Evolution Oriented Monitoring oriented to Security Properties for Cloud Applications

Jamal Toutouh
University of Malaga
Málaga, Spain
jamal@lcc.uma.es

Antonio Muñoz
University of Malaga
Málaga, Spain
amunoz@lcc.uma.es

Sergio Nesmachnow
Universidad de la República
Montevideo, Uruguay
sergion@fing.edu.uy

ABSTRACT

Internet is changing from an information space to a dynamic computing space. Data distribution and remotely accessible software services, dynamism, and autonomy are prime attributes. Cloud technology offers a powerful and fast growing approach to the provision of infrastructure (platform and software services) avoiding the high costs of owning, operating, and maintaining the computational infrastructures required for this purpose. Nevertheless, cloud technology still raises concerns regarding security, privacy, governance, and compliance of data and software services offered through it. Concerns are due to the difficulty to verify security properties of the different types of applications and services available through cloud technology, the uncertainty of their owners and users about the security of their services, and the applications based on them, once they are deployed and offered through a cloud. This work presents an innovative and novel evolution-oriented, cloud-specific monitoring model (including an architecture and a language) that aim at helping cloud application developers to design and monitor the behavior and functionality of their applications in a cloud environment.

KEYWORDS

Cloud Computing, Monitoring rules, Dynamic Verification, Security Properties, Event-Sequence Language

ACM Reference Format:
Jamal Toutouh, Antonio Muñoz, and Sergio Nesmachnow. 2018. Evolution Oriented Monitoring oriented to Security Properties for Cloud Applications. In *ARES 2018: International Conference on Availability, Reliability and Security, August 27–30, 2018, Hamburg, Germany*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3230833.3232856

1 INTRODUCTION

Despite the many advantages cloud technology offers, it still raises significant concerns regarding security, privacy, governance, and compliance of data and services offered through it. Such concerns arise from the inherent difficulty to control processes and data that are stored and used in platforms that are managed and controlled by third-parties [1].

ARES 2018, August 27–30, 2018, Hamburg, Germany

ACM ISBN 978-1-4503-6448-5/18/08…$15.00
https://doi.org/10.1145/3230833.3232856

We claim that evolution is a necessary ability for applications and individual services. Nevertheless, there are bugs that need to be overcome, i.e., threats overlooked in the development phase can be later identified and require including countermeasures, new functionalities, improved performance, or environment changes may trigger the evolution of applications and services. In all these cases, engineers must address evolution with few or even no information about the behavior of the software component. This lack of support for a security focused evolution process covering the full life cycle of systems allows vulnerabilities to be exposed and exploited by malicious users, before developers are able to fix them.

Evolution should take the spotlight in the development and operation of cloud applications. Now, evolution is considered a secondary activity in the life cycle of applications, but we argue that this is not in the line with the needs of a large percentage of cloud applications. Among the main reasons why applications are hosted in the cloud we highlight, first, the need of making them available to a very large number of users that use multiple devices, and second, the high costs of maintaining and evolving such type of applications in comparison with traditional standalone or client-server applications. Likewise, it is common that the nature of services provided by these applications also requires more frequent updates to add new functionalities or new access interfaces. Finally, the heterogeneity and unpredictability of modern computing and communication infrastructures and platforms on which the applications actually run, is also an important reason that requires adaptation and evolution. In order to be applicable as a driver of security and efficiency in cloud computing, application evolution approaches and mechanisms must be made more dynamic, taking advantage of the characteristics of the cloud scenario, while avoiding the introduction of new security risks.

The aforementioned topic has been rarely addressed in the related literature. Few researches have focused on developing new tools and mechanisms to improve the efficiency and security of systems evolution across the whole software life cycle. In fact, security and evolution are tightly interrelated concepts. On the one hand, poorly controlled evolution is the source of many security weaknesses and other errors that have a negative impact on the system security. On the other hand, evolution is a crucial feature to maintain the security of systems. The current lack of support for a security-focused evolution process covering the full life cycle of computing systems results in (i) slow response times that allow vulnerabilities to be known and exploited by malicious users before developers are able to fix them, and (ii) high maintenance costs, downtime, and security incident handling.

Run-time monitoring has become an essential element whenever high levels of assurance are required. Monitoring can be useful for

different purposes such as prevention of harm when an strange behavior is detected, collection of information from both the application and the environment where it is running, etc. The current concept of monitoring focuses on the runtime supervision and control of applications, allowing the early detection of operation problems of individual application instances and supporting the automated reconfiguration of these applications. In our context, as we have already mentioned, there are not appropriate pre-deployment controls for the services running on a cloud. Therefore, runtime analysis and control become an essential tool for comprehensive support of the security of cloud software, especially when focusing on evolution. Consequently, in this article we focus on developing the concept of evolution-oriented monitoring to monitor systems to obtain feedback and inform the evolution process. These processes are themselves subject to security requirements and they need to ensure that the privacy of different stakeholders is preserved whilst sufficient information is communicated to developers to guide evolution.

Many authors have studied evolution topics from different perspectives, focusing on classifying types of evolution and also defining abstract models of evolution [7–10]. Some initiatives have focused on requirement evolution [11–13] more than system evolution. A key research contribution in this area is a better understanding of the origins of change [14, 15]. Roshandel et al. [16] presented an approach for managing architectural evolution in sync with code. Yu et al. [17] defined an approach based on ignoring all information but the source code, and applying reverse engineer requirements from there on an as-needed basis.

In the area of evolution of software systems, Kephart [2] proposed autonomic software that self-configures, self-repairs, self-optimizes and self-protects. Oreizy [3] proposes an architectural approach for adaptive software systems that adapt to the environment to meet their intended purpose through a monitor-diagnose-compensate feedback loop. Following this approach, a reflective middleware was presented by Kon et al. [4]. Rainbow et al. [5] proposed an architecture-based framework that enables self-adaptation based on two models: (i) an externalized approach and (ii) software architecture models. A number of authors have been interested in software evolution, most of them focusing of helping developers understand how to change the system. However, very few efforts are dedicated to address the full life cycle of systems, being just some of them interested in engineering.

In this line of work, this article describes a proposal for an evolution-oriented monitoring model for cloud systems focused on security properties.

The article is organized as follows. Section 2 presents the main features of the proposed evolution-oriented monitoring for specifications and implementations. The proposed model for evolution of cloud applications is described in Section 3. The ongoing work about monitoring applications is described in Section 4. Finally, Section 5 presents the conclusions and formulates the main lines for current and future work.

2 EVOLUTION ORIENTED CAPABILITIES

In this work, we propose a framework that provides support for evolution of both specifications and implementations by means of evolution-oriented monitoring mechanisms that enable the analysis of experiences (e.g. runtime data) gathered during the deployment of solutions (services and components) on different platforms. We envisage such mechanisms being based on secondary analysis of primitive monitoring data, aimed at identifying several issues, including:

- Gaps in the descriptions of solutions (e.g. missing preconditions and other assumptions about the operational context of a solution).
- Faulty implementations of solutions (e.g. implementations that do not satisfy certain solution properties under specific conditions).
- Faulty or incomplete models for solutions (e.g. missing state transitions).
- Improvements and enhancements in the solution composition plans.

The first level of monitoring (application-specific monitoring carried out by the *Local Application Surveillance* (LAS) is based on logic reasoning and deterministic rules. For the higher levels (interaction between applications carried out by the *Intra Platform Surveillance* (IPS) and monitoring of different instances of the same application running on different platforms carried out by the *Global Application Surveillance* (GAS) we propose adopting probabilistic reasoning, which fits better the nature of the rules to deal with in these levels. Precisely, most of the evolution-oriented knowledge is generated in these two levels.

The addition of the most abstract monitoring level allows obtaining information that cannot be obtained using normal application monitoring. In particular, the IPS allows our model to deal with potential problems caused by the interaction between different applications in the same platform. For example, if the frequency of correlation between events generated in two monitored applications is statistically significant our model may conclude that there is an unforeseen interaction between those applications and take appropriate measures. On the other hand, GAS supports maintenance and evolution of specific applications and detect problems with non-compliant implementations, as well as problems in the modeling. With an isolated application, monitoring can detect a failure, which is by definition an inconsistency between the model and the implementation of the application (considering that it includes all supporting layers of the cloud stack). However, it is not possible to determine whether the problem is actually in the model or in the implementation. With the vertical monitoring that GAS performs over different instances of the same application, our model can indeed determine the origin of the inconsistency. Suppose that GAS receives, from a given application instance, events indicating repeated violations of a monitoring rule indicating an illegal transition between two states. This would indicate that the application instance does not conform to its application model. On the contrary, if the violations come from different instances of the same application, these violations would most likely indicate an error in the model of the application.

3 EVOLUTION OF CLOUD APPLICATIONS

The evolution of cloud applications is a complex task due to the difficulty of knowing the specific problems that applications have

and how correctly evolve then in order to obtain a new enhanced application without unexpected security problems. It is relevant to take into account that those problems increase since the cloud applications are running over different operating systems and architectures, which means different software and hardware infrastructures. Our approach addressed these problems by using the monitoring architecture described in the previous section. This section describes how the architecture works and how the evolution of cloud applications by using our approach is performed.

3.1 Evolution of applications running on Cloud

Data needed for the evolution is obtained by using the monitoring infrastructure, from different architectures (hardware and software) where the same cloud application is running. CloudMon[18] infrastructure enable applications to forward those data that allow identifying and studying the different configurations and behaviors that were active when the cloud application failed or a monitor rule was violated.

The CloudMon workflow consists of creating an instance of LAS specific for each application. The idea is that every application loads its Application Behavior Security Model in its associated LAS in such a way that LAS detects what to monitor is described in the event declaration part. LAS sends to a subscriber (a daemon process) the list of events to subscribe. After that, rules are checked and if any rule is violated then the reaction for that specific rule is performed. IPS can act as privacy filter to decide whether sending the set of events to their related GAS or not. At this level, events become rule violations. If a rule is violated, it is reported to application provider in order to be analyzed and study if a common error in many instances of its application or only for one instance.

The local monitors (LAS) of each system check the correct working and behavior of a specific cloud application. If the application violates any rule included in the list of the monitor, it reacts by executing the (re)action specified in the rule and sends the information to the IPS. Each IPS receives reports from different cloud application running in the same platform. When rules are violated, IPS sends the information to GAS, as it means that the cloud application has a security vulnerability or flaw in the design and must be evolved. The information sent to the cloud application developer, that is to GAS, is a precise and focused report that describes the problem, the system configuration, the functionality that violated the rule, the environment where the cloud application is running, and every other relevant detail. Using all this information, the developer can analyze the problem and extract the problematic functionalities of the application.

To follow the proposed approach, new cloud applications should be designed for monitoring. For this purpose, specific parts of software must have attached a set of rules, similarly to the Proof Carrying Code [6] approach. Applications must have the structure depicted in Figure1. Obviously, the application code and data parts are essential for runtime. As it was previously described, the *Application Behavior Security Model* (ABSM) enables a mechanism for communicating application specific events. For instance, applications write on a specific file such as monitoring log and the Application Interaction Security Model (AISM) is used to define the monitoring of problems derived from the interaction of this application with others simultaneously running at the same platform

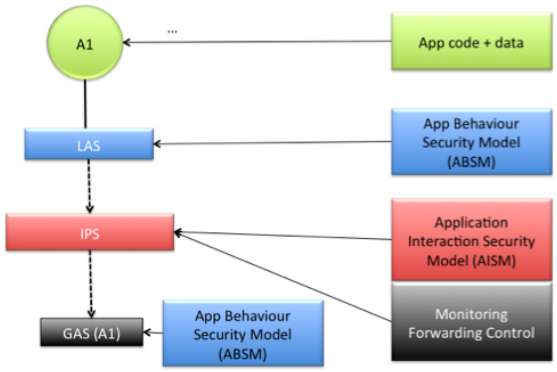

Figure 1: Application Instrumented Code Interlinked.

Let us introduce an example including an application interaction security model that describes the problem. Let suppose that App1 uses key K1 from key-store KS1 for encryption, and there is evidence that none KS1 or App1 does ever leak K1. App2 also uses K1 from KS1, but we know it is vulnerable to an attack that can extract K1. In this case, App1 becomes vulnerable too and application designer and/or developer should be warned. In this example AISM, is described in [18] as a set of rules used to declare what to monitor regarding the use of shared resources. These rules are mostly based on abduction, focused on interaction and not application-specific. Thus, if we observe that App2 fails more than 60% of the times after App1 fails, we can conclude that App1 and App2 have an unexpected interaction.

The Monitoring Forwarding Control is fully editable, configurable, and parameterizable by the user. This component provides the user the decision about what to report to the application provider, keeping the user privacy. We assume that user is a gambler, but he prefer to keep it as a secret. Then, an specific parameter of a violated rule reflects that the access to the file "gambling.avi" is not granted. Let us introduce the next example: rule R1 is violated by App1 and after one second, rule R2 is violated three times by App2. This fact reflects that a correlation exists. The next step is to notify someone that R1 is always violated after R3 is violated three times. Nevertheless, this notification must be done in a privacy-respectful system and based on a policy that must be known beforehand. We notice after our analysis that it is necessary to forward only that required information for performance, privacy, and efficiency issues. As an additional measure, and tailored for specific cases the application code, ABSM, AISM, and Monitoring Forwarding Control parts can be signed.

The evolution-aware cloud applications are created using a specific methodology and they have a particular structure. Applications are enabled to be checked for malfunctioning, attacks, security vulnerabilities, and other issues. The monitoring rules for each application are specific and designed by expert users who know the

expected behavior and possible problems that the application can present when executing in a cloud system. Besides the application information and behavior, the monitoring rules must check the environment behavior where the cloud application is running. For example, there must be rules in charge of checking the RAM memory consumed by the application, the behavior of other applications running in the same system, etc. This functionality is possible thanks to an important characteristic of the proposed evolution-aware cloud applications model: Applications are created using modules for each specific functionalities. This means that the application is composed of modules that cover specific functionalities or areas of the application. For example, a cloud application for messaging would be composed of a module for sending messages, a module for the encryption of data, a module for the receiving of data, etc.

Figure 2 describes the structure of an evolution-aware cloud application. It is structured in independent modules that can be replaced if any of them fails, is vulnerable to a malicious attack, or if the monitor detects a security vulnerability, etc. This is one of the most important characteristics of these applications: the modules can be replaced if necessary and the application will continue working successfully without notice, although some functionalities can be disabled temporarily.

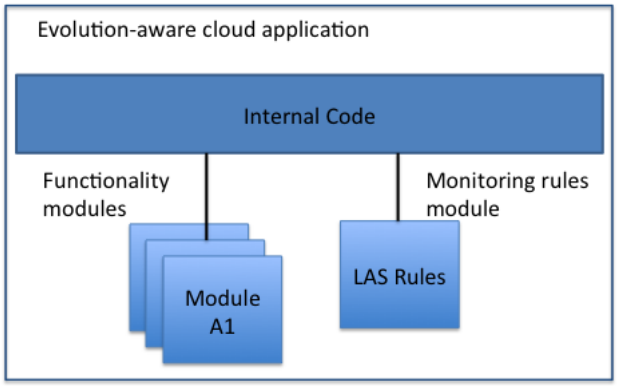

Figure 2: Evolution-aware cloud application structure.

One of the most important modules of a cloud application using the proposed model is the one that has the information of the monitoring rules, which implies that each application created with this structure contains all the monitoring rules in a single package and it can be updated with additional and/or different rules if the system evolves or security threats that were not predicted are found.

3.2 The proposed evolution process

The evolution process of the evolution-aware cloud applications is a critical and necessary functionality that allows cloud applications to evolve, adapt, and fix errors or security vulnerabilities detected in some instances of the applications running in specific system configurations or infrastructures. The evolution process takes into account the distributed and heterogeneous functionalities of the cloud applications, allowing these applications to avoid future risks and threats that are not yet detected by some instances of the application.

Evolution process is composed of two different processes. These two parts complement each other and helps the developer to control and evolve the cloud application. The main two functionalities are *prevention* and *evolution*.

Prevention is a local action that happens only in the cloud application instance that has detected the vulnerability. Thus application executes the reactions defined in the rule violation description, in order to minimize the error and data vulnerability.

Evolution is a general response for all the instances of the cloud application. The cloud application developer uses the information collected by the monitoring infrastructure and updates and/or manages the threated modules. These modules and the corresponding information are then deployed and all the instances of the cloud application can benefit from these changes.

For example, if application A detects a rule violation, the system reacts with the two-phase functionality. First, the local monitor reacts to the violation by executing the response code defined in the rule (stopping the application, disabling some modules, alerting the user, etc.); second, the local monitor (LAS) derives the information to be forwarded to the cloud application developer using the Monitoring Forwarding mechanism. This mechanism will use the information to create an evolution version of the application. The developer updates and manages the vulnerable modules and forwards information to all the instances of the cloud application. By means of this mechanism, even those cloud applications that have not detected the threat are protected against it. Figure 3 shows the evolution process for our proposal of evolution-aware cloud applications based on two phases.

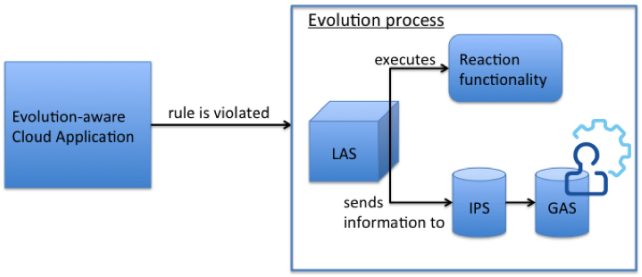

Figure 3: Evolution Process for Evolution-aware Cloud Applications.

Evolution has two phases. The first evolution phase is the adaptation of the application following the behavior model described in the monitoring rules of the application. These responses are specified in the LAS monitoring rules, which are the first ones executed when an application violates a rule. If the LAS detects that an application is having a security problem or has an unexpected behavior (i.e, it violates any of the rules), then it will execute the reactions specified for each violated rule. For example, LAS can close the application, disable the sending module of the application (preventing the application to send raw private data, etc.), restart the application, send a message to the system administrator, etc. These responses and reactions are specified in the monitoring rules by expert users who know which are the more dangerous assets and vulnerabilities of the application, allowing the system a quick

response to unexpected error or attacks and, preventing the harm to the system or the user. As the first reaction to errors and attacks, this first evolution phase is performed in real time, preventing attacks before any real threat occurs. In addition, as this rules and reactions are described in a specific module of the evolution-aware cloud applications, when the application creators receive more information about errors, attacks, security vulnerabilities, and other issues from the applications' interactions in real systems, they can create a new set of rules and reactions and update the application with the new information.

The second phase of the evolution-aware process for cloud applications consists on changing, managing, and updating the modules of the application. This phase complements and works in parallel with the first phase of the evolution process. In this phase, feedback information and monitoring rules of all the different instances of cloud applications are received and used to create or update their modules in the case that is required. A cloud application has several instances running in different environments and systems architectures. These instances have their LAS checking for the correct behavior of the applications and the data they use. When errors are detected, the LAS reacts executing the adaptability rules defined in the monitoring rules, and sends data about the error to the IPS.

The monitor engine receives as input information from the same application running in different platforms. Then the IPS (Intra Platform Surveillance) module receives many reports of an error or security vulnerability in an specific functionality the developers of the application use this information to evolve the application by updating or changing the module or set of them that fails. The information of all the different threats and reports from the execution of the cloud application are collected by the GAS. Cloud application developers access this information in order to obtain the necessary data for the evolution of the application, but keeping user privacy. Once the developers receive the information of the malfunction module, they react disabling that module in the application (or taking other preventive procedures) and, if necessary, changing the module by another one that offers the same or similar functionalities and can be used in that cloud application (regarding compatibility). This action is critical, as developers have to check that the new module is not only secure, but its use does not create new security vulnerabilities or errors in the application. Developers check this issue by testing the application including the new module internally, checking for violation of existing rules and the new rules included in the new module. This functionality allows users of the application to continue using the cloud application while developers fix the errors or security vulnerabilities detected in the failing module. Although the new module would not offer the same functionality and would not work as good as the former one, it will allow using the majority of functionalities of the cloud application. In the meanwhile, developers use the information from the monitors to fix the errors of the application by updating or creating a new module that replaces the one failed. A very important step of this process is the creation of new rules for the new module. Developers create new rules and reactions for the module and insert them in the monitoring module of the application. These new rules are based on the existing ones, and update them with the information of the detected threats, environment data where the application failed, and other relevant information. Once finished, developers evolve the cloud application by changing the old module or replacing it with the new one and update the rules module for the LAS. Users can continue working with the cloud application that is more secure and resilient after the procedure, as it has been evolved to check and work with the new issues founded thanks to the monitoring infrastructure.

3.3 Life cycle of the evolution process

The life cycle of the proposed evolution process is described in detail in this subsection.

Lets suppose that a group of end users are working with their company private data using a cloud application in their system. The local monitoring infrastructure detects an error in the application concerning the visibility of the private data by a malicious user. The CloudMon framework detects the problem before any harm is caused. The first reaction is executing the prevention response specified in the violated rule. This response was defined by the application developer, who knew what was the best course of action to avoid data loss or intrusion of malicious users. Lets suppose the reaction specified was to stop using the communications module of the cloud application, because if the data was sent without security, then it is better to avoid this functionality. Using this approach, users can continue using the cloud application without risk to expose the private data to malicious users. Only the vulnerable module is disabled, so users can work without worrying a malicious user access the company private data. Besides, users know that the information and data of the violated rule and the discovered vulnerability are being used by the cloud application developers to fix the threat. The CloudMon infrastructure sends data to protect the private information and data from the users, allowing for a major confidence and use of this system.

Together with the previous process, as the LAS reacts and executes the prevention action specified in the rule, it sends the information of the problem, violated rule, environment, cause, data of the application, the system configuration where it is running, and other data. This component can receive reports from both the same and different threats of the same application. If LAS receives several messages regarding the same threat, it analyzes the data, preserving the privacy of users, and sends the information to the Global Application Surveillance (GAS). GAS is in charge to check and digest information from local monitors, analyzes it, stores and extracts the key information in order to send it to the cloud application developer. Through the GAS, a user can obtain all the data relating to the detected threats and vulnerabilities of the cloud application. Then, the user identifies the problem and threat and can start working in the solution. The developer studies the risk of threat and, if necessary, can deploy a temporal module that offers the same functionality (in a lower tier) than the vulnerable module. By means of this mechanism, users of the cloud application can use the functionality of the affected module without compromising their private data. As this new module was not created specifically for the cloud application where it is used, it has to be checked against the threats detected in the previous module, to check that it does not bring new unexpected errors or vulnerabilities to the system. This part is very important, since if the new module has

more vulnerabilities than the previous one, the capacity and usability of the cloud application can be worse than before, thus there is no real benefit of applying the process.

The proposed approach guarantees the confidentiality and privacy of the end users' data, as the monitors do not send any of this information to the cloud application developers. The developer creates a new version of the application that fixes the security threat or vulnerability. It is usually done by creating a new version of the threated module or, if necessary, some more modules. After creating the new module, the cloud application developer also creates new rules that can assure the correct functionality of the system with the new module, as it can bring new functionality or has a new data structure. Also, new rules have their equivalent reactions described. They are specified in the rules module of the application, which will be evolved with the rest of the application. Finally, once the developer has checked that the new version fix the threat, it is uploaded to the cloud application providers. The users receive a message indicating that a new version of the application is up and ready to be evolved. When the application evolves, it changes the malfunction module (or modules) and updates the rules module with the new rules and reactions for the evolved modules.

The end user can continue working with the application, with her privacy and confidentiality data protected. The information their monitor provided was used to secure and evolve not only the user application but also all its different instances that are used by different users in different environments. The benefit of this functionality is that even the users that have not experienced the error or security threat are now updated and security-enhanced to this new version. For example, let us suppose that a company is using a cloud application that stores the private financial data of the company. If the monitor detects the error but cannot send this information to the LAS, IPS, or GAS, the cloud application developer will not have enough information to fix the error. The company, I.T. workers, can fix the threat internally, e.g., by fixing ports and disabling some permission, but this will not help the rest of the users of the cloud application or even assure that the application is fully secured. As the same time, if users fix locally the application in one of their computing resources, other resources of the same company will still be vulnerable to the threat and can affect the whole company system, because it does not include reaction procedures defined for the rule violation.

CloudMon provides a solution to the aforementioned problem. In case that an instance of a cloud application is compromised, developers use the information derived from the execution of a particular action to fix the application. In such a way, all instances of the application will be fixed and secured before malicious users can attempt to attack other computing resources or application instances using a security bug in many cases. Once the error has been identified in one instance, all other instances can be secured against the threat only upgrading their applications.

4 ONGOING WORK

Currently, we are working on some specific developments and improvements to the proposed evolution approach.

One of the main lines of ongoing work is related to enhance the CloudMon approach in order to provide more support for application developers. For this target, we have included as a milestone to provide engineering approaches that help developers incorporate monitoring specifications into applications and novel programming models that produce monitoring-ready applications with built-in support for being monitored. Today, cloud applications are developed using tools designed for traditional computing systems, without offering adequate support for developing applications that can be securely and efficiently adapted to the different and possibly evolving characteristics of the cloud infrastructures used to execute them. Applications could easily and systematically evolve, using a "social" strategy that takes advantage of the knowledge gathered from the execution of different instances in different clouds. This lack of cloud-oriented engineering and programming models has some important consequences. First, cloud applications might not be able to take full advantage of existing cloud security mechanisms, as these mechanisms might not be standardized or have a level of complexity that make it hard for application developers to understand and exploit them. Second, security mechanisms hard-coded into applications may interfere with cloud security and other application and infrastructure management mechanisms in ways that reduce the overall level of application security and resilience. Third, assumptions about the cloud infrastructure on which an application was originally planned to be deployed on may have influenced its design in ways that make it difficult (if not impossible) migrating the application to different clouds, even if the latter are part of the same cloud federation.

Due to the aforementioned reasons, developing and programming cloud applications should be based on a clear model of the needed infrastructure. Moreover, the process of developing secure and evolvable applications for the cloud does frequently not only deal with technical problems, but it also involves social and organizational aspects that must be considered in a systematic way from the beginning of the engineering process. Requirements are not limited to technical needs to be satisfied, but also to policies and restrictions that have to be guaranteed in line with the social/organizational setting and within the business policies and processes that will be adopted.

Furthermore, security requirements and policies that affect cloud applications may change during their lifetime. When this situation happens, applications must evolve to ensure the satisfaction of the new security needs whilst making effective use of the available cloud mechanisms. Proper monitoring and evolution techniques must be used at runtime to guarantee continuous alignment between technical evolution of the application and security requirements and properties emerging at social and organizational level. Additionally, we consider that in order to take full advantage of the proposed approach, a way to represent the different underlying trust models used in each cloud computing platform must be established. Such trust model will be used by applications to become aware of and to adapt themselves to their current execution environment. Trust models that are currently assumed for applications executing on platforms owned and/or controlled by application providers themselves are not always adequate when the same applications are to be deployed on a cloud infrastructure, mainly because actors, roles responsibilities, and capabilities changes. For

instance, responsibilities that were originally assigned to the application provider, such as ensuring that changes in the platform do not affect the application or that application code itself is not modified, are transferred to the cloud provider in the new approach. Therefore, new trust models must be defined that are tailored to the cloud computing model and can be supported by specific trust and security mechanisms integrated into cloud infrastructures. These new trust models should also consider different attack models that have been redefined in the cloud environment taking advantage of the previously mentioned changes in roles and responsibilities of the cloud setting.

5 CONCLUSIONS

This article proposes an approach for managing software evolution for cloud applications. The model is based on runtime analysis and control mechanisms that streamline the evolution processes of cloud-based software, and considers evolution along the full application engineering life cycle, in order to guarantee the security and resilience of evolving systems. In order to achieve these goals, the model provides a three-layered monitoring infrastructure to facilitate the operation of monitoring activities and the correct treatment of the monitoring information by different parties.

The proposed architecture increases the security and reliability of cloud computing by making easier the task of identifying the origin of design-flaws. This feature is performed by LAS, IPS, and GAS components, which are able to monitor each application separately or as a whole. In addition, components capture precise and specific information on attacks, errors, and malfunctioning and improve the efficiency (in terms of time) required to identify and fix errors. A proper set of monitoring rules helps with error identification before it further propagates in the application and becomes harder to track. This architecture enables the evolution of applications at runtime supporting the evolution of both specifications and implementations by means of evolution-oriented monitoring mechanisms that enable the analysis of experiences (e.g. runtime data) gathered during the deployment of solutions (services and components) on different platforms.

REFERENCES

[1] Schadt, E. et al. Computational Solutions to Large-Scale Data Management and Analysis. Nature reviews. Genetics 11.9 (2010): 647âĂŞ657. PMC. Web. 18 May 2018.

[2] Kephart, J., Chess, D., “The Vision of Autonomic Computing”, IEEE Computer 36(1), January 2003, 41-50.

[3] Oreizy, P., Gorlick, M., Taylor, R., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D., and Wolf, A., “An Architecture-Based Approach to Self-Adaptive Software” IEEE Intelligent Systems 14, 1999, 54-62.

[4] Kon, F., Costa, F., Blair, G., Campbell, R.H., “The case for reflective middleware”, Communications of the ACM 45(6), 2002, 33âĂŞ38.

[5] Garlan, D., Cheng, S.W., Huang, A.C., Schmerl, B., Steenkiste, P., “Rainbow: architecture based self-adaptation with reusable infrastructure”, IEEE Computer 37(10), October 2004, 46âĂŞ54.

[6] Colby C., Lee P., Necula G.C. (2000) A Proof-Carrying Code Architecture for Java. In: Emerson E.A., Sistla A.P. (eds) Computer Aided Verification. CAV 2000. Lecture Notes in Computer Science, vol 1855. Springer, Berlin, Heidelberg.

[7] Lehman, M., “On Understanding Laws, Evolution, and Conservation in the Large Program Life Cycle”, Journal of Systems and Software 1, 1980, 213âĂŞ221.

[8] Mens, T., Buckley, J., Zenger, M. and Rashid, A. “Towards a Taxonomy of Software Evolution”, In Proceedings of the 1st Workshop on Unanticipated Software Evolution, 2003.

[9] Buckley, J., Mens, T., Zenger, T., Rashid, A. and Kniesel, G. “Towards a Taxonomy of Software Change: Research Articles”, Journal of Software Maintenance and Evolution, 17(5):309âĂŞ332, 2005.

[10] Fernandez-Ramil, J., Perry, D., Madhavji, N.H. (eds.) “Software Evolution and Feedback: Theory and Practice”, Wiley, Chichester, 2006.

[11] Harker, S.D.P., Eason, K.D., Dobson, J.E. “The change and evolution of requirements as a challenge to the practice of software engineering”, IEEE International Symposium on Requirements Engineering, January 1993, 266âĂŞ272.

[12] Lam, W., Loomes, M., “Requirements evolution in the midst of environmental change: A managed approach”, Euromicro Conf. on Software Maintenance and Reengineering, Florence, Italy, March 1998, 121âĂŞ127.

[13] Felici, M., “Observational Models of Requirements Evolution”, PhD thesis, University of Edinburgh, 2004, Available at http://homepages.inf.ed.ac.uk/mfelici/doc/IP040037.pdf

[14] Stark, G., Skillicorn, A., Ameele, R., “An examination of the effects of requirements changes on software releases”, Crosstalk: J. of Defence Software Eng., Dec. 1998, 11âĂŞ16.

[15] Wiegers, K., “Automating requirements management”, Software Development Magazine 7(7), July 1999.

[16] Roshandel, R., Van Der Hoek, A., Mikic-Rakic, M., Medvidovic, N., “Mae âĂŞa system model and environment for managing architectural evolution”, ACM Trans. On Software Engineering and Methodology 13(2), 2004, 240âĂŞ276.

[17] Yu, Y., Wang, Y., Mylopoulos, J., Liaskos, S., Lapouchnian, A., “Reverse engineering goal models from legacy code”, International Conference on Requirements Engineering (REâĂŹ05), Paris, September 2005, 363âĂŞ372.

[18] Muñoz, A; Gonzalez, J; Maña, A, "A Performance-Oriented Monitoring System for Security Properties in Cloud Computing Applications". Accepted for publishing in "The Computer Journal". doi:10.1093.