IPHV8I1051X
International Journal Of Advanced Research and Innovation -Vol.8, Issue .I
ISSN Online: 2319 – 9253
Print: 2319 – 9245

PROVIDING DISTRIBUTED SERVICE INTEGRITY ATTESTATION FOR SAAS CLOUD

[#1]CH.HARSHINI, M.Tech Student,
[#2]DR.S.DURGA BHAVANI, Professor,
Dept of Computer Science,
SCHOOL OF INFORMATION TECHNOLOGY, JNTU-HYDERABAD, TELANGANA, INDIA.

ABSTRACT— A cloud computing is the recently emerging technology which can provide many service based on "pay as you go" way that can be accessed through internet. The services provided by the cloud are software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS). Software as a service can provide their applications from the application service provide through massive cloud computing environment. Due to the sharing nature ,it is vulnerable to malicious attacker. To identify the malicious, there are many techniques that can be compared through the survey with and without any special hardware or kernel support. SaaS clouds are vulnerable to malicious attacks because of their sharing nature. Many security frameworks have been developed to address cloud security issues like IntTest, Privacy Proxy, Trusted virtual data center, Placement and Extraction method for Exploring Information Leakage, Stateful Dataflow Processing, Building Privacy-Conscious Composite Web Services, Anomaly Extraction and Mitigation using Efficient- Web Miner Algorithm. Brief Study on the above frame works are explained below.

***Keywords*—:** *Distributed Service, Integrity attestation, Cloud computing, Multitenanat.*

I. INTRODUCTION

Cloud computing is a technology helps us to keep up data and its application by using internet and central remote servers [18]. Cloud computing has greater flexibility and availability at lower cost. The four deployment models operated by cloud computing are the: Public Cloud, Private Cloud, Community Cloud, and Hybrid Cloud. Private cloud -- The cloud infrastructure is operated solely for an organization. It may be managed by the organization or a third party and may exist on premise or off premise. Community cloud -- The cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns. It may be managed by the organizations or a third party and may exist on premise or off premise. Public cloud -- The cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling the cloud services and the comparison of private and public cloud. Hybrid cloud -- The cloud infrastructure is a composition of two or more clouds (private, community, or public). There are different types of cloud service providers like Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS).Here we are discussing about SaaS Cloud system.

Software as a Service (SaaS) is a software distribution model in which applications are hosted by a vendor or service provider and made this is available to customers over a network.SaaS is becoming an increasingly prevalent delivery model as underlying technologies that support Web services and service-oriented architecture (SOA) and many other new developmental approaches. SaaS service are suffered from many malicious attacks hence they need security. Below are the various frameworks proposed to provide security.

Here, we present IntTest, a novel integrated service integrity attestation framework for multitenant cloud systems. IntTest does not assume trusted entities on third-party service provisioning sites or require application modifications rather provides a practical service integrity attestation scheme. IntTest builds upon previous work RunTest and AdapTest but can provide stronger malicious attacker Pinpointing power than previous tests. Specifically, both RunTest and AdapTest as well as traditional majority voting schemes need to assume that benign service providers take majority in every service function, assumption makes the test easier to get the solution. To invalidate this assumption multiple malicious attackers may launch colluding attacks on certain targeted service functions, in large-scale multitenant cloud. In order to overcome this, IntTest takes a holistic approach by systematically examining both consistency and inconsistency relationships among different service providers within the entire cloud system. The per-function consistency graph analysis can limit the scope of damage which is caused by colluding attackers, while the global inconsistency graph analysis can effectively show those attackers that try to compromise 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 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 that are produced by malicious attackers.
- We conduct both analytical study and experimental results to quantify the accuracy and overhead of the scheme.

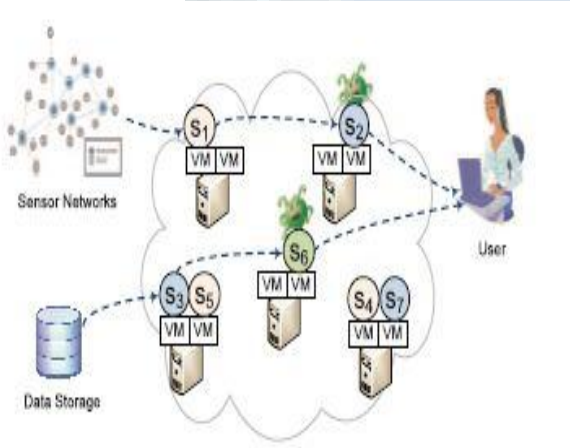

Fig. 1: Service integrity attack in cloud-based data processing.

II. PRELIMINARY

Here we introduce the software-as-a service (SaaS) cloud system model. Then we describe our problem formulation including the service integrity attack model and our key assumptions.

2.1 SaaS Cloud System Model

It develops upon the concepts of Software as a Service (SaaS) and Service Oriented Architecture (SOA) which allows application service providers (ASPs) to deliver their applications via large-scale cloud computing infrastructures. Amazon Web Service (AWS) and Google App Engine are examples to provide a set of application services supporting enterprise applications and big data processing. A distributed application service can be dynamically composed from individual service components provided by different ASPs. For example, a disaster assistance claim processing application consists of voice-over-IP (VoIP) analysis component, email analysis component, community discovery Component, and clustering and joins components. Our work focuses on data processing services which have become increasingly popular with applications in any real world usage domains such as business intelligence, security surveillance, and scientific computing. Each service component, denoted by ci, provides a specific data processing function, denoted by fi, such as sorting, filtering, correlation, or data mining utilities. Each service component can have one or more input ports for receiving input data tuples, denoted by di, and one or more output ports to emit output tuples. In a large-scale SaaS cloud, the same service function can be provided by different ASPs. Those functionally equivalent service components exist because: i) service providers may create replicated service components for load balancing and fault tolerance purposes; and ii) popular services may attract different service providers for profit. To support automatic service composition, we can deploy a set of portal nodes that serve as the gateway for the user to access the composed services in the SaaS cloud. The portal node can aggregate different service components into composite services based on the user's requirements. For security protection, the portal node can perform authentication on users to avoid malicious users from disturbing normal service provisioning. Different from other open distributed systems such as peer-to-peer networks and volunteer computing environments, SaaS cloud systems possess a set of unique features. First, third-party ASPs typically do not want to reveal the internal implementation details of their software services for intellectual property protection. Thus, it is difficult to only rely on challenge-based attestation scheme where the verifier is assumed to have certain knowledge about the software implementation or have access to the software source code. Second, both the cloud infrastructure provider and third-party service providers are autonomous entities. It is impractical to impose any special hardware or secure kernel support on individual service provisioning sites. Third, for privacy protection, only portal nodes have global information about which service functions are provided by which service providers in the SaaS cloud. Neither cloud users nor individual ASPs have the global knowledge about the SaaS cloud such as the number of ASPs and the identifiers of the ASPs offering a specific service function.

2.2 Problem Formulation

For a given SaaS 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

IPHV8I1051X
International Journal Of Advanced Research and Innovation -Vol.8, Issue .I
ISSN Online: 2319 – 9253
Print: 2319 – 9245

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:

1) 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;

2) 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.

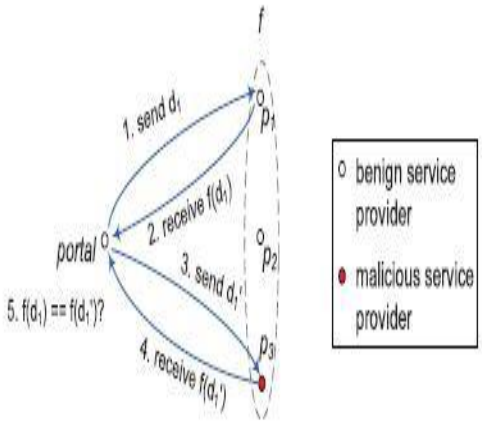

Fig. 2. Replay-based consistency check

Assumptions:

1. 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. 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.

2. 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. We can also easily extend our attestation framework to support stateful data processing services, 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 and are excluded from our malicious attack detection.

III. ATTACK MODEL

In a shared cloud infrastructure, malicious attackers can pretend to be legitimate service providers to give fake service instances or compromise vulnerable benign service instances by exploiting their security roles. It focuses on detecting the service integrity attack where a malicious (or compromised) service instance gives deceptive data processing results. To escape detection, malicious attackers may want to perform selective cheating. That is, they can misbehave on a selective subset of received data while pretending to be benign on other received data. Thus, the attack detection scheme must be able to capture misbehavior that are both unpredictable and occasional without losing scalability. Although we can perform integrity attestation on all service instances all the time, the overhead of integrity attestation would be very high, especially for high throughput data processing services in large-scale cloud systems. Thus, an effective attack detection scheme must perform sneaky attestation, which can prevent attackers from gaining knowledge about our attestation scheme (i.e., when and which set of data will be attested.). Otherwise, the attacker can compromise the integrity of selective data processing results without being detected at all.

IV. DESIGN AND ALGORITHMS

In this section, we first present the basis of the IntTest system: probabilistic replay-based consistency check and the integrity attestation graph model. We then describe the integrated service integrity attestation scheme in detail. Next, we present the result auto-correction scheme.

4.1 Baseline Attestation Scheme

In order to detect service integrity attack and pinpoint malicious service providers, our algorithm relies on replay-based consistency check to derive the 4 consistency/inconsistency relationships between service providers. For example, Figure 2 shows the consistency check scheme for attesting three service providers p1, p2,

and p3 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 d′ 1, a duplicate of d1 to p3 and gets back the result f(d′ 1). The portal then compares f(d1) and f(d′ 1) 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 _. For scalability, we propose randomized probabilistic attestation, an attestation technique that randomly replays a subset of input data for attestation. For composite dataflow processing services consisting of multiple service hops, each service hop is composed of a set of functionally equivalent service providers. Specifically, for an incoming toppled, the portal may decide to perform integrity attestation with probability put. If the portal decides to perform attestation on di, the portal first sends di to a predefined service path p1 → p2... → pl providing functions f1 → f2... → fl. After receiving the processing result for di, the portal replays the duplicate(s) of di on alternative service path(s) such as p′ 1 → p′ 2... → p′ l, where p′ j provides the same function fj as pj. The portal may perform data replay on multiple service providers to perform concurrent attestation. After receiving the attestation results, the portal compares each intermediate result between pairs of functionally equivalent service providers pi and p′ i. If pi and p′ i receive the same input data but produce different output results, we say that pi and p′ i are inconsistent. Otherwise, we say that pi and p′ i are consistent with regard to function fi. For example, let us consider two different credit score service providers p1 and p′ 1. Suppose the distance function is defined as two credit score difference is no more than 10. If p1 outputs 500 and p′ 1 outputs 505 for the same person, we say p1 and p′ 1 are consistent. However, if p1 outputs 500 and p′ 1 outputs 550 for the same person, we would consider p1 and p′ 1 to be inconsistent. We evaluate both intermediate and final data processing results between functionally equivalent service providers to derive the consistency/inconsistency relationships. For example, if data processing involves a sub-query to a database, we evaluate both the final data processing result along with the intermediate sub-query result. Note that although we do not attest all service providers at the same time, all service providers will be covered by the randomized probabilistic attestation over a period of time. With replay-based consistency check, we can test functionally equivalent service providers and obtain their consistency and inconsistency relationships. We employ both the consistency graph and inconsistency graph to aggregate pair-wise 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. Figure 4(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

IPHV8I1051X
International Journal Of Advanced Research and Innovation -Vol.8, Issue .I
ISSN Online: 2319 – 9253
Print: 2319 – 9245

relationships by integrating inconsistency links across functions. Figure 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.

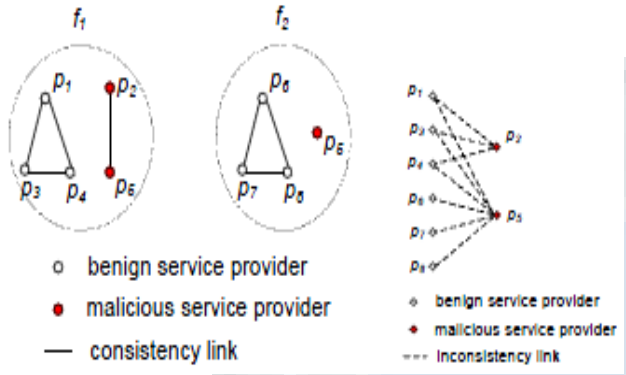

Fig. 3. Attestation graphs.

Definition4: 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. In order 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. The portal updates the counters each time when it receives attestation results. At any time, if the counter for consistency results has the same value with that for the total attestation data, there is a consistency link between this pair of service providers. Otherwise, there is an inconsistency link between them.

4.2 Integrated Attestation Scheme

We now present our integrated attestation graph analysis algorithm.

Step 1: Consistency graph analysis.

We first examine per-function consistency graphs to pinpoint suspicious service providers. The consistency links in per-function consistency graphs can tell which set of service providers keep consistent with each other on a specific service function. Given any service function, since benign service providers always keep consistent with each other, benign service providers will form a clique in terms of consistency links. For example, in Figure 3(a), p1, p3 and p4 are benign service providers and they always form a consistency clique. In our previous work [26], we have developed a clique based algorithm to pinpoint malicious service providers. If we assume that the number of benign service providers is larger than that of the malicious ones, a benign node will always stay in a clique formed by all benign nodes, which has size larger than $\lfloor k/2 \rfloor$, where k is the number of service providers provisioning the service function. Thus, we can pinpoint *suspicious* nodes by identifying nodes that are outside of all cliques of size larger than $\lfloor k/2 \rfloor$. For example, in Figure 3(a), p2 and p5 are identified as suspicious because they are excluded from the clique of size 3.

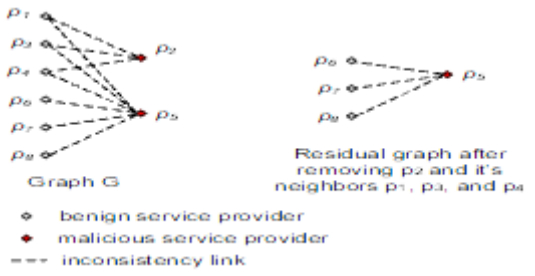

Fig. 4. Inconsistency graph G and its residual graph.

However, strategically colluding attackers can try to take majority in a specific service function to escape the detection. Thus, it is insufficient to examine the per function consistency graph only. We need to integrate the consistency graph analysis with the inconsistency graph analysis to achieve more robust integrity attestation.

Step 2: Inconsistency graph analysis. Given an inconsistency graph containing only the inconsistency links, there may exist different possible combinations of the benign node set and the malicious node set. However, if we assume that the total number of malicious service providers in the whole system is no more than K, we can pinpoint a subset of truly malicious service providers. Intuitively, given two service providers connected by an inconsistency link, we can say that at least one of them is malicious since any two benign service providers should always agree with each other. Thus, we can derive the lower bound about the number of malicious service providers by examining the minimum vertex cover of the inconsistency graph. The minimum vertex cover of a graph is a minimum set of vertices such that each edge of the graph is incident to at least one vertex in the set. For example, in Figure 3(b), p2 and p5 form the minimum vertex cover. We present two propositions as part of our approach. The proofs for these propositions can be found in section 1 of the online supplementary material.

Proposition 1: Given an inconsistency graph G, let CG be a minimum vertex cover of G. Then the number of malicious service providers is no less than |CG|. We now define the residual inconsistency graph for a node pi as follows.

Definition 5: The *residual inconsistency graph* of node pi is the inconsistency graph after removing the node pi and all of links adjacent to pi. For example, Figure 4 shows the residual inconsistency graph after removing the node p2. Based on the lower bound of the number of malicious

IPHV8I1051X
International Journal Of Advanced Research and Innovation -Vol.8, Issue .I
ISSN Online: 2319 – 9253
Print: 2319 – 9245

service providers and Definition 5, we have the following proposition for pinpointing a subset of malicious nodes.

Proposition 2: Given an integrated inconsistency graph G and the upper bound of the number of malicious service providers K, a node p must be a malicious service provider if and only if $|Np| + |CG' p| > K$ (1)

where |Np| is the neighbor size of p, and |CG′ p | is the size of the minimum vertex cover of the *residual* inconsistency graph after removing p and its neighbors from G. For example, in Figure 3(b), suppose we know the number of malicious service providers is no more than 2. Let us examine the malicious node p2 first. After we remove p2 and its neighbors p1, p3, and p4 from the inconsistency graph, the residual inconsistency graph will be a graph without any link. Thus, its minimum vertex cover is 0. Since p2 has three neighbors, we have 3 + 0 > 2. Thus, p2 is malicious. Let us now check out the benign node p1. After removing p1 and its two neighbors p2 and p5, the residual inconsistency graph will be a graph without any link and its minimum vertex cover should be 0. Since p1 has two neighbors, Equation 1 does not hold. We will not pinpoint p1 as malicious in this step. Note that benign service providers that do not serve same functions with malicious ones will be isolated nodes in the inconsistency graph, since they will not be involved in any inconsistency links. For example, in Figure 5, nodes p4, p5, p6 and p7 are isolated nodes since they are not associated with any inconsistency links in the global inconsistency graph. Thus, we can remove these nodes from the inconsistency graph without affecting the computation of the minimum vertex cover. We now describe how to estimate the upper bound of the number of malicious service providers K. Let N denote the total number of service providers in the system. Since we assume that the total number of malicious service providers is less than that of benign ones, the number of malicious service providers should be no more than ⌊N/2⌋. According proposition 1, the number of malicious service providers should be no less than the size of the minimum vertex cover |CG| of the global inconsistency graph. Thus, K is first bounded by its lower bound |CG| and upper bound ⌊N/2⌋. We then use an iterative algorithm to tighten the bound of K. We start from the lower bound of K, and compute the set of malicious nodes, as described by Proposition 2, denoted by . Then we gradually increase K by one each time. For each specific value of K, we can get a set of malicious nodes. With a larger K, the number of nodes that can satisfy |Ns|+|CG′ s | > K becomes less, which causes the set to be reduced. When = ∅, we stop increasing K, since any larger K cannot give more malicious nodes. Intuitively, when K is large, fewer nodes may satisfy Equation 1. Thus, we may only identify a small subset of malicious nodes. In contrast, when K is small, more nodes may satisfy Equation 1, which may mistakenly pinpoint benign nodes as malicious. To avoid false positives, we want to pick a large enough K, which can pinpoint a set of true malicious service providers.

Step 3: Combining consistency and inconsistency graph analysis results.

Let Gi be the consistency graph generated for service function fi, and G be the global inconsistency graph. Let Mi denote the list of suspicious nodes by analyzing per function consistency graph Gi (i.e., nodes belonging to minority cliques), and denotes the list of suspicious nodes by analyzing the global inconsistency

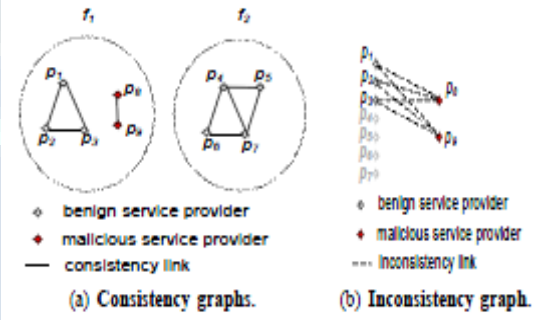

Fig. 5. Isolated benign service providers in the global inconsistency graph.

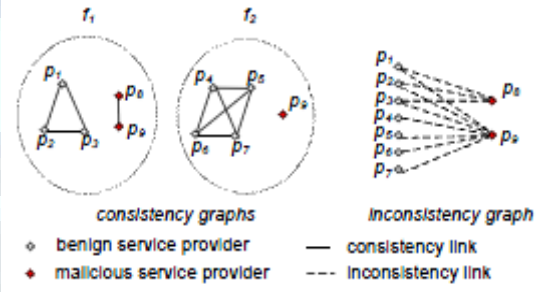

Fig. 6. An example for integrated graph analysis with two malicious nodes.

graph G, given a particular upper bound of the number of malicious nodes K. We examine per-function consistency graphs one by one. Let i denote the subset of that serves function fi. If i ∩ Mi 6= ∅, we add nodes in Mi to the identified malicious node set. The idea is that since the majority of nodes serving function fi have successfully excluded malicious nodes in i, we could trust their decision on proposing Mi as malicious nodes. Pseudo-code of our algorithm can be found in section 1 of the online supplemental material. For example, Figure 6 shows both the per-function consistency graphs and the global inconsistency graph. If the upper bound of the malicious nodes K is set to 4, the inconsistency graph analysis will capture the malicious node p9 but will miss the malicious node p8. The reason is that p8 only has three neighbors and the minimum vertex cover for the residual inconsistency

IPHV8I1051X
International Journal Of Advanced Research and Innovation -Vol.8, Issue .I
ISSN Online: 2319 – 9253
Print: 2319 – 9245

graph after removing p8 and its three neighbors is 1. Note that we will not pinpoint any benign node as malicious according to Proposition 2. For example, the benign node p1 has two neighbors and the minimum vertex cover for the residual graph after removing p1 and its two neighbors p8 and p9 will be 0 since the residual graph does not include any link. However, by checking the consistency graph of function f1, we find 1 = {p9} has overlap with the minority clique M1 = {p8, p9}. We then infer p8 should be malicious too. Note that even if we have an accurate estimation of the

number of malicious nodes, the inconsistency graph analysis scheme may not identify *all* malicious nodes. However, our integrated algorithm can pinpoint more malicious nodes than the inconsistency graph only algorithm. An example showing how our algorithm can pinpoint more malicious nodes than the inconsistency graph only algorithm can be found in section 1 of the online supplemental material.

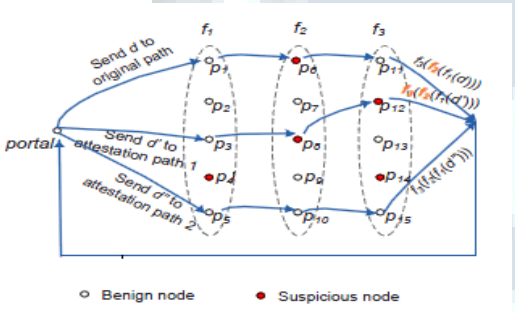

Fig.7. Automatic data correction using attestation data processing results.

4.3 Result Auto-Correction

IntTest can not only pinpoint malicious service providers but also automatically correct corrupted data processing results to improve the result quality of the cloud data processing service, illustrated by Figure 7. Without our attestation scheme, once an original data item is manipulated by any malicious node, the processing result of this data item can be corrupted, which will result in degraded result quality. IntTest leverages the attestation data and the malicious node pinpointing results to detect and correct compromised data processing results. Specifically, after the portal node receives the result f(d) of the original data d, the portal node checks whether the data d has been processed by any malicious node that has been pinpointed by our algorithm. We label the result f(d) as "suspicious result" if d has been processed by any pinpointed malicious node. Next, the portal node checks whether d has been chosen for attestation. If d is selected for attestation, we check whether the attestation copy of d only traverses good nodes. If it is true, we will use the result of the attestation data to replace f(d). For example, in Figure 7, the original data d is processed by the pinpointed malicious node s6 while one of its attestation data d is only processed by benign nodes. The portal node will use the attestation data result f(d′′) to replace the original result that can be corrupted if s6 cheated on d.

V.CONCLUSION

In this paper, we have presented the design and implementation of IntTest, a novel integrated service integrity attestation framework for multi-tenant 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 auto correction to automatically correct compromised results to improve the result quality. We have implemented IntTest 10 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 light-weight, which imposes low performance impact to the data processing processing services running inside the cloud computing infrastructure.

REFERENCES

[1] Garay.J and Huelsbergen.L, "Software integrity protection using timed executable agents," in Proceedings of ACM Symposium on Information, Computer and Communications Security (ASIACCS), Taiwan, Mar. 2006.
[2] Juan Du Daniel J. Dean, Yongmin Tan, Xiaohui Gu, Senior and Ting Yu Scalable Distributed Service Integrity Attestation for Softwareas- a-Service Clouds .
[3] Du.J, Wei.W, Gu.X, and Yu.T, "Runtest: Assuring Integrity of Dataflow Processing in Cloud Computing Infrastructures," Proc.ACM Symp. Information, Computer and Comm. Security (ASIACCS),2010.
[4] Du.J, Shah.N, and Gu.X, "Adaptive Data-Driven Service Integrity Attestation for Multi-Tenant Cloud Systems," Proc. Int'l Workshop Quality of Service (IWQoS), 2011. Virtual Computing Lab, http://vcl.ncsu.edu/, 2013.
[5] Ho et al.T, "Byzantine Modification Detection in Multicast Networks Using Randomized Network Coding," Proc. IEEE Int'l Symp. Information Theory (ISIT), 2004.
[6] Hwang.I, "A Survey of Fault Detection, Isolation, and Reconfiguration Methods," IEEE Trans. Control System Technology, vol. 18,no. 3, pp. 636-653, May 2010.

[7] Lamport.L, Shostak.R, and Pease.M, "The Byzantine Generals Problem," ACM Trans. Programming Languages and Systems, vol. 4,no. 3, pp. 382-401, 1982

[8] Shi.E, Perrig.A, and Doorn.L.V, "Bind: A fine-grained attestation service for secure distributed systems," in Proceedings of the IEEE Symposium on Security and Privacy, 2005.

[9] Xu.W, Venkatakrishnan.V. N, Sekar.R, and Ramakrishnan .I. V, "A framework for building privacy-conscious composite web services," in IEEE International Conference on Web Services, Chicago, IL, Sep. 2006, pp. 655–662.

[10] Zhang.H, Savoie.M,Campbell.S, Figuerola.S, von Bochmann.G, and Arnaud.B.S, "Service-oriented virtual private networks for grid applications," in IEEE International Conference on Web Services, Salt Lake City, UT, Jul. 2007, pp. 944–951.