The Potential of Sampling for Dynamic Analysis

Joseph L. Greathouse  Todd Austin

Advanced Computer Architecture Laboratory
University of Michigan
Dynamic Security Analysis

- Finds flaws in programs as they run
  - Makes programs more robust
- Insert checks around instructions

```c
y = x->data;
*w += y;
z = 75/y;
```
Dynamic Security Analysis

- Finds flaws in programs as they run
  - Makes programs more robust
- Insert checks around instructions

```c
check(x!=NULL);
y = x->data;
check(w!=NULL);
*w += y;
check(y!=0);
z = 75/y;
```
Dynamic Information Flow Tracking

1. Inputs are untrusted
2. Propagate untrusted status while executing
3. Check trustedness for safety
Dynamic Information Flow Tracking

[Diagram showing 'Data', 'Untrusted', and 'Input']
Dynamic Information Flow Tracking

Input

\( x = \text{read\_input()} \)
Dynamic Information Flow Tracking

\[ x = \text{read}_\text{input}() \]
Dynamic Information Flow Tracking

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

Data
Untrusted
Dynamic Information Flow Tracking

Input

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

\[ y = x \times 1024 \]

\[ z = y \times 75 \]
Dynamic Information Flow Tracking

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

Input

validate(x)

\[ y = x \times 1024 \]

\[ z = y \times 75 \]
Dynamic Information Flow Tracking

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

\[
y = x \times 1024
\]

\[
z = y \times 75
\]

\[
w = x + 42
\]

\[
\text{validate}(x)
\]
Dynamic Information Flow Tracking

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

Input

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

validate(x)

\[ y = x \times 1024 \]

\[ z = y \times 75 \]

Check z

Check w
Dynamic Information Flow Tracking

Data
Untrusted

Input

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

validate(x)

Only works on current dynamic control path

\[z = y \times 75\]

Check z

Check w

\[w = x + 42\]
Problem: Dynamic Analyses are Slow

Assertion Checking
- DIFT: 150x
- 2x

Race Detection
- Symbolic Execution: 200x
- 100x

Atomicity Checking
- 400x
Dynamic Analysis Sampling

- Lower overheads by skipping some analyses

Graph:
- Ideal Detection Accuracy (%) vs. Overhead
- No Analysis to Complete Analysis
- Overhead increases, Ideal Detection Accuracy (%) increases linearly
Dynamic Analysis Sampling

- Lower overheads by skipping some analyses

![Graph showing ideal detection accuracy versus overhead. The graph is a straight line indicating an increase in detection accuracy with increasing overhead.]
Sampling Allows Distribution

Ideal Detection Accuracy (%)

Overhead

Developer
Sampling Allows Distribution

- Ideal Detection Accuracy (%)
- Overhead

Beta Testers

Developer
Sampling Allows Distribution

<table>
<thead>
<tr>
<th>Developer</th>
<th>End Users</th>
<th>Beta Testers</th>
</tr>
</thead>
</table>

Ideal Detection Accuracy (%)

Overhead

0 25 50 75 100

0 25 50 75 100

End Users Beta Testers
Sampling Allows Distribution

Many users testing at little overhead see more errors than one user at high overhead.
Sampling Assertion Checking

- Perform a random subset of checks

Static Code

```c
check(x!=NULL);

y = x->data;
check(w!=NULL);

*w += y;
check(y!=0);

z = 75/y;
```
Sampling Assertion Checking

- Perform a random subset of checks

```c
check(x!=NULL);

y = x->data;

check(w!=NULL);

*w += y;

check(y!=0);

z = 75/y;
```
## Sampling Assertion Checking

- Perform a random subset of checks

### Dynamic #1

```
check(x!=NULL);
y = x->data;
*w += y;
z = 75/y;
```

### Dynamic #2

```
y = x->data;
check(w!=NULL);
*w += y;
z = 75/y;
```

### Dynamic #3

```
y = x->data;
check(y!=0);
*w += y;
check(w!=NULL);
z = 75/y;
```
**Sampling Assertion Checking**

- Perform a random subset of checks

<table>
<thead>
<tr>
<th>Dynamic #1</th>
<th>Dynamic #2</th>
<th>Dynamic #3</th>
</tr>
</thead>
<tbody>
<tr>
<td>check(x!=NULL);</td>
<td>check(w!=NULL);</td>
<td>check(y!=0);</td>
</tr>
<tr>
<td>y = x-&gt;data;</td>
<td>y = x-&gt;data;</td>
<td>y = x-&gt;data;</td>
</tr>
<tr>
<td>*w += y;</td>
<td>*w += y;</td>
<td>*w += y;</td>
</tr>
<tr>
<td>z = 75/y;</td>
<td>z = 75/y;</td>
<td>z = 75/y;</td>
</tr>
</tbody>
</table>

- 1/1000 checks: ~3x slowdown $\rightarrow$ 30% overhead
Sampling DIFT

- Must sample dataflows instead of instructions
Sampling DIFT

- Must sample dataflows instead of instructions
Sampling DIFT

- Must sample dataflows instead of instructions
- 0.1-10% of faults: 10% overhead
Advanced Sampling Techniques

- Cold region Hypothesis
  - Sample “cold” code at higher rate

- New atomicity violation detection
  - Trace atomic regions and correlate crashes
Future Research Directions

- Sampling for more dynamic analyses
- New types of analyses because of sampling
- Studies on users slowdown acceptance
Needed Engineering Efforts

- Push-button sampling mechanisms for Valgrind, DynamoRIO, Pin, etc.

- Libraries to integrate bug reporting into analysis tools

- Back-end infrastructure for handling distributed, sampled bug reports