IJDCST @Aug-Sept-2015, Issue- V-3, I-6, SW-23
ISSN-2320-7884 (Online)
ISSN-2321-0257 (Print)

Verification of Scalable Distributed Service Integrity for Software-as-a-Service Clouds

Ramalakshmi Golla[1], I Tabitha[2], Sayeed Yasin[3]

[1]M.Tech (CSE), Nimra College of Engineering & Technology, A.P., India.

[2]Asst. Professor, Dept. of Computer Science & Engineering, Nimra College of Engineering & Technology, A.P., India.

[3]Head of the Department, Dept. of Computer Science & Engineering, Nimra College of Engineering & Technology, A.P., India.

Abstract — Cloud is providing 3 types of services IaaS, PaaS and SaaS. Software as a Service (SaaS) is a software distribution model in which applications are hosted by a vendor or service provider and made available to customers over a network, typically the Interne. However, due to their sharing nature, SaaS clouds are vulnerable to malicious attacks SaaS cloud systems enable application service providers to deliver their applications via massive cloud computing infrastructures.. In this paper, we present IntTest, a scalable and effective service integrity attestation framework for SaaS clouds. IntTest provides a novel integrated attestation graph analysis scheme that can provide stronger attacker pinpointing power than previous schemes. Moreover, IntTest can automatically enhance result quality by replacing bad results produced by malicious attackers with good results produced by benign service providers. We have implemented a prototype of the IntTest system and tested it on a production cloud computing infrastructure using IBM System S stream processing applications. Our experimental results show that IntTest can achieve higher attacker pinpointing accuracy than existing approaches. IntTest does not require any special hardware or secure kernel support and imposes little performance impact to the application, which makes it practical for large-scale cloud systems.

Keywords — Distributed service integrity attestation, cloud computing, secure distributed data processing

I. INTRODUCTION

The cloud computing concept is simple: it enables you to run computer applications over the Internet, removing the need to buy, install or manage your own servers. You can simply run your company's IT operations with just a browser and an Internet connection .Cloud computing has emerged as a cost-effective resource leasing paradigm, which obviates the need for users maintain complex physical computing infrastructures by themselves. Software-as-a-service (SaaS) clouds (e.g., Amazon Web Service (AWS) [1] and Google AppEngine [2]) build upon the concepts of software as a service [3] and service-oriented architecture (SOA) [4], [5], which enable application service providers (ASPs) to deliver their applications via the massive cloud computing infrastructure. In particular, our work focuses on data stream processing services [6], [7], [8] that are considered to be one class of killer applications for clouds with many real-world applications in security surveillance, scientific computing, and business intelligence. However, cloud computing infrastructures are often shared by ASPs from different security domains, which make them vulnerable to malicious attacks [9], [10]. For example, attackers can pretend to be legitimate service providers to provide fake service components, and the service components provided by benign service providers may include security holes that can be exploited by attackers. Our work focuses on service integrity attacks that cause the user to receive untruthful data processing results, illustrated by Fig. 1. Although confidentiality and privacy protection problems have been extensively studied by previous research [11], [12], [13], [14], [15], [16], the service integrity attestation problem has not been properly addressed. Moreover, service integrity is the most prevalent problem, which needs to be addressed no matter whether public or private data are processed by the cloud system. Although previous work has provided various software integrity attestation solutions [9], [10], [11],[12], those techniques often require special trusted hardware or secure kernel support, which makes them difficult to be deployed on large-scale cloud computing infrastructures. Traditional Byzantine fault tolerance (BFT) techniques [14], [15] can detect arbitrary misbehaviors using full-time majority voting (FTMV) over all replicas, which however incur high overhead to the cloud system.

In this paper, we present IntTest, a new integrated service integrity attestation framework for multitenant cloud systems. IntTest provides a practical service integrity attestation scheme that does not assume trusted entities on third-party service provisioning sites or require application modifications. IntTest builds upon our previous work RunTest [16] and AdapTest [7] but can provide stronger malicious attacker pinpointing power than RunTest and AdapTest. Specifically, RunText and AdapTest as well as traditional majority voting schemes need to assume that benign service providers take majority in every service function. However, in large-scale multitenant cloud systems, multiple malicious attackers may launch colluding attacks on certain targeted service functions to invalidate the assumption. To address the challenge, IntTest takes aholistic approach by systematically examining both consistency and inconsistency relationships among different service providers within the entire cloud system.

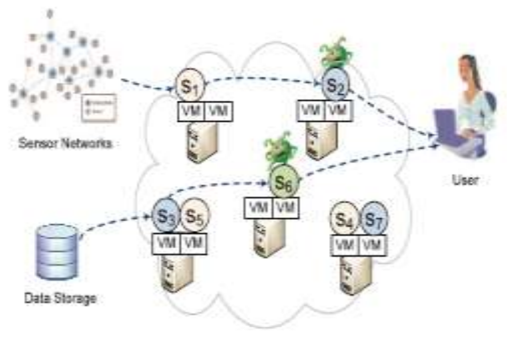
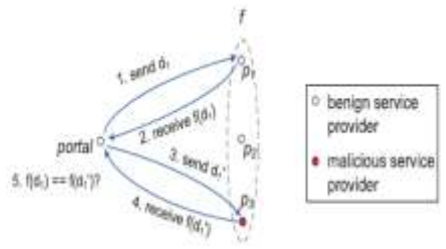


Fig. 1. **Service integrity attack in cloud-based data processing. Si denotes different service component and VM denotes virtual machines**

The per-function consistency raph analysis can limit the scope of damage caused by colluding attackers, while the global inconsistency graph analysis can effectively expose those attackers that try to compromise many service functions. Hence, IntTest can still pinpoint malicious attackers even if they become majority for some service functions. By taking an integrated approach, IntTest can not only pinpoint attackers more efficiently but also can suppress aggressive attackers and limit the scope of the damage caused by colluding attacks. Moreover, IntTest provides result auto correction that can automatically replace corrupted data processing results produced by malicious attackers with good results produced by benign service providers. Specifically, this paper makes the following contributions:

- We provide a scalable and efficient distributed service integrity attestation framework for large scale cloud computing infrastructures.
- We present a novel integrated service integrity attestation scheme that can achieve higher pinpointing accuracy than previous techniques.



- We describe a result auto correction technique that can automatically correct the corrupted results produced by malicious attackers.
- We conduct both analytical study and experimental evaluation to quantify the accuracy and overhead of the integrated service integrity attestation scheme.

We have implemented a prototype of the IntTest system and tested it on NCSU’s virtual computing lab (VCL) [8], a production cloud computing infrastructure that operates in a similar way as the Amazon elastic compute cloud (EC2) [9]. The benchmark applications we use to evaluate IntTest are distributed data stream processing services provided by the IBM System S stream processing platform [8], [3], an industry strength data stream processing system. Experimental results show that IntTest can achieve more accurate pinpointing than existing schemes (e.g., RunTest, AdapTest, and full-time majority voting) under strategically colluding attacks. IntTest is scalable and can reduce the attestation overhead by more than one order of magnitude compared to thetraditional full-time majority voting scheme.

II. PROBLEM STATEMENT

Given a SaaS cloud system, the goal of IntTest is to pinpoint any malicious service provider that offers an untruthful service function. IntTest treats all service components as black boxes, which does not require any special hardware or secure kernel support on the cloud platform. We now describe our attack model and our key assumptions as follows:

Attack model. A malicious attacker can pretend to be a legitimate service provider or take control of vulnerable service providers to provide untruthful service functions. Malicious attackers can be stealthy, which means they can misbehave on a selective subset of input data or service functions while pretending to be benign service providers on other input data or functions. The stealthy behavior makes detection more challenging due to the following reasons:

- The detection scheme needs to be hidden from the attackers to prevent attackers from gaining knowledge on the set of data processing results that will be verified and therefore easily escaping detection; and

- The detection scheme needs to be scalable while being able to capture misbehavior that may be both unpredictable and occasional.

In a large-scale cloud system, we need to consider colluding attack scenarios where multiple malicious attackers collude or multiple service sites are simultaneously compromised and controlled by a single malicious attacker. Attackers could sporadically collude, which means an attacker can collude with an arbitrary subset of its colluders at any time. We assume that malicious nodes have no knowledge of other nodes except those they interact with directly. However, attackers can communicate with their colluders in an arbitrary way. Attackers can also change their attacking and colluding strategies arbitrarily. Assumptions we first assume that the total number of malicious service components is less than the total number of benign ones in the entire cloud system. Without this assumption, it would be very hard, if not totally impossible, for any attack detection scheme to work when comparable ground truth processing results are not available.

Fig.2. Replay-based consistency check.

However, different from RunTest, AdapTest, or any previous majority voting schemes, IntTest does not assume benign service components have to be the majority for every service function, which will greatly enhance our pinpointing power and limit the scope of service functions that can be compromised by malicious attackers. Second, we assume that the data processing services are input-deterministic, that is, given the same input, a benign service component always produces the same or similar output (based on a user-defined similarity function). Many data stream processing functions fall into this category [8]. We can also easily extend our attestation framework to support stateful data processing services [8], which however is outside the scope of this paper. Third, we also assume that the result inconsistency caused by hardware or software faults can be marked by fault detection schemes [3] and are excluded from our malicious attack detection.

III. RELATED WORK

To detect service integrity attack and pinpoint malicious service providers, our algorithm relies on replay-based consistency check to derive the consistency/inconsistency relationships between service providers. For example, Fig. 2 shows the consistency check scheme for attesting three service provider's p1, p 2, and p 3 that offer the same service function f. The portal sends the original input data d1 to p1 and gets back the result f(d1). Next, the portal sends d0, a duplicate of d1 to p3 and gets back the result f(d_0).

The portal then compares f(d1) and f(d0) to see whether p1 and p3 are consistent. The intuition behind our approach is that if two service providers disagree with each other on the processing result of the same input, at least one of them should be malicious. Note that we do not send an input data item and its duplicates (i.e., attestation data) concurrently. Instead, we replay the attestation data on different service providers after receiving the processing result of the original data. Thus, the malicious attackers cannot avoid the risk of being detected when they produce false results on the original data. Although the replay scheme may cause delay in a single tuple processing, we can overlap the attestation and normal processing of consecutive tuples in the data stream to hide the attestation delay from the user. If two service providers always give consistent output results on all input data, there exists consistency relationship between them. Otherwise, if they give different outputs on at least one input data, there is inconsistency relationship between them. We do not limit the consistency relationship to equality function since two benign service providers may produce similar but not exactly the same results. For example, the credit scores for the same person may vary by a small difference when obtained from different credit bureaus. We allow the user to define a distance function to quantify the biggest tolerable result difference.

Definition 1. For two output results, r1 and r2, which come from two functionally equivalent service providers, respectively, result consistency is defined as either r1 = r2, or the distance between r1 and r2 according to user-defined distance function D(r1,r2) falls within a threshold ɗ.

For scalability, we propose randomized probabilistic attestation, an attestation technique that randomly replays a subset of input data for attestation. For composite data-flow processing services consisting of multiple service hops, each service hop is composed of a set of unction ally equivalent service providers. Specifically, for an upcoming tuple $\boldsymbol{d_i}$, the portal may decide to perform integrity attestation with probability $\boldsymbol{p_u}$. If the portal decides to perform attestation on $\boldsymbol{d_i}$, the portal first sends di to a pre-defined service path $p_1 \longrightarrow p_2 \cdots \longrightarrow p_l$ providing functions $f_1 \longrightarrow f_2 \cdots \longrightarrow f_l$. After receiving the processing result for $\boldsymbol{d_i}$, the portal replays the duplicates of $\boldsymbol{d_i}$, on alternative service path(s) such as $\acute{p}_1 \longrightarrow \acute{p}_2 \cdots \longrightarrow \acute{p}_j$ providing functions f_j as $\acute{p}_j$. The portal may perform data replay on multiple service providers to perform concurrent attestation.

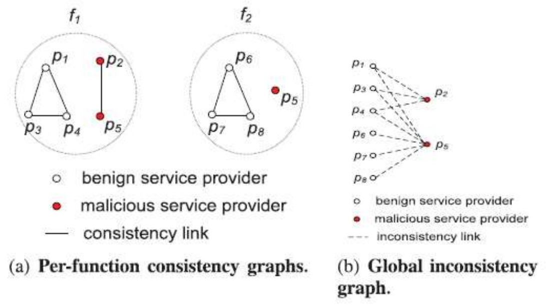

f1
p1
p2
p3
p4
p5
f2
p6
p5
p7
p8
benign service provider
malicious service provider
consistency link
(a) Per-function consistency graphs.
p1
p3
p4
p6
p7
p8
p2
p5
benign service provider
malicious service provider
inconsistency link
(b) Global inconsistency graph.

Fig. 3. Attestation graphs.

With replay-based consistency check, we can test functionally equivalent service providers and obtain their consistency and inconsistency relationships. Fig.3. Attestation graphs both the we employ consistency graph and inconsistency graph to aggregate pairwise attestation results for further analysis. The graphs reflect the consistency/inconsistency relationships across multiple service providers over a period of time. Before introducing the attestation graphs, we first define consistency links and inconsistency links.

Definition 2. A consistency link exists between two service providers who always give consistent output for the same input data during attestation. An inconsistency link exists between two service providers who give at least one inconsistent output for the same input data during attestation.

We then construct consistency graphs for each function to capture consistency relationships among the service providers provisioning the same function. Fig 3 (a) shows the consistency graphs for two functions. Note that two service providers that are consistent for one function are not necessarily consistent for another function. This is the reason why we confine consistency graphs within individual functions.

Definition 3. A per-function consistency graph is an undirected graph, with all the attested service providers that provide the same service function as the vertices and consistency links as the edges.

We use a global inconsistency graph to capture inconsistency relationships among all service providers. Two service providers are said to be inconsistent as long as they disagree in any function. Thus, we can derive more comprehensive inconsistency relationships by integrating inconsistency links across functions. Fig. 3(b) shows an example of the global inconsistency graph. Note that service provider p5 provides both functions f1 and f2. In the inconsistency graph, there is a single node p5 with its links reflecting inconsistency relationships in both functions f1 and f2.

Definition 4. The global inconsistency graph is an undirected graph, with all the attested service providers in the system as the vertex set and inconsistency links as the edges. The portal node is responsible for constructing and maintaining both per-function consistency graphs and the global inconsistency graph. To generate these graphs, the portal maintains counters for the number of consistency results and counters for the total number of attestation data between each pair of service providers.

IV. RESULTS AND ANALYSIS

We first investigate the accuracy of our scheme in pinpointing malicious service providers. Fig. 4(a) compares our scheme with the other alternative schemes (i.e., FTMV, PTMV, and RunTest) when malicious service providers aggressively attack different number of service functions. Inthis set of experiments, we have 10 service functions and 30 service providers. The number of service providers in each service function randomly ranges in [1, 8].

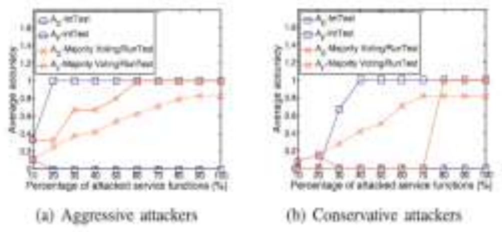


Fig 4.Malicious attackers pinpointing accuracy comparison with 20 percent service providers being malicious.

Each benign service provider provides two randomly selected service functions. The data rate of the input stream is 300 tuples per second. We set 20 percent of service providers as malicious. After the portal receives the processing result of a new data tuple, it randomly decides whether to perform data attestation. Each tuple has 0.2 probability of getting attested (i.e., attestation probability P_u ¼ 0:2), and two attestation data replicas are used (i.e., number of total data copies including the original data r ¼ 3). Each experiment is repeated three times. We report the average detection rate and false alarm rate achieved by different schemes. Note that RunTest can achieve the same detection accuracy results as the majority voting based schemes after the randomized probabilistic attestation covers all attested service providers and discovers the majority clique [6]. In contrast, IntTest comprehensively examines both perfection consistency graphs and the global inconsistency graph to make the final pinpointing decision. We observe that IntTest can achieve much higher detection rate and lower false alarm rate than other alternatives. Moreover, IntTest can achieve better detection accuracy when malicious service providers attack more functions. We also observe that when malicious service providers attack aggressively, our scheme can detect them even though they attack a low percentage of service functions Fig. 4(b) shows the malicious service provider detection accuracy results under the conservative attack scenarios. All the other experiment parameters are kept the same as the previous experiments. The results show that IntTest can consistently achieve higher detection rate and lower false alarm rate than the other alternatives. In the conservative attack scenario, as shown by fig. 4(b), the false alarm rate of IntTest first increases when a small percentage of service functions are attacked and then drops to zero quickly with more service functions are attacked. This is because when attackers only attack a few service functions where they can take majority; they can hide themselves from our detection scheme while tricking our algorithm into labeling benign service providers as malicious. However, if they attack more service functions, they can be detected since they incur more inconsistency links with benign service providers in the global inconsistency graph. Note that majority voting-based schemes can also detect malicious attackers if attackers fail to take majority in the attacked service function. However, majority voting-based schemes have high false alarms since attacks can always trick the schemes to label benign service providers as malicious as long as attackers can take majority in each individual service function

V. CONCLUSION

In this paper, we have presented the design and implementation of IntTest, a novel integrated service integrity attestation framework for multitenant software-as-a-service cloud systems. IntTest employs randomized replay-based consistency check to verify the integrity of distributed service components without imposing high overhead to the cloud infrastructure. IntTest performs integrated analysis over both consistency and inconsistency attestation graphs to pinpoint colluding attackers more efficiently than existing techniques. Furthermore, IntTest provides result autocorrect ion to automatically correct compromised results to improve the result quality. We have implemented IntTest and tested it on a commercial data stream processing platform running inside a production virtualized cloud computing infrastructure. Our experimental results show that IntTest can achieve higher pinpointing accuracy than existing alternative schemes. IntTest is lightweight, which imposes low-performance impact to the data processing services running inside the cloud computing infrastructure.

REFERENCES

[1] Amazon Web Services, http://aws.amazon.com/, 2013.

[2] Google App Engine, http://code.google.com/appengine/, 2013.

[3] Software as a Service, http://en.wikipedia.org/wiki/Software as a Service, 2013.

[4] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, Web Services Concepts, Architectures and Applications (Data-Centric Systems and Applications). Addison-Wesley Professional, 2002.

[5] T. Erl, Service-Oriented Architecture (SOA): Concepts, Technology, and Design. Prentice Hall, 2005.

[6] T.S. Group, "STREAM: The Stanford Stream Data Manager," IEEE Data Eng. Bull., vol. 26, no. 1, pp. 19-26, Mar. 2003.

[7] D.J. Abadi et al., "The Design of the Borealis Stream Processing
Engine," Proc. Second Biennial Conf. Innovative Data Systems Research (CIDR '05), 2005.

[8] B. Gedik et al., "SPADE: The System S Declarative Stream Processing Engine," Proc. ACM SIGMOD Int'l Conf. Management Of Data (SIGMOD '08), Apr. 2008.

[9] S. Berger et al., "TVDc: Managing Security in the Trusted Virtual
Datacenter," ACM SIGOPS Operating Systems Rev., vol. 42, no. 1, pp. 40-47, 2008.

[10] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, You Get Off My Cloud! Exploring Information Leakage in Third-Party Compute Clouds," Proc. 16th ACM Conf. Computer and Communications Security (CCS), 2009.

[11] W. Xu, V.N. Venkatakrishnan, R. Sekar, and I.V. Ramakrishnan,"A Framework for Building Privacy-Conscious Composite Web Services," Proc. IEEE Int'l Conf. Web Services, pp. 655-662, Sept. 2006.

[12] P.C.K. Hung, E. Ferrari, and B. Carminati, "Towards Standardized Web Services Privacy Technologies," IEEE Int'l Conf. Web Services, pp. 174-183, June 2004.

[13] L. Alchaal, V. Roca, and M. Habert, "Managing and Securing Web Services with VPNs," Proc. IEEE Int'l Conf. Web Services, pp. 236- 243, June 2004.

[14] H. Zhang, M. Savoie, S. Campbell, S. Figuerola, G. von Bochmann, and B.S. Arnaud, "Service-Oriented Virtual Private Networks for Grid Applications," Proc. IEEE Int'l Conf. Web Services, pp. 944-951, July 2007.

[15] M. Burnside and A.D. Keromytis, "F3ildCrypt: End-to-End Protection of Sensitive Information in Web Services," Proc. 12th Int'l Conf. Information Security (ISC), pp. 491-506, 2009.

[16] I. Roy et al., "Airavat: Security and Privacy for MapReduce," Proc. Seventh USENIX Conf. Networked Systems Design and Implementation (NSDI), Apr. 2010.