Practical Verification of System Integrity in Cloud Computing Environments

Trent Jaeger
Penn State NSRC Industry Day
April 27\textsuperscript{th}, 2012
Overview

- Cloud computing even replaces physical infrastructure

- Is it safe to trust these services?
Reasons to Doubt

- History has shown they are **vulnerable to attack**
  - SLAs, audits, and armed guards offer few guarantees
  - **Insiders** can subvert even hardened systems

Data Loss Incidents

<table>
<thead>
<tr>
<th>Year</th>
<th>Incidents</th>
</tr>
</thead>
<tbody>
<tr>
<td>'06</td>
<td>641</td>
</tr>
<tr>
<td>'07</td>
<td>770</td>
</tr>
<tr>
<td>'08</td>
<td>986</td>
</tr>
<tr>
<td>'09</td>
<td>695</td>
</tr>
<tr>
<td>'10</td>
<td>678</td>
</tr>
<tr>
<td>'11</td>
<td>903</td>
</tr>
</tbody>
</table>

Incident Attack Vector

- **External** 54%
- **Insider** 16%
- **Unknown** 7%
- **Accidental** 23%

Credit: The Open Security Foundation datalossdb.org
Blind Trust

• Services are essentially black boxes
  ‣ Are the data produced from a trustworthy system?
  ‣ Will the system protect client data?

Need to verify the service’s integrity
Integrity Monitoring

- We need to monitor the system’s integrity
  - Define criteria for a trustworthy system
  - Verify the system meets those criteria

- How do we measure the system’s configuration?
- Will the service remain trustworthy?
System Complexity

- Systems are complex
  - Systems run several layers of independent software
  - Systems often span multiple hosts
Cloud Complexity

- Cloud environments add further challenges
  ‣ Opaque, Dynamic
Research Goal

Construct a mechanism to monitor cloud-hosted services to ensure they satisfy a broad range of integrity criteria with minimal verification overhead.
Research Experience

System Integrity

- System-Wide CW Integrity
  - In Preparation
- Automate Hook Placement
  - In Submission
- Process Firewall
  - In Submission
- Detect Name Vulnerabilities
  - USENIX Security 2012
- Compute Attack Surface
  - ACM ASIACCS 2012
- Name Resolution Defense
  - EC2ND 2011
- Cut-Mediation Equivalence
  - Euro Symp Program 2010

Trustworthy Cloud Computing

- Verify Cloud App Integrity
  - In Preparation
- Integrity Verification Proxy
  - Trust 2012
- NetROTI
  - IEEE S&P Mag 2011
- Cloud Verifier
  - ACM CCSW 2010
- Cloud Security
  - IEEE S&P Mag 2010
- VM Verifier
  - ACSAC 2009
- sCore
  - ACSAC 2007
Integrity Problem

- Programs are deployed in a variety of environments
  - Where they may be expected to process inputs from both trusted and untrusted sources
- How can a program protect its integrity under these circumstances?

![Program Data Flows Diagram]

- Attack Surface:
  - Trusted Files
  - Network Input
  - Local Files/IPC
  - Downstream Processing
System Integrity

• Classical integrity (Biba) prevents information flow from low to high
  • Excepting trusted processes
System Integrity

- Classical integrity (Biba) prevents information flow from low to high
  - Excepting trusted processes
  - In many cases, processes do receive low object data
    - E.g., services receive untrusted requests
  - How do we justify this trust?
- **Insight**: Info flow integrity alone is too simplistic
Name Resolution Integrity

• How do we know that a process obtains resources in a manner that protects its integrity?
  ‣ E.g., adversaries may modify the system namespace
  ‣ Leads to several types of attacks
    • File squatting, directory traversal, TOCTTOU,…
• Sponsor: NSF
Name Resolution Attack (1)

- Bash script predictable temporary file

**Victim:**

```bash
script.sh:
...
echo $tmpstate > /tmp/somefile
```

**Adversary:**

```bash
/* Create inode for name /tmp/somefile before victim tries to create this inode */
```

V: Request: Resolve /tmp/somefile

A: Reply: adversary-owned inode

Introduce untrusted resource

```
Nameserver
```

Name1
/tmp/somefile
```
Adversary File
Obj2
```
Name Resolution Attack (2)

- Time-of-check-to-time-of-use (TOCTTOU) attack
  - After system call to check adversary resource
  - Adversary redirects victim to privileged resource

**Victim:**
```c
obj_stat = stat("name2");
/* Check obj_stat properties */
/* open obj */
obj = open("name2");
```

**Adversary:**
```c
/* Change name2 to point to obj2 from obj1 */
```

**Diagram:**
- **V** (Victim)
- **A** (Adversary)
- **Nameserver**
- **Request: Resolve Name2**
- **Reply: Obj2**
- **Introduce untrusted bindings**

**Observation:**
```
obj_stat != obj
```
Find Vulnerabilities

• Find name resolution vulnerabilities in programs
  ‣ So programs can be fixed to perform the correct checks
    
    ```
    if ((stat("/var/mail/root", st)) == 0 || IS_ISLNK(st->st_mode))
    ```
  ‣ Or access control policies can be tightened
    
    ```
    root@mantra:/var# ls -l | grep mail
    drwxr-xr-x 2 root mail 4096 2012-03-29 02:05 mail
    ```
  ‣ Or system mechanisms can prevent such attacks
    • Process Firewall (discuss later)
STING: Resolution Tester

• Runtime analysis with active adversary in the OS
  ‣ OS models adversaries using access control permissions

• Only produce attack test case when that program’s adversary can modify directory (i.e., change bindings or resources) used in resolution
  ‣ Dynamically changes namespace to generate attack test case
  ‣ Later detects if program was vulnerable to the attack test case or not
  ‣ Finally, rolls back changes to namespace
STING: Resolution Tester

- Only about 15% of name resolutions are accessible to adversaries and only 2% are vulnerable

<table>
<thead>
<tr>
<th>Adversary model</th>
<th>Total Resolutions</th>
<th>Adversary Access</th>
<th>Vulnerable</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC - Ubuntu</td>
<td>690</td>
<td>134 (19.4%)</td>
<td>21 (3.0%)</td>
</tr>
<tr>
<td>DAC - Fedora</td>
<td>514</td>
<td>66 (12.8%)</td>
<td>5 (1.0%)</td>
</tr>
</tbody>
</table>

- Found 21 unknown vulnerabilities and 5 w/o patches
Resource Access Integrity

• Just the process of obtaining resources is fraught with danger
  ‣ Malicious names
  ‣ Malicious modification of namespace bindings (STING)
  ‣ Maliciously-timed resource delivery (races)

• Access control, sandboxing, capabilities, host IDS techniques cannot prevent these attacks

• **Insight**: Need to protect individual program entrypoints when using names to retrieve resources
Resource Access Integrity

• Access Control
  ‣ \texttt{Authorize( process, resource, op )}

• But to enforce resource access integrity
  ‣ Names: \texttt{Authorize( process, entry, resource, op )}
  ‣ Bindings: \texttt{Authorize( process, entry, resource, bindings, op )}
  ‣ Races: \texttt{Authorize( process, entry, syscall trace, resource, bindings, op )}

• All this information is available to systems to protect processes from well-known attacks
Process Firewall

• Gather “context” in which system call is made to check against invariants
  ‣ Program entrypoint
  • Support binaries and scripts
  ‣ Bindings and resources
  ‣ System call traces
  • Only need a few

• Result is like a **firewall for the syscall interface**

![Diagram of Process Firewall architecture]

Figure 2: The Process Firewall architecture: Shaded elements are specially developed for Process Firewall rule generation, installation, and processing. Unshaded elements in the Process Firewall proper are standard firewall components.
Process Firewall

• Can it be efficient enough?
  ‣ <4% on macrobenchmarks and about 10% or less on all microbenchmarks

<table>
<thead>
<tr>
<th>Syscall</th>
<th>DIS</th>
<th>BASE</th>
<th>FULL</th>
<th>CONCACHE</th>
<th>LAZYCON</th>
<th>INSTSP</th>
<th>AGGR</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>11.675</td>
<td>11.681 (0.05)</td>
<td>12.641 (8.27)</td>
<td>12.666 (8.48)</td>
<td>11.865 (1.62)</td>
<td>11.873 (1.69)</td>
<td>11.865 (1.63)</td>
</tr>
<tr>
<td>stat</td>
<td>12.545</td>
<td>12.609 (0.51)</td>
<td>26.403 (110.46)</td>
<td>23.207 (85)</td>
<td>21.981 (72.21)</td>
<td>13.872 (10.57)</td>
<td>13.97 (11.35)</td>
</tr>
<tr>
<td>read</td>
<td>11.767</td>
<td>11.805 (0.32)</td>
<td>15.332 (30.29)</td>
<td>14.823 (25.97)</td>
<td>13.704 (16.46)</td>
<td>11.982 (1.82)</td>
<td>12.021 (2.15)</td>
</tr>
<tr>
<td>write</td>
<td>11.763</td>
<td>11.794 (0.26)</td>
<td>13.602 (15.63)</td>
<td>13.096 (11.33)</td>
<td>12.123 (3.06)</td>
<td>11.975 (1.80)</td>
<td>12.035 (2.31)</td>
</tr>
<tr>
<td>fstat</td>
<td>11.826</td>
<td>11.857 (0.26)</td>
<td>15.529 (31.31)</td>
<td>14.897 (25.96)</td>
<td>13.719 (16.01)</td>
<td>13.106 (10.82)</td>
<td>13.32 (12.63)</td>
</tr>
<tr>
<td>open+close</td>
<td>24.632</td>
<td>24.722 (0.36)</td>
<td>44.54 (80.82)</td>
<td>39.789 (61.53)</td>
<td>37.511 (52.28)</td>
<td>26.113 (6.01)</td>
<td>26.92 (9.29)</td>
</tr>
<tr>
<td>fork+exit</td>
<td>104.326</td>
<td>104.392 (0.06)</td>
<td>112.371 (7.71)</td>
<td>110.918 (6.31)</td>
<td>106.303 (1.89)</td>
<td>104.959 (0.60)</td>
<td>104.743 (0.40)</td>
</tr>
<tr>
<td>fork+execve</td>
<td>664</td>
<td>667.05 (0.45)</td>
<td>903.714 (36.10)</td>
<td>861.571 (29.75)</td>
<td>818.142 (23.21)</td>
<td>670.589 (0.99)</td>
<td>673.58 (1.44)</td>
</tr>
<tr>
<td>fork+sh -c</td>
<td>1461.875</td>
<td>1465.375 (0.23)</td>
<td>1934.666 (32.34)</td>
<td>1856 (26.96)</td>
<td>1766 (20.80)</td>
<td>1475 (0.89)</td>
<td>1476.87 (1.02)</td>
</tr>
</tbody>
</table>

No rules → All rules, but no optimization → All rules, optimized
Protect Program Integrity

• OK, so what about the cases where client (adversary) data is let into a service process?
  ‣ Program needs to ensure that operations that result from this data are authorized

• People have been adding authorization hooks to a variety of programs over the last 10 years
  ‣ Operating systems: Linux, BSD, Xen
  ‣ Programs: X server, Apache, Postgres, D-Bus, gconf, ...

• Problems: missing hooks, too coarse, can’t decide, ...

• Sponsor: NSF and HP Innovation Research Program
Authorization Hook Placement

• Request integrity results when authorization hooks to mediate all “security-sensitive operations”

• **Insight:** We identify that security-sensitive operations are determined by user “choice”
  ‣ Select an object from a container of objects available to all
  ‣ Select a code branch to take by affecting branch predicate

• We built analyses to find these and produce comprehensive hook placements automatically
Authorization Hook Placement

- Static taint tracking to find effects from requests
- Find objects: objects chosen from containers by request inputs
- Find ops: code branches chosen by predicates on request inputs
- All but 3 hooks found in X with 1 false positive
System Integrity Safety (AFOSR)

- **Objects (includes subjects)**
  - **Label integrity**: must be verified to satisfy integrity properties for that label (Clark-Wilson integrity verification procedures)

- **Processes (per entrypoint)**
  - **Entrypoint integrity**: Only some may be accessible to adversary (attack surface)
  - **Resource access integrity**: must process resource requests to satisfy integrity invariants (e.g., name resolution and more)
  - **Request integrity**: all security-sensitive operations due to low integrity input must be authorized by security policy (authorization)
  - **Data structure integrity**: all untrusted inputs that are used to update process data must satisfy data structure requirements
  - **Value integrity**: The resultant values must be well-formed
System-Wide Integrity

- Use available mandatory access control and network policies to evaluate comprehensive integrity system-wide
  - Sponsor: AFRL and ARL
System-Wide Integrity

- Use available mandatory access control and network policies to evaluate comprehensive integrity system-wide
  - Sponsor: AFRL and ARL
System-Wide Integrity

- Use available mandatory access control and network policies to evaluate comprehensive integrity system-wide
System-Wide Integrity

- Use available mandatory access control and network policies to evaluate comprehensive integrity system-wide
Find Entrypoint Mediation

- Use available mandatory access control and network policies to evaluate comprehensive integrity system-wide.

![Diagram showing entrypoint mediation](Image)
Generate Integrity Policy

• **Goal**: Find minimal mediation necessary to enforce information flow integrity system-wide

• **Insight**: Graph Cut-Mediation Equivalence [ESOP 2010]

• Find minimal number of entrypoint mediations necessary to protect integrity
  ▸ Where request, data structure, value integrity is necessary

• Find minimal number of entrypoint mediations and integrity verification procedures to protect integrity
  ▸ Covers label integrity as well

• **Approximates Clark-Wilson integrity system-wide**
Integrity Monitor Concept

- **Integrity monitor** similar to a reference monitor
  - Mediate access to service based on integrity criteria

- **Challenges**
  - Where do we measure integrity-relevant events?
  - How do we verify ongoing integrity?
  - How can we deploy this in a cloud environment?
TCG Remote Attestation

- Trusted Platform Module (TPM)
  - PCRs store event measurements
  - Protected key pair uniquely identifies platform
  - Enables remote attestation of recorded events

\[
\text{QUOTE}(N, PCR, AIK)
\]
Measurement Limitations

- Administrator decides what to measure
  - Difficult to verify arbitrary criteria
  - Too little or irrelevant information
Attestation Limitations

• Attestation limits reporting **timeliness**
  ‣ Made worse by **slow hardware**
  ‣ High demand for attestations
Improved Monitoring

- **Insight:** Move integrity monitor to the cloud node
  - Avoid having to poll for attestations
  - Measure only integrity-relevant events
  - Bind cloud node integrity to installer and image (NetROTI)
Implementation

- **Client**: (1) Register criteria, (2) Verify IVP, (4) Connect
- **Channel Mediator**: (2) Verify IVP, (3) Verify VM, (5) Report Violation
- **IVP**: (3) Verify VM
- **VM**: Monitor VM
- **Cloud Node**: Modules, Integrity Monitor

---

Systems and Internet Infrastructure Security Laboratory (SIIS)
Evaluation

• Tested integrity monitor functionality
  ‣ Built Apache VM to approximate Clark-Wilson integrity
  ‣ Performed attacks on loadtime / runtime criteria
    • Loaded modified VM components / configurations
  ‣ Used x86 compatibility exploit to modify kernel data
    • Disabled SELinux enforcement (detected)
    • Modified binary handler function pointers (detected)
    • Replaced MAC enforcement policies (detected)
Performance

- 1.5% overhead for applications -- Apache and distcc
- <8% overhead on microbenchmarks

VMI Findings

- Measure enforcement
  - Changes infrequently
- `gdb/ptrace` are inefficient

- Exploring more efficient VMI approach

<table>
<thead>
<tr>
<th>Operation</th>
<th>Mean (± 95% CI) (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Watchpoint Trigger</td>
<td></td>
</tr>
<tr>
<td>VM Exit and Entry</td>
<td>0.006 (± 0.000)</td>
</tr>
<tr>
<td>QEMU overhead</td>
<td>0.496 (± 0.081)</td>
</tr>
<tr>
<td>GDB overhead</td>
<td>0.327 (± 0.054)</td>
</tr>
<tr>
<td>Monitor Overhead</td>
<td>0.172 (± 0.028)</td>
</tr>
<tr>
<td>Runtime Modules</td>
<td></td>
</tr>
<tr>
<td>Collect LIM Hash</td>
<td>66.76 (± 0.215)</td>
</tr>
<tr>
<td>Read kernel variable</td>
<td>0.132 (± 0.002)</td>
</tr>
</tbody>
</table>
Cloud Roadmap

• Built proof-of-concept in Eucalyptus [CCSW ’10]
  ‣ Preliminary evaluation on distributed compiler

• Splitting IVP into a two-tiered verifier
  ‣ Integrating IVP into OpenStack
  ‣ Support VM migration
  ‣ Plan to submit to ACM SoCC ’12

![Diagram of Cloud Roadmap]

- Client specifies integrity criteria
- Monitor cloud verifier integrity
- Monitor VM integrity
- Monitor cloud node integrity

System and Internet Infrastructure Security Laboratory (SIIS)
Thank you

Trent Jaeger (tjaeger@cse.psu.edu)
http://www.trentjaeger.com/
SIIS Laboratory (http://siis.cse.psu.edu)
Operating System Security

Trent Jaeger, The Pennsylvania State University

Operating systems provide the fundamental mechanisms for securing computer processing. Since the 1960s, operating systems designers have explored how to build "secure" operating systems — operating systems whose mechanisms protect the system against a motivated adversary. Recently, the importance of ensuring such security has become a mainstream issue for all operating systems. In this book, we examine past research that outlines the requirements for a secure operating system and research that implements example systems that aim for such requirements. For system designs that aimed to satisfy these requirements, we see that the complexity of software systems often results in implementation challenges that we are still exploring to this day. However, if a system design does not aim for achieving the secure operating system requirements, then its security features fail to protect the system in a myriad of ways. We also study systems that have been retrofitted with secure operating system features after an initial deployment. In all cases, the conflict between function on one hand and security on the other leads to difficult choices and the potential for unwise compromises. From this book, we hope that systems designers and implementers will learn the requirements for operating systems that effectively enforce security and will better understand how to manage the balance between function and security.