Decoupling Dynamic Information Flow Tracking with a Dedicated Coprocessor

Hari Kannan, Michael Dalton, Christos Kozyrakis

Presenter: Yue Zheng
Yulin Shi
Outline

• Motivation & Background
• Hardware DIFT overview
• DIFT Coprocessor Design
• Prototype System
• Evaluation
• Conclusion
• Discussion points
Outline

• Motivation & Background
• Hardware DIFT overview
• DIFT Coprocessor Design
• Prototype System
• Evaluation
• Conclusion
• Discussion points
Motivation & Background

• Dynamic Information Flow Tracking (DIFT)
  • Tag data from untrusted source
  • Track tainted data propagation
  • Check unsafe tainted data usage

Vulnerable C Code

```
int idx = tainted_input;
buffer[idx] = x; // memory corruption
```

Why is the pointer dereference potentially dangerous?

Tainted Pointer Dereference

```
TRAP
```

Why is the pointer dereference potentially dangerous?
Why is the pointer dereference potentially dangerous?

- Buffer Overflow Attack

Vulnerable C Code

```c
char buf[126];
strcpy(buf,str);
```

Stack grows this way

![Stack diagram showing buffer overflow](image-url)
Motivation & Background

• Software DIFT
  
  • Use Dynamic Binary Translation (DBT) to implement DIFT
    • Avoid recompilation
    • Introduce **significant overheads**

• Limitation
  • **Incompatible** with self-modifying and multithreaded programs
Outline

• Motivation & Background
• Hardware DIFT overview
• DIFT Coprocessor Design
• Prototype System
• Evaluation
• Conclusion
• Discussion points
Hardware DIFT overview

• In-core DIFT

• Minimize runtime overhead
• Require significant modifications to processor structure
Hardware DIFT overview

• Offloading DIFT

- Offer the flexibility of analysis in software
- Introduce significant overheads
  - halve the throughput, double the power consumption
- Require pipeline changes
Hardware DIFT overview

• Off-core DIFT

- DIFT synchronizes with main core only on system calls
- Eliminate the changes to processor
- Allow pairing with multiple processor designs
Outline

• Motivation & Background
• Hardware DIFT overview
• DIFT Coprocessor Design
• Prototype System
• Evaluation
• Conclusion
• Discussion points
DIFT Coprocessor Design – Security Model

• Security Model

```
R1       &tainted_input
load R2   M[R1]
add R4    R2 + R3
store M[R4] R5
```

Tainted Pointer Dereference

System Call

TRAP
DIFT Coprocessor Design - Architecture

Main Core Processor VS Coprocessor

Main Core Processor

- Handle data
- 32 bits
- Complexity

Coprocessor

- Handle tag propagation and check
- 4 bits
- Simple
DIFT Coprocessor Design - Architecture

- Four-stage pipeline: Decode, Execution, TagCheck, WriteBack
DIFT Coprocessor Design - Interface

- **Coproces sor Setup**
  - Software control security policies

- **Instruction Flow Information**
  - PC, Instruction Encoding and Memory Address

- **Decoupling**
  - Greater or equal processing rate to avoid full queue

- **Security Exceptions**
  - Asynchronous interrupts and run in trusted mode
Outline

• Motivation & Background
• Hardware DIFT overview
• DIFT Coprocessor Design
• Prototype System
• Evaluation
• Conclusion
• Discussion points
Prototype

• Hardware
  • SPARC V8 Processor
  • FPGA Board

• Software
  • Gentoo Linux 2.6

• Design Statistics
  • 7.64% area overhead
Outline

• Motivation & Background
• Hardware DIFT overview
• DIFT Coprocessor Architecture
• Prototype System
• Evaluation
• Conclusion
• Discussion points
## Evaluation

### Security Evaluation

<table>
<thead>
<tr>
<th>Program</th>
<th>Language</th>
<th>Attack</th>
<th>Detected Vulnerability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gzip</td>
<td>C</td>
<td>Directory traversal</td>
<td>Open tainted directory</td>
</tr>
<tr>
<td>Tar</td>
<td>C</td>
<td>Directory traversal</td>
<td>Open tainted directory</td>
</tr>
<tr>
<td>Scry</td>
<td>PHP</td>
<td>Cross-site scripting</td>
<td>Tainted <code>&lt;script&gt;</code> tag</td>
</tr>
<tr>
<td>Htdig</td>
<td>C++</td>
<td>Cross-site scripting</td>
<td>Tainted <code>&lt;script&gt;</code> tag</td>
</tr>
<tr>
<td>Polymorph</td>
<td>C</td>
<td>Buffer overflow</td>
<td>Tainted code ptr dereference</td>
</tr>
<tr>
<td>Sendmail</td>
<td>C</td>
<td>Buffer overflow</td>
<td>Tainted data ptr dereference</td>
</tr>
<tr>
<td>Quotactl syscall</td>
<td>C</td>
<td>User/kernel pointer dereference</td>
<td>Tainted pointer to kernelspace</td>
</tr>
<tr>
<td>SUS</td>
<td>C</td>
<td>Format string bug</td>
<td>Tainted ‘%n’ in syslog</td>
</tr>
<tr>
<td>WU_FTPD</td>
<td>C</td>
<td>Format string bug</td>
<td>Tainted ‘%n’ in vfprintf</td>
</tr>
</tbody>
</table>

- Independent of programming language
- Wide workload range

- High level attacks
- Low level attacks

- Common Utilities
- Servers
- Web apps
- Kernel code
Evaluation

• Performance Evaluation – Execution time

- 512-byte tag cache
- 6-entry queue

Runtime overhead < 1%
Evaluation

• Performance Evaluation – Scaling the tag cache
Evaluation

• Performance Evaluation – Scaling the decoupling queue

![Graph showing runtime overhead vs queue size]

- 16-byte tag cache
- Tag miss ↑
Evaluation

• Processor/Coprocessor Performance Ratio

![Graph showing relative overhead vs ratio of main core's clock to coprocessor's clock.]

- $3.8\%$ for gzip
- $11.7\%$ for gcc
- $11.7\%$ for twolf

- 16-entry queue

Can be paired with various main cores
Outline

• Motivation & Background
• Hardware DIFT overview
• DIFT Coprocessor Architecture
• Prototype System
• Evaluation
• Conclusion
• Discussion points
Conclusion

• DIFT: a promising security technique

• Proposed an off-core, decoupling coprocessor for DIFT
  • Provide the same security features as in-core DIFT
  • Reduce DIFT implementation cost drastically
  • Has low area and performance overheads

• Developed a full-system prototype
  • Protect real-world Linux applications
Questions?
Debate

• Is a wider-issue coprocessor better than a single-issue coprocessor for 3-way superscalar processors?

• Is it worth to add a checkpoint scheme to DIFT to provide reliable recovery? (A checkpoint scheme allows the system to rollback for recovery)