Graph:  
- **ssh_transmit**
- **Throughput with no analysis**

**Throughput (MB/s)**

- Maximum

**Maximum % Time in Analysis**

0 20 40 60 80 100
Accuracy with Background Tasks

netcat_receive running with benchmark

% Chance of Detecting Exploit

- Apache
- Eggdrop
- Lynx
- ProFTPD
- Squid

Maximum Allowed Overhead %

10% 25% 50% 75% 90%

0 10 20 30 40 50 60 70 80 90 100

0.38 0.4

University of Michigan