Accelerating Dynamic Software Analyses

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

February 20, 2012
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
  - >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)

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)
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

len2=thread_local->mylen;
ptr=malloc(len2);

if(ptr==NULL)

memcpy(ptr, data2, len2)

LEAKED

ptr

TIME
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)

LEAKED

ptr
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

LONG run time

In-House Test Server(s)
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
- Memory Checking (e.g. MemCheck) - 5-50x
- Symbolic Execution - 10-200x
- Taint Analysis (e.g. TaintCheck) - 2-200x
- Dynamic Bounds Checking - 10-80x
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

- Data
- Meta-data

Input
Example: Taint Analysis

Input

x = read_input()
Example: Taint Analysis

```
x = read_input()
```

Diagram:
- Data
- Meta-data
- Input
- Associate
- $x = \text{read\_input()}$
Example: Taint Analysis

\[ y = x \times 1024 \]

Example: Taint Analysis

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

\[ y = x \times 1024 \]

Propagate

Data

Meta-data
Example: Taint Analysis

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

Input:

```
x = read_input()
```

Data
Meta-data

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

\[ \text{Example: Taint Analysis} \]

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

\[ \text{Clear} \]
Example: Taint Analysis

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

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

Input

Data
Meta-data
Example: Taint Analysis

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

Check \(w\)
Example: Taint Analysis

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

Input

validate(x)

Check w

Check a

Check z

Data

Meta-data

Example: Taint Analysis
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

Meta-Data Detection
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

![Diagram showing Native Application, Instrumented Application, Meta-Data Detection, and Non-Shadowed Data.]
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

Native Application

Instrumented Application

Meta-Data Detection

Shadowed Data

- Only Analyze Shadowed Data

Native Application

Instrumented Application

Meta-Data Detection

Shadowed Data
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

![Diagram showing Native Application, Instrumented Application, and 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

Meta-Data Detection
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

[Diagram showing native application connected to instrumented application with 'No meta-data' label]

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

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
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:**

```
netcat_transmit  | 150x
netcat_receive   | 10x
ssh_transmit     | 100x
ssh_receive      | 50x
```
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)
Thread 1
mylen=small
if(ptr==NULL)
len1=thread_local->mylen;
ptr=malloc(len1);
mempcpy(ptr, data1, len1)

Thread 2
mylen=large
if(ptr==NULL)
len2=thread_local->mylen;
ptr=malloc(len2);
mempcpy(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)
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->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)

Thread 2
mylen=large

ptr write-shared?

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)

len2=thread_local->mylen;

ptr=malloc(len2);

memcpy(ptr, data2, len2)

Thread 2
mylen=large

Interleaved Synchronization?

if(ptr==NULL)
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>Phoenix</th>
<th>PARSEC</th>
<th>Race Detector Slowdown (x)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>histogram</td>
</tr>
<tr>
<td></td>
<td></td>
<td>kmeans</td>
</tr>
<tr>
<td></td>
<td></td>
<td>matrix_multiply</td>
</tr>
<tr>
<td></td>
<td></td>
<td>string_match</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pca</td>
</tr>
<tr>
<td></td>
<td></td>
<td>word_count</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GeoMean</td>
</tr>
<tr>
<td></td>
<td></td>
<td>blackscholes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bodytrack</td>
</tr>
<tr>
<td></td>
<td></td>
<td>facesim</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ferret</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fgmime</td>
</tr>
<tr>
<td></td>
<td></td>
<td>raytrace</td>
</tr>
<tr>
<td></td>
<td></td>
<td>swaptions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fluidanimate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>vips</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x264</td>
</tr>
<tr>
<td></td>
<td></td>
<td>canneal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dedup</td>
</tr>
<tr>
<td></td>
<td></td>
<td>streamcluster</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GeoMean</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

```
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);
```
“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);
```

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);
```

```
Thread-local data
NO SHARING
```

```
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)
  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)
```
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
Very Little Inter-Thread Sharing

% Write-Sharing Events

<table>
<thead>
<tr>
<th>Phoenix</th>
<th>PARSEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>histogram</td>
<td>0.5</td>
</tr>
<tr>
<td>kmeans</td>
<td>1</td>
</tr>
<tr>
<td>linear_regression</td>
<td>1.5</td>
</tr>
<tr>
<td>matrix_multiply</td>
<td>2</td>
</tr>
<tr>
<td>pca</td>
<td>2.5</td>
</tr>
<tr>
<td>string_match</td>
<td>3</td>
</tr>
<tr>
<td>word_count</td>
<td></td>
</tr>
<tr>
<td>blackscholes</td>
<td></td>
</tr>
<tr>
<td>bodytrack</td>
<td></td>
</tr>
<tr>
<td>facesim</td>
<td></td>
</tr>
<tr>
<td>freqmine</td>
<td></td>
</tr>
<tr>
<td>raytrace</td>
<td></td>
</tr>
<tr>
<td>swaptions</td>
<td></td>
</tr>
<tr>
<td>fluidanimate</td>
<td></td>
</tr>
<tr>
<td>vips</td>
<td></td>
</tr>
<tr>
<td>x264</td>
<td></td>
</tr>
<tr>
<td>canneal</td>
<td></td>
</tr>
<tr>
<td>dedup</td>
<td></td>
</tr>
<tr>
<td>streamcluster</td>
<td></td>
</tr>
</tbody>
</table>

67
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

Multithreaded Application

Inter-thread Sharing Monitor

Software Race Detector
Use Demand-Driven Analysis!
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

Inter-thread Sharing Monitor
Use Demand-Driven Analysis!

Inter-thread Sharing Monitor

Local Access

Software Race Detector

Multi-threaded Application

ON
OFF
Finding Inter-thread Sharing

- Virtual Memory Watchpoints?
Finding Inter-thread Sharing

- Virtual Memory Watchpoints?

![Diagram](image-url)
Finding Inter-thread Sharing

- Virtual Memory Watchpoints?
Finding Inter-thread Sharing

- Virtual Memory Watchpoints?
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

Cache

Perf. Ctrs
1
0
0
0
```
Hardware Sharing Detector

- Hardware Performance Counters

Pipeline

Cache

Perf. Ctrs

2
0
0
0
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>
<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></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>
<th>Perf. Ctrs</th>
<th>PEBS</th>
<th>Debug Store</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>-1</td>
<td>FAULT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></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>-</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></td>
</tr>
<tr>
<td></td>
<td>-1</td>
<td>Armed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

PEBS: Armed
Hardware Sharing Detector

- Hardware Performance Counters

![Diagram showing Hardware Sharing Detector with Pipeline, Cache, Perf. Ctrs, PEBS, and Debug Store]

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

- Hardware Performance Counters

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

- **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
3. SW Race Detection
4. 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

NO

YES
Demand-Driven Analysis on Real HW

1. HITM Interrupt?
   - NO
   - YES

2. Execute Instruction
   - NO
   - YES

3. Analysis Enabled?
   - NO
   - YES

4. SW Race Detection
   - NO
   - YES

5. Sharing Recently?
   - NO
   - YES

6. Disable Analysis
   - NO
   - YES
Demand-Driven Analysis on Real HW

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

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

- Execute Instruction
- SW Race Detection
- HITM Interrupt?
  - NO
  - > 97%
- Analysis Enabled?
  - NO
  - YES
- Enable Analysis
- Disable Analysis
- Sharing Recently?
  - NO
  - YES
On-Demand Analysis on Real HW

- Execute Instruction

- Analysis Enabled?
  - YES
  - NO

- HITM Interrupt?
  - YES
  - NO

- Enable Analysis

- SW Race Detection

- Sharing Recently?
  - YES
  - NO

- Disable Analysis

- > 97%

- < 3%
Performance Increases

The diagram compares the demand-driven analysis speedup of Phoenix and PARSEC. The x-axis represents the demand-driven analysis, while the y-axis shows the speedup (x). The bars vary in height, indicating different speedup values for each analysis.

Notable comparisons include:
- Phoenix shows a significantly higher speedup for programs like "raytrace" and "fluidanimate".
- PARSEC generally has lower speedups across most applications.

The GeoMean calculation provides a single value that represents the overall performance increase for both tools, summarizing the data.

Programs such as "histogram", "kmeans", "linear_regression", "matrix_multiply", "string_match", "word_count", "blackshole", "bodytrack", "fatesim", "ferret", "freqmine", "raytrace", "swapions", "vips", "x264", "canneal", "dedup", "streamcluster", and "GeoMean" are analyzed in the diagram.
Performance Increases

Phoenix | PARSEC

Speedup (x)

Demand-driven Analysis

histogram, kmeans, matrix_multiply, string_match, word_count, GeoMean, blackscholes, bodytrack, facesim, ferret, freqmne, raytrace, swapions, fluidanimate, vips, x264, canmean, dedup, streamcluster, GeoMean
Demand-Driven Analysis Accuracy

Demand-driven Analysis Speedup (x)

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

[Graph showing speedup (x) for various benchmarks such as histogram, kmeans, matrix_multiply, pca, word_count, GeoMean, blackscholes, bodytrack, facesim, ferret, raytrace, swaptions, fluidanimate, vips, x264, canneal, dedup, streamcluster, GeoMean.]

- histogram: 1/1
- kmeans: 2/4
- matrix_multiply: 3/3
- pca: 4/4
- word_count: 3/3
- GeoMean: 4/4
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

**Ideal Detection Accuracy (%)**

- **Overhead**
  - No Analysis
  - Complete Analysis

Graph shows a linear relationship between Overhead and Detection Accuracy.
Reducing Overheads Further: Sampling

- Lower overheads by skipping some analyses

![Graph showing the relationship between Ideal Detection Accuracy (%) and Overhead]

---

117
Reducing Overheads Further: Sampling

![Graph showing the relationship between Overhead and Ideal Detection Accuracy (percentage)]
Reducing Overheads Further: Sampling

![Diagram showing the relationship between ideal detection accuracy and overhead, with Beta Testers and Developer icons.]
Sampling Allows Distribution

<table>
<thead>
<tr>
<th>Ideal Detection Accuracy (%)</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Developer</td>
</tr>
<tr>
<td>75</td>
<td>Beta Testers</td>
</tr>
<tr>
<td>50</td>
<td>Beta Testers</td>
</tr>
<tr>
<td>25</td>
<td>End Users</td>
</tr>
<tr>
<td>0</td>
<td>End Users</td>
</tr>
</tbody>
</table>
Many users testing at little overhead see more errors than one user at high overhead.
Cannot Naïvely Sample Code
Cannot Naïvely Sample Code

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

```
a += y
z = y * 75
y = x * 1024
a += y
```
Cannot Naïvely Sample Code

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

```
x = read_input()
```

```
Skip Instr.
```

Diagram: Flowchart showing the execution of code with input and validation steps.
Cannot Naïvely Sample Code

Input

x = read_input()

validate(x)

y = x * 1024

w = x + 42

a += y

z = y * 75
Cannot Naïvely Sample Code

\[
a += y
\]

\[
z = y \times 75
\]

\[
y = x \times 1024
\]

\[
w = x + 42
\]

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

\[
\text{Check } w
\]

\[
\text{Check } z
\]

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

Input

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

\[ y = x \times 1024 \]

\[ w = x + 42 \]

\[ z = y \times 75 \]

\[ a += y \]

\[ a \] (Check a)

\[ z \] (Check z)

False Positive

False Negative

Validate(x)

x = read_input()
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
Dataflow Sampling Example

\[ a += y \]

\[ y = x \times 1024 \]

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

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

Input

Skip Dataflow
Dataflow Sampling Example

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

```
x = read_input()
```

```python
Dataflow Sampling Example
```
Dataflow Sampling Example

\[ a += y \]

\[ z = y \times 75 \]

\[ y = x \times 1024 \]

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

validate\( (x) \)
Dataflow Sampling Example

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

Input

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

validate(x)

\[ y = x * 1024 \]

\[ a += y \]

\[ z = y * 75 \]

\[ w = x + 42 \]
Dataflow Sampling Example

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

x = read_input()
validate(x)
```

Input

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

![Sampling Analysis Tool Diagram]

- Native Application
- Instrumented Application
- Operating System
Dataflow Sampling

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

![Sampling Analysis Tool Diagram]

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**
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
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

![Graph showing throughput vs. maximum % time in analysis for ssh_receive. The throughput decreases as the maximum % time in analysis increases. There is a dashed line representing throughput with no analysis.](image-url)
Accuracy with Background Tasks

ssh_receive running in background

% Chance of Detecting Exploit

Maximum % Time in Analysis

- Apache
- Eggdrop
- Lynx
- ProFTPD
- Squid
Accuracy with Background Tasks

*ssh_receive* running in background

**Graph Description:**
- **Y-axis:** % Chance of Detecting Exploit
- **X-axis:** Maximum % Time in Analysis
- **Categories:**
  - Apache
  - Eggdrop
  - Lynx
  - ProFTPD
  - Squid

**Data Points:**
- **10%:** 0.1
- **25%:** 0.7
- **50%:** 2
- **75%:**
  - Apache: 40
  - Eggdrop: 80
  - Lynx: 60
  - ProFTPD: 50
  - Squid: 100
- **90%:**
  - Apache: 60
  - Eggdrop: 80
  - Lynx: 80
  - ProFTPD: 70
  - Squid: 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