Abstract

The number of applications being deployed using the **PaaS!** (**PaaS!**) cloud computing model is increasing. Despite the security controls implemented by cloud service providers, we expect intrusions to harm these applications. We present Shuttle, a novel intrusion recovery service where security intrusions in **PaaS!** applications are removed and tolerated. **PaaS!** providers are capable to allow their customers to recover from intrusions in their applications using Shuttle.

Our approach allows undoing changes to the state of **PaaS!** applications due to intrusions, without loosing the effect of legitimate operations performed after the intrusions took place. We combine a record-and-replay approach with the elasticity provided by cloud offerings to recover applications deployed on various instances and backed by distributed databases. To recover applications from intrusions, the service loads a database snapshot taken before the intrusion and replays the subsequent requests, in concurrently as possible, while continuing to execute incoming requests. Shuttle is available without setup and configuration to the **PaaS!** application developers. The proposed service removes security intrusions due to software flaws or corrupted user requests and supports corrective and preventive maintenance of applications deployed in **PaaS!** cloud computing platform.

We present an experimental evaluation of Shuttle on Amazon Web Services (AWS). We show Shuttle can replay 1 million requests in around 10 minutes and that it is possible to duplicate the number of requests replayed per second by increasing the number of application servers from 1 to 3.

Keywords

- Intrusion Recovery
- Intrusion Tolerance
- Dependability
- Cloud Computing
- Platform as a Service
- Distributed Database Systems

1 Introduction

PaaS! is a cloud computing model that supports automated configuration and deployment of applications [**?** **?** **?** **?**]. While the **IaaS!** (**IaaS!**) model is being much used to obtain computation resources and services on demand [**?** **?**], **PaaS!** aims to reduce the cost of software deployment and maintenance abstracting the underlying infrastructure. The model defines a well tested and integrated environment in which clients (*tenants*) design, implement, deploy and run their applications on a managed cloud infrastructure through a set of middleware services. Examples of these services are load-balancing, automatic server configuration and storage. These services are paid-per-usage and turn the application easy to deploy and scale. **PaaS!** platforms are provided either by cloud providers, such as Windows Azure [**?**], Google App Engine [**?**], Heroku [**?**], Openshift [**?**] and Amazon Elastic Beanstalk [**?**], or by open source projects [**?** **?** **?**]. Besides natural metrics such as cost and performance, the success of **PaaS!** systems will also be established by their features, for instance capability to recover from intrusions.

1.1 Problem Statement

The number of applications running in cloud computing platforms, including those based on the **PaaS!** model, is increasing rapidly. Many of these applications are critical for their companies and contain valuable ~~informations~~information, so the exploitation of vulnerabilities is attractive and profitable. Consequently, the risk of intrusion is high. An intrusion happens when an attacker exploits a vulnerability successfully. Intrusions are considered faults. Faults may cause system failure and, consequently, application downtime ~~which has~~ and significant business losses [**?**]. The recent case of the cloud-based Code Spaces service is conspicuous: hackers deleted most of its data and backups, leading to the termination of the service [**?**].

CSP! (**CSP!**) implement several security controls. Most of these controls aim to prevent and detect intrusions: access control, firewalls, intrusion detection and prevention systems, network access control, vulnerability scanning, etc. Despite the importance of these mechanisms, applications often contain design or configuration vulnerabilities that let intrusions happen [**?** **?**]. Complexity and budget/time constraints [**?**], weak users passwords or bad security policies are known causes of these problems. The recent case of the bash bug (or Shellshock) shows that there are other reasons such as legacy software being used in ways that were unpredictable when it was developed [**?**]. Attackers can spend years developing new ingenious and unanticipated attack methods having access to what protects the application. On the opposite side, guardians have to predict new methods to mitigate vulnerabilities and to solve attacks in few minutes to prevent intrusions.

Much research has been done on mechanisms to tolerate Byzantine faults, including intrusions [**?**

? ?]. However, most of these techniques do not prevent application level attacks or user mistakes. For instance, if attackers steal legitimate user credentials, they are able to modify the state of the applications violating their security policy. In summary, there are several paths for intrusions to happen, even if mechanisms to prevent or tolerate them are used.
We assume intrusions can happen and their effects need to be removed from the applications' state. This removal is often done manually by system administrators. Administrators have to detect the intrusion, understand the parts of the state compromised directly by the intrusion or contaminated by operations that used compromised state, and clean the state manually. For instance, most of full-backup solutions revert the intrusion effects but require extensive system administrator effort to ~~replay the~~ restore the effect of the legitimate actions. This process is error-prone, often takes long and causes application downtime [**?**]. Intrusion recovery systems aim to automate these steps and mitigate these issues.

Previous intrusion recovery systems targeted operating systems [**? ?**], databases [**? ?**], web applications [**? ? ?**] and other services [**?**]. Yet, none of them was designed for cloud applications, which are often deployed in multiple servers and use background databases. Furthermore, most cause downtime, which is undesirable in online services.

1.2 Goals and Main Contributions

The primary goal of this thesis is to design, develop and evaluate a system to recover from ~~security~~ intrusions in cloud computing. In particular, the system shall ~~support to keep~~ allow tenants to keep their **PaaS!** (**PaaS!**) applications operational despite intrusions. The ~~system shall accept that~~ idea is to accept intrusions can happen, ~~remove their effects~~ thus to provide a system to remove their from the application's state and restore the state's integrity.

The approach followed in this dissertation consists in recovering the applications state when intrusions happen, instead of trying to prevent them from happening. Intrusion recovery systems do not aim to substitute prevention but to be an additional security mechanism. Similarly to fault tolerance, ~~they~~ we accept that faults occur and have to be processed. ~~Namely~~However, we aim to decrease the applications' Mean Time to Repair (MTTR), not the Mean Time to Failure (MTTF). Doing so, we expect to increase the applications' availability, which is given by $Availability = MTTF/(MTTF + MTTR)$.

The main contribution of this dissertation is a novel *intrusion recovery service* for **PaaS!** systems, named *Shuttle*. Shuttle is a service that aims to make **PaaS!** applications operational despite intrusions, helping tenants to recover their applications from software flaws and malicious, or accidental, corrupted user requests without requiring application downtime during the process. When an intrusion is detected, tenants can use this service, which is offered by the **CSP!**, to remove intrusions' effects and recover the integrity of their applications. In this dissertation, we ~~concern~~ are concerned with the applications' availability and the integrity of their state, not their confidentiality. Consequently, the proposed service does not aim to ~~avoid~~ deal with information leaks.

Shuttle assumes a client-server model in which clients communicate with the servers in the cloud using **HTTP!** (**HTTP!**)/**HTTPS!** (**HTTPS!**) ~~.~~ or protocols encapsulated on top of them (e.g., **SOAP!**, **REST!**). For each application deployed in the **PaaS!** system, Shuttle records the requests issued by clients and creates periodic snapshots of the application database.

After detection of the intrusion, Shuttle loads the snapshot that precedes the beginning of the intrusion and replays only the legitimate requests to recreate an intrusion-free application state. Requests are replayed asynchronously and, whenever possible, concurrently. Even so, the recovery process is deterministic because operations to each data item must have the same order as on first execution.

Dependencies established at database level during the requests' first execution are used to create independent clusters of requests that can be replayed concurrently. We propose a branching mechanism to maintain the service available continuing to execute incoming requests while replaying the requests.

We introduce a novel approach to remove the intrusion effects in which the **PaaS!** controller terminates the current application instances, launches new instances and deploys an ~~update~~ updated software version, which may fix previous flaws.

Unlike previous intrusion recovery systems, Shuttle is provided as a service to developers and tenants of **PaaS!** applications. Consequently, it can be well tested and available without depending

on being correctly setup by the application developers. We also leverage the elasticity of **PaaS!** infrastructures to reduce the service costs and the recovery period. Specifically, Shuttle is designed to allocate more servers during the recovery period to accommodate the throughput of requests being replayed, and release them ~~when~~ at the end, with a proportional impact on service costs. The ~~rapid and continuous decline~~ decline retirei continuo e rápido, faz mais sentido e é mais seguro in computation and storage costs in public cloud providers makes affordable to store user requests, to use database snapshots and to replay previous user requests.

We propose, to the best of our knowledge, the first intrusion recovery service for **PaaS!** applications. The main contributions of this dissertation are the following:

- a new intrusion recovery approach provided as a service integrated in a **PaaS!** system and taking into consideration applications running in various instances backed by distributed databases;
- a method to order the replayed user requests considering their accesses to databases;
- accomplishing intrusion recovery without service downtime using a branching mechanism;
- leveraging the resource elasticity and pay-per-use model in **PaaS!** environments to record and launch multiple clients to replay previous non-malicious user requests as concurrently as possible to reduce the recovery time and costs;
- a mechanism to do a globally transaction-consistent snapshot of **NoSQL!** databases;
- an approach to remove intrusions redeploying the applications; este é o que está em excesso mas acho-o relevante para a tese...

1.3 Thesis Structure

This document is structured as follows. In Chapter **??**, we present the fundamental concepts and previous intrusion recovery proposals. In Chapter **??**, we describe briefly the architecture of **PaaS!** systems and the proposed architecture for intrusion recovery service. In Chapter **??**, we describe the platform and components of the prototype of Shuttle. The work is evaluated in Chapter **??** and concluded in Chapter **??**.