Backdoors: A Remote Healing Architecture for Cluster-based Systems

Florin Sultan
Laboratory for Network Centric Computing
http://discolab.rutgers.edu
A fatal exception 0E has occurred at 0028:C00068F8 in VxD VMM(01) + 000059F8. The current application will be terminated.

* Press any key to terminate the application.
* Press CTRL+ALT+DEL to restart your computer. You will lose any unsaved information in all applications.

Press any key to continue
Dang, frozen again.
The Only Way Out...
...Not Good for All
What Do We Need?

- Monitor system health
  - OS/application failures
  - DoS attack, overload
  - intrusion
- Take action to heal the system
  - repair damaged state, clean-up corrupted state
  - extract and recover good state
  - contain fault/attack
  - repel intrusion
- Where should these operations be performed?
Self-Healing

- Consumes processor cycles (intrusive)
- Relies on processor availability
  - hang failures make healing impossible
- Relies on OS resources
  - sensitive to resource depletion/unavailability
- Relies on system integrity
  - state may be corrupted
  - system may be compromised by an attacker
Alternative: Remote Healing

- Perform healing from another system
  - target system must allow remote access
  - the monitor system must be trusted

- Can we make remote healing nonintrusive?
  - no extra load on the target system
  - no reliance on target resources (processor, OS, etc.)
Target Failures

- OS/application hangs or cannot sustain service
  - hw: processor, network, disk, etc.
  - OS: driver bug, deadlock, resource exhaustion, etc.
  - DoS attack, overload
- Memory still available, yet not accessible via conventional paths (IP stack, console, etc.)
- Solution
  - monitor and detect failures
  - recover or repair software state of the affected system
The Backdoor

**backdoor**: a hidden software or hardware mechanism, usually created for testing and troubleshooting

--American National Standard for Telecommunications
The Backdoor (BD) Architecture

Target System

- Processor
- Memory
- I/O devices

Backdoor

Monitor System
Outline

- Introduction
- Remote Healing in Clusters of Computers
- Backdoor Architecture
- Case Study: Recovery in Internet Services
- Prototype
- Conclusions
Internet Services Today

- Commercial shift in using the Internet
  - e-commerce, banking, trading, auctioning, etc.
  - transactional, time-critical services
  - economic incentive to fault tolerance and service continuity
Cluster-based Internet Services

Diagram showing the relationship between clients, servers, and monitors in a cluster-based service architecture.
Cluster-based Internet Services
Remote Healing in Clusters

- **Goal:** survivability of live service state
  - OS and application-specific
- **Target:** state critical to service continuity
- **Remote monitoring and diagnosis**
  - detect failure, bad state, attack, intrusion
- **Remote intervention**
  - recovery of useful state from failed nodes
  - in-place repair of bad state
Backdoor-based Remote Healing

Private secure network

Diagram showing M, I/O, P, and BD nodes connected through a private secure network.
Backdoor Architecture Principles

1. Bidirectional access
   - both remote input and output operations must be supported

2. Remote memory access
   - memory must be accessible remotely
   - remote I/O?

3. Availability
   - failure must not impair BD

4. Nonintrusive operation
   - BD operations must not involve processors of the target system
5. Transparency
- BD operation must not be visible to target

6. Access control
- monitor and target negotiate access permissions at the beginning
- target cannot “close” the BD afterwards

7. Tamper resistance
- target cannot modify the result of a BD operation

Question: How can we implement Backdoor using existing technologies?
Remote Memory Communication (RMC)

- Remote DMA (RDMA) Read/Write operations
- Remote processor not involved
- RMC-based networking technologies: VIA, InfiniBand, etc.
Backdoor with RMC

Target

- CPU
- Memory
- RMC NIC

Monitor

- CPU
- Memory
- RMC NIC

Connections:
- MONITOR (RDMA-R)
- REPAIR/RECOVER (RDMA-R/W)
# RMC Compliance with BD Principles

<table>
<thead>
<tr>
<th>Feature</th>
<th>Compliance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bidirectional access</td>
<td>Y</td>
</tr>
<tr>
<td>Remote memory access</td>
<td>Y</td>
</tr>
<tr>
<td>Availability</td>
<td>Y</td>
</tr>
<tr>
<td>Nonintrusiveness</td>
<td>Y</td>
</tr>
<tr>
<td>Transparency</td>
<td>Y?</td>
</tr>
<tr>
<td>Access control</td>
<td>Y-</td>
</tr>
<tr>
<td>Tamper resistance</td>
<td>Y</td>
</tr>
</tbody>
</table>
Remote Healing Architecture

Application

Critical state

Detection

Action

API

Passive Gateway

Monitoring

Repair / Recovery

Active Gateway

Monitoring

Repair / Recovery

OS

BD

Nonintrusive remote access

Feb 12, 2004

COANS Seminar Spring 2004
Monitoring over RMC-BD

Monitor: progress, anomalous events, integrity constraints, etc.
Repair over RMC-BD

Target

Monitor

CPU → Memory → RMC NIC

Correct State

CPU ← Memory ← RMC NIC

Repaired State
Recovery over RMC-BD

Target

Monitor

Recoverable State

Recovered State

CPU

Memory

RMC NIC

CPU

Memory

RMC NIC
And Possibly More...

- Remote control of I/O devices
  - access state in peripheral devices, e.g., OS swap space
- Dynamically inject code/data in a live system
  - test, diagnosis, repair handlers
  - fast system reboot through OS memory overlay
  - fast restart of application components (micro-reboot)
- Monitor for intrusion/attack detection
Case Study: Recovery In Internet Services

- Remote healing is not just RMC!
  - RMC provides just a way of access
- Requires OS support
  - Failure Detection
  - Session Recovery
OS Support

- **Monitoring: Progress Box (PB)**
  - progress counter: \{scalar value, update deadline\}
  - PB = set of progress counters in OS memory
  - API to allocate and update progress counters in PB
  - monitor reads PB, checks counters, detects stalls

- **Recovery: State Box (SB)**
  - encapsulates per-session server state
  - API to export/import application state to/from SB
  - backup node reads SB, reinstates session, resumes service
Failure Detection with PB

- Target system updates progress counters in PB
  - Examples: interrupts (global, per-device), context switches, connections accepted, etc.

- Monitor process
  - scans remote PB, checks counters, detects stalls

![Diagram of the system with labeled components: Target OS, ProgressBox, and Monitor.]
Recovery With SB

- Fine-grained, essential service state
- Application-specific components (SB_APP)
  - E.g., document name, offset in document, etc.
- OS-specific components (SB_IO)
  - E.g., send/receive TCP buffers
- An SB can be distributed over multiple processes (multi-tier servers)
- Backup node extracts SB from a failed node and reinstates it locally
SB Structure

Front-end server process

Back-end server process

TCP state

App state

pipe state

App state

C1

C2

C3

SB_APP

SB_IO

SB1

SB2

SB3
Backdoors Prototype

- Implemented using Myrinet NICs with modified firmware
  - remote Read/Write DMA
  - remote OS locking (syscalls, interrupt handlers)
- Modified FreeBSD kernel
  - Progress Box
  - State Box
- Modified server applications
A Realistic Sample Application: Multi-tier Auction Service (RUBiS)

Front-End (FE)
- Apache web server

Middle Tier (MT)
- Tomcat + JBoss

Back-End
- MySQL DB
Recoverable RUBiS

\[
\begin{align*}
SB & = \{\text{reqID, req}\} \\
SB & = \{\text{reqID, tid, result}\}
\end{align*}
\]
Experimental Evaluation

- 2.4 GHz, 1 GB RAM, 1Gbps Ethernet, Myrinet LanaiX 133 MHz PCI
- Fault injection
  - synthetic freeze: halt CPU, disable device interrupts, disable network interface, trap to kernel debugger
  - emulated crashes in buggy network drivers
- Experiments
  - Microbenchmarks
  - Failover correctness
  - Failover throughput and latency
Microbenchmarks

- Monitor CPU usage, sampling a 100-counter PB
  - 46% worst-case (infinite loop)
  - < 5% @ 10 ms, < 1% @ 100 ms
  - High sampling rates possible

- Low overhead SB API
  - export/import: < 30 us
  - extract + reinstate a 10 KB front-end SB: 358 us
Failure-free Overhead

![Graph showing Failure-free Overhead](image)

- **Base**
- **Recoverable FE**
- **Recoverable FE+MT**

The graph illustrates the performance of different systems under varying loads. The x-axis represents the number of clients, while the y-axis shows the number of requests per minute. The lines indicate how each system handles load increases, with **Base** being the least effective and **Recoverable FE+MT** showing the best performance.
Failover Correctness

- **Workload/run:** 600 requests from 200 clients
  - request = DB queries + DB table update
- **Two correctness tests across crash & recovery**
  - End-to-end consistency (crash invisible to client)
  - Database integrity (exactly-once semantics preserved)
- All crash-test runs were validated
Failover Throughput (FE+MT crash)
Failover Latency (FE+MT crash)
Related Work

- DEC WRL Titan system [Mogul ‘86]
- Recovery Box [Baker ‘93]
- Rio reliable file cache [Chen ‘96]
- Online OS reconfiguration [Soules ‘03]
- Virtual machines [Bressoud ‘95, Dunlap ‘02]
- Automatic repair of data structures [Demski ‘03]
Conclusions

- **Backdoor**: system architecture for nonintrusive remote healing
  - monitoring without using processor cycles
  - repair, recovery even when remote processor is not available
- **BD prototype**: for transparent recovery of active service sessions in cluster-based Internet services
Current and Future Work

- Remote repair of OS state
- OS support and API for healing-conscious applications
  - programmer performs application-specific monitoring, repair and recovery
- Securing the BD
  - low-level access control through BD Guard entities implemented in firmware
- Remote control of I/O devices
The People Behind Backdoors

- Aniruddha Bohra
- Stephen Smaldone
- Yufei Pan
- Iulian Neamtiu (Maryland)
- Pascal Gallard (IRISA/INRIA)
- Liviu Iftode
Thank you!

http://discolab.rutgers.edu/bda