An Automated SMT-based Security Framework for Supporting Migrations in Cloud Composite Services

Mohamed Oulaaffart, Remi Badonnel, Christophe Bianco

An Automated SMT-based Security Framework for Supporting Migrations in Cloud Composite Services

Mohamed Oulaaffart
RESIST Research Team
LORIA / INRIA Nancy Grand Est,
University of Lorraine, CNRS
615 Rue du Jardin-Botanique
54600 Villers-les-Nancy, France
mohamed.oulaaffart@loria.fr

Remi Badonnel
RESIST Research Team
LORIA / INRIA Nancy Grand Est,
University of Lorraine, CNRS
615 Rue du Jardin-Botanique
54600 Villers-les-Nancy, France
remi.badonnel@loria.fr

Christophe Bianco
RESIST Research Team
LORIA / INRIA Nancy Grand Est,
University of Lorraine, CNRS
615 Rue du Jardin-Botanique
54600 Villers-les-Nancy, France
christophe.bianco@loria.fr

***Abstract*—The growing maturity of orchestration languages is contributing to the elaboration of cloud composite services, whose resources may be deployed over different distributed infrastructures. These composite services are subject to changes over time, that are typically required to support cloud properties, such as scalability and rapid elasticity. In particular, the migration of their elementary resources may be triggered by performance constraints. However, changes induced by this migration may introduce vulnerabilities that may compromise the resources, or even the whole cloud service. In that context, we propose an automated SMT[1]-based security framework for supporting the migration of resources in cloud composite services, and preventing the occurrence of new configuration vulnerabilities. We formalize the underlying security automation based on SMT solving, in order to assess the migrated resources and select adequate counter-measures, considering both endogenous and exogenous security mechanisms. We then evaluate its benefits and limits through large series of experiments based on a proof-of-concept prototype implemented over the CVC4 commonly-used open-source solver. These experiments show a minimal overhead with regular operating systems deployed in cloud environments.**

***Index Terms*—Cloud Security, Resource Migration, Security Automation, Composite Services, Vulnerability Management**

I. Introduction

Cloud infrastructures enable the building of elaborated services through the composition and configuration of multiple computing resources, such as virtual machines, network devices, and software components. These elementary resources may typically be deployed across different infrastructures supported by one or several cloud provider(s), and are subject to changes over time [1]. This dynamics increases the complexity of management tasks and may lead to potential vulnerabilities that may compromise the resources, or even the whole cloud composite service. In particular, the cold and hot migrations of cloud resources are currently facilitated by recent advances on virtualization techniques [2], allowing to transfer the resource(s) of a cloud composite service from a given provider (or a given infrastructure) to another provider (or infrastructure) [3]. This process is often motivated by performance and cost objectives with regard to the cloud properties, such as high scalability, rapid elasticity, resource pooling and on-demand self-service [4]. However, it may directly impact the protection of cloud services and increase their exposure to security attacks. Indeed, the different changes that affect the migrated resources and their dependencies may involuntarily generate vulnerabilities that are then exploitable by malicious users to potentially cause important damages, including loss, disclosure, and even tampering of data [5].

We propose in this paper an automated SMT-based security framework for supporting migrations in cloud composite services, such as those orchestrated with the Topology and Orchestration Specification for Cloud Applications (TOSCA) [6]. We have already argued in favor of such security automation by showing how it contributes and fits with risk management in [7]. Our objective here is to exploit SMT solving for automatically assessing the configuration changes that affect the resources of cloud services during their migration and determining adequate counter-measures. After overviewing the considered strategy and the underlying challenges, we will describe the proposed SMT-based security framework, its main components and their interactions for enabling security automation based on verification techniques. It relies on three main phases. The first phase establishes a projection of the resource configuration induced by the migration, considering the changes that affect the migrated resource and its context. It then assesses this configuration by exploiting the knowledge provided by vulnerability descriptions. A vulnerability description can be seen as a logical combination of tests to be performed on a given configuration, each test associating a configuration parameter to an expected state. The third phase consists in determining potential counter-measures to be executed on the resource itself, or to be activated in

Supported by the project CONCORDIA that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 830927. We would like to thank Professor Olivier Festor for his very helpful comments and advices on the submitted article.

[1]Satisfiability Modulo Theories

its environment to prevent the observed vulnerabilities, and maintain the protection of the whole cloud composite service.

The main contributions of this paper include: (1) the design of a security framework for supporting the resource migration in cloud composite services, considering different security knowledge sources, (2) the SMT-based formalization supporting the assessment of migrated resources and the selection of counter-measures addressing identified vulnerabilities, (3) the development of a proof-of-concept prototype using the CVC4 solver as a back-end service, and (4) the performance evaluation of our solution based on extensive experiments.

The remainder of this paper is therefore organized as follows. First, Section II gives an overview of existing work in the area of cloud security automation. Section III then describes the considered security framework and illustrates its operation through a concrete example. Section IV formalizes the underlying security automation based on SMT solving. Section V details the proof-of-concept prototype implemented on top of the CVC4 open-source solver, and the performance evaluation based on extensive series of experiments. Finally, Section VI provides the conclusion and points out future research perspectives.

II. Related work

Automating the protection of cloud services and their resources is a major challenge, already investigated in the literature through several security approaches. This challenge is increased by the growing complexity, distribution and dynamics of cloud composite services. These services are leveraged by dedicated orchestration languages, such as the OpenStack Heat Orchestration Template (HOT) [8], the OASIS TOSCA [6], and the AWS CloudFormation language [9]. OASIS TOSCA constitutes an open-source orchestration language, in comparison to proprietary solutions such as AWS CloudFormation, and is characterized by a more advanced expressivity than OpenStack HOT, in order to build and operate flexible and interoperable applications. While orchestration languages facilitate their management, these cloud composite services are facing a high exposure to attacks, in particular during the migration of their resources [10].

Existing solutions to protect cloud resources include endogenous security mechanisms that directly impact the resources, such as generating specific cloud images with a low attack surface, modifying the internal parametrization to prevent vulnerable configurations, and exploiting certification techniques for guaranteeing the resource behaviours. For instance, the authors of [11] propose to extend an orchestration language to drive the generation of protected unikernel images, corresponding to lightweight virtual machines composed of only the strict necessary packages and libraries. In this case, any configuration changes require the re-generation of new virtual machines. Safe configuration methods, such as developed in [12], consist in analyzing and modifying the current configuration of a given resource to maintain it in a state compliant with the security policy, based on best practices or vulnerability descriptions. This enables the identification of potential unsafe configuration states, and select corrective operations, if available, to modify the configuration of the concerned cloud resources before their migration. Furthermore, certification solutions, such as [13], [14], enable the elaboration of a trusted ecosystem for cloud resources. They consist in exploiting certificates guaranteeing the resource behaviours. These approaches are initiated with the certification of cloud resources in a controlled environment. After the deployment of cloud services, the resources are continuously tested in order to control their behaviours and maintain the validity of certificates. These security mechanisms are restricted by the operations that are allowed on the considered resources, such as the availability of security patches to be applied. This may considerably reduce the possible counter-measures with respect to a given resource.

Complementarily, exogenous security mechanisms consist in protecting the cloud resources by exploiting external security functions [15]. For instance, the authors of [16] propose a security orchestrator implemented based on the NFV MANO[2] architecture to deploy network security functions in front of the considered resources. It is centered on access control rules that are specified using an orchestration language extension. Audit approaches such as [17], [18] aim at evaluating cloud service providers before the migration of resources, and then during the operation of resources. Evidence may typically be collected from the cloud service providers, in order to determine the level of trust with respect to the infrastructure of a given provider, and its capability to comply with the expected security policy. In addition, approaches such as [19] consider formal verification methods applied to security chains that are built from different security functions, such as intrusion detection systems, firewalls, and data leakage prevention mechanisms. The objective is to synthesize automatically security chains by analyzing the network traffic of resources and inferring behavioral models, and to detect any potential inconsistencies and redundancies that may occur in the rules of given security chains, through the exploitation of model checking techniques. However, they do not take into account the internal configuration of resources.

In order to reinforce the security of cloud resources, several approaches have focused on trust features for cloud infrastructures, in particular considering the virtualization systems used in these environments. Trusted computing technologies provide hardware and software support for secure storage and software integrity protection. Paired with encryption mechanisms, it aims at guaranteeing a trusted environment during the interactions with other infrastructures, such as the interactions required by the migration of cloud resources. For instance, the authors of [20] introduce a remote attestation protocol specifically designed for supporting the interactions amongst a federation of cloud infrastructures. The migration of resources are only allowed amongst infrastructures that are part of the defined federation. In the same manner, the authors of [21] propose a solution to prevent the migration of virtual machines

[2]Management And Network Orchestration

to untrusted platforms, without requiring a third party entity to issue trust certificates. Complementary to these solutions, our approach goes a step forward by leveraging the knowledge provided by orchestration languages to support cloud resource migrations, in order to prevent configuration vulnerabilities that may occur even in fully trusted cloud environments.

III. Security Automation Approach

We will now describe our security automation approach for supporting the migration of cloud resources that may affect cloud composite services. After overviewing the challenges related to this automation, we will present the considered framework, detail its main building blocks and illustrate its operation based on a practical example.

A. Automation challenges

The migration of cloud composite services consists in reallocating one or several resources of the considered services from one cloud infrastructure to another cloud infrastructure owned by the same cloud service provider, or by an alternative one. It may be motivated by several resource optimization strategies, such as load balancing amongst providers to improve scalability, consolidation to maximize the efficient usage of servers, or co-location to minimize the latency amongst cloud infrastructures. These migrations are currently leveraged by the advances on virtualization techniques, that increase their performance at runtime, without requiring the instantiation of a new cloud resource. Architectural patterns of cloud-native applications, contrarily to monolithic applications, also contribute to their flexible operations. However, these migrations may introduce vulnerabilities, and therefore introduce new security automation challenges with that respect. Figure 1 illustrates the case of a cloud composite service hosted over three different cloud service providers, noted $csp1$, $csp2$, and $csp3$. This composite service is built from a set of elementary resources, noted c_i, and represented by orange nodes with a continuous border. For instance, the resources c_5 to c_9 are deployed over the second cloud service provider $csp2$. Let consider that the cloud resource c_9, that may typically stand for a micro-service, undergoes a migration m_1 (represented by the black arrow) from the cloud provider $csp2$ to the cloud provider $csp3$, motivated by performance considerations. The resource after migration is represented by the orange node c'_9 with a dotted border. The considered migration supposes a contextual change for the resource, that may increase its exposure to security attacks, and may imply dedicated countermeasures, such as the exogeneous security functions, represented by nodes f_1 and f_2, and corresponding here to firewalls. In that context, security automation is required to support the migration of such a resource in cloud composite services. Amongst the main challenges related to this automation, it should first take benefits from the specification of cloud composite services, in particular the orchestration language, that provides important knowledge on the context of the migrated resource, but also on the dependencies that may exist amongst the migrated resource and the other resources that compose

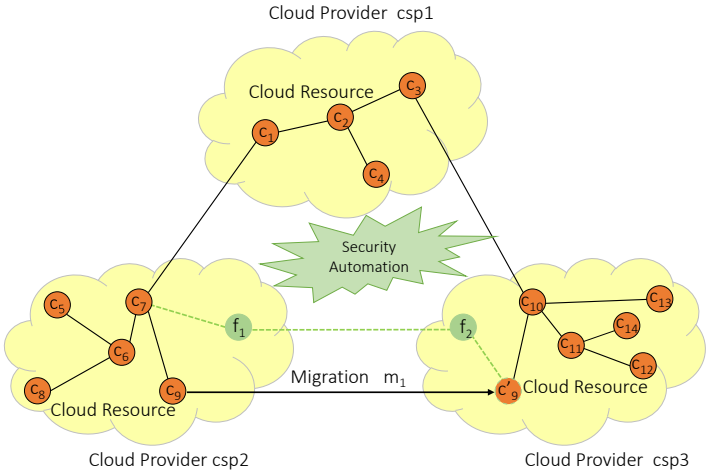

Fig. 1: Illustrative example of security automation for supporting the migration of a resource, noted c_9, in a cloud composite service, using two exogenous security functions, noted f_1 and f_2

the service. The changes affecting the resource may impact the security of the whole composite service. Second, this automation should exploit the knowledge sources corresponding to vulnerability descriptions, in order to automatically assess the resource configuration and identify potential configuration vulnerabilities with their severity. Third, it should take into account both the endogenous mechanisms that may be available for the considered resource (such as an update of the resource), and the exogenous mechanisms that may be supported by cloud service providers, in order to reduce the attack surface.

B. Security framework for cloud migration

In order to address these automation challenges, we have designed a security framework for supporting resource migrations in cloud composite services. The different building blocks of this framework and their interactions are illustrated on Figure 2. The framework is triggered when a resource of the composite service undergoes a migration. This composite service, represented on the top of the figure, is orchestrated using the TOSCA orchestration language, which describes its elementary resources as well as their relationships. Concretely speaking, the TOSCA language specifies a cloud service as a TOSCA topology which defines a set of TOSCA nodes (e.g. a web micro-service node) that are interconnected by a set of TOSCA relationships (e.g. the interconnection to a MySQL database node). An extract of such a TOSCA specification with different nodes (in orange color) and relationships (in blue color) is detailed in Figure 3, corresponding to a high-level representation without versions and parameters. As previously mentioned, the TOSCA language is both open-source and characterized by a high expressivity to support flexible and interoperable applications. Our framework also relies on complementary knowledge sources serving as inputs, namely vulnerability descriptions, that specify vulnerable configura-

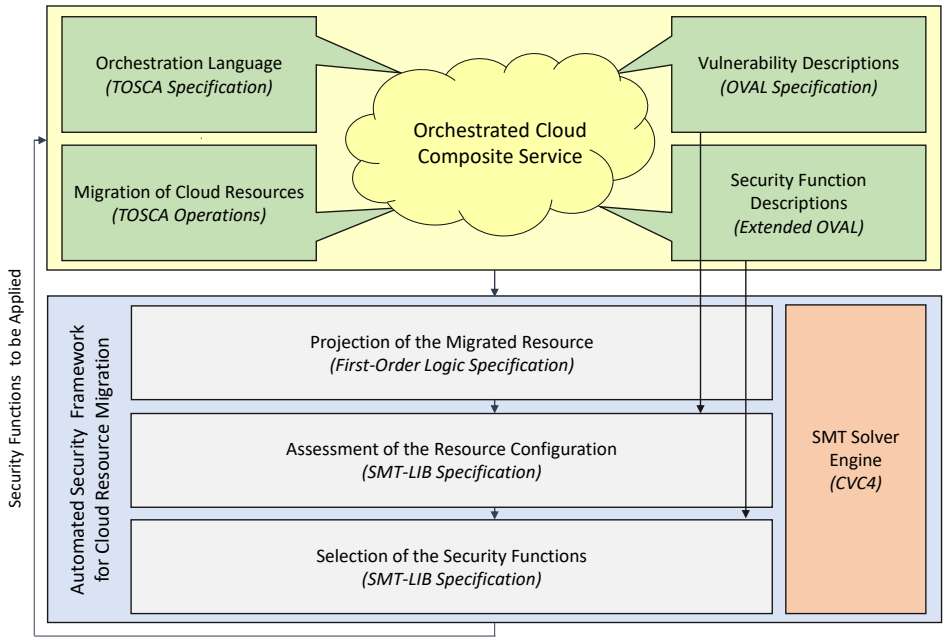

Fig. 2: Main building blocks of the considered security framework for cloud migration

tions using the Open Vulnerability and Assessment Language (OVAL) [22], and security function descriptions that reuse the same specification to define the counter-measures that can be applied on the cloud resource in an endogenous or exogenous manner. The OVAL language, part of the Security Content Automation Protocol (SCAP) [23] developed by NIST, has become the de-facto standard for describing configuration vulnerabilities in a machine readable manner. Complementing Common Vulnerabilities and Exposures (CVE) definitions that only provide literal descriptions, it specifies each vulnerability as a logical combination of tests/conditions that if observed on the target system, the security problem described by such vulnerability is present on that system. Each OVAL definition corresponds to a criterion that logically combines a set of OVAL tests. Each OVAL test in turn examines an OVAL object (e.g. an Apache Tomcat web server) looking for a specific OVAL state (e.g. a given version for this server). Components found in the system matching the OVAL object are called OVAL items. These items are compared against the specified OVAL state in order to build the OVAL test result. The overall result for the criterion specified in the OVAL definition are built using the results of each referenced OVAL test.

Our framework is structured into three main building blocks. The first one stands for the projection of the migrated resource. The objective is to determine the new configuration of the migrated resource, as well as the other impacted resources using the specification of the cloud composite service. This projection is performed prior to the effective migration, in order to prevent the occurrence of vulnerable configurations that may expose the composite service to security attacks. The analysis of dependencies permits to identify the other resources that may be impacted by the migration, considering the TOSCA specification. The second building block corresponds to the assessment of the resource configuration based on vulnerability descriptions. The analysis relies on the projection of the migrated resource. It permits to compare the resource configuration expected after the migration to a given set of vulnerability descriptions, and to determine whether the projection matches one or several of the specified vulnerable configurations. The severity of vulnerabilities can also be determined based on the Common Vulnerability Scoring System (CVSS) associated to OVAL vulnerability descriptions, and providing a quantification of the ease and impact of exploits. The third building block corresponds to the selection of counter-measures. It only applies when the assessment block has identified a vulnerable configuration. In that case, the objective is to select adequate security functions to prevent the occurrence of the vulnerability when the migration is effective. The considered counter-measures include security functions that are provided by the cloud service providers, or that can directly be executed on the resource itself. We have formalized the underlying security automation using SMT solving to assess the resource configuration and determine adequate counter-measures. In particular, the two last building blocks exploit a SMT solver, namely the CVC4 solver, as a back-end service. According to recent benchmarking on opensource solvers [24], the CVC4 solver provides better overall performance than other SMT solvers, such as Z3 or VeriT.

C. Operation through an illustrative example

Let us consider a concrete example with an e-commerce cloud composite service involving a set of micro-services and

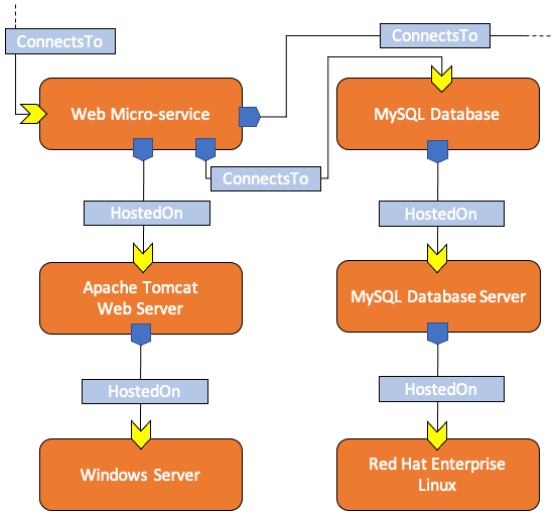

Fig. 3: Extract of a TOSCA topology specification describing a cloud service composed of different nodes and relationships (high-level representation without versions and parameters)

backend databases, orchestrated with the TOSCA language. This composite service integrates a micro-service resource (corresponding to the c_9 node), implemented in the form of a web application built with the Struts open source framework, that extends the Java Servlet API. The micro-service interacts with other micro-services and backend MySQL databases, corresponding to nodes c_1 and c_6 on Figure 1. It is deployed over an Apache Tomcat web server 9.0.29 running on top of a virtualized Windows Server 2019 instance. Due to performance optimization, the micro-service undergoes a migration to a new cloud service provider, that offers the same version of the operating system, but supports the version 9.0.18 of the Apache Tomcat server, as well as the version 2.3.31 of the Struts framework. Our security framework first established the projection of the migrated resource, by considering the new versions of these resources. It then initiates an assessment of the projected resource configuration based on vulnerability descriptions. This assessment is automatically performed by translating the projected configuration and the vulnerability descriptions in the form of first-order logic expressions, that are then interpreted by the CVC4 solver to identify potential matches. It detects one configuration vulnerability referring to CVE-2017-5638 for the cloud resource, and due to the new Struts framework version. This vulnerability can be exploited to perform remote code execution, by allowing a remote attacker to inject operating system commands into the web application through the content-type header. As the configuration is identified as vulnerable, the security framework then executes the selection of counter-measures, formalized as a satisfiability issue, to determine which corrective operations are possible, using again the SMT solver. Based on the security function descriptions, it identifies two potential counter-measures to address the considered vulnerability. The first one is endogenous, and corresponds to the execution of a software patch which updates the version of the Struts framework to version 2.5.26, while the second one is exogenous and corresponds to the activation of a web application firewall (WAF) with specific rules. As the cloud service provider does not allow clients to modify its WAF rules, a software patch is finally executed on the considered resource to update the Struts framework, and prevent the occurrence of the vulnerability before the migration is performed.

IV. Automation based on SMT solving

In order to support this framework, we formalize the underlying security automation based on SMT solving. The assessment of the migrated resource, and the selection of security functions to be applied, is modeled as a satisfiability problem. We consider a cloud service composed of $C = \{c_1, c_2, \dots\}$ standing for the set of resources (also called components). Each component c_i, such as a Struts micro-service, is in turn characterized by a set of properties $P = \{p_1, p_2, \dots\}$ that can be seen as unary predicates $p_i(c)$ defined for the considered component and its environment. For instance, a predicate can relate to the version of the Struts framework. These predicates permit to define the properties that the component possesses, as well as to specify the properties to be observed for the configuration assessment. We then introduce $S = \{s_1, s_2, \dots\}$ the set of component states describing in a compact manner a set of properties required to be observed over a cloud component, which can be defined as follows:

1) if $p_i \in P$, then $p_i \in S$ with $i \in \mathbb{N}$
2) if $\alpha, \beta \in S$, then $(\alpha \diamond \beta) \in S$ with $\diamond \in \{\wedge, \vee\}$
3) if $\alpha \in S$, then $(\neg\alpha) \in S$.

In the meantime, the function $state : C \rightarrow S$ takes a cloud component $c \in C$ as input and returns its current state $s \in S$. For instance, $state(c_i)$ can provide the different properties related to the Struts micro-service, such as the versions of the Struts framework and of the Apache Tomcat webserver.

A. Assessment of the migrated resource

The migration of the component c_i from a cloud service provider csp_i to another cloud service provider csp_j may lead to new configuration vulnerabilities. We introduce the following definitions related to the migration:

- $M = \{m_1, m_2, \dots\}$ denotes the set of migrations that can be applied over a cloud composite service during its operation, they may concern one or several components c_i of the given composite service.
- $impact : M \rightarrow S \equiv$ function that takes a migration $m \in M$ as input and returns a state $s \in S$ that projects the affected characteristics corresponding to the migration, independently from the considered component.
- $\Pi : S \times S \rightarrow S \equiv$ function that takes a projected state $s_1 \in S$ together with a component state $s_2 \in S$ as inputs and returns s_2 updated with the properties of $s1$.
- $migrate : C \times M \rightarrow S \equiv$ function that corresponds to a migration $m \in M$ applied on a component $c \in C$,

and that returns the state of the component c after the operation of the given migration.

The assessment of the migrated resource is performed based on a set of vulnerability descriptions $V = \{v_1, v_2, \ldots, v_m\}$. For instance, one of these descriptions refers to the CVE-2017-5638 presented in the illustrative example. As each vulnerability description can be specified as a logical formula corresponding to a state $s \in S$, the vulnerability dataset can be seen as a disjunction of logical formulas given by $\phi = v_1 \vee v_2 \ldots \vee v_n = \bigvee(v_i)$ with $v_i \in V$. Considering these definitions, we specify an assessment function $\Phi : S \rightarrow Boolean$ that determines if a component $c \in C$ is vulnerable under V, meaning that the assessment of ϕ over the state of c is true. As a consequence, we assess the projection of the migrated resource under V using Equation 1, where the projection is obtained with the Π function applied to $impact(m_i)$ and $state(c)$.

$$\Phi(\Pi(impact(m_i), state(c)))with m_i \in M, c \in C \quad (1)$$

Depending on the results of this assessment, the resource migration can be effectively performed on the cloud composite service using the $migrate(c, m)$ function.

B. Selection of counter-measures

When vulnerabilities are identified, the security framework initiates the selection of counter-measures. We consider $F = F_{ex} \cup F_{en} = \{f_1, f_2, \ldots\}$, the set of available security functions, with F_{en} and F_{ex} standing respectively for the endogenous and exogenous ones. For instance, the web application firewall used to protect the Struts micro-service is specified as an exogenous security function $f \in F_{ex}$. We then consider the following definitions to support the selection:

- $future : F \rightarrow S \equiv$ function that takes a subset of security functions $F' \subset F$ as input and returns a state $s \in S$ that projects the affected characteristics after the application of the considered security functions.
- $activate : F \times C \rightarrow S \equiv$ function that corresponds to the effective application of security functions $F' \subset F$ on the component c and returns the new component state.

We then exploit the assessment function Φ to support the selection of counter-measures using Equation 2, where $\Pi(impact(m_i), state(c))$ provides the resource projection after migration, and the Π function is used a second time on this projection to assess the resource state after the application of the security functions using $future(F')$.

$$\Phi(\Pi(future(F'), \Pi(impact(m_i), state(c)))) \quad m_i \in M, F' \subset F, c \in C \quad (2)$$

Once the security functions are properly selected by the security framework, their application can be effectively performed using the $activate(F', c)$ function over the cloud composite service, in a proactive or reactive manner.

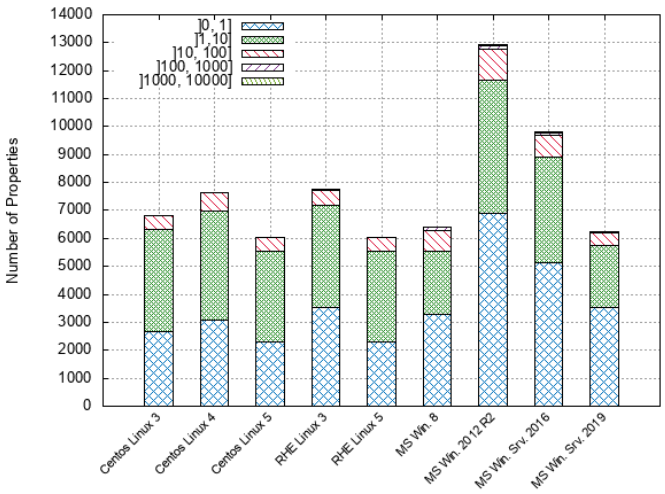

Fig. 4: Distribution of properties specified by vulnerability descriptions for each considered vulnerability database

V. Performance evaluation

We have then evaluated the performance of the proposed security framework through extensive series of experiments. In that context, we have developed a proof-of-concept prototype written in Python, which supports the different blocks of the framework, and is built on top of the open-source CVC4 SMT solver. The prototype internally generates a SMT-LIB (Satisfiability Modulo Theories LIBrary) specification, which can also be interpreted by other SMT solvers, such as Z3 and VeriT. We have performed the experiments over a regular laptop equipped with a 2Ghz Intel Core i5 processor and 8GB of RAM memory. We have considered a simple TOSCA specification, and have used vulnerability descriptions coming from the official OVAL repository, considering the specifications for various operating systems corresponding to the largest datasets, namely CentOS Linux, RedHat Enterprise Linux and Microsoft Windows with different versions. The same OVAL formalism is exploited for defining the tests that are used to characterize the configuration states and their properties, and for specifying the description of security functions that are applicable for the cloud resource, such as the application of patches or the activation of a web application firewall with specific security rules. Selecting a cloud service provider amongst multiple ones is a challenge in itself, and is considered as out of the scope of this paper.

A. Distribution of vulnerability properties

In a first preliminary series of experiments, we were interested in assessing the distribution of properties (or tests) amongst the considered vulnerability descriptions, each OVAL test being associated to a given property. The objective was to quantify the redundancy of tests that are used to evaluate properties for the vulnerability datasets. For that, we have analyzed the OVAL datasets for the different operating systems, and measured the occurrence of tests for each of them.

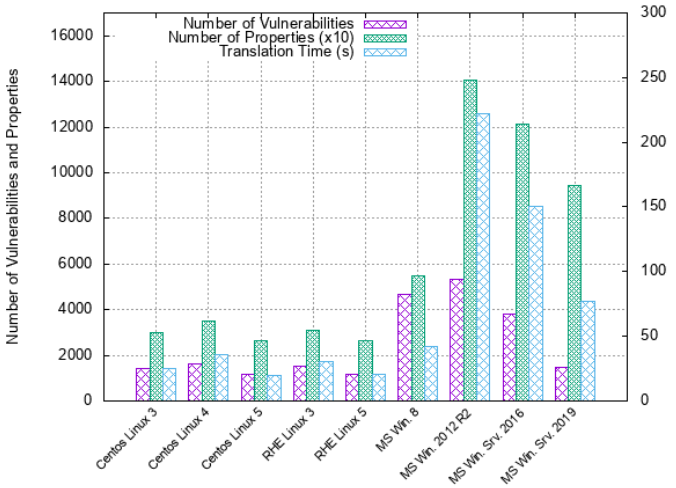

Fig. 5: Translation time of vulnerability descriptions into logical specifications for each considered vulnerability dataset

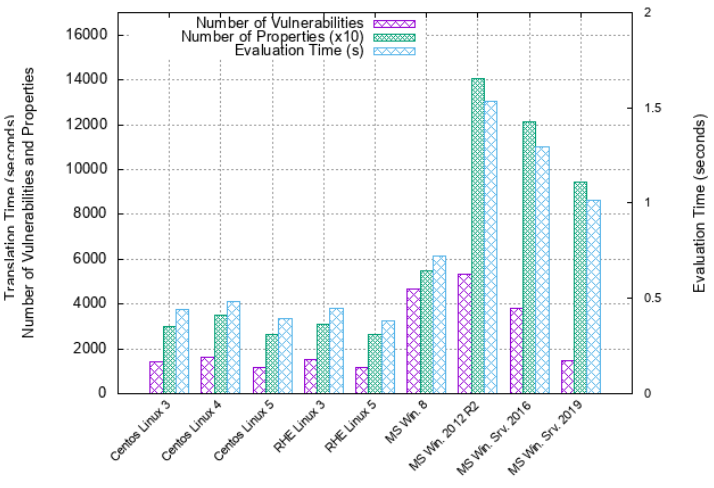

Fig. 6: Evaluation time for assessing the projection of the migrated cloud resource with respect to each considered dataset

Figure 4 describes this distribution per operating system, with the horizontal axis indicating the system and its version, and the vertical axis corresponding to the number of properties. We considered five different occurrence categories using a logarithmic scale, with tests that are present respectively once, up to 10, up to 100, up to 1000, and up to 10000 times in a vulnerability dataset. We have observed that a large proportion of tests (or properties) are present between 2 and 10 times, representing on average 44,15% for the different datasets. The other highest categories are less significant with around 8,22% of tests that occur between 11 and 100 times, and less than 1% on average for tests occurring up to 1000 and 10000 times. The distributions are relatively homogeneous amongst operating systems, with the highest values of occurrences being observed with the Microsoft Windows operating system. This distribution analysis shows that more than half of the tests or properties (around 53% on average) occur more than once in the different datasets, arguing in favor of our solution based on SMT solving, which is usually efficient in simplifying such satisfiability problems with redundant constraints.

B. Performance of migrated resource assessment

In next series of experiments, we wanted to quantify the time required for translating the vulnerability descriptions into logical specifications, and for assessing the projection of the cloud migrated resource with respect to the different vulnerability datasets. Figure 5 describes the time required by the security framework to automatically translate a vulnerability dataset V into a logical specification with a conjunctive normal form that is then directly interpretable by the SMT solver. Our framework generates a conjunctive normal form by construct, in order to prevent any additional pre-processing to be performed by the solver afterwards. The figure details the translation time for each operating system dataset, with regard to the number of vulnerabilities and properties. We can observe that this translation time remains below 225 seconds, corresponding to the largest dataset with Microsoft Windows 2012R2. It is mainly dependent on the number of properties, which relates itself to the number of vulnerability descriptions.

In the meantime, Figure 6 depicts the evaluation time required for assessing the projection of the migrated resource with respect to a given vulnerability dataset. The time includes the interpretation of the SMT-LIB specification together with the solving of the satisfiability issue. We can observe a linear behavior with respect to the number of tests (or properties). The minimal evaluation time has been obtained with RedHat Entreprise Linux 5, corresponding to the smallest dataset, while the maximal time corresponds again to Microsoft Windows 2012R2, which is the largest dataset. The evaluation time is clearly smaller than the translation time, and is kept under 1,6 seconds for all the vulnerability datasets, confirming the feasibility of our approach. The translation activity should rather be performed proactively, as soon as the publication of new vulnerability descriptions.

C. Performance of counter-measure selection

In a last series of experiments, we wanted to quantify the assessment time required for selecting a given security function (or a set of security functions). In particular, we have looked at the frequency of the properties that are impacted by a given security function, and how it influences the assessment time to determine whether the new configuration of the migrated resource is vulnerable or not with regard to this security function. In that context, we have considered three different scenarios, corresponding to security functions that impact respectively the most frequent properties (high scenario), the less frequent properties (low scenario), and properties with an averaged frequency (medium scenario) in the considered vulnerability datasets, as illustrated on Figure 7 describing the assessment time for selecting a security function

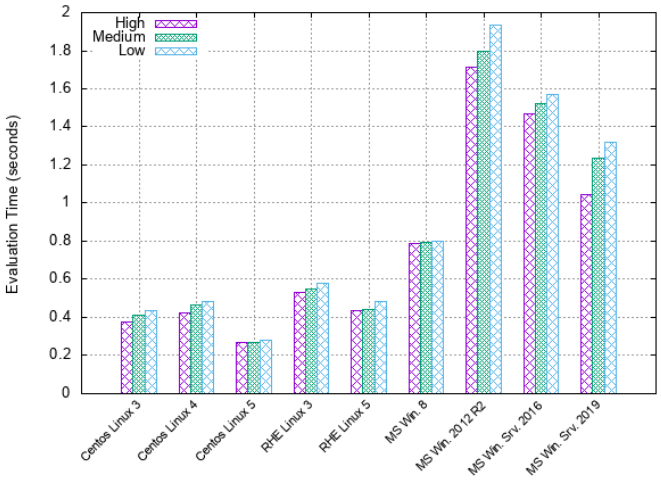

Fig. 7: Evaluation time for selecting a security function depending on the frequency of impacted properties

with the different vulnerability datasets. These scenarios have been chosen to guarantee an homogeneous distribution of considered properties, amongst the less frequent, moderately frequent and most frequent categories. We expected that the security functions corresponding to the scenario with the highest frequencies provides the best assessment time for selecting security functions, which was indeed the case in our experiments performed with the different vulnerability datasets, by considering various cloud resource migrations. In addition, we can observe the influence of the dataset size. In particular, the differences amongst the three scenarios are the highest for the largest datasets with regard to the evaluation time. For instance, this difference is of less than 8% for the smallest vulnerability datasets, while it reaches around 15% for the largest vulnerability datasets. The selection of security functions might therefore be prioritized with respect to the frequency of properties that are impacted by such functions, in addition to other performance criteria, such as the cost induced by a given security function provided by a cloud provider. Considering the modeling and results presented in [25] on live migration times with a virtual machine of 1024 MB and different migration link speeds (from 100 Mbps to 10 Gbps), we quantified the average overhead for both the assessment and selection phases in comparison to these different baseline scenarios. While using a regular laptop, we observed overheads of less than 10%, as shown in Table I which depicts on average the migration times and overheads. This means that the time required for assessing a migrated resource and selecting the countermeasures has a low impact on the overall time for performing the live migration.

TABLE I: Assessment and selection overhead considering different live migration baseline scenarios with a 1024 MB virtual machine

Migr. Link	Migr. Time	Assess. Overhead	Select. Overhead
100 Mbps	277,2s	0,25%	0,30%
1 Gbps	31,6s	2,14%	2,56%
10 Gbps	7,7s	8,22%	9,74%

VI. Conclusions and Future Work

The large-scale deployment of cloud composite services has been leveraged by the growing maturity of orchestration languages. The advances on virtualization techniques facilitate the migration of resources that compose these services, but such changes may also introduce new vulnerabilities and increase the exposure to attacks. In that context, we have proposed a SMT-based security framework for automatically supporting the migration of resources in cloud composite services. It bridges the gap between the orchestration language that specifies the cloud composite services, and the vulnerability descriptions that define the configuration states that should be prevented for the migrated resource. We have described the different building blocks of the framework and their interactions. They operate according to three main phases, namely the projection of the migrated resource, the assessment of the corresponding configuration based on vulnerability datasets, and the selection of adequate security functions when this configuration appears to be vulnerable. We have shown the framework applicability based on an illustrative example corresponding to a Struts micro-service, part of a cloud composite service, undergoing a migration amongst cloud service providers. This security automation has been formalized using SMT solving, and implemented into a proof-of-concept prototype written in Python on top of the CVC4 solver. Finally, we have performed large series of experiments based on existing vulnerability descriptions coming from the official OVAL repository, and corresponding to different operating systems. This enabling to evaluate the benefits and limits of our proposed framework, with respect to various aspects, such as the formalization into the SMT-LIB format interpretable by the solver, the assessment time of the projection of the migrated resource, and the automated selection of security functions that can be both endogenous or exogenous to the cloud resource. These different results compared to live migration baseline scenarios confirm the feasibility of our proposed solution.

As future work, we are interested in evaluating complementary criteria for supporting the selection of security functions, including the different costs that are associated to their activation, such as the additional delays that they may introduce for the operation of the considered services. We are also planning to further investigate the extension of the TOSCA orchestration language with respect to such resource migrations in cloud composite services. In particular, we would like to evaluate its usage for supporting the interactions with multiple cloud providers based on a trusted third party.

References

[1] J. P. Barrowclough and R. Asif, “Securing Cloud Hypervisors: A Survey of the Threats, Vulnerabilities, and Countermeasures,” *Security and Communication Networks*, 2018.

[2] J. Narantuya, H. Zang, and H. Lim, "Service-aware cloud-to-cloud migration of multiple virtual machines," *IEEE Access*, vol. 6, pp. 76 663–76 672, 2018.

[3] M. C. S. Filho, C. de Castro Monteiro, P. R. M. Inácio, and M. M. Freire, "Approaches for Optimizing Virtual Machine Placement and Migration in Cloud Environments: A Survey," *Journal on Parallel Distributed Computing*, vol. 111, pp. 222–250, 2018.

[4] N. S. Dey and T. Gunasekhar, "A Comprehensive Survey of Load Balancing Strategies Using Hadoop Queue Scheduling and Virtual Machine Migration," *IEEE Access*, vol. 7, pp. 92 259–92 284, 2019.

[5] D. Fernandes, L. Soares, J. Gomes, M. Freire, and P. Inácio, "Security Issues in Cloud Environments - a Survey," *International Journal of Information Security*, p. 113–170, 2013.

[6] C. L. Paul Lipton. (2020) TOSCA Simple Profile in YAML Version 1.3. [Online]. Available: https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/os/TOSCA-Simple-Profile-YAML-v1.3-os.html

[7] M. Oulaaffart, R. Badonnel, and O. Festor, "Towards Automating Security Enhancement for Cloud Services," in *Proceedings of the International Symposium on Integrated Network Management (IM 2021), IFIP Digital Library*, 2021.

[8] A. Esposito, B. Di Martino, and G. Cretella, "Defining Cloud Services Workflow: a Comparison between TOSCA and OpenStack Hot," in *Proceedings of the International Conference on Complex, Intelligent, and Software Intensive Systems*, 2015.

[9] AmazonWebServices, "AWS CloudFormation - API Reference," p. 283.

[10] N. Shahapure and P. Jayarekha, "Virtual Machine Migration Based Load Balancing for Resource Management and Scalability in Cloud Environment," *International Journal of Information Technology*, 2018.

[11] M. Compastié, R. Badonnel, O. Festor, and R. He, "A TOSCA-Oriented Software-Defined Security Approach for Unikernel-Based Protected Clouds," in *Proceedings of the IEEE Conference on Network Softwarization (NetSoft 2019)*, 2019, pp. 151–159.

[12] M. Barrere, R. Badonnel, and O. Festor, "A SAT-based Autonomous Strategy for Security Vulnerability Management," in *Proc. of the IEEE Network Operations and Management Symposium (NOMS)*, 2014.

[13] M. Anisetti, C. A. Ardagna, and E. Damiani, "Security Certification of Composite Services: A Test-Based Approach," in *Proceedings of the IEEE International Conference on Web Services (ICWS 2013)*, 2013.

[14] M. Anisetti, C. Ardagna, E. Damiani, and F. Gaudenzi, "A Semi-Automatic and Trustworthy Scheme for Continuous Cloud Service Certification," *IEEE Transactions on Services Computing*, Jan. 2017.

[15] V. Sciancalepore, F. Z. Yousaf, and X. Costa-Perez, "z-TORCH: An Automated NFV Orchestration and Monitoring Solution," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, 2018.

[16] M. Pattaranantakul, R. He, Z. Zhang, A. Meddahi, and P. Wang, "Leveraging Network Functions Virtualization Orchestrators to Achieve Software-Defined Access Control in the Clouds," *IEEE Transactions on Dependable and Secure Computing*, Dec. 2018.

[17] U. M. Ismail, S. Islam, and H. Mouratidis, "Cloud Security Audit for Migration and Continuous Monitoring," in *Proceedings of the International Conference on Trust and Trustworthy Computing (TrustCom)*, vol. 1, Aug. 2015.

[18] K. W. Ullah, A. S. Ahmed, and J. Ylitalo, "Towards Building an Automated Security Compliance Tool for the Cloud," in *Proceedings of the International Conference on Trust and Trustworthy Computing (TrustCom)*, 2013, pp. 1587–1593.

[19] N. Schnepf, R. Badonnel, A. Lahmadi, and S. Merz, "Automated Verification of Security Chains in SDN Networks with Synaptic," in *Proceedings of the Conference on Network Softwarization (NetSoft 2017)*, 2017.

[20] A. Celesti, A. Salici, M. Villari, and A. Puliafito, "A Remote Attestation Approach for a Secure Virtual Machine Migration in Federated Cloud Environments," in *Proceedings of the International Symposium on Network Cloud Computing and Applications*, 2011, pp. 99–106.

[21] M. Aslam, C. Gehrmann, and M. Björkman, "Security and Trust Preserving VM Migrations in Public Clouds," in *Proceedings of the IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, 2012, pp. 869–876.

[22] J. Baker, M. Hansbury, and D. Haynes, "The OVAL Language Specification, Version 5.11.2," *MITRE Corporation*, 2016.

[23] D. Waltermire, S. Quinn, K. Scarfone, and A. Halbardier, "The Technical Specification for the Security Content Automation Protocol (SCAP): SCAP version 1.3," *NIST Special Publication*, vol. 800, p. 126, 2018.

[24] H. Barbosa, C. Barrett, F. Bobot, and M. Brain. (2020) About CVC4. [Online]. Available: https://cvc4.github.io/

[25] S. Akoush, R. Sohan, A. Rice, A. W. Moore, and A. Hopper, "Predicting the Performance of Virtual Machine Migration," in *Proceedings of the IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (IEEE MASCOTS)*. IEEE Computer Society, 2010, p. 37–46.