Safe Loading
A Foundation for Secure Execution of Untrusted Programs

Mathias Payer, Tobias Hartmann, Thomas R. Gross
Department of Computer Science
ETH Zurich, Switzerland
Motivation

Application code

Kernel
Motivation

ld.so

Code

Application code

Sandbox

Kernel
Motivation

ld.so
Code

Application code

Sandbox

Kernel
Trusted RUNtime Environment

- Trusted loader
- application
- library
- library
- ...

TRuE

Sandbox

Kernel
Outline

- Motivation
- **Attack and execution model**
- Trusted Runtime Environment
- Evaluation
- Related work
- Conclusion
Attack constraints

- Attacker tries to *escalate* privileges with:
  - Code injection
  - Code reuse (ROP / JOP*)
  - Data attacks

* ROP – Return Oriented Programming: Shacham, CCS'07
  JOP – Jump Oriented Programming: Bletsch et al., ASIACCS'11
Attack constraints

- Attacker tries to **escalate** privileges with:
  - Code injection
  - Code reuse (ROP / JOP*)
  - Data attacks

- Application is killed on policy violation

* ROP – Return Oriented Programming: Shacham, CCS'07
  JOP – Jump Oriented Programming: Bletsch et al., ASIACCS'11
Execution model

- Application is untrusted (not malicious)
  - Symbol table and ELF information used in sandbox

- Secure execution uses
  - Secure loader to bootstrap application
  - Sandbox to protect from any code-based and data-based exploits
Outline

• Motivation
• Attack model
• Trusted Runtime Environment
  • Security architecture
  • Safe loading
  • Sandbox & System call policy
  • Implementation
• Evaluation
• Related work
• Conclusion
Security architecture
Secure loader

Traditional loading:

1d.so

???

regular sandbox

transl. 1d.so

application

library

...

late interception

loader as black box

TRuE:

Secure loader

safe sandbox

application

library

...

TRuE:
Secure loader

TRuE:
- Late interception
- Secure loader

Traditional loading:
- ld.so
- Black box
- Information & safe sandbox instantiation

transf. ld.so
- Application
- Library
- ...

13
SFI sandbox

Dynamic translator:
- Translates individual basic blocks
- Checks branch targets and origins
- Weaves guards into translated code

Sandbox

Original code

1
2
3
4

Translated code

1'
2'
3'

System call policy

Kernel
SFI sandbox

Dynamic translator

- Translates individual basic blocks
- Checks branch targets and origins
- Weaves guards into translated code

Protects from code-based and data-based attacks
TRuE: implementation

- Prototype implementation (open source)
  - Focus on IA32 and Linux
  - Concept works for any ISA and operating system

- Small trusted computing base

<table>
<thead>
<tr>
<th></th>
<th>Code</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure loader</td>
<td>5,400</td>
<td>2,100</td>
</tr>
<tr>
<td>Sandbox</td>
<td>15,200*</td>
<td>3,200</td>
</tr>
</tbody>
</table>

*4,900 LOC for the translation tables
Outline

- Motivation
- Attack model
- Trusted Runtime Environment
- **Evaluation**
  - Security discussion
  - SPEC CPU2006 performance
- Related work
- Conclusion
Security discussion

• Two execution domains
  • Privileged sandbox domain
  • Unprivileged application domain (traps into sandbox)

• Sandbox ensures code integrity
  • Protection from code-injection and return oriented programming
  • Policy protects from jump oriented programming and data attacks

• Secure loader enables safe program instantiation
  • Low complexity (bare bone functionality)
  • API for requests from the application
Security discussion

- Two execution domains
  - Privileged sandbox domain
  - Unprivileged application domain (traps into sandbox)

- Sandbox ensures code integrity
  - Protection from code-injection and return oriented programming
  - Policy protects from jump oriented programming and data attacks

- Secure loader enables safe program instantiation
  - Low complexity (bare bone functionality)
  - API for requests from the application

Protects unmodified, binary applications from attacks
SPEC CPU 2006 performance

• Benchmarks execute with well-defined policy
  • On Ubuntu Jaunty
  • Intel Xeon E5520 CPU at 2.27GHz
  • GCC version 4.3.3

• Three configurations:
  • native
  • Secure loader (without sandbox)
  • TRuE (secure loader plus sandbox)
## SPEC CPU 2006 performance

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Secure Loading</th>
<th>TRuE</th>
</tr>
</thead>
<tbody>
<tr>
<td>400.perlbench</td>
<td>-0.3%</td>
<td>85%</td>
</tr>
<tr>
<td>401.bzip2</td>
<td>-0.1%</td>
<td>4.9%</td>
</tr>
<tr>
<td>429.mcf</td>
<td>-0.1%</td>
<td>0.5%</td>
</tr>
<tr>
<td>464.h264ref</td>
<td>-0.3%</td>
<td>41%</td>
</tr>
<tr>
<td>433.milc</td>
<td>0.1%</td>
<td>3.7%</td>
</tr>
<tr>
<td><strong>Average</strong>*</td>
<td><strong>-0.1%</strong></td>
<td><strong>15%</strong></td>
</tr>
</tbody>
</table>

* Average is calculated over all 28 SPEC CPU2006 benchmarks
## SPEC CPU 2006 performance

### Low performance impact

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Secure Loading</th>
<th>TRuE</th>
</tr>
</thead>
<tbody>
<tr>
<td>perlbench</td>
<td>-0.3%</td>
<td>85%</td>
</tr>
<tr>
<td>bzip2</td>
<td>-0.1%</td>
<td>4.9%</td>
</tr>
<tr>
<td>mcf</td>
<td>-0.1%</td>
<td>0.5%</td>
</tr>
<tr>
<td>h264ref</td>
<td>-0.3%</td>
<td>41%</td>
</tr>
<tr>
<td>milc</td>
<td>0.1%</td>
<td>3.7%</td>
</tr>
</tbody>
</table>

**Average***

-0.1% 15%

*Average is calculated over all 28 SPEC CPU2006 benchmarks*
Outline

- Motivation
- Attack model
- Trusted Runtime Environment
- Evaluation
- Related work
- Conclusion
Related work

- System call interposition (Janus, AppArmor)
  - Only system calls checked, code is unchecked
- Software-based fault isolation (Libdetox, Vx32, Strata)
  - Only a sandbox is not enough, additional guards and system call authorization needed, no loader information
- Static binary translation (Google's NaCL, PittSFIELD)
  - Limits the ISA, static, special compilers needed
- Full system translation (VMWare, QEMU, Xen)
  - Management overhead, data sharing problem
Outline

- Motivation
- Attack model
- Trusted Runtime Environment
- Evaluation
- Related work
- Conclusion
Conclusion

• TRuE protects from code-based and data-based exploits
  • **Secure loader** extracts information
  • **Sandbox** protects from code-based and data-based exploits

• Trusted secure loader increases security
  • Application needs no privileges to map code executable
  • Knowledge of program structure enables new guards

• TRuE protects unmodified applications in user-space
Questions?

http://nebelwelt.net/projects/TRuE/
Software based fault isolation

- Translates individual basic blocks
- Checks branch targets and origins
- Weaves guards into translated code

Original code

1
2
3
4

Mapping table

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1'</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2'</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3'</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Code cache

1'
2'
3'

Dynamic translator

Indirect control flow transfers use a dynamic check to verify target and origin
Implementation alternatives

- Static binary translation
  - No second protected domain
  - No dynamic library/module support
  - Restricted ISA

- Regular loader, hidden sandbox
  - Sandbox hidden by modifying loader data-structures
  - Loader treated as black-box
Malicious applications

- No information about internal control flow
  - Coarse-grained protection at system call level

- Application can use CPU time (inside the app)
  - System call policy protects from malicious system calls