2013 IEEE Sixth International Conference on Cloud Computing

Framework for Assessing Cloud Trustworthiness

Curt Wu, Steve Marotta
Charles River Analytics
Cambridge, USA
{cwu, smarotta}@cra.com

When applications or data reside in a public cloud, trustworthiness can be compromised due to lack of control over the underlying infrastructure. Most public cloud infrastructures cannot be instrumented or modified to independently verify data integrity. To support verifiable access to applications and data residing in cloud infrastructures, we are developing a framework that treats the cloud as a black box and assesses its trustworthiness from the trusted cloud client. Our solution generates and performs diagnostic tests to assess the trustworthiness of cloud-based applications. Diagnostic tests for data objects stored in the cloud are based on a separate cryptographic hash-based check that verifies their data integrity.

Keywords-cloud computing; security; trustworthiness, data integrity

I. INTRODUCTION

When applications or data reside in a public cloud, trustworthiness can be compromised due to lack of control over the underlying infrastructure. Equipment that is processing, storing, and transmitting data may be vulnerable or already compromised. Security policies and technologies may be outdated. Untrusted software may be sharing the same platform. An application or data object residing in a public cloud is clearly at greater risk than one residing in a private cloud. The cloud user needs to understand the level of trustworthiness in the information returned from the cloud to better understand the likelihood of correct results, and to better assess the probability of mission success.

Because the user has limited access to the public cloud infrastructure, the user must treat the cloud as a black box that cannot be instrumented or modified. Therefore, the user must rely on black-box testing to assess the trustworthiness of cloud-based transactions. To ensure the integrity of these diagnostic tests, all tests must be executed outside of the cloud where they can analyze the output from this black box in a trustworthy environment (i.e., locally). Also, the user cannot assess the trustworthiness of a cloud-based transaction in advance of its execution. Each time an application executes or a data object is accessed, different cloud resources may be used. Therefore, a cloud testing framework must repeatedly assess the trustworthiness of underlying cloud resources for each cloud-based transaction.

To assess the trustworthiness of a black box system, a tool must compare the output of the system against expected values. The diagnostic tests must be based on the application code, so their successes and failures will be accurate indicators of the success and failure of the application.

Finally, any solution must be automated for scalability. One of the primary reasons for implementing a cloud-based solution is to provide scalability, so a test framework that requires extensive manual intervention or includes a performance bottleneck will effectively mitigate a key benefit of the cloud. Therefore, it is not feasible to manually implement diagnostic tests for each application or data object that is designed to be run or stored in a cloud. The cloud testing framework must automatically select and integrate diagnostic tests within the transaction. If diagnostic tests fail and time permits, the framework should automatically restart the transaction in an attempt to execute it on more reliable components of the cloud infrastructure.

II. TECHNICAL APPROACH

To support verifiable access to applications and data residing in public cloud infrastructures, we are developing a Framework for Assessing Cloud Trustworthiness (FACT). The framework treats the cloud as a black box and assesses trustworthiness at the cloud client where tests can be executed within a trusted environment. To ensure that diagnostic tests faithfully and comprehensively test the functionality of the application in the cloud, they are based on the acceptance tests already generated during the application development and testing lifecycle. The diagnostic tests are integrated with the local application client, so they run within a single process. The exact nature and extent of the integration depends on the properties of the test suite, the parameters of the mission supported by the application, and the properties of the cloud. The results of the diagnostic tests are evaluated at the client. If a diagnostic test fails, the framework will attempt to redeploy the cloud application on alternative cloud resources and rerun the instrumented client. This process may need to be repeated until all diagnostic tests pass, or until time constraints are exceeded. Diagnostic tests for data objects stored in the cloud are based on a separate cryptographic hash-based check that verifies their data integrity. As with the diagnostic tests for applications, the diagnostic tests for data objects are evaluated outside of the cloud. We are implementing the following modules:

- Test analysis module
- Selection optimization module
- Instrumentation module
- Execution module
- Verifiable storage adapter
- Data integrity verifier
- Web server and file system in an enterprise-level cloud

A. *Verification of Cloud Applications*

To prepare the test suite to be inserted into the client and to extract the properties of the acceptance tests, we are developing a *test analysis module*. This module automatically infers which tests apply to each application method. This module also calculates the relative time required to run each test, so FACT can better understand the impact of the diagnostic tests on overall execution time when the tests are integrated into the application. This information will guide the selection of tests if timing constraints dictate that only a subset of the test suite can be run. Although these times will be highly dependent on the actual execution platform, comparing relative execution times will provide insight into which tests run quickly (e.g., in a few milliseconds) and which tests impact overall execution time (e.g., several seconds or even minutes).

To create a test integration plan, we are developing a *selection optimization module* that meets the constraints imposed by the mission or business process (e.g., deadlines, priorities) while taking into account the test properties. The optimization function will also incorporate properties of the cloud (e.g., perceived trustworthiness, performance guarantees) if they are known. The goal of the optimization is to balance trustworthiness (i.e., test coverage) vs. performance (i.e., timely application execution). Because different optimization functions may be better suited for different situations and missions, we have designed an extensible module that dynamically selects and applies an optimization strategy. The selection will be performed automatically based on pre-defined rules or overridden manually by a security administrator.

We are developing an *instrumentation module* to integrate the application client with the selected diagnostic tests based on the test integration plan. The module integrates the tests directly with a binary modification tool into the application client binary, so the tests and application run within the same process. The tests are then more likely to run on and therefore evaluate the same hardware and software as the application client when they access the application in the cloud. Evaluation of test success (i.e., determination of pass/fail) is separated from the primary test code (i.e., the invocation of the cloud-based application) for better test management and failure response. Because unit tests can vary greatly in their structure (i.e., theoretically any language construct could be used), it may not be feasible to automatically separate the test and evaluation for all cases. However we will automate this process as much as is feasible and limit manual intervention to exceptional cases.

We are developing an *execution module* to run the instrumented client and monitor its progress. If a test fails, execution is aborted immediately to prevent invalid data from affecting the mission or business process. The client can be reinstrumented (if needed) and rerun. If it is possible to specify or constrain how the application is deployed in the cloud, the framework will attempt to use different hardware/software within the cloud or even a different cloud infrastructure altogether. We will indirectly assess the trustworthiness of the cloud by monitoring the applications that run in it. As applications successfully execute, the cloud trustworthiness increases; as tests fail, the cloud trustworthiness decreases.

B. *Verification of Cloud Data*

To verify the data integrity of data objects in the cloud, we are developing a *verifiable storage adapter* and a *data integrity verifier*. The verifiable storage adapter will provide an API that is a superset of the cloud storage API. In addition to the standard write command (e.g., "put()"), the extended API will include an alternative version (e.g., "putVerified()") that also calculates a checksum of the data object to be written and sends it to the data integrity verifier. The data integrity verifier will immediately verify that the checksum of the data object retrieved from the cloud matches the checksum provided. Furthermore, the data integrity verifier will maintain a table of object identifiers and checksum values. Periodically, the data integrity verifier will verify the data integrity of each object based on its checksum value. If there is a discrepancy, the object will be flagged as invalid, and the monitor will send out appropriate error messages.

C. *Evaluation*

To evaluate the feasibility of our framework, we are implementing a *web server and file system in an enterprise-level cloud*. We will apply realistic simulated attacks and faults directly to the cloud software stack. We will define specific performance metrics (e.g., execution time, fault detection rate) to evaluate the performance of the framework. We will design our evaluation cloud to replicate the functionality and interface of the commercial cloud solutions, so our approach will be transferable. We will also evaluate the performance of our framework in a commercial cloud. We will not be able to introduce attacks and faults to a commercial cloud, but we will be able to evaluate the execution time of the tests in a commercial cloud environment.

III. OPEN QUESTIONS

We have a number of open questions that we are investigating:

- How are application tests typically structured?
- To what extent should we address the range of testing styles? For what cases should we provide fully automated test extraction? Semi-automated?
- How prevalent are errors in different cloud infrastructures?
- To what extent are different types of users willing to tolerate invalid data?
- How well does our approach to data verification scale?
- How applicable are these technologies to other non-cloud environments where remote attestation of data integrity is useful?

ACKNOWLEDGMENT

This work is being supported by the Air Force Research Laboratory (AFRL) under contract FA8750-13-C-0028.