RIFLE: An Architectural Framework for User-Centric Information-Flow Security

Neil Vachharajani · Matthew J. Bridges
Jonathan Chang · Ram Rangan
Guilherme Ottoni · Jason A. Blome · George A. Reis
Manish Vachharajani · David I. August

Liberty Research Group
Princeton University
Information-Flow Security in the Real World

IRS provides tax forms

IRS.gov

Financial Info Barrier

TaxPrep, Inc. provides software patches

TaxPrep.com

Alice enters her financial information

Alice

Tax Prep Software
Information-Flow Security in the Real World

IRS provides tax forms

IRS.gov

TaxPrep, Inc. provides software patches

TaxPrep.com

Alice enters her financial information

Financial Info Barrier

Alice

Tax Prep Software
Information-Flow Security in the Real World

All programs must be assumed unsafe
- **Malicious** programs intentionally leak information
- **Buggy** programs that unintentionally leak information

User-Centric Information-Flow Security

1. Users want to establish their *own* security policy
   - CIA's security needs differ from Joe Average's

2. Users want data-dependent security policies
   - Web browser with web search form data
   - Web browser with banking login form data

3. Users should not have to sacrifice security for functionality
   - All programs should be secure or securable
   - Only security holes that will be realized are significant
Definition of Security: Non-Interference

Integrity

- Untrusted inputs should not affect trusted outputs
- Example: prevent input from being executed [Suh 04, Crandall 04]

Confidentiality [Denning 76, Myers 97, Myers 99, Tse 04]

- High security inputs should not affect low security outputs
- Example: tax preparation software

Key mechanism: tracking flow of information through code

- Integrity/confidentiality are dual
- Policies and enforcement rely on information flow
Information-Flow Security: Tainting Data

- Used in Perl’s “taint” mode and other works
  [Denning 76, Suh 04, Crandall 04]

\[
\begin{align*}
\text{add } r4 &= r1, r2 \\
\text{add } r5 &= r4, r3 \\
\text{div } r6 &= r5, 3 \\
\text{sc } .\text{write}, r6
\end{align*}
\]

1. Program inputs are **tainted** or **labeled** with a security class
2. Labels propagate through computation
3. Certain operations enforce a security policy by verifying operand labels for security
Problems with the Taint Solution

- Control Flow Can Leak Information!

```
mov r2=0
bnez r1, L1
L1: mov r2=1
sc .write, r2
```
# User-Centric Information-Flow Security

## Essential for User-Centric IFS

- Deal breaker for User-Centric IFS
- Fundamentally Impossible

<table>
<thead>
<tr>
<th>Method</th>
<th>Leaks Information</th>
<th>Policy Enforcement</th>
<th>Programmer Support</th>
<th>Conservative</th>
<th>Performance Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Taint [Suh 2004]</td>
<td>✗</td>
<td>Dynamic</td>
<td>No</td>
<td>No</td>
<td>Moderate</td>
</tr>
<tr>
<td>Static Systems [Denning 76, Myers 97, Myers 99]</td>
<td>Rate Limited</td>
<td>Static</td>
<td>Yes</td>
<td>Yes</td>
<td>None</td>
</tr>
<tr>
<td>Static with Runtime Principles [Tse 2004]</td>
<td>Rate Limited</td>
<td>Hybrid</td>
<td>✗</td>
<td>Yes</td>
<td>Little</td>
</tr>
<tr>
<td>Ideal User-Centric</td>
<td>✗</td>
<td>Dynamic/Hybrid</td>
<td>No</td>
<td>✗</td>
<td>None</td>
</tr>
<tr>
<td>RIFLE</td>
<td>Rate Limited</td>
<td>Dynamic</td>
<td>No</td>
<td>Yes</td>
<td>Moderate</td>
</tr>
</tbody>
</table>

- Rate Limited
- Dynamic
- No
- Yes
- Moderate
- None
- Little
- Ideal
Naïve “Solution”: Taint the Program Counter

- Ops have *implicit* PC operand
- Label PC like other operands
- PC should be declassified after branch merge

Code can leak information whether it is executed or not!
RIFLE: The Big Picture

Compiler

Unannotated Source Code

Programmer’s System

Compiler

Binary Translator

Compiled Code

Base ISA

Runtime Environment (policy enforcer)

User’s System

End User

Compiled Code

Secure ISA
Naïve Binary Translation

mov r2=0, PC

bnez PC=r1, L1

L1: mov r2=1, PC

PC=declassify PC
sc .write, r2, PC
Naïve Binary Translation

1. Force every if to have an else

```
mov r2=0, PC

bnez PC=r1, L1
```

```
L1: mov r2=1, PC
```

```
PC=declassify PC
sc .write, r2, PC
```
Naïve Binary Translation

1. Force every if to have an else
2. On each side of the branch, modify same variables

\[
\text{mov } r2=0, PC
\]

\[
\text{bnez } PC=r1, L1
\]

\[
\text{mov } r2=r2, PC
\]

\[
\text{L1:mov } r2=1, PC
\]

\[
\text{PC=declassify } PC
\]

\[
\text{sc .write, r2, PC}
\]
Naïve Binary Translation

- But, what about memory?

```
mov r2=&x, PC
bnez PC=r1, L1

st M[r1]=M[r1], PC
L1: st M[r1]=1, PC
```

No Memory Dependence since r1 == 0

Possible Memory Dependence

```
PC=declassify PC
sc .write, M[r2], PC
```
RIFLE Binary Translation

Key Insight: Handle implicit flows at data use, not data definition.

```
mov r2=&x
mov s10=s1
bnez r1, L1
```

L1:<s10> st M[r1]=1

<s10> sc .write,M[r2]

Control Dependence
Possible Memory Dependence
Results: Security

Word Count (wc)
- Function calls and returns
- Global pointer, stack pointer

PGP – identified unexpected information flows!
- Key ring – each key labeled with a unique label
- Plain text – colored with a unique label
- Cipher text –
  - Expected: labeled with key’s label and plain text label
  - Actual: labeled with label of all keys up to used key and plain text label
Hardware Implementation & Optimizations

- All instructions create explicit flows
  - Use shadow registers/memory to store security labels
  - Augment processor data path to track explicit flows
- Transformation inserts redundant security register defines
  - Many instructions added
  - Many security registers needed

Before Opti

```
add r1=0,1
mov s50 = s10
mov s60 = s50
(r10) jump L2
L1: <s50>(r1) jump L3
... L2: <s60> add r1=0,0
      jump L1
```

After Opti

```
add r1=0,1
s50 = mov s10
(r10) jump L2
L1: <s50>(r1) jump L3
... L2: <s50> add r1=0,0
      jump L1
```
Results: Performance

Validated Itanium 2 model built in the Liberty Simulation Environment

Normalized Runtimes

Double Cache
Original Cache

164.gzip 175.vpr 181.mcf 186.crafty 197.parser 256.bzip2 300.twolf thttpd wc mpeg2dec Geo Mean

http://www.liberty-research.org
Conclusions & Future Work

• User-centric information flow security empowers *users*
  • User (not programmer) tailored security policy
  • Data-based (not program-based) security
  • Any program (no need for special languages) can be secured

• User-centric information flow security is possible

• RIFLE provides user-centric information-flow security by:
  • Tracking flow and enforcing policies *dynamically*
  • Using static “hints” via binary translation to establish security

• Future work
  • Improved performance – more optimization, hardware acceleration
  • JVM implementation – for broadened applicability
  • Declassification – allowing user-controlled data “leaks”