Cloud Resilient Architecture (CRA)
-Design and Analysis

Glynis Dsouza
Hamid Alipour
Salim Hariri
Youssif-Al-Nashif

Mohamed Eltoweissy

NSF Center for Autonomic Computing
University of Arizona

Pacific National NorthWest Laboratory
Presentation Outline

- Cybersecurity and Cloud Security Challenges
- Experimental Environment and Setup
- Cloud Resilient Architecture (CRA)
  - Moving Target Defense Approach
- CRA Design and Evaluation
- Conclusions
Cybersecurity Challenges

- Current cybersecurity technologies failed to secure and protect our cyberspace resources and services
  - They are mainly signature base, manual intensive and ad-hoc;
  - According to Phishing Activity Trends Report, data-stealing increased from 36% in 2010 to more than 45% in April 2011.
- Cyber attacks can get costly if not resolved quickly. The average time to resolve a cyber attack is 18 days, with an average cost to participating organizations of $415,748. Results show that malicious insider attacks can take more than 45 days on average to contain.
- Information theft continues to represent the highest external cost, followed by the costs associated with business disruption
  - Cyber crimes are intrusive and common occurrences.
    - Caused by malicious code, denial of service, stolen or hijacked devices and malicious insiders
Cloud Security Challenges

Challenging research problem due to many interdependent tasks

Reasons for challenge

– Software Monoculture
– Dynamic Environment
– Social Networking
Software Monoculture

- Easy for attacker to study behavior of system and generate attacks
- Vulnerabilities in one software can propagate to a great extent
Cloud Resilient Architecture (CRA)

Software Behavior Encryption

- Spatio Temporal Behavior Encryption
- Moving Target Defense

Self Management

- Automatic Detection and Recovery Capabilities
Moving Target Defense

Vision

- Create, evaluate and deploy mechanisms and strategies that are diverse, continually shift, and change over time to increase complexity and costs for attackers, limit the exposure of vulnerabilities and opportunities for attack, and increase system resiliency (Source: "CyberSecurity Game-Change Research and Development Recommendations")
Moving Target Defense

Three Common Techniques of MTD
- Address Space Randomization
- Instruction Set Randomization
- Data Randomization
MTD Attack Lifetime

Successful Attack

Thwarted Attack

Versions

Small time window for attacker
Software Behavior Encryption

- Diversity
  - Hot Shuffling software variants at runtime
  - Variants are functionally equivalent, behaviorally different

- Redundancy
  - Multiple replicas on different physical hardware

- Shuffling
Software Behavior Encryption

- Diverse VM's
- Diverse Operating systems
- Diverse Programming Languages

Diversity

Software Behaviour Encryption

Redundancy

Task Variants: Service Behavior Encryption (SBE)

SBE (Task A, Phases: p_1, p_2, Versions: v_1, v_2) →
T\_1(p\_1, v\_1) \cdot T\_2(p\_2, v\_2) \cdot T\_3(p\_3, v\_3) \cdot T\_4(p\_4, v\_4) \cdot T\_5(p\_5, v\_5) \cdot T\_6(p\_6, v\_6) \cdot T\_7(p\_7, v\_7) \cdot T\_8(p\_8, v\_8) \cdot T\_9(p\_9, v\_9) \cdot T\_10(p\_10, v\_10) \cdot T\_11(p\_11, v\_11) \cdot T\_12(p\_12, v\_12) \cdot T\_13(p\_13, v\_13) \cdot T\_14(p\_14, v\_14) \cdot T\_15(p\_15, v\_15) \cdot T\_16(p\_16, v\_16) \cdot T\_17(p\_17, v\_17) \cdot T\_18(p\_18, v\_18) \cdot T\_19(p\_19, v\_19) \cdot T\_20(p\_20, v\_20)

SBE (Task B, Phases: p_1, p_2, Versions: v_1, v_2) →
T\_1(p\_1, v\_1) \cdot T\_2(p\_2, v\_2) \cdot T\_3(p\_3, v\_3) \cdot T\_4(p\_4, v\_4) \cdot T\_5(p\_5, v\_5) \cdot T\_6(p\_6, v\_6) \cdot T\_7(p\_7, v\_7) \cdot T\_8(p\_8, v\_8) \cdot T\_9(p\_9, v\_9) \cdot T\_10(p\_10, v\_10) \cdot T\_11(p\_11, v\_11) \cdot T\_12(p\_12, v\_12) \cdot T\_13(p\_13, v\_13) \cdot T\_14(p\_14, v\_14) \cdot T\_15(p\_15, v\_15) \cdot T\_16(p\_16, v\_16) \cdot T\_17(p\_17, v\_17) \cdot T\_18(p\_18, v\_18) \cdot T\_19(p\_19, v\_19) \cdot T\_20(p\_20, v\_20)
Self Management

1: while true do
2:     if (any change in management policies) then
3:         read Controller configuration file and update Controller.policies
4:     end if
5:     Currentstate ← Observer.Sensors.getCurrentState()
6:     Events← Observer.Sensors.getReceivedEvents()
7:     AnalysisDecision← Observer.analyzeKnowledge(Currentstate, Events)
8:     if (AnalysisDecision is abnormal) then
9:         Controller.actionsList = Controller.Policies.check(Events, AnalysisDecision)
10:        Controller.executeActions()
11:        LocalEvents = Controller.actionsList.getEvents()
12:     end if
13:     if (LocalEvents ≠ null) then
14:         Controller.reportEventsToPredefinedObservers()
15:     end if
16: end while
Experimental Environment

- IBM BladeCenter HS22 based Private Cloud
- University of Arizona’s Autonomic Computing Lab
- Evaluated on a three node cluster
- Each node has four versions
- Version consists of combination of:
  - Operating System
  - Programming Language
  - E.g. <Linux, C++>, <Windows, Java>
Illustrative Example

- MapReduce Job is divided into three phases
  - First Map on Input 1
  - Second Map on Input 2
  - Final Map/Reduce on above two outputs
- All three nodes run the three phases independently
- Slaves are in Hadoop MapReduce ‘Single-Node Cluster’ setup
- At the beginning of each phase, a version is selected randomly by each of the three masters.
Application used - MapReduce

- Large-Scale Data Processing
- MapReduce provides
  - Automatic parallelization & distribution
- MapReduce Wordcount program
Recovery Block - fault tolerance


Checkpoint based recovery technique
Acceptance test

- At the end of each phase, results are given by the slave to the Master
- Master runs local acceptance test
  - Output of each Map phase is in the form of <keyword 1>
  - Output of the final Map/Reduce phase is in the form of <keyword count>
  - Any other outcome is considered as failure

- If the acceptance test fails, output is requested from the other masters
Acceptance Test

- **Phase 1**: 1st Map Job
  - V3
  - TA₁

- **Phase 2**: 2nd Map Job
  - V1
  - TA₂

- **Phase 3**: Final Map/Reduce
  - V1
  - TA₃

- **Machine 1**: Acceptance Test
- **Machine 2**: Acceptance Test
- **Machine 3**: Acceptance Test

- Checkpoint-select the best output for next step
- Checkpoint-select the best output for next step
- Checkpoint-select the best output for the final output

**Final Output**
Task Management Algorithm

1: Initialize SBE
2: Controller.generatePolicies(SBE.getTasksAndPhasesVersionList());
3: Observer.setTasksPhaseSensors();
4: Observer.startTasksPhaseSensors();
5: for (phase=1 to SBE.numberOfPhases) do
6:    exitFlag←0
7:    for (task=1 to SBE.numberOfReplicas) do
8:        Controller.launchTask(task,phase);
9:    end for
10:   while (exitFlag=0) do
11:      while (true) do
12:         state←Observer.getAllTasksCheckpoint(phase)
13:         if (state='A task finished this phase successfully') then
14:             Controller.killThisPhaseForAllTasks(state,phase)
15:             Controller.updateAllTaskReplicasWithTheSuccessfulOutput();
16:         exitFlag←1
17:         breakLoop()
18:         else if (state='A task is running abnormaly') then
19:             taskWithAbnormality←state.getAbnormalTask()
20:             Controller.killAbnormalTask(taskWithAbnormality,phase)
21:             SBE.reassignPhasesVersionsForAbnormalTask(taskWithAbnormality,phase)
22:             Controller.relaunchTask(taskWithAbnormality,phase)
23:         end if
24:      end while
25:   end while
26: end for
Case 1: Resilience against Dos Attacks

Denial of Service attack on Windows VM-6

<table>
<thead>
<tr>
<th></th>
<th>Without DoS attack</th>
<th>With DoS attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without MTD</td>
<td>95</td>
<td>615</td>
</tr>
<tr>
<td>With MTD</td>
<td>105</td>
<td>105</td>
</tr>
</tbody>
</table>
## Case 2: Resilience against Insider Attacks

Compromise attack on Linux VM-1

<table>
<thead>
<tr>
<th></th>
<th>Response Time (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Without Insider attack</td>
</tr>
<tr>
<td>Without MTD</td>
<td>95</td>
</tr>
<tr>
<td>With MTD</td>
<td>105</td>
</tr>
<tr>
<td>% increase in response</td>
<td></td>
</tr>
<tr>
<td>time with MTD</td>
<td></td>
</tr>
</tbody>
</table>
Conclusions

- We cannot build perfect cloud security systems,
- CRA based on MTD provides a promising approach to build intrusion tolerance Cloud Services
- Anomaly behavior analysis will help driving when to change the environment and respond optimally to attacks or malicious events.
- Autonomic computing (Self Management) can be the underlying paradigm to implement Cloud applications and services
Questions??
Thank You