Accelerating Dynamic Software Analyses

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

December 2, 2011
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
  - $\frac{1}{3}$ 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
  - >⅓ from viruses, network intrusion, etc.

Cataloged Software Vulnerabilities


- CVE Candidates
- CERT Vulnerabilities
Example of Modern Bug

Nov. 2010 OpenSSL Security Flaw
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)
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

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 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)

leitung
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
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
Runtime Overheads: How Large?

- Data Race Detection (e.g. Inspector XE) 2-300x
- Taint Analysis (e.g. TaintCheck) 2-200x
- Memory Checking (e.g. MemCheck) 5-50x
- 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
Example: Taint Analysis

```
x = read_input()
```

**Input**

**Data**

**Meta-data**
Example: Taint Analysis

```
x = read_input()
```

Input

Associate

Data

Meta-data
Example: Taint Analysis

```
x = read_input()
y = x * 1024
```
Example: Taint Analysis

Input

\( x = \text{read}\textunderscore\text{input}() \)

\( y = x \times 1024 \)

\( a += y \)

\( z = y \times 75 \)
Example: Taint Analysis

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

```
x = read_input()

validate(x)
```

Input

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

Clear
Example: Taint Analysis

x = read_input()

validate(x)

y = x * 1024

w = x + 42

z = y * 75

a += y
Example: Taint Analysis

\[
a += y
\]
\[
z = y \times 75
\]
\[
y = x \times 1024
\]
\[
w = x + 42
\]
\[
\text{Input}
\]
\[
\text{Data}
\]
\[
\text{Meta-data}
\]
\[
\text{validate}(x)
\]
\[
\text{Check } w
\]

Example: Taint Analysis

\[
x = \text{read_input}()
\]
\[
y = x \times 1024
\]
\[
a += y
\]
\[
z = y \times 75
\]
\[
w = x + 42
\]
Example: Taint Analysis

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

**Check \(a\)**

**Check \(z\)**

**Check \(w\)**
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

- Meta-Data Detection

- Native Application

- Instrumented Application
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

![Diagram showing native and instrumented applications with meta-data detection]

- Native Application
- Instrumented Application
- Meta-Data Detection
- Non-Shadowed Data
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

Native Application

Instrumented Application

Non-Shadowed Data

Meta-Data Detection
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

Native Application

Instrumented Application

Shadowed Data

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

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

Native Application

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

\[ \text{V} \rightarrow \text{P} \]
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
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:**
  - System: Taint Analysis
    - Slowdown (normalized): 101.7x
  - System: On-Demand Taint Analysis
    - Slowdown (normalized): 1.98x

- **Results when everything is tainted:**
  - Bar chart showing slowdown for:
    - netcat_transmit
    - netcat_receive
    - ssh_transmit
    - ssh_receive
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

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

Thread 2
mylen=large

ptr write-shared?

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
- ptr write-shared?
- 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);

Interleaved Synchronization?
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)

Interleaved Synchronization?
**SW Race Detection is Slow**

<table>
<thead>
<tr>
<th>Race Detector Slowdown (x)</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>string_match</td>
<td></td>
<td></td>
</tr>
<tr>
<td>word_count</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GeoMean</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>ferret</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fregmine</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>
<tr>
<td>GeoMean</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

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

```c
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);

Thread-local data
NO SHARING
```

```
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)

Thread-local data
NO SHARING

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

Shared data
NO INTER-THREAD SHARING EVENTS
```
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);
```
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>0</td>
<td>0.5</td>
</tr>
<tr>
<td>kmeans</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>linear_regression</td>
<td>0</td>
<td>1.5</td>
</tr>
<tr>
<td>matrix_multiply</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>pca</td>
<td>0</td>
<td>2.5</td>
</tr>
<tr>
<td>string_match</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>word_count</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>blackscholes</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>bodytrack</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>facesim</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>freqmine</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>raytrace</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>swaptions</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>fluidanimate</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>vips</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>x264</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>canneal</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>dedup</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>streamcluster</td>
<td>0</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
- Software Race Detector
- Local Access

Inter-thread Sharing Monitor
Use Demand-Driven Analysis!

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

Inter-thread Sharing Monitor

Inter-thread Sharing Monitor

Software Race Detector

Inter-thread sharing

Multithreaded Application Interthread sharing
Use Demand-Driven Analysis!

Inter-thread Sharing Monitor

Inter-thread sharing

Software Race Detector

Multi-threaded Application Software Race Detector

Mixed
Use Demand-Driven Analysis!

Inter-thread Sharing Monitor
Use Demand-Driven Analysis!

Multi-threaded Application Software Race Detector

Inter-thread Sharing Monitor
Use Demand-Driven Analysis!

Inter-thread Sharing Monitor

Local Access

Software Race Detector

Multi-threaded Application
Use Demand-Driven Analysis!

Multi-threaded Application

Local Access

Inter-thread Sharing Monitor

Software Race Detector
Finding Inter-thread Sharing

- Virtual Memory Watchpoints?
Finding Inter-thread Sharing

- Virtual Memory Watchpoints?

![Diagram showing memory sharing and fault](image)
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></td>
<td>0</td>
</tr>
<tr>
<td></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>
</tbody>
</table>
Hardware Sharing Detector

- Hardware Performance Counters

Pipeline  ->  Perf. Ctrs

Cache

1
0
0
0
Hardware Sharing Detector

- Hardware Performance Counters

![Diagram showing Pipeline and Cache with Perf. Ctrs: 2, 0, 0, 0]
Hardware Sharing Detector

- Hardware Performance Counters

```
<table>
<thead>
<tr>
<th>Pipeline</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perf. Ctrs</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></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

![Diagram showing hardware sharing detector with Pipeline and Cache blocks connected to performance counters and PEBS/Debug Store flags.](image-url)
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></td>
</tr>
<tr>
<td></td>
<td>-1</td>
<td>-</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

- **Pipeline**
  - Perf. Ctrs: 2
  - -1
  - 1
  - 0

- **Cache**
  - PEBS: Armed
  - Debug Store:

- PEBS Armed
  - -
  - -
Hardware Sharing Detector

- Hardware Performance Counters

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

- **Hardware Performance Counters**

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

- **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**
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

1. Execute Instruction
2. Analysis Enabled?
   - YES: SW Race Detection
   - NO: (Next step or decision)
3. Sharing Recently?
Demand-Driven Analysis on Real HW

Execute Instruction

Analysis Enabled?

YES

SW Race Detection

Sharing Recently?

YES
Demand-Driven Analysis on Real HW

- Execute Instruction
- Analysis Enabled?
  - YES: SW Race Detection
  - NO: Disable Analysis

- Sharing Recently?
  - YES: Analysis Enabled?
Demand-Driven Analysis on Real HW

Flowchart:

1. HITM Interrupt? (NO → Execute Instruction)
2. Execute Instruction
3. Analysis Enabled? (NO → HITM Interrupt?; YES → SW Race Detection)
4. SW Race Detection
5. Sharing Recently? (NO → NO; YES → Disable Analysis)
6. Disable Analysis (NO → YES; YES → Analysis Enabled?)

Decision Points:
- HITM Interrupt?
- Analysis Enabled?
- Sharing Recently?
Demand-Driven Analysis on Real HW

1. HITM Interrupt?
   - NO
     - NO
     - YES
     - YES
      - Enable Analysis
      - SW Race Detection
      - Sharing Recently?
      - NO
      - YES
      - Disable Analysis
      - Analysis Enabled?
      - YES
      - Execute Instruction
      - NO

   - YES
     - Enable Analysis
     - SW Race Detection
     - Sharing Recently?
     - NO
     - YES
Demand-Driven Analysis on Real HW

- Execute Instruction
- Disable Analysis
- Analysis Enabled?
- YES
- NO
- SW Race Detection

- HITM Interrupt?
- YES
- Enable Analysis
- NO

- Sharing Recently?
- NO
- YES
Demand-Driven Analysis on Real HW

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

HITM Interrupt?

NO

YES
Performance Increases

![Graph showing performance increases for different applications]

- **Phoenix**
- **PARSEC**

The graph illustrates the demand-driven analysis and speedup (x) for various applications, including GeoMean, blackscholes, bodytrack, facesim, ferret, freqmine, raytrace, swaptions, fluidanimate, vips, x264, canneal, dedup, streamcluster, and GeoMean.
Performance Increases

![Graph showing performance increases for Phoenix and PARSEC]{fig:performance_increases}

Phoenix  
PARSEC

51x
Demand-Driven Analysis Accuracy
Demand-Driven Analysis Accuracy
Demand-Driven Analysis Accuracy

![Graph showing demand-driven analysis accuracy with speedup (x) values for various benchmarks and the GeoMean.](image-url)
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](image)

- Ideal Detection Accuracy
- Overhead

- No Analysis
- Complete Analysis
Reducing Overheads Further: Sampling

- Lower overheads by skipping some analyses

![Graph showing ideal detection accuracy vs. overhead]

Ideal Detection Accuracy (%) vs. Overhead

- Overhead
- Ideal Detection Accuracy (%)

SPEED

ACCURACY
Reducing Overheads Further: Sampling

[Graph showing a linear relationship between Ideal Detection Accuracy (%) and Overhead, with a note for Developer.]
Reducing Overheads Further: Sampling

![Graph showing the relationship between ideal detection accuracy and overhead with Beta Testers and Developer icons.](Image)

<table>
<thead>
<tr>
<th>Ideal Detection Accuracy (%)</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

Beta Testers

Developer
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
Cannot Naïvely Sample Code

```
# Python code snippet
a += y
y = x * 1024
x = read_input()
```
Cannot Naïvely Sample Code

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

ON

OFF

Input

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

\[ y = x \times 1024 \]

\[ a += y \]

\[ z = y \times 75 \]

Skip Instr.
Cannot Naïvely Sample Code

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

\[
\begin{align*}
    a & += y \\
    y & = x \times 1024 \\
    w & = x + 42 \\
    x & = \text{read_input()} \\
    z & = y \times 75 \\
    \text{validate}(x) &
\end{align*}
\]
Cannot Naïvely Sample Code

\[ a += y \]
\[ z = y \times 75 \]
\[ y = x \times 1024 \]
\[ w = x + 42 \]
\[ \text{Input} \]

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

\[ \text{Check } w \]

\[ \text{Check } z \]

\[ \text{Check } a \]
Cannot Naïvely Sample Code

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

Validate(x)

Input

```
x = read_input()
```

False Positive

Check w

False Negative

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

[Image of a light switch with labels 'ON' and 'OFF', and a button labeled 'Input']
Dataflow Sampling Example

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

Input

\[ a += y \]
Dataflow Sampling Example

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

Input

Skip Dataflow
Dataflow Sampling Example

\[ a \ += \ y \]

\[ z = y \times 75 \]

\[ y = x \times 1024 \]

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

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

1. $x = \text{read\_input()}$
2. $y = x \times 1024$
3. $w = x + 42$
4. $z = y \times 75$
5. $a += y$

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

\[ a += y \]
\[ z = y \times 75 \]
\[ y = x \times 1024 \]
\[ w = x + 42 \]
\[ \text{validate}(x) \]
\[ x = \text{read_input}() \]
\[ \text{Check } w \]
\[ \text{Check } z \]
\[ \text{Check } a \]

False Negative
Dataflow Sampling

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

![Diagram of 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

**Operating System**
Dataflow Sampling

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

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

![Diagram of Sampling Analysis Tool with Native Application, Instrumented Application, and Operating System]
Dataflow Sampling

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

![Sampling Analysis Tool Diagram]

**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
Dataflow Sampling

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

**Sampling Analysis Tool**

- 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
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

- Apache
- Eggdrop
- Lynx
- ProFTPD
- Squid

Maximum % Time in Analysis

- 10%
- 25%
- 50%
- 75%
- 90%
BACKUP SLIDES
## Accuracy on Real Hardware

<table>
<thead>
<tr>
<th></th>
<th>kmeans</th>
<th>facesim</th>
<th>ferret</th>
<th>freqmine</th>
<th>vips</th>
<th>x264</th>
<th>streamcluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>W→W</td>
<td>1/1 (100%)</td>
<td>0/1 (0%)</td>
<td>-</td>
<td>-</td>
<td>1/1 (100%)</td>
<td>-</td>
<td>1/1 (100%)</td>
</tr>
<tr>
<td>R→W</td>
<td>-</td>
<td>0/1 (0%)</td>
<td>2/2 (100%)</td>
<td>2/2 (100%)</td>
<td>1/1 (100%)</td>
<td>3/3 (100%)</td>
<td>1/1 (100%)</td>
</tr>
<tr>
<td>W→R</td>
<td>-</td>
<td>2/2 (100%)</td>
<td>1/1 (100%)</td>
<td>2/2 (100%)</td>
<td>1/1 (100%)</td>
<td>3/3 (100%)</td>
<td>1/1 (100%)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Spider Monkey-0</th>
<th>Spider Monkey-1</th>
<th>Spider Monkey-2</th>
<th>NSPR-1</th>
<th>Memcached-1</th>
<th>Apache-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>W→W</td>
<td>9/9 (100%)</td>
<td>1/1 (100%)</td>
<td>1/1 (100%)</td>
<td>3/3 (100%)</td>
<td>-</td>
<td>1/1 (100%)</td>
</tr>
<tr>
<td>R→W</td>
<td>3/3 (100%)</td>
<td>-</td>
<td>1/1 (100%)</td>
<td>1/1 (100%)</td>
<td>1/1 (100%)</td>
<td>7/7 (100%)</td>
</tr>
<tr>
<td>W→R</td>
<td>8/8 (100%)</td>
<td>1/1 (100%)</td>
<td>2/2 (100%)</td>
<td>4/4 (100%)</td>
<td>-</td>
<td>2/2 (100%)</td>
</tr>
</tbody>
</table>
Width Test