Auditing Cloud Service Level Agreement on VM CPU Speed

Ryan Houlihan, Xiaojiang Du, Chiu C. Tan, Jie Wu
Temple University

Mohsen Guizani
Qatar University
A Service Level Agreement (SLA) is a contract formed between a cloud service provider (CSP) and a user which specifies,

- in measurable terms, what resources the CSP will provide the user. (e.g. CPU speed, storage size, network bandwidth)
CSP is a profit driven enterprise, there is a great incentive for the CSP to cheat on the SLA.

CSP cannot guarantee to audit the SLA and to verify that the SLA is being met.
Third Party Auditor (TPA) [1][2] is a framework that is highly beneficial for three reasons:

- Highly flexible and scalable: easily extended to cover a variety of metrics (e.g. memory allocation, CPU usage).
- Support testing for multiple users: increase the accuracy of the cloud testing.
- Remove the auditing and verification burden from the user.
Contributions

- Develop a novel algorithm for auditing CPU allocation using a TPA framework to verify the SLA is met.
- Use real experiments to demonstrate the effectiveness of our algorithm for detecting CSP cheating on the SLA metric of CPU speed.
Threat Model - CSP

- CSP has complete control over all its own resources which include physical machines, VMs, hypervisor, etc.
- CSP is able to access and modify any data held on the VM (e.g. timestamp)
- CSP will only perform cheating if the benefit is greater than the cost.
Threat Model - TPA

- The TPA can be trusted by the user to properly carry out the auditing functions while auditing the CSP and verifying the SLA.
- TPA can obtain hypervisor source code from CSP to ensure that it does not exhibit malicious behavior.
- The TPA must be able to ensure the integrity of the hypervisor. This is provided by Trusted Platform Group (TCG) [3]. The framework for ensuring hypervisor integrity is provided by Hypersentry [4].
- Communication time between the cloud system and the TPA is 200 ms or less.
Auditing Test Requirement

- Run generic computational task: not easily detected as an audit.
- Perform redundant time recording: able to detect the modification of input/output by the cloud system.
- Assure the execution of computational task: compute the SHA-1 hash [5] of a NxN matrix.
Implementation

- Initialization:
  - VM mirroring: create a VM on auditing system that mirrors the specifications of the one on the cloud system.
  - NxN matrix creation and upload: create two NxN matrices for multiplication on the TPA, then upload onto the VM on the cloud system.
Auditing Test Execution (on the cloud VM):

- Output signal to terminal that multiplication will begin and record the time, $t_{2-i}$.
- Perform a matrix multiplication where $C = A \times B$.
- Record the elapsed time, $e_{2-i}$, and output to the terminal that the multiplication has ended.
- Compute the SHA-1 hash of the resulting matrix $C$, represented as SHA-1[C].
- Output the time to compute the matrix multiplication, $e_{2-i}$ and SHA-1[C] to the terminal.
- Shift each element of matrix $A$ and $B$ by one.
- Repeat the previous steps $X - 1$ more times where $i$ is equal to the current transposition matrix multiplication being performed.
Auditing Test Execution (on the TPA VM):

- Record the time, T.
- Initialize and execute the auditing test on the cloud VM.
- Watch the output from the cloud VM terminal. Compute the time elapsed between the signal that the multiplication has started and the signal that the multiplication has ended, \( e_{1-i} \). Also record the hash value, SHA-1[C], and the execution time, \( e_{2-i} \) as reported by the cloud VM.
- Record the elapsed time, E, for the entire execution of the test.
Verification (Communication overhead can be neglected):

- Sum up the $e_{2-i}$ value, $\sum_i^X e_{2-i}$ for all tests run on the cloud VM. And compare this to the value of E.
- we take $\sum_i^X e_{1-i}$ and compare it to $\sum_i^X e_{2-i}$.
- The hash of the resulting matrix C (SHA-1[C]) from the tests on the TPA’s VM should match the SHA-1[C] values produced on the cloud.
Testing

- Background:
  - Ubuntu Server 12.04 LST with Xen DOM-0 Hypervisor 4.1 x64.
  - 4 Gigs of ram and a Intel Q6600 Quad Core processor.
  - The VM used was given one processor with a clock of 1.0 Ghz as well as 1 Gigabyte of RAM.
  - 1000x1000 matrix of doubles.
Testing

- Results:

<table>
<thead>
<tr>
<th>Average Time 100% CPU:</th>
<th>6.7361414708</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AVERAGE (s)</td>
</tr>
<tr>
<td>100% CPU (Run 1):</td>
<td>6.727433531</td>
</tr>
<tr>
<td>100% CPU (Run 2):</td>
<td>6.7399728319</td>
</tr>
<tr>
<td>100% CPU (Run 3):</td>
<td>6.7398816269</td>
</tr>
<tr>
<td>100% CPU (Run 4):</td>
<td>6.7372778932</td>
</tr>
<tr>
<td>90% CPU:</td>
<td>7.5026936102</td>
</tr>
<tr>
<td>80% CPU:</td>
<td>8.4290672141</td>
</tr>
<tr>
<td>70% CPU:</td>
<td>9.628378256</td>
</tr>
<tr>
<td>60% CPU:</td>
<td>11.242350495</td>
</tr>
<tr>
<td>85% CPU 15% TTL:</td>
<td>6.9142334212</td>
</tr>
<tr>
<td>85% CPU 30% TTL:</td>
<td>7.0991599864</td>
</tr>
<tr>
<td>70% CPU 15% TTL:</td>
<td>7.165942121</td>
</tr>
<tr>
<td>70% CPU 30% TTL:</td>
<td>7.6036018606</td>
</tr>
</tbody>
</table>

6/8/2014
Testing

Results:

- The average time to run a single transpose matrix multiplication based on the percent cheating (100%-CPU Cap %). As the % cheating increases the average run time increases linearly, as expected.

6/8/2014
Results:

- The average time to run a single transpose matrix multiplication based on the percent cheating (100%-CPU Cap %) and the % time the cheating lasts. As the % cheating or the % time of cheating increases, the average run time increases as expected.


The End

Thank You

Q&A