Disruption-Resilient Publish and Subscribe

Noor Ahmed[1,2], and Bharat Bhargava[1]
[1]*Purdue University*
[2]*Air Force Research Laboratory/RI*
{ahmed24, bbshail}@purdue.edu

Keywords: Cloud Computing, OpenStack, Byzantine Fault Tolerant, Publish and Subscribe Middleware.

Abstract: Publish and Subscribe (pub/sub) dissemination paradigm has emerged as a popular means of disseminating selective time-sensitive information. Through the use of event service or *broker*, published information is filtered to disseminate only to the subscribers interested in that information. Once a *broker* is compromised, information can be delivered unfiltered, dropped, delayed, perhaps colluding among the *brokers* in virtualized cloud platforms. Such disruptive behavior is known as Byzantine faults. We present a Disruption–Resilient Publish and Subscribe (DRPaS) system designed to withstand faults through continuously refreshing the virtual instances of the broker. DRPaS combines advances in cloud management software stack *(i.e., OpenStack nova and neutron)* to control the *broker's* susceptibility window of disruption. Preliminary experimental results show that the defensive security solutions enabled by the underlying cloud computing fabric is simpler and more effective than the ones implemented at the application/protocol level to withstand disruptions.

1 INTRODUCTION

Publish and Subscribe (pub/sub) dissemination paradigm has emerged as a popular means of disseminating filtered messages across large numbers of subscribers and publishers. It's a dissemination paradigm that has attracted many applications: financial trading systems, cloud infrastructures to interconnect components, and distributed clustering (i.e. RabitMQ), as service buses used in Service Oriented Architectures, Yahoo Message Borker, OracleJMS, IBM-Websphere, Jboss, and many others.

In pub/sub, typically, a broker(s) mediates the exchange of topic or content-based messages between the producers (publishers) and consumers (subscribers). Subscribers register their topic of interest to the broker in which is then filtered against the incoming messages from the publishers and forwarded to them upon match, thereby, eliminating the need for a priori connection between the message publishers and subscribers.

However, such many-to-many loose coupling data sharing model between the subscribers and the publishers mediated by a broker have a major security issue. Once the broker is compromised, messages can be dropped or not delivered at all, delayed or delivered unfiltered. Most importantly, replicated brokers can collude to disrupt the entire operation. These malicious behaviors is known as Byzantine faults; a faulty model where the system deviates from the protocol specification and enters into undesired states.

Mayer et. al. (Mayer, 2011) evaluated the robustness of pub/sub systems in eight architectural dimensions and argued the criticality of the rational behaviour. For decades, replication have been the corner stone for achieving reliability and robustness of the brokers. For example, crash failure resiliency issues and reliability in pub/sub systems using replication have been studied in (Kazemzadeh, 2009).

With the growing trend of cloud computing adaptations, replicating brokers on a highly dynamic virtualized cloud environment is undeniably cost effective. However, it's increasingly challenging to guarantee the reliability and robustness of the brokers on these platforms due to the increase in the attack surface (Manadhata, 2011) – the set of ways/entries an adversary can exploit/penetrate the systems.

Chang et al, in their position paper (Chang, 2012), noted the lack of studies of BFT-based pub/sub systems in the literature and pointed out that building such as system is difficult, perhaps even impossible. The key challenge is that the BFT system's run time execution model is ordered (client request are processed in persistent FIFO model) in contrast to the loosely coupled nature of pub/sub system.

A crash tolerant Paxos-based system, referred as

P2S, was recently proposed by the same authors (Chang, 2014). Others have approached this problem using overlay networks. To name a few, a consensus replication model is proposed in (Jehl, 2013) which was noted that the protocol somewhat deviates from the traditional message forwarding standards. Another overlay networks based on neighbourhoods is proposed in (Kazemzadeh, 2013).

With the ever-increasing sophisticated targeted attacks that employ zero-day exploits, protocol-level solutions tend to be defeated when attacks originate outside the application, i.e. OS kernel. We believe shifting from a perceived over-emphasis on improving existing protocol-centric solutions, to better enable fault–resiliency while reflecting the underlying computing fabric, is critical.

Our primary focus in this work is to leverage the capabilities provided by the underlying cloud computing fabric to address the robustness issues in the context of pub/sub which has not yet sufficiently explored. We build our solution in OpenStack cloud software stack (OpenStack, 2015), more specifically, the *nova* API which allows provisioning and de–provisioning VM/brokers instance (dubbed VM/broker refresh), and the *neutron* for the networking reconfiguration of the brokers.

Thus, the main contribution of our work is the following:

- We introduce a disruption-resilient framework for publish and subscribe.
- We introduce a generic VM refreshing algorithm for event-based replicated systems on virtualized cloud platforms.

We have organized the paper as follows; we first give a brief overview of a pub/sub and discuss the threat scope and assumptions. We present DRPaS system design and implementation in section 3. In section 4, we show a preliminary experimental evaluations and report our results. Finally, we discuss the related work followed by our conclusion in sections 5 and 6 respectively.

2 BACKGROUND

Within this section, we give a brief overview of pub/sub system and the threat models that we consider.

2.1 Publish and Subscribe Systems

Eugster et. al. (Eugster et.al., 2003) gave a fair treatment of the pub/sub models and their interaction patterns, highlighting the decoupled nature of publishers and subscribers in time, space, and synchronization. Many variants of the paradigm are especially adapted a variety of applications and network models. In this work, we are interested in highly available replicated brokers/pub/sub systems distributed across heterogeneous clusters.

Pub/sub ecosystem consists of three agent interactions: subscribers, publishers, and broker. Typically, a broker mediates the exchange of topic or content-based messages between the producers (publishers) and consumers (subscribers). In general, pub/sub systems handle aperiodic and periodic messages of heterogeneous sizes and formats in near-real time.

One of the proven methods of implementing a fault-tolerant (i.e., Byzantine Faults) replicated services is through State Machine (SM) approach (Scheneider, 1990). However, the main challenge of implementing Byzantine Fault Tolerant (BFT) protocol in pub/sub systems is that BFT protocols are based on SM which complies a well ordered finite state automata transitions, for example, client requests enter the system which is then executed in ordered fashion among the replica and then a consensus is reached for the response. Unlike pub/sub, the client requests/events are processed in loosely coupled fashion, for example, events are published out of order and brokered by the broker independent of the subscribers that are interested in these events.

2.2 Scope and Threat Model

Many variants of the pub/sub paradigm have been proposed and each is being specially adapted to specific application and network models (Eugster et.al., 2003). We consider replicated n brokers where $n>1$ nodes/replica typically used for high availability cluster settings.

Our attack model considers an adversary that has fully (system privilege) compromised a broker undetected by the traditional defensive mechanisms, a valid assumption in the cyber space. The adversaries' advantage, in this case, is the unbounded time which enables him to eventually disrupt the system. The fundamental premise of DRPaS is to eliminate the adversaries' advantage of time through the use of the underlying computing fabric.

We consider the standard assumptions of Byzantine fault models (Lamport, 1982) . Typically, replicated brokers with Byzantine fault models demonstrate arbitrary faults that deviate from the correctness protocol. We consider a Byzantine broker misbehavior described in (Chang, 2012). A compromised broker can: 1) impact the performance by delaying

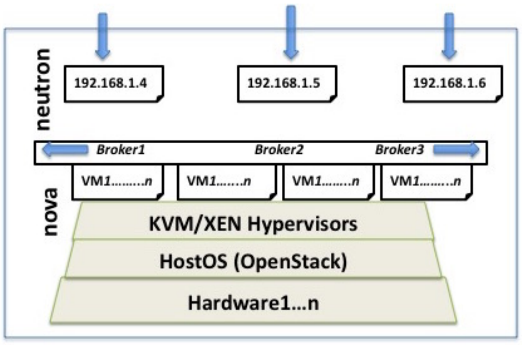

Figure 1: Logical System View of 3 Brokers on Virtualized Cloud Platform. Arrows on top represent the entry point of the application, and those on the sides of the application layer (*Broker 1..n*) represent the elastic computing model of the brokers provisioned on any of the underlying VM.

publications, 2) effect system integrity by tempering published contents, violating message reordering and corrupting forwarding tables, and 3) even cause system outage.

In this work, we consider publishers and subscribers (clients) are trusted, for simplicity, and illustrate VM refresh techniques for impeding malicious attacks. Dealing with misbehaving clients that can wage denial of service attacks, for example, will be considered in our future work.

We assume that hypervisors/VMMs and the cloud software management stack (OpenStack) and it's dependent libraries are secure.

3 SYSTEM DESIGN AND IMPLEMENTATION

Our design is motivated by the modularized, pluggable and structured cloud computing fabric, i.e, stacked hardware, host OS, guest VM/OS's, and reconfigurable networks. In this section we will describe our system design approach and discuss our algorithms.

3.1 System Design

The logical system view depicted in Figure 1 illustrates the building blocks of a cloud platform. These blocks can be viewed as three logical layers; i) the bottom three blocks (Hardware1..n, HostOS, and Hypervisors) which we call it the foundation layer, ii) *guest/VM* layer which consists of the VM1..n blocks and the impeded applications (Broker1, Broker2, and Broker3), and iii) the networking layer.

We adopted a cross layer vertical design that simultaneously operate on two logical layers of the cloud platform; a *guest/VM* and the network layer. The *guest/VM* layer aims for broker VM instance refreshes while the networking component aims to dynamically reconfigure the network at runtime. It's intuitive to see that such scheme has the benefit of terminating a compromised broker within a given time frame, therefore, a successful and/or in progress attack will have a limited impact on the system.

Since attacks originate at the entry point, externally visible *192.168.1.x* IP for clients, by refreshing the underlying VM (*broker1...n*), we circumvent any attack crafted or vulnerability exploited to a given system (i.e., hardware, Host and guest OSs and the Hybervisor). Note that the brokers typically communicate with local IP address similar to those found in LAN settings.

3.2 Implementation

We implemented our algorithms with *bash* shell script using OpenStack (OpenStack, 2015) *nova* api, an open source cloud management software stack. OpenStack is popular in the commercial world, for instance, RackSpace (RackSpace, 2015), a public cloud platform built with OpenStack used by many well-established businesses like Netflix.

OpenStack provides modularized components that simplify cloud management. In this work, we leveraged *nova compute* for provisioning the VMs/brokers, *neutron* for networking, *glance* for the VM image management, and *horizon* dashboard for visualization.

Algorithm 1 VM/Broker Refreshing Algorithm

```
Input: targetBroker
Output: Refreshed targetBroker
1: $flIP \leftarrow targetBroker_{floatIP}$
2: $nova\ delete\ targetBroker$
3: $targetBroker \leftarrow nova\ boot < options >$
4: nova floating-ip-associate($targetBroker_{ID}$, $flIP$)
```

In Algorithm 1, we first save the broker's externally feasible IP address known as *floating IP* in line 1. We then delete the broker in line 2, and in line 3 we create a new VM instance with *options* like; specific port ID with selected *fix* IP address (LAN), OS type, cluster, geographic location, file to run after boot, etc. Finaly, in line 4, we associate the floating IP from the old VM saved in line 1 to the one created in line 3. Note that *nova ip-associate <options>* is an implementation of Software Defined Networking (SDN) in the *neutron* component.

Clearly, this algorithm is also suitable for any replicated and non-replicated system deployed on virtualized cloud platform using some form of *nova* implementation for *provisioing* and *de-provisioning* VM instances, and an SDN implementation. Further, the algorithm can be used in a random refresh fashion or timely based (i.e. 5 minute) intervals as we illustrate in the experiment section.

4 EVALUATION

We randomly refresh broker/VM instance, thus, our experiment is targeted on evaluating how fast we can refresh brokers in order to reduce the exposure window time of an attack to succeed and disrupt the system.

4.1 Experimental Setup

Our experimental cloud platform uses a private cloud built on OpenStack software on a cluster of 10 machines (Dell Z400) with Intel Xeon 3.2 GHz Quad-Core with 8GB of memory connected with 1 gigabit Ethernet switch. The 10 machines were used as one controller and networking node, and 9 compute nodes. The 9 compute nodes allow us provisioning 36 virtual CPU's (vCPU) which equals upto a pool of 18 small vm instances, 2 vCPU per instance.

We used Ubuntu 14.04 for clients and the replicas/servers in all our experiments. Note that the OS's of both replicas and clients can be any OS image that Openstack supports. We deployed *RabitMQ* (RabitMQ, 2015) pub/sub brokers in distributed fashion. *RabitMQ* is a widely adopted open source and commercially supported content-based message brokering. RabitMQ is used in may applications such the financial trading systems, SOA Service buses, inter cloud component interconnections, etc.

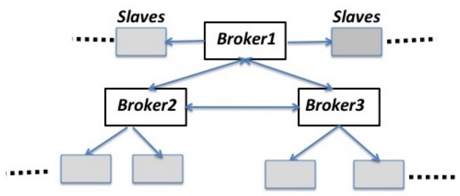

Figure 2: Replicated Broker Topology

There are numerous ways on setting highly available replicated brokering. For simplicity, we set up 3 brokers with 6 slaves (2 each master) as depicted in Figure 2. Within a RabbitMQ cluster, queues (message topics) are singular structures that exists only on one node (the master node) in which is then mirrored (replicated) across multiple nodes to address high availability. Each mirrored queue consists of one master and one or more slaves that can be synchronised, with the idea of the slave replacing the master when it fails (built-in reliability scheme). Thus, the key motivating factor for our solution approach, the existence of the built-in reliability schemes in the protocol enables the brokers and clients to reconnect after a short disconnect.

4.2 Experimental Results

Figure 3 shows the exposure window time line. The *x-axis* show a five minute blocks. In each block we have 3 physical nodes numbered *1, 2, 3* in which the applications/brokers are deployed on (depicted as circles).

In the first block, we show 3 nodes depicting the 3 replicas deployed for the experiment. The 3 replicas/VM can be on any hardware, say, hardware #1, #2 and #3 out of the 9 nodes of our private cloud infrastructure setting. To illustrate the concept, we refreshed a brokers in 5 minute intervals marked *0 to 5min* by the end of each block.

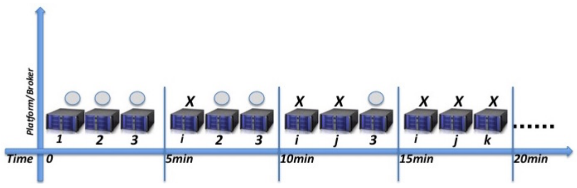

Figure 3: Illustration of a 3 broker/VM exposure window of 5 minute intervals.

At the start of the experiment, the brokers *broker*$_1$, *broker*$_2$ and *broker*$_3$ represented in circles are on their respected hardware among our cloud nodes, node#1, node#2 and node#3 respectively. Note that each node can host more than one broker instance.

After 5 minutes, we refreshed *broker*$_1$ on one of the other 6 nodes, $node_i$ in this case. At the end of the next 5 minutes (10 minute block), we did similarly to *broker*$_2$ and mapped on the other 6 nodes of the platforms, $node_j$ in this case. Similarly in the consequent 5 minute blocks. The process of refreshing a VM took only 50 to 60 seconds. We consider evaluating the performance impact and the application queue updates (creation/deletion) with large number of clients in our future work.

It's intuitive to see that defending the 3 replicas in first block for it's entire run time is extremely challenging compared to when defending them in one of the 5 minute blocks. The rationale behind this is that in each block there is at least one broker replica is on yet unknown (to the attacker) platform, and another one (or more) soon to be refreshed, thus, reducing the exposure attack window of the overall system.

Therefore, capabilities enabled by the underlying cloud platform seamlessly adds high visibility on system's runtime to prevent disruptive faulty behavior. The preliminary result show the feasibility of the unprecedented security capabilities enabled by underlying computing fabric to defend against unknown enemies.

5 RELATED WORK

There has been a wide array of research on tackling the reliability of pub/sub systems at the application layer. (Mayer, 2011), discussed over decades of works on replication techniques and a taxonomy of Byzantine faults in pub/sub systems considering failure scenarios, however, all these studies address reachability issues, i,e, link/node crashes and fast recovery in the overlay brokers.

Recent protocol level solution approaches include the crash tolerant Paxos-based proposed by (Chang, 2014), the State Machine consensus replication model presented in (Jehl, 2013), and the overlay network based on neighbourhoods (Kazemzadeh, 2013).

All of these solutions address BFT-resiliency in the context of pub/sub by modifying the pub/sub messaging protocol. In contrast, our generic solution approach can be applied to any pub/sub system, perhaps, non-pub/sub replicated systems to resist BFT-faults without any modification to the protocol while considering the underlying computing fabric.

6 CONCLUSIONS

We showed the capabilities enabled by the underlying computing fabric are simpler and effective than the ones implemented in the protocol to defend against modern sophisticated attacks. The practicality and the effectiveness of the proposed scheme is illustrated with a widely adopted open source cloud management software stack (*OpenStack*) and replicated publish and subscribe *(RabitMQ)* system deployed on a realistic private cloud setting.

Future works will address remote hardware and software attestation, and trace-based performance analysis of VM refreshes across multiple cloud providers. We will consider integrating *Virtual Machine Introspection* for fine tuning the VM refresh rate while injecting/detecting attacks to determine the limits impose by the cloud platforms for an acceptable exposure window.

ACKNOWLEDGEMENTS

Authors would like to sincerely thank Jim Hanna for the cloud platform support and Matthew Paulini for his editorial proofing contributions, both at AFRL/RI. Special thanks for the anonymous reviewers for their feedback.

REFERENCES

Ahmed, N., and Bhargava, B. 2015. Towards Targeted Intrusion Detection Deployments in Cloud Computing. In the *Int. Journal of Next-Generation Computing Vol. 6, No 2 (2015), IJNGC - JULY 2015.*

Chang, T., and Meling, H., 2012. Byzantine Fault-Tolerant Publish/Subscribe: A Cloud Computing Infrastructure In the *Proceedings of the Symposium on Reliable and Distributed Systems. October 2012.*

Chang, T., Duan, S., Meling, H., Peisert, S., and Zhang, H., 2014. P2S: A Fault-Tolerant Publish/Subscribe Infrastructure In the *Proceedings of DEBS, May 2014*

Eugster, P., Felber, P., Guerraoui, R., and Kermarrec, M., 2003. The Many Faces of Publish/Subscribe. In the *Proceedings of the ACM CSUR 35, 2 (June 2003).*

Jehl, L. ,and Meling, H., 2013. Towards Byzantine Fault Toletant Publish/Subscribe: A State Machine Approach In the *Proceedings of HotDep, November 2013*

Kazemzadeh, S. R., and Jacobsen, H., 2013. PubliyPrime: Exploiting Overly Neighbourhood to Defeat Byzantine Publish/Subscribe Brokers. TR University of Toronto, May 2013

Kazemzadeh, S. R., and Jacobsen, H., 2009. Reliable and Highly Available Publish and Subscribe In the *Proceedings of the Symposium on Reliable and Distributed Systems. October 2009.*

Lamport, L., Shostak, R., and Pease, M. 1982. The Byzantine Generals Problem. In *ACM Trans. Program. Lang. Syst., 4(3): 382–401, 1982.*

Manadhata, P.K., and, Wing, J.M., (2011). An Attack Surface Metric In the *IEEE Trans. Software Engineering, 37, 371-386, 2011.*

Mayer, R. T., Brunie, L., Coquil, D., and Kosh, H. 2011. Evaluating the Robustness of Publish and Subscribe Systems. In *the Proceedings of IEEE Int. Conf. of (3PGCIC). pp. 75-82. 2011.*

OpenStack(2015).`https://www.openstack.org/`

RabitMQ (2015).`https://www.rabbitmq.com/ha.html`

RackSpace,(2015).`https://www.rackSpace.com/`

Schneider, F., 1990. Implementing Fault-Tolerant Services using the State Machine Approach: A tutorial. In *ACM Computing Surveys (CSUR) 22.4 (1990): 299-319.*