CPSC 426/526
Accountability

Ennan Zhai
Computer Science Department
Yale University

Credits: Some slides from Andreas Haeberlen’s SOSP’07 and OSDI’10 talks
Motivation

- Cheating is a serious problem in itself
  - Multi-billion-dollar industry

- A more general problem:
  - Alice relies on software that runs on a third-party machine
  - Examples: Competitive system (auction), federated system...
  - How does Alice know if the software running as intended?
Dealing with faults is difficult in practice.

- How to detect faults?
- How to identify the faulty nodes?
- How to convince others that a node is (not) faulty?
Learning from the 'offline' world

- Relies on accountability
- Example: Banks

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commitment</td>
<td>Signed receipts</td>
</tr>
<tr>
<td>Tamper-evident record</td>
<td>Double-entry bookkeeping</td>
</tr>
<tr>
<td>Inspections</td>
<td>Audits</td>
</tr>
</tbody>
</table>

- Can be used to detect, identify and convince
- Recall: Fault-tolerance work focused on tolerance

- Goal: A general+practical system for accountability
Outline

- Introduction
- What is accountability?
- PeerReview
- Accountable VM
Ideal accountability

- Fault := Node deviates from expected behavior
Ideal accountability

- Fault := Node deviates from expected behavior
- Recall that our goal is to
  - detect faults
  - identify the faulty nodes
  - convince others that a node is (or is not) faulty
Ideal accountability

- Fault := Node deviates from expected behavior
- Recall that our goal is to
  - detect faults
  - identify the faulty nodes
  - convince others that a node is (or is not) faulty

- Can we build a system:

  Whenever a node is faulty in any way, the system generates a proof of misbehavior against that node
Can we detect all faults?

- **Problem:** Faults that affect only a node's internal state

- **Focus on observable faults:**
  - Log information

- This allows us to detect faults without introducing any trusted components
Can we always get a proof?

- Three possible causes:
  - A never sent X
  - B refuses to accept X
  - X was lost by the network

- Cannot get misbehavior proof!

- Generalize to verifiable evidence:
  - a proof of misbehavior, or
  - a challenge that the node cannot answer

- What if, after a long time, no response has arrived?
  - Does not prove the fault, but we can suspect the node
Practical accountability

- We propose the following definition of a distributed system with accountability:

  Whenever a fault is observed by a correct node, the system eventually generates verifiable evidence against a faulty node.

- This is useful:

  Any (!) fault that affects a correct node is eventually detected and linked to a faulty node.
Outline

- Introduction
- What is accountability?
- PeerReview
- Accountable VM
PeerReview

- Adds accountability to a given system
  - Implemented as a library
  - Provides secure record, commitment, auditing, etc.

Assumptions:
1. System can be modeled as collection of deterministic state machines
2. Nodes have reference implementations of the state machines
3. Correct nodes can eventually communicate
4. Nodes can sign messages
PeerReview

- Adds accountability to a given system
  - Implemented as a library
  - Provides secure record, commitment, auditing, etc.

- Assumptions:
  1. System can be modeled as collection of deterministic state machines
  2. Nodes have reference implementations of the state machines
  3. Correct nodes can eventually communicate
  4. Nodes can sign messages
PeerReview in High-Level

- All nodes keep a log of their inputs & outputs
  - Including all messages
- Each node has a set of **witnesses**, who audit its log periodically
- If the witnesses detect misbehavior, they
  - generate evidence
  - make the evidence available to other nodes
- Other nodes check evidence, report fault
PeerReview detects tampering

- What if a node modifies its log entries?
- Log entries form a hash chain
  Inspired by secure histories [Maniatis02]
- Signed hash is included with every message
  ⇒ Node commits to its current state
  ⇒ Changes are evident
PeerReview detects inconsistencies

- What if a node
  - keeps multiple logs?
  - forks its log?

- Check whether the signed hashes form a single hash chain
PeerReview detects faults

- How to recognize faults in a log?
- Assumption:
  - Node can be modeled as a deterministic state machine
- To audit a node:
  - Replay inputs to a trusted copy of the state machine
  - Check outputs against the log
Recall: Working Process

- All nodes keep a log of their inputs & outputs
  - Including all messages
- Each node has a set of witnesses, who audit its log periodically
- If the witnesses detect misbehavior, they
  - generate evidence
  - make the evidence available to other nodes
- Other nodes check evidence, report fault
PeerReview

- Accountability is an approach to handling faults in decentralized systems
  - detects faults
  - identifies the faulty nodes
  - produces evidence

- PeerReview: A system that enforces accountability
  - Offers provable guarantees and is widely applicable
PeerReview is widely applicable

- **App #1: NFS server in the Linux kernel**
  - Many small, latency-sensitive requests
    - Tampering with files
    - Lost updates

- **App #2: Overlay multicast**
  - Transfers large volume of data
    - Freeloading
    - Tampering with content

- **App #3: P2P email**
  - Complex, large, decentralized
    - Denial of service
    - Attacks on DHT routing
How much does PeerReview cost?

- Dominant cost depends on number of witnesses $W$
  - $O(W^2)$ component
What is the problem of PeerReview
Outline

- Introduction
- What is accountability?
- PeerReview
- Accountable VM
Scenario: Multiplayer game

- Alice decides to play a game of Counterstrike with Bob and Charlie
Could Bob be cheating?

- In Counterstrike, ammunition is local state
  - Bob can manipulate counter and prevent it from decrementing
  - Such cheats (and many others) do exist, and are being used
Goal: Accountability

We want Alice to be able to

- Detect when the remote machine is faulty
- Obtain evidence of the fault that would convince a third party

Challenges:

- Neither Alice nor Bob may understand how the software works
  - Binary only - no specification of the correct behavior
Bob runs Alice's software image in an AVM.

AVM maintains a log of network in-/outputs.

Alice can check this log with a reference image.

- AVM correct: Reference image can produce same network outputs when started in same state and given same inputs.
- AVM faulty: Otherwise.

What if Bob manipulates the log?

How can Alice find this execution, if it exists?

Alice must trust her own reference image.
Tamper-evident logging

- Message log is tamper-evident [PeerReview]
  - Log is structured as a hash chain
  - Messages contain signed authenticators

- Result: Alice can either...
  - ... detect that the log has been tampered with, or 😊
  - ... get a complete log with all the observable messages 😊

- 474: SEND(Alice, Firing)
  - 473: SEND(Charlie, Got ammo)
  - 472: RECV(Alice, Got medipack)
  - 471: SEND(Charlie, Moving left)
  - ...
Auditing and replay

371: SEND(Alice, Firing)
370: SEND(Alice, Firing)
369: SEND(Alice, Firing)
368: Mouse button clicked
367: SEND(Alice, Got medipack)
366: Mouse moved left

373: SEND(Alice, Firing)
372: SEND(Alice, Firing)
371: SEND(Alice, Firing)
370: SEND(Alice, Firing)
369: SEND(Alice, Firing)
368: Mouse button clicked
367: SEND(Alice, Got medipack)
366: Mouse moved left
...

Modification
Evidence

Alice
Bob
AVM properties

- **Strong accountability**
  - Detects faults
  - Produces evidence
  - No false positives

- **Works for arbitrary, unmodified binaries**
  - Nondeterministic events can be captured by AVM Monitor

- **Alice does not have to trust Bob, the AVMM, or any software that runs on Bob's machine**
  - If Bob tampers with the log, Alice can detect this
  - If Bob's AVM is faulty, ANY log Bob could produce would inevitably cause a divergence during replay

If it runs in a VM, it will work
Methodology

- A prototype AVMM
  - Based on logging/replay engine in VMware Workstation 6.5.1
  - Extended with tamper-evident logging and auditing

- Evaluation: Cheat detection in games
  - Setup models competition / LAN party
  - Three players playing Counterstrike 1.6
  - Nehalem machines (i7 860)
  - Windows XP SP3
# AVMs can detect real cheats

If the cheat needs to be installed in the AVM to be effective, AVM can trivially detect it

- **Reason**: Event timing + control flow change
- **Examined**: real 26 cheats from the Internet; all detectable

---

<table>
<thead>
<tr>
<th>Event</th>
<th>BC</th>
<th>EIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECV(Alice, Hit)</td>
<td>59</td>
<td>0x861e</td>
</tr>
<tr>
<td>SEND(Alice, Fire@((2,7)))</td>
<td>54</td>
<td>0x2d16</td>
</tr>
<tr>
<td>Mouse button clicked</td>
<td>49</td>
<td>0xc43e</td>
</tr>
<tr>
<td>Interrupt received</td>
<td>44</td>
<td>0x6771</td>
</tr>
<tr>
<td>RECV(Alice, Jumping)</td>
<td>37</td>
<td>0x570f</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Bob's log

---

Event timing (for replay)
When auditing a player after a one-hour game,

- How big is the log we have to download? 148 MB
- How much time is needed for replay? ~1 hour
Online auditing

- **Idea:** Stream logs to auditors during the game
  - **Result:** Detection within seconds after fault occurs
  - **Replay** can utilize unused cores; frame rate penalty is low
Extentions

- Play and replay:
  - NetReview
  - TimingReview
Extentions

- Play and replay:
  - NetReview
  - TimingReview

- Problems
  - Privacy concerns
  - Efficiency
  - Deployment
Questions?