Accelerating Dynamic Software Analyses

Joseph L. Greathouse
Ph.D. Candidate
Advanced Computer Architecture Laboratory
University of Michigan

December 1, 2011
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
  - >1/3 from viruses, network intrusion, etc.
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
  - >1/3 from viruses, network intrusion, etc.

Cataloged Software Vulnerabilities

- CVE Candidates
- CERT Vulnerabilities

![Chart showing cataloged software vulnerabilities from 2000 to 2008](chart.png)
Example of Modern Bug

Nov. 2010 OpenSSL Security Flaw
Example of Modern Bug

Thread 1
mylen=small

\[
\text{if}(\text{ptr}==\text{NULL})
\]

\[
\text{len1=thread\_local->mylen;}
\]

\[
\text{ptr=malloc(len1);}\]

\[
\text{memcpy(ptr, data1, len1)}\]

Thread 2
mylen=large

\[
\text{if}(\text{ptr}==\text{NULL})
\]

\[
\text{len2=thread\_local->mylen;}
\]

\[
\text{ptr=malloc(len2);}\]

\[
\text{memcpy(ptr, data2, len2)}\]
Example of Modern Bug

Thread 1
mylen=small

if(ptr==NULL)

len1=thread_local->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)

Thread 2
mylen=large

if(ptr==NULL)
len2=thread_local->mylen;
ptr=malloc(len2);
memcpy(ptr, data2, len2)

ptr
∅
Example of Modern Bug

Thread 1
mylen=small

if(ptr==NULL)

len1=thread_local->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)

Thread 2
mylen=large

if(ptr==NULL)
len2=thread_local->mylen;
ptr=malloc(len2);
memcpy(ptr, data2, len2)

ptr
∅
**Example of Modern Bug**

Thread 1
mylen=small

\[
\text{if}(\text{ptr}==\text{NULL})
\]

\[
\text{len1}=\text{thread}_\text{local}->\text{mylen};
\]

\[
\text{ptr} = \text{malloc}(\text{len1});
\]

\[
\text{memcpy}(\text{ptr}, \text{data1}, \text{len1})
\]

Thread 2
mylen=large

\[
\text{if}(\text{ptr}==\text{NULL})
\]

\[
\text{len2}=\text{thread}_\text{local}->\text{mylen};
\]

\[
\text{ptr} = \text{malloc}(\text{len2});
\]

\[
\text{memcpy}(\text{ptr}, \text{data2}, \text{len2})
\]
**Example of Modern Bug**

Thread 1
mylen=small

if(ptr==NULL)

len1=thread_local->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)

Thread 2
mylen=large

if(ptr==NULL)

len2=thread_local->mylen;
ptr=malloc(len2);
memcpy(ptr, data2, len2)

LEAKED
Example of Modern Bug

Thread 1
mylen=small
if(ptr==NULL)

len1=thread_local->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)

Thread 2
mylen=large
if(ptr==NULL)
len2=thread_local->mylen;
ptr=malloc(len2);

memcpy(ptr, data2, len2)

ptr

LEAKED
Example of Modern Bug

Thread 1
mylen=small

if(ptr==NULL)

len1=thread_local->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)

Thread 2
mylen=large

if(ptr==NULL)
len2=thread_local->mylen;
ptr=malloc(len2);
memcpy(ptr, data2, len2)

TIME

ptr

LEAKED
Example of Modern Bug

Thread 1
mylen=small
if(ptr==NULL)
len1=thread_local->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)

Thread 2
mylen=large
if(ptr==NULL)
len2=thread_local->mylen;
ptr=malloc(len2);
memcpy(ptr, data2, len2)

memcpy(ptr, data2, len2)

LEAKED
Dynamic Software Analysis

- Analyze the program as it runs
  + System state, find errors on any executed path
  - LARGE runtime overheads, only test one path
Dynamic Software Analysis

- Analyze the program as it runs
  - System state, find errors on any executed path
  - LARGE runtime overheads, only test one path
Dynamic Software Analysis

- Analyze the program as it runs
  - System state, find errors on any executed path
  - LARGE runtime overheads, only test one path
Dynamic Software Analysis

- Analyze the program as it runs
  + System state, find errors on any executed path
  - LARGE runtime overheads, only test one path

Analysis Instrumentation

Developer

Analysis Results

In-House Test Server(s)
Runtime Overheads: How Large?

- **Data Race Detection** (e.g. Inspector XE)  
  2-300x

- **Memory Checking** (e.g. MemCheck)  
  5-50x

- **Taint Analysis** (e.g. TaintCheck)  
  2-200x

- **Dynamic Bounds Checking**  
  10-80x

- **Symbolic Execution**  
  10-200x
Outline

- Problem Statement

- Background Information
  - Demand-Driven Dynamic Dataflow Analysis

- Proposed Solutions
  - Demand-Driven Data Race Detection
  - Sampling to Cap Maximum Overheads
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: Taint Analysis

Input

Data

Meta-data
Example: Taint Analysis

```
x = read_input()
```

Input

Data

Meta-data
Example: Taint Analysis

```
x = read_input()
```

Diagram:
- Data
- Meta-data
- Input
- Associate

- `x = read_input()`
Example: Taint Analysis

\[ y = x \times 1024 \]
Example: Taint Analysis

```
# Source code
a += y
z = y * 75
y = x * 1024
```

Diagram:
- **Input**: $x = \text{read\_input}()$
- **Data**: $y = x \times 1024$
- **Meta-data**: $a += y$, $z = y \times 75$
Example: Taint Analysis

```
Input

x = read_input()

validate(x)

y = x * 1024

z = y * 75

a += y
```

Data

Meta-data

Clear
Example: Taint Analysis

```
val x = read_input()

validate(x)

val y = x * 1024

val w = x + 42

val a += y

val z = y * 75
```
Example: Taint Analysis

```
a += y
z = y * 75
y = x * 1024
w = x + 42
```
Example: Taint Analysis

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

Input

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

```
validate(x)
```

```
y = x * 1024
```

```
a += y
```

```
z = y * 75
```

Check a

Check z

Check w

Check w

Check z

Check a

Meta-data

Data
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

Native Application

Instrumented Application

Meta-Data Detection
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

![Diagram showing native and instrumented applications, non-shadowed data, and meta-data detection]
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

Native Application

Instrumented Application

Meta-Data Detection

Shadowed Data
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

Native Application

Instrumented Application

Meta-Data Detection
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

Native Application

Instrumented Application

Meta-Data Detection
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

Native Application

Instrumented Application

Meta-Data Detection
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

Native Application

Instrumented Application

No meta-data

Meta-Data Detection
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

![Diagram of Native Application and Instrumented Application]

Meta-Data Detection
Finding Meta-Data

- No additional overhead when no meta-data
  - Needs hardware support
- Take a fault when touching shadowed data
Finding Meta-Data

- No additional overhead when no meta-data
  - Needs hardware support
- Take a fault when touching shadowed data
- Solution: Virtual Memory Watchpoints
Finding Meta-Data

- No additional overhead when no meta-data
  - Needs hardware support
- Take a fault when touching shadowed data
- Solution: Virtual Memory Watchpoints

V→P
Finding Meta-Data

- No additional overhead when no meta-data
  - Needs hardware support
- Take a fault when touching shadowed data
- Solution: Virtual Memory Watchpoints

\[ V \rightarrow P \]
Finding Meta-Data

- No additional overhead when no meta-data
  - Needs hardware support
- Take a fault when touching shadowed data
- Solution: Virtual Memory Watchpoints
Finding Meta-Data

- No additional overhead when no meta-data
  - Needs hardware support
- Take a fault when touching shadowed data
- Solution: Virtual Memory Watchpoints
Finding Meta-Data

- No additional overhead when no meta-data
  - Needs hardware support
- Take a fault when touching shadowed data
- Solution: Virtual Memory Watchpoints
**Results by Ho et al.**

- **Imbench Best Case Results:**

<table>
<thead>
<tr>
<th>System</th>
<th>Slowdown (normalized)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Taint Analysis</td>
<td>101.7x</td>
</tr>
<tr>
<td>On-Demand Taint Analysis</td>
<td>1.98x</td>
</tr>
</tbody>
</table>

- **Results when everything is tainted:**

![Graph showing system slowdown](image-url)
Outline

- Problem Statement

- Background Information
  - Demand-Driven Dynamic Dataflow Analysis

- Proposed Solutions
  - Demand-Driven Data Race Detection
  - Sampling to Cap Maximum Overheads
Software Data Race Detection

- Add checks around every memory access
- Find inter-thread sharing events
- Synchronization between write-shared accesses?
  - No? Data race.
Example of Data Race Detection

Thread 1
mylen=small

if(ptr==NULL)
len1=thread_local->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)

Thread 2
mylen=large

if(ptr==NULL)
len2=thread_local->mylen;
ptr=malloc(len2);
memcpy(ptr, data2, len2)
Example of Data Race Detection

Thread 1
mylen=small

if(ptr==NULL)
len1=thread_local->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)

Thread 2
mylen=large

if(ptr==NULL)
len2=thread_local->mylen;
ptr=malloc(len2);
memcpy(ptr, data2, len2)
Example of Data Race Detection

Thread 1
mylen=small

if(ptr==NULL)
len1=thread_local->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)

Thread 2
mylen=large

if(ptr==NULL)
len2=thread_local->mylen;
ptr=malloc(len2);
memcpy(ptr, data2, len2)
Example of Data Race Detection

Thread 1
mylen=small

if(ptr==NULL)
len1=thread_local->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)

Thread 2
mylen=large

if(ptr==NULL)
len2=thread_local->mylen;
ptr=malloc(len2);
memcpy(ptr, data2, len2)
Example of Data Race Detection

Thread 1
mylen = small
if (ptr == NULL)
len1 = thread_local->mylen;
ptr = malloc(len1);
memcpy(ptr, data1, len1)

Thread 2
mylen = large
if (ptr == NULL)
len2 = thread_local->mylen;
ptr = malloc(len2);
memcpy(ptr, data2, len2)
Example of Data Race Detection

Thread 1
mylen=small

\[ \text{if(ptr==NULL)} \]
\[ \text{len1=thread\_local->mylen;} \]
\[ \text{ptr=malloc(len1);} \]
\[ \text{memcpy(ptr, data1, len1)} \]

Thread 2
mylen=large

\[ \text{ptr write-shared?} \]

\[ \text{if(ptr==NULL)} \]
\[ \text{len2=thread\_local->mylen;} \]
\[ \text{ptr=malloc(len2);} \]
\[ \text{memcpy(ptr, data2, len2)} \]
Example of Data Race Detection

Thread 1
mylen=small

if(ptr==NULL)
len1=thread_local->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)

Thread 2
mylen=large

if(ptr==NULL)
len2=thread_local->mylen;
ptr=malloc(len2);
memcpy(ptr, data2, len2)

ptr write-shared?
Example of Data Race Detection

Thread 1
mylen=small

if(ptr==NULL)
len1=thread_local-&gt;mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1);

Thread 2
mylen=large

if(ptr==NULL)
len2=thread_local-&gt;mylen;
ptr=malloc(len2);
memcpy(ptr, data2, len2);
Example of Data Race Detection

Thread 1

\[
\text{mylen} = \text{small}
\]

\[
\text{if}(\text{ptr}==\text{NULL})
\]

\[
\text{len1} = \text{thread\_local->mylen};
\]

\[
\text{ptr} = \text{malloc}(\text{len1});
\]

\[
\text{memcpy}(\text{ptr}, \text{data1}, \text{len1})
\]

Thread 2

\[
\text{mylen} = \text{large}
\]

\[
\text{if}(\text{ptr}==\text{NULL})
\]

\[
\text{len2} = \text{thread\_local->mylen};
\]

\[
\text{ptr} = \text{malloc}(\text{len2});
\]

\[
\text{memcpy}(\text{ptr}, \text{data2}, \text{len2})
\]

Interleaved Synchronization?
SW Race Detection is Slow

![Bar chart showing race detector slowdown for Phoenix and PARSEC benchmarks. The x-axis represents benchmarks such as "histogram", "kmeans", "matrix_multiply", "pca", etc., and the y-axis represents race detector slowdown (x). The GeoMean is also shown for each category.](chart.png)
Inter-thread Sharing is What’s Important

“Data races ... are failures in programs that access and update shared data in critical sections” – Netzer & Miller, 1992

```c
if(ptr==NULL)
    len1=thread_local->mylen;
    ptr=malloc(len1);
    memcpy(ptr, data1, len1);

if(ptr==NULL)
    len2=thread_local->mylen;
    ptr=malloc(len2);
    memcpy(ptr, data2, len2);
```
Inter-thread Sharing is What’s Important

“Data races ... are failures in programs that access and update shared data in critical sections” – Netzer & Miller, 1992

```c
if(ptr==NULL)
len1=thread_local->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)

if(ptr==NULL)
len2=thread_local->mylen;
ptr=malloc(len2);
memcpy(ptr, data2, len2)
```
Inter-thread Sharing is What’s Important

“Data races ... are failures in programs that access and update shared data in critical sections” – Netzer & Miller, 1992

if(ptr==NULL) { len1=thread_local->mylen; ptr=malloc(len1); memcpy(ptr, data1, len1); }
Inter-thread Sharing is What’s Important

“Data races ... are failures in programs that access and update shared data in critical sections” – Netzer & Miller, 1992

```
if(ptr==NULL)
len1=thread_local->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)
```

```
if(ptr==NULL)
len2=thread_local->mylen;
ptr=malloc(len2);
memcpy(ptr, data2, len2)
```

Thread-local data

NO SHARING
Inter-thread Sharing is What’s Important

“Data races ... are failures in programs that access and update shared data in critical sections” – Netzer & Miller, 1992

```c
if (ptr == NULL)
    len1 = thread_local->mylen;
ptr = malloc(len1);
memcpy(ptr, data1, len1)

len2 = thread_local->mylen;
ptr = malloc(len2);
memcpy(ptr, data2, len2)
```
“Data races ... are failures in programs that access and update shared data in critical sections” – Netzer & Miller, 1992

```c
if (ptr == NULL)
    len1 = thread_local->mylen;
    ptr = malloc(len1);
    memcpy(ptr, data1, len1);

if (ptr == NULL)
    len2 = thread_local->mylen;
    ptr = malloc(len2);
    memcpy(ptr, data2, len2);
```
Inter-thread Sharing is What’s Important

“Data races ... are failures in programs that access and update shared data in critical sections” – Netzer & Miller, 1992

```c
if(ptr==NULL)
    len1=thread_local->mylen;
    ptr=malloc(len1);
    memcpy(ptr, data1, len1)

if(ptr==NULL)
    len2=thread_local->mylen;
    ptr=malloc(len2);
    memcpy(ptr, data2, len2)
```
Inter-thread Sharing is What’s Important

“Data races ... are failures in programs that **access and update shared data** in critical sections” – Netzer & Miller, 1992

```c
if(ptr==NULL)
    len1=thread_local->mylen;
    ptr=malloc(len1);
    memcpy(ptr, data1, len1);

len2=thread_local->mylen;
ptr=malloc(len2);
memcpy(ptr, data2, len2);
```
Very Little Inter-thread Sharing

<table>
<thead>
<tr>
<th>% Write-Sharing Events</th>
<th>Phoenix</th>
<th>PARSEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>histogram</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kmeans</td>
<td></td>
<td></td>
</tr>
<tr>
<td>linear_regression</td>
<td></td>
<td></td>
</tr>
<tr>
<td>matrix_multiply</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pca</td>
<td></td>
<td></td>
</tr>
<tr>
<td>string_match</td>
<td></td>
<td></td>
</tr>
<tr>
<td>word_count</td>
<td></td>
<td></td>
</tr>
<tr>
<td>blackscholes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bodytrack</td>
<td></td>
<td></td>
</tr>
<tr>
<td>facesim</td>
<td></td>
<td></td>
</tr>
<tr>
<td>freqmine</td>
<td></td>
<td></td>
</tr>
<tr>
<td>raytrace</td>
<td></td>
<td></td>
</tr>
<tr>
<td>swaptions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fluidanimate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vips</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x264</td>
<td></td>
<td></td>
</tr>
<tr>
<td>canneal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dedup</td>
<td></td>
<td></td>
</tr>
<tr>
<td>streamcluster</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Use Demand-Driven Analysis!

Multi-threaded Application

Software Race Detector

Inter-thread Sharing Monitor
Use Demand-Driven Analysis!

Multi-threaded Application

Local Access

Software Race Detector

Inter-thread Sharing Monitor
Use Demand-Driven Analysis!

Software Race Detector

Local Access

Inter-thread Sharing Monitor

Multi-threaded Application Software Race Detector Local Access Inter-thread Sharing Monitor
Use Demand-Driven Analysis!

- Multi-threaded Application
- Inter-thread sharing
- Software Race Detector
- Inter-thread Sharing Monitor
Use Demand-Driven Analysis!

- Multi-threaded Application Software Race Detector
- Inter-thread sharing
- Inter-thread Sharing Monitor
Use Demand-Driven Analysis!

Inter-thread Sharing Monitor
Use Demand-Driven Analysis!

Inter-thread Sharing Monitor
Use Demand-Driven Analysis!

Inter-thread Sharing Monitor

Local Access

Software Race Detector

Multi-threaded Application
Use Demand-Driven Analysis!

Inter-thread Sharing Monitor

Local Access

Software Race Detector

Multi-threaded Application

Use Demand-Driven Analysis!
Finding Inter-thread Sharing

- Virtual Memory Watchpoints?
Finding Inter-thread Sharing

- Virtual Memory Watchpoints?

![Diagram showing fault with two blocks of memory]

FAULT
Finding Inter-thread Sharing

- Virtual Memory Watchpoints?
Finding Inter-thread Sharing

- Virtual Memory Watchpoints?

Inter-Thread Sharing

FAULT
Finding Inter-thread Sharing

- Virtual Memory Watchpoints?

  - ~100% of accesses cause page faults
Finding Inter-thread Sharing

- Virtual Memory Watchpoints?
  - ~100% of accesses cause page faults

- Granularity Gap
Finding Inter-thread Sharing

- Virtual Memory Watchpoints?
  - ~100% of accesses cause page faults

- Granularity Gap
- Per-process not per-thread
- Must go through the kernel on faults
- Syscalls for setting/removing meta-data
Hardware Sharing Detector

- Hardware Performance Counters

<table>
<thead>
<tr>
<th></th>
<th>Perf. Ctrs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline</td>
<td>0</td>
</tr>
<tr>
<td>Cache</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
Hardware Sharing Detector

- Hardware Performance Counters

```
Pipeline

Cache

Perf. Ctrs

1
0
0
0
```
Hardware Sharing Detector

- Hardware Performance Counters

<table>
<thead>
<tr>
<th></th>
<th>Pipeline</th>
<th>Cache</th>
<th>Perf. Ctrs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
Hardware Sharing Detector

- Hardware Performance Counters

<table>
<thead>
<tr>
<th>Pipeline</th>
<th>Perf. Ctrs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache</th>
<th>Perf. Ctrs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
Hardware Sharing Detector

- Hardware Performance Counters

<table>
<thead>
<tr>
<th></th>
<th>Perf. Ctrs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>-1</td>
</tr>
<tr>
<td>Cache</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
Hardware Sharing Detector

- Hardware Performance Counters

<table>
<thead>
<tr>
<th>Pipeline</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Perf. Ctrs

- PEBS

- Debug Store

- Pipeline Counters:
  - 2
  - -1
  - 1
  - 0

- Cache Counters:

- PEBS Indicator: FAULT
Hardware Sharing Detector

- Hardware Performance Counters

<table>
<thead>
<tr>
<th></th>
<th>Perf. Ctrs</th>
<th>PEBS</th>
<th>Debug Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline</td>
<td>2</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Pipeline</td>
<td>-1</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Cache</td>
<td>1</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Cache</td>
<td>0</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>
# Hardware Sharing Detector

## Hardware Performance Counters

<table>
<thead>
<tr>
<th></th>
<th>Perf. Ctrs</th>
<th>PEBS</th>
<th>Debug Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline</td>
<td>2</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-1</td>
<td>Armed</td>
<td></td>
</tr>
<tr>
<td>Cache</td>
<td>1</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>
Hardware Sharing Detector

- Hardware Performance Counters

<table>
<thead>
<tr>
<th>Pipeline</th>
<th>Perf. Ctrs</th>
<th>PEBS</th>
<th>Debug Store</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>-</td>
<td>EFLAGS</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Armed</td>
<td>EIP</td>
</tr>
<tr>
<td>Cache</td>
<td>1</td>
<td>-</td>
<td>RegVals</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>-</td>
<td>MemInfo</td>
</tr>
</tbody>
</table>

Precise Fault
Hardware Sharing Detector

- **Hardware Performance Counters**

  - Pipeline
  - Cache
  - Perf. Ctrs: 2-3-1-0
  - PEBS: Armed
  - Debug Store: EFLAGS, EIP, RegVals, MemInfo, Precise Fault

- **Intel’s HITM event: W→R Data Sharing**

  - Core 1: S-M
  - Core 2: S-I
Hardware Sharing Detector

- **Hardware Performance Counters**

  - Pipeline
  - Cache
  - Perf. Ctrs
    - 2
    - 0
    - 1
    - 0
  - PEBS
    - Armed
  - Debug Store
    - EFLAGS
    - EIP
    - RegVals
    - MemInfo
    - Precise Fault

- **Intel’s HITM event: W→R Data Sharing**

  - Core 1
    - S
    - M
  - Core 2
    - S
    - I
Hardware Sharing Detector

- Hardware Performance Counters

- Intel’s HITM event: W→R Data Sharing
Potential Accuracy & Perf. Problems

- Limitations of Performance Counters
  - HITM only finds W→R Data Sharing
  - Hardware prefetcher events aren’t counted

- Limitations of Cache Events
  - SMT sharing can’t be counted
  - Cache eviction causes missed events
  - False sharing, etc…

- PEBS events still go through the kernel
Demand-Driven Analysis on Real HW

Execute Instruction
Demand-Driven Analysis on Real HW

Execute Instruction

Analysis Enabled?
Demand-Driven Analysis on Real HW

- Execute Instruction
  - Analysis Enabled?
    - YES
      - SW Race Detection
Demand-Driven Analysis on Real HW

Execute Instruction

Analysis Enabled?

YES

SW Race Detection

Sharing Recently?
Demand-Driven Analysis on Real HW

Execute Instruction

Analysis Enabled?

YES

SW Race Detection

Sharing Recently?

YES

YES

YES
Demand-Driven Analysis on Real HW

1. Execute Instruction
2. Analysis Enabled?
   - YES → SW Race Detection
   - NO → Disable Analysis
3. Sharing Recently?
   - NO → Analysis Enabled?
   - YES
Demand-Driven Analysis on Real HW

HITM Interrupt?

NO

Execute Instruction

Analysis Enabled?

NO

SW Race Detection

YES

Disable Analysis

Sharing Recently?

NO

YES
Demand-Driven Analysis on Real HW
Demand-Driven Analysis on Real HW

- Execute Instruction
- Analysis Enabled?
  - NO
  - YES: SW Race Detection
  - NO: Disable Analysis
- Sharing Recently?
  - NO
  - YES: Hitm Interrupt?
    - NO
    - YES: Enable Analysis
  - YES: NO
Demand-Driven Analysis on Real HW

- Execute Instruction
- Analysis Enabled?
- SW Race Detection
- Sharing Recently?
- Enable Analysis
- Disable Analysis

HITM Interrupt?

NO

YES

NO

YES

NO

YES

NO

YES
Performance Increases

![Graph showing performance increases for Phoenix and PARSEC](image)
Performance Increases

Demand-driven Analysis Speedup (x)

<table>
<thead>
<tr>
<th>Phoenix</th>
<th>PARSEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>51x</td>
<td></td>
</tr>
</tbody>
</table>

- histogram
- kmeans
- matrix_multiply
- string_match
- word_count
- GeoMean
- blackscholes
- bodytrack
- facesim
- ferret
- freqmine
- raytrace
- swaptions
- fluidanimate
- vips
- x264
- canmeal
- dedup
- streamcluster
- GeoMean
Demand-Driven Analysis Accuracy

[Bar chart showing demand-driven analysis speedup (x) for various benchmarks. Each benchmark is represented with a bar, and the GeoMean is shown at the end.]
Demand-Driven Analysis Accuracy
Demand-Driven Analysis Accuracy

Accuracy vs. Continuous Analysis: 97%
Outline

■ Problem Statement

■ Background Information
  ▶ Demand-Driven Dynamic Dataflow Analysis

■ Proposed Solutions
  ▶ Demand-Driven Data Race Detection
  ▶ Sampling to Cap Maximum Overheads
Reducing Overheads Further: Sampling

- Lower overheads by skipping some analyses

![Graph showing ideal detection accuracy vs. overhead]

- No Analysis
- Complete Analysis

Ideal Detection Accuracy (%) vs. Overhead
Reducing Overheads Further: Sampling

- Lower overheads by skipping some analyses

![Graph showing ideal detection accuracy vs. overhead]

Ideal Detection Accuracy (%) vs. Overhead
Reducing Overheads Further: Sampling

![Graph showing the relationship between Ideal Detection Accuracy (%) and Overhead. The graph is a straight line indicating an increase in accuracy as overhead increases. The y-axis represents Ideal Detection Accuracy (%) ranging from 0 to 100, and the x-axis represents Overhead. A developer icon is placed at the top right corner of the graph.]
Reducing Overheads Further: Sampling

Ideal Detection Accuracy (%) vs Overhead

Beta Testers

Developer
Sampling Allows Distribution

Ideal Detection Accuracy (%) vs. Overhead

- End Users
- Beta Testers
- Developer
Many users testing at little overhead see more errors than one user at high overhead.
Cannot Naïvely Sample Code

Input
**Cannot Naïvely Sample Code**

```plaintext
Input

x = read_input()

y = x * 1024

a += y
```
Cannot Naïvely Sample Code

```
a += y
z = y * 75
y = x * 1024
x = read_input()
```

Input

```
a += y
```

```
z = y * 75
```

Skip Instr.
Cannot Naïvely Sample Code

\[ a += y \]
\[ z = y \times 75 \]
\[ y = x \times 1024 \]
\[ x = \text{read_input()} \]
\[ \text{validate}(x) \]

Skip Instr.
Cannot Naïvely Sample Code

\[ a += y \]
\[ z = y \times 75 \]
\[ x = \text{read\_input}() \]
\[ y = x \times 1024 \]
\[ w = x + 42 \]
\[ \text{validate}(x) \]
Cannot Naïvely Sample Code

- \( x = \text{read}\_\text{input}() \)
- \( y = x \times 1024 \)
- \( w = x + 42 \)
- \( a += y \)
- \( z = y \times 75 \)

Input

Validate(x)

False Positive

Check w

False Negative

Check z

Check a

Check z
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

ON

OFF
Dataflow Sampling Example

\[ a += y \]
\[ y = x \times 1024 \]
\[ x = \text{read\_input()} \]

Input

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

```
a += y
z = y * 75
y = x * 1024
x = read_input()
```

Input

```
x = read_input()
y = x * 1024
```

Skip Dataflow

```
a += y
z = y * 75
```
Dataflow Sampling Example

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

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

\[
\begin{align*}
\text{Input} & \\
\text{validate}(x) & \rightarrow \text{x = read_input()} \\
y = x \times 1024 & \rightarrow \text{a += y} \rightarrow \text{z = y \times 75} \\
w = x + 42 &
\end{align*}
\]
Dataflow Sampling Example

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

Input

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

validate(x)

Check w

Check z

Check a
Dataflow Sampling Example

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

Input

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

validate(x)

Check w

Check z

Check a

False Negative
Dataflow Sampling

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

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

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

---

**Sampling Analysis Tool**

- **Native Application**
- **Instrumented Application**

---

**Operating System**
Dataflow Sampling

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

![Sampling Analysis Tool Diagram]

- Native Application
- Instrumented Application

Meta-data
Dataflow Sampling

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

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

Sampling Analysis Tool

- Native Application
- Instrumented Application
- OH Threshold

Operating System
Dataflow Sampling

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

---

**Sampling Analysis Tool**

- Native Application
- Instrumented Application

---

Operating System
Dataflow Sampling

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

---

**Sampling Analysis Tool**

- **Native Application**
- **Instrumented Application**

**Clear meta-data**

**Operating System**
Dataflow Sampling

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

Throughput (MB/s) vs. Maximum % Time in Analysis for ssh_receive.

Throughput with no analysis is shown as a dashed line at 20 MB/s.
Accuracy with Background Tasks

*ssh_receive* running in background

% Chance of Detecting Exploit

![Graph with data points for Apache, Eggdrop, Lynx, ProFTPD, and Squid.](chart.png)

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
Performance Difference
Width Test