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 (0xFD3094C2,0x00000001,0xFBBE7617,0x00000000)

*** SPCMDCON.SYS - Address FBBE7617 base at FBBE5000, DateStamp 3d6dd67c
Software Errors Abound

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

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

```
A problem has been detected and Windows has been shut down to prevent damage to your computer.
The problem seems to be caused by the following file: $POMDCON.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:
```
```
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.

![CERT-Cataloged Vulnerabilities](image)
Security Vulnerability Example

- Buffer overflows a large class of security vulnerabilities

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

- Buffer overflows a large class of security vulnerabilities

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

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

- Buffer overflows a large class of security vulnerabilities

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

If `read_input()` reads >256 ints
Concurrency Bugs Also Matter

Thread 1
mylen=small

Thread 2
mylen=large

if(ptr == NULL) {
    len=thread_local->mylen;
    ptr=malloc(len);
    memcpy(ptr, data, len);
}
Concurrency Bugs Matter **NOW**

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

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

\[
\text{len2}=\text{thread}\_\text{local}->\text{mylen};
\]
\[
\text{ptr}=\text{malloc}(\text{len2});
\]
\[
\text{memcpy}(\text{ptr}, \text{data2}, \text{len2})
\]

\[\text{ptr} \emptyset\]
**Concurrency Bugs Matter NOW**

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
∅
Concurrency Bugs Matter **NOW**

**Thread 1**
- mylen = small

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

**Thread 2**
- mylen = large

```c
if(ptr == NULL) {
    len2 = thread_local->mylen;
    ptr = malloc(len2);
}
```

Here `len2` is allocated and `ptr` is initialized with the `len2` value. This setup illustrates the concurrency bug where `len2` is not thread-local, leading to race conditions or data corruption.
if(ptr==NULL)
    ptr = malloc(len2);
len2 = thread_local->mylen;
memcpy(ptr, data2, len2)
Concurrency Bugs Matter NOW

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
Concurrency Bugs Matter **NOW**

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
Concurrency Bugs Matter **NOW**

### Thread 1
- mylen = small

```c
if(ptr == NULL)
    memcpy(ptr, data2, len2)
```

### Thread 2
- mylen = large

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

---

**TIME**

`len1 = thread_local->mylen;`

`ptr = malloc(len1);`

`memcpy(ptr, data1, len1)`

---

**ptr**

LEAKED
Concurrency Bugs Matter **NOW**

Thread 1
mylen=small

```c
if(ptr==NULL)
    memcpy(ptr, data2, len2)
```

Thread 2
mylen=large

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

**TIME**

LEAKED
One Layer of a Solution

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

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

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

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

- Data
- Meta-data

Input
Example Dynamic Dataflow Analysis

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

```
x = read_input()
```

[Diagram showing data flow and associations]
Example Dynamic Dataflow Analysis

\$y = x \times 1024\$

\$x = \text{read}\_\text{input}()\$

\$y = x \times 1024\$

Data

Meta-data

Propagate
Example Dynamic Dataflow Analysis

- Data
- Meta-data

Input

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

\[ y = x \times 1024 \]

\[ a += y \]

\[ z = y \times 75 \]
Example Dynamic Dataflow Analysis

Data
Meta-data

Input

x = read_input()

validate(x)

Clear

y = x * 1024

a += y

z = y * 75
Example Dynamic Dataflow Analysis

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

Input

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

validate(x)

\[ a += y \]
\[ z = y * 75 \]
\[ y = x * 1024 \]
\[ w = x + 42 \]
Example Dynamic Dataflow Analysis

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

Input

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

\[ y = x \times 1024 \]

\[ a += y \]

\[ z = y \times 75 \]

validate(x)

\[ w = x + 42 \]

Check \( w \)

Check \( z \)

Check \( a \)
Distributed Dynamic Dataflow Analysis

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

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

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

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

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

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

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

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

- Lower overheads by skipping some analyses

![Graph showing ideal detection accuracy (%) vs. overhead. The graph depicts a linear relationship where lower overheads correspond to lower detection accuracy, and higher overheads correspond to higher detection accuracy. The graph includes labels for 'No Analysis' and 'Complete Analysis' at the extremes of the overhead axis.]
Our Solution: Sampling

- Lower overheads by skipping some analyses
Sampling Allows Distribution

Ideal Detection Accuracy (%) vs Overhead

Developer
Sampling Allows Distribution

![Graph showing ideal detection accuracy against overhead with Beta Testers and Developer nodes.](image-url)
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
Cannot Naïvely Sample Code

```
a += y
y = x * 1024
x = read_input()
a += y
```
Cannot Naïvely Sample Code

Input

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

\[ y = x \times 1024 \]

\[ a \text{ += } y \]

\[ z = y \times 75 \]

Skip Instr.
Cannot Naïvely Sample Code

\[
a += y
\]

\[
z = y \times 75
\]

\[
y = x \times 1024
\]

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

\[
\text{Input}
\]

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

\[
\text{Skip Instr.}
\]

\[
a += y
\]

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

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

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

```
Input
```

```
Output
```

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

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

```
Input
```

```
Output
```

```
```
Cannot Naïvely Sample Code

Input

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

\[ y = x \times 1024 \]

\[ w = x + 42 \]

\[ z = y \times 75 \]

\[ a += y \]

\[ \text{check } a \]

\[ \text{check } z \]

\[ \text{check } w \]

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

\[ x = \text{read\_input}() \]
Cannot Naïvely Sample Code

\[
\begin{align*}
  a & \leftarrow y \\
  z & \leftarrow y \times 75 \\
  y & \leftarrow x \times 1024 \\
  w & \leftarrow x + 42
\end{align*}
\]

\begin{align*}
  x & = \text{read\_input()} \\
  \text{validate}(x) \\
  \text{Check } w
\end{align*}

Check \( a \)

Check \( z \)

False Positive

False Negative
Our Solution: Sample **Data**, not Code

- Sampling must be aware of meta-data

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

- Sampling must be aware of meta-data

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

Input
Dataflow Sampling Example

\[ a += y \]

\[ y = x \times 1024 \]

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

Input

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

\[ y = x \times 1024 \]

\[ a \text{ } += \text{ } y \]

\[ z = y \times 75 \]

Skip Dataflow
Dataflow Sampling Example

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

\[ \text{Input} \]

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

\[ y = x \times 1024 \]

\[ a += y \]

\[ z = y \times 75 \]

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

\[
\begin{align*}
    a & += y \\
    z & = y \times 75 \\
    y & = x \times 1024 \\
    w & = x + 42 \\
\end{align*}
\]

\[
\begin{align*}
    x & = \text{read}_\text{input}() \\
    \text{validate}(x) & \\
\end{align*}
\]
Dataflow Sampling Example

Input

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

\[ y = x \times 1024 \]

\[ z = y \times 75 \]

\[ a += y \]

\[ w = x + 42 \]

Check \( w \)

Check \( z \)

Check \( a \)
Dataflow Sampling Example

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

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

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

Check \( w \)

False Negative
Mechanisms for Dataflow Sampling (1)

- Start with demand analysis

---

**Demand Analysis Tool**

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

---

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

- Start with demand analysis

Demand Analysis Tool

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

Operating System
Mechanisms for Dataflow Sampling (1)

- Start with demand analysis

**Demand Analysis Tool**

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

**Meta-data**
Mechanisms for Dataflow Sampling (1)

- Start with demand analysis
Mechanisms for Dataflow Sampling (1)

- Start with demand analysis

**Demand Analysis Tool**

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

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

- Start with demand analysis

Demand Analysis Tool

- Native Application
- Instrumented Application (e.g., Valgrind)
- No meta-data

Operating System
Mechanisms for Dataflow Sampling (1)

- Start with demand analysis

**Demand Analysis Tool**

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

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

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

![Diagram of Sampling Analysis Tool with Native Application and Instrumented Application connected to Operating System]
Mechanisms for Dataflow Sampling (2)

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

![Sampling Analysis Tool](image)

Sampling Analysis Tool

- Native Application
- Instrumented Application

Operating System
Mechanisms for Dataflow Sampling (2)

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

![Diagram showing Sampling Analysis Tool, Native Application, Instrumented Application, and Meta-data]

**Sampling Analysis Tool**

- Native Application
- Instrumented Application

**Meta-data**
Mechanisms for Dataflow Sampling (2)

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

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

![Sampling Analysis Tool Diagram]

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

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

Sampling Analysis Tool

- Native Application
- Instrumented Application

Operating System
Mechanisms for Dataflow Sampling (2)

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

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

![Sampling Analysis Tool Diagram]

- **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 (1)

ssh_receive

Throughput (MB/s)

Maximum % Time in Analysis

Throughput with no analysis
Performance of Dataflow Sampling (2)

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

Throughput with no analysis is shown in the inset box.

The graph shows a significant decrease in throughput as the maximum percentage of time spent in analysis increases.
Performance of Dataflow Sampling (3)

Throughput (MB/s) vs. Maximum % Time in Analysis for ssh\_transmit.

- Throughput with no analysis.

The graph shows a decreasing trend in throughput as the maximum percentage of time spent in analysis increases.
Accuracy at Very Low Overhead

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

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

netcat_receive running with benchmark

% Chance of Detecting Exploit

Maximum Allowed Overhead %

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

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

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

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

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

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