A Framework for the Attack Tolerance of Cloud Applications Based on Web Services

Georges Ouffoué, Fatiha Zaïdi, Ana R Cavalli, Huu Nghia Nguyen

electronics

MDPI

Article

A Framework for the Attack Tolerance of Cloud Applications Based on Web Services †

Georges Ouffoué [1], Fatiha Zaïdi [2], Ana R. Cavalli [3,4] and Huu Nghia Nguyen [3,*]

1 APL Data Center, Montrouge 92120, France; georges.ouffoue@apl-france.fr
2 Univer Paris-Saclay, CNRS, Laboratoire de recherche en informatique, Orsay 91405, France; zaidi@lri.fr
3 Montimage Paris, Paris 75013, France; ana.cavalli@it-sudparis.eu
4 Institut Polytechnique de Paris, Telecom SudParis, Evry 91000, France
* Correspondence: huunghia.nguyen@me.com

† This paper is an extended version of our paper published in The 31st IFIP International Conference on Testing Software and Systems.

Received: 21 October 2020; Accepted: 15 December 2020; Published: date

Abstract: Information systems of companies and organizations are increasingly designed using web services that allow different applications written in different programming languages to communicate. These systems or some parts of them are often outsourced on the cloud, first to leverage the benefits of cloud platforms (e.g., scalability) and also to reduce operational costs of companies as well. However, web services as well as cloud platforms may be the target of attacks that alter their security, and the security of web services is not completely addressed. The solutions proposed in the literature are sometimes specific to certain types of attacks and they cannot ensure the attack tolerance of web services. Attack tolerance can be defined as the capability of a system to function properly with minimal degradation of performance, even if the presence of an attack is detected. As such, we claim that, to achieve attack tolerance, one should detect attacks by a continuous monitoring and mitigate the effects of these attacks by reliable reaction mechanisms. For this aim, an attack tolerance framework is proposed in this paper. This framework includes the risks analysis of attacks and is based on diversification and software reflection techniques. We applied this framework to cloud applications that are based on web services. After describing the core foundation of this approach, we express such cloud applications as choreographies of web services according to their distributed nature. The framework has been validated through an electronic voting system. The results of these experiments show the capability of the framework to ensure the required attack tolerance of cloud applications.

Keywords: resilience; attack tolerance; monitoring; passive tests; runtime verification; web services and cloud; software reflection

1. Introduction

Computer systems are now at the heart of all business functions (accounting, customer relations, production, etc.) and more generally in everyday life. These systems consist of heterogeneous applications and data. They are sometimes described through modular architectures that integrate and compose them in order to meet the needs of the organization. Service Oriented Architectures (SOA) are suitable for this purpose.

These architectures are distributed and facilitate the communication between heterogeneous systems. The main components of such architectures are web services. A web service is a collection of open protocols and standards for exchanging data between systems. Thus, software applications written in different programming languages and running on different platforms can therefore use web

services to exchange data. These services can be internal and only concern one organization. However, with technological advances in communication networks especially the Internet and the expansion of online services via cloud computing or simply the interconnection of IT systems, the need to expose services to the outside world is growing. Cloud computing for example enables sharing of IT resources (computing, storage, networks, etc.) on demand over the Internet. These services are often deployed on the basis of smaller components (containers, virtual machines, etc.) deployed on a single site or on several geographically distributed sites. They can also be provided by several different cloud service providers (multi-cloud applications).

However, web services, as with many other technologies taking advantage of the Internet, are also facing attacks on availability, integrity, and confidentiality of platforms and user data. Moreover, web services deployed in the cloud inherit their vulnerabilities. Indeed, they are vulnerable to different risks that have to be evaluated [1]. Recently, new attacks exploiting cloud vulnerabilities such as side-channel, VM escape, hacked interfaces and APIs, and account hijacking [2–4] are considerably reducing the effectiveness of traditional detection and prevention systems (e.g., firewall, intrusion detection systems, etc.) available in the market. Web services deployed in the cloud also inherit their vulnerabilities.

As mentioned above, cyber attacks are multiplying and becoming more and more sophisticated. We show that, to better tolerate and limit the impact of these attacks, the monitoring of the information systems is of paramount importance for any organization. Traditional intrusion detection systems are deployed to identify and inhibit attacks as much as possible. Usually, the detection of anomalies is based on the comparison of observed behaviors with previously established normal behaviors. An alert is raised when these two behaviors differ. In the case of dysfunction of the information systems, they are able to act accordingly. Moreover, monitoring makes it possible to analyze in real time the state of the computer system and the state of the computer network for preventive purposes.

However, we believe that the monitoring and detection of attacks require an awareness of the risks that the system might be exposed to. As such, it is mandatory to include risk management in the monitoring strategy in order to reduce the probability of failure or uncertainty. Risk management attempts to reduce or eliminate potential vulnerabilities, or at least reduce the impact of potential threats by implementing controls and/or countermeasures. In the case in which it is not possible to eliminate the risk, mitigation mechanisms should be applied to reduce their effects.

In this paper, we adopt a new end-to-end security approach based on risk analysis, formal monitoring, software diversity and software reflection. This approach integrates security of cloud applications based on web services at all levels: design, specification, development, deployment, and execution.

At the design phase, we model cloud applications as choreographies of web services in order to benefit from the distributed nature of cloud applications. Besides when deploying choreographies, one should ensure that these choreography are realizable and the participants of these choreographies act according to the requirements. Choreographies are written as process algebras and formally verified and projected on the peers. Each participant of that choreography is deployed in one container or virtual machine and diversified. The basic idea is to have variants of the components of the system and these variants react and replace themselves when one of these components is compromised due to the effects of an attack. Skeletons of the corresponding services are generated by *ChorGen*, a new Domain Specific Language (DSL) we propose.

To anticipate attacks and better monitor the system, we leverage the traditional risk management loop to build a risk-based monitoring that integrates risks into monitoring. This risk analysis helps identify the attacks most likely to be executed against the system. Once these attacks are identified, we rely on software reflection to monitor the system and detect the attacks. Reflection is a software engineering technique that helps a program to monitor, analyze, and adjust its behavior dynamically.

We propose an attack tolerance framework (offline and online) for cloud applications based on the web services in [5]. This paper is an extension of that work covering different aspects, which are not addressed in this first version. In particular, we enrich the related work and the risk analysis sections,

by giving more details on the work carried out in these areas and by presenting in more detail our results. In summary, the contributions of the paper are the following:

- We detail the methodological aspects (models, assumptions, and implementation) and all the steps that led to the construction of this approach (Sections 3.5 and 4). In short, a more complete framework is proposed in this paper.
- We illustrate the approach through a concrete case study: an electronic vote system (Section 5). Experiments on this use case highlight the attack tolerance capability of the whole framework. The use case is also detailed. In particular, we present how conformance of the roles with respect to the choreography is achieved.
- We discuss improvements and research avenues in the area of attack tolerance (Section 6).

The paper is organized as follows. We review the main attack tolerance techniques and the principal issues related to web services in Section 2. Section 3 fully describes the risk-based monitoring methodology. In this section, we also detail remediation strategies to apply corrective actions for mitigating the impact. Following the above methodology, an attack tolerance framework for cloud applications is presented in Section 4. In Section 5, we present a concrete case study: an electronic vote system. Experiments on this use case highlight the attack tolerance capability of the whole framework. Conclusions and future enhancements of this work are given in Section 6.

It should be mentioned that this paper is focused on the design of a framework for attack tolerance of cloud applications, and the proposed approach could be adapted to network security [6], but this subject is out of the scope of the paper.

2. Attack Tolerance for Web Services

This section first presents attack tolerance and existing techniques highlighting the main issues that remain unsolved. We explore software formal methods as well, in order to disclose their benefits for attack tolerance. Finally we provide an overview of web services security issues and the proposed attack tolerance approaches.

2.1. Attack Tolerance Techniques

The attack tolerance concept comes from fault tolerance, a term used in dependability [7]. Dependability is a generic notion that measures the trustworthiness of a system, so that the users have a justified trust in the service delivered by that system [8]. It mainly includes four components: reliability, maintainability, availability, and security. Dependability has emerged as a necessity in particular with industrial developments. The goal is to build systems that are reliable and contain near-zero defaults. As IT systems are facing both diversified and sophisticated intrusions, intrusion tolerance can be considered as one of the crucial attributes of dependability to be taken into account.

Attack tolerance of a system is then the ability of that system to continue to function properly with minimal degradation of performance, despite intrusions or malicious attacks. Several approaches and techniques are proposed in the literature. The goal of the work in [9] is to identify common techniques for building highly available intrusion tolerant systems. The authors mentiond that a major assumption of intrusion tolerance is that IT systems can be faulty and compromised and the main challenge consists in continuing to provide (possibly degraded) services when attacks are present. In addition, the main techniques used for attack tolerance are presented in [10]. Voting and dynamic reconfiguration are some examples of these techniques.

Furthermore, there are several solutions that provide attack tolerance using one or a combination of such techniques [11–15]. Constable *et al.* [16] explored how to build distributed systems that are attack-tolerant by design. The idea is to implement systems with equivalent functionality that can respond to attacks in a more safe way. Roy *et al.* [17] proposed an attack tree, which they named attack-countermeasure tree. The aim is to model and analyze cyber attacks and countermeasures. They used this tree to allow automation of the attack scenarios.

Other approaches have been developed to cope with intrusion-tolerance. In [18], the authors proposed a hybrid authorization service. The main contribution of that study is the introduction of an Intrusion tolerance authorization scheme. In this scheme, the system is able to distribute proofs of authorization to the participants of the system.

Besides, La [10], Nguyen and Sood [19] classified ITS (Intrusion Tolerant Systems) architectures into four categories:

- **Detection-triggered** architectures build multiple levels of defense to increase system survivability. Most of them rely on an intrusion detection that triggers reactions mechanisms.
- **Algorithm-driven** systems employ algorithms such as the voting algorithm, threshold cryptography, and fragmentation redundancy scattering (FRS) to harden their resilience.
- **Recovery-based** systems assume that, when a system goes online, it is compromised. Periodic restoration to a former good state is necessary.
- **Hybrid** systems combine different architectures mentioned above.

The main conclusion of this section is that the complementary combination of these architectures can lead to the design of more efficient architectures. Our intrusion tolerance approach for web services in this paper also combines the attack tolerance mechanisms in a coherent manner by incorporating new detection methods.

2.2. Formal Methods

One of the open issues in software engineering is the correct development of computer systems. We want to be able to design safe systems. The secure design of software refers to techniques based on mathematics for the specification, development, and verification of software and hardware systems. The use of a secure design is especially important in reliable systems where, due to safety and security reasons, it is important to ensure that errors are not included during the development process. Secure designs are particularly effective when used early in the development process, at the requirements and specification levels, but can be used for a completely secure development of a system. One of the advantages of using a secure representation of systems is that it allows rigorously analyzing their properties. In particular, it helps to establish the *correctness* of the system with respect to the specification or the fulfilment of a specific set of requirements, to check the semantic *equivalence* of two systems, to analyze the *preference* of a system over another one with respect to a given criterion, to predict the possibility of *incorrect behaviors*, to establish the *performance* level of a system, etc. Formal methods are well suited to address the above mentioned issues as there are based on mathematical foundations that support reasoning.

There are two different categories of formal methods, static analysis, and dynamic analysis [20]. In static analysis, the code is not executed but some properties are proven. Dynamic analysis consists of executing the code or simulating it in order to reveal bugs. Software testing consists in comparing the result of a program with the expected result. A particular type of dynamic analysis is formal monitoring which remains more used for the detection of attacks. This is why we deeply present formal monitoring.

What Is Monitoring?

Monitoring is the process of dynamically collecting, interpreting, and presenting metrics and variables related to a system's behavior in order to perform management and control tasks [21]. The idea behind monitoring is to measure and observe performance, connectivity, security issues, application usage, data modifications, and any other variable that permits determining the current status of the entity being monitored. By keeping a constant view of the different entities, we can obtain a real-time status of Key Performance Indicators (KPI) or Service Level Agreements (SLA) compliance as well as faults and security breaches. In addition, security requirements can be specified using different formalisms as regular expressions, temporal logic formulas, etc. Monitoring can be performed

in several domains that include user activity, network and Internet traffic, software applications, services, and security. The monitoring processes should not disturb the normal operation of the protocol, application, or service under analysis.

The general processes involved in monitoring are: the definition of the detection method to track and label events and measurements of interest; the transmission of the collected information to a processing entity; the filtering and classification; and, finally, the generation decisions associated to the results obtained after the evaluation [21]. Regarding how to collect events and measurements, monitoring techniques can be classified into three main categories: active, passive, and hybrid approaches.

Active monitoring: The System Under Observation (SUO) is stimulated in order to obtain responses to determine its behavior under certain circumstances or events. This technique permits directing requests to the concerned entities under observation. However, it presents some drawbacks. The injection of requests towards the SUO might affect its performance. This will vary depending on the amount of data required to perform the desired tests or monitoring requests. For large amounts of data, the SUO processing load might increase and produce undesirable effects. Secondly, the injected information might also influence the measurements that are being taken, for example incurring in additional delay. Lastly, active monitoring injects data that could be considered invasive. In a network operator context, it could limit its use and applicability [22].

Passive monitoring: It consists in capturing a copy of the information produced by the SUO without a direct interaction [23]. Runtime verification can be also considered as a form of passive monitoring [24]. This technique reduces the overhead required on active monitoring. Conversely, certain delay should be considered when analyzing large amounts of data. Additionally, in some cases, it is not always possible to perform real-time monitoring because of required offline data post-processing [23]. This technique has the advantage over the active approach of not performing invasive requests [25–27].

We have seen that the formal methods make it possible to check that the system is working properly according to the expected specifications. We also note the benefits of monitoring information systems. Probes provide valuable information about the state of the system. In conclusion, monitoring can contributes to attack tolerance.

2.3. Security Issues Related to Web Services

Web services are the target of Cyber attacks. Web services face several attacks. The main attacks such as XML DoS are these listed in [28–30]. Moreover, web services are increasingly used to develop Enterprise Service Oriented Architectures. These services are often deployed in the cloud. Indeed, Sharma *et al.* [31] showed interest in deploying web services in the cloud. They pointed out that deploying web services in the cloud increases the availability and reliability of these services and reduces the messaging overhead. In fact, the resources, provided per demand in the cloud with great elasticity, satisfy the requirements of the service consumers. In conclusion, web services deployed in the cloud or used for building cloud applications inherit the vulnerabilities of the cloud platforms (Table 1).

Moreover, few studies have been conducted to transpose the techniques and framework cited in the previous section to web services. Sadegh and Azgomi [32] and Ficco and Rak [29] presented attack tolerant web service architectures based on diversity techniques presented above. These solutions protect essentially against XML DoS attacks. While these approaches are interesting, they do not address the specificity of services-based application deployed on cloud platforms. The solutions are attack-specific. Moreover, for this kind of application, it is necessary to integrate security in all the process steps, i.e., from modeling to deployment. We need a more efficient intrusion-tolerant mechanism.

Table 1. Cloud-based attacks.

	Attacks	
Types	**Data loss Attacks**	• Data loss • Cache-based side channel attacks • Unauthorized access to data • Data exfiltration • Metadata modification • Data sniffing/spoofing • Others
	Virtualization based attacks	• Detecting a virtualized environment • Identifying the hypervisor • VM hopping • Malicious VM creation • VM escape • Others

2.4. Discussion

To cope with all these issues, it is necessary to consider information security as a permanent issue that needs to be managed in order to obtain attack-tolerant web services. In this work, we design an attack tolerant system that integrates intrusion detection methods, formal methods, and diverse defense strategies. By means of constant monitoring, we provide an attack-tolerant framework, so that potential security breaches within can be dynamically detected and appropriate mitigation measures can be activated on-line, thus reducing the effects of the detected attacks. As a result, we ensure a total attack tolerance attack tolerance for applications based on web services deployed in the cloud. Moreover, even though currently many companies continue to build their business applications using a Service Oriented Architecture (SOA) approach, micro service architectures will become the standard in the years to come. We believe our approach in this paper may suit new development paradigms such as micro services.

3. Risk-Based Monitoring Methodology

We leverage the risk management loop to build our risk-based monitoring loop, as depicted in Figure 1. Indeed, this risk-based monitoring solution can be summarized by the following objectives:

1. Identification of system assets
2. Risk analysis to categorize threats that can exploit system vulnerabilities and result in different levels of risks
3. Threat modeling,
4. System monitoring to detect potential occurrences of attacks
5. Remediation strategies to apply corrective actions for mitigating the impact of the attack on the target system

Steps 1–5 are described in detail below.

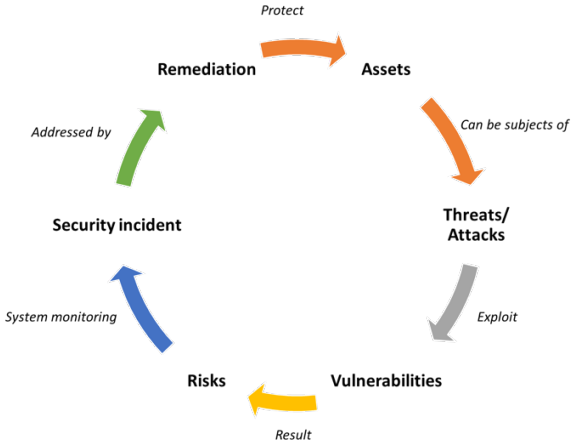


Figure 1. Risk-based monitoring loop.

3.1. Identifying Assets

Assets are defined as proprietary resources of value and necessary for its proper functioning. We distinguish business-level assets from system assets. In terms of business assets, we mainly find information (e.g., credit card numbers) and processes (e.g., transaction management or account administration). The business assets of the organization are often entirely managed through the information system. System assets include technical elements, such as hardware, business-critical applications, and their corresponding databases and networks, as well as the computer system environment, such as users or buildings. System assets can also represent some attributes or properties of the system such as the data integrity and availability. This is particularly true for cloud services consumers. As such, no company can afford to lose these assets.

3.2. Risk and Vulnerability Analysis

Risk is the possibility or likelihood that a threat will exploit a vulnerability resulting in a loss, unauthorized access, or deterioration of an asset. A threat is a potential occurrence that can be caused by anything or anyone and can result in an undesirable outcome. Natural occurrences, such as floods or earthquakes, accidental acts by an employee, or intentional attacks can all be threats to an organization. A vulnerability is any type of weakness that can be exploited. The weakness can be due to, for example, a flaw, limitation, or the absence of a security control.

Thus, after identifying valuable assets, it is necessary to perform vulnerability analysis. This type of analysis attempts to discover weaknesses in the systems with respect to potential threats. For example, in the context of access control, vulnerability analysis attempts to identify the strengths and weaknesses of the different access control mechanisms and the potential of a threat to exploit these weaknesses.

Common Attack Pattern Enumeration and Classification (CAPEC) [33] provides a database of known patterns of attacks that have been employed to exploit known weaknesses in cyber systems. It represents attack patterns in three different ways: hierarchical representation via attack mechanisms or attack domains, representation according to the relations to the external factors, and representation according to the relations to the specific attributes. It can help to advance community understanding and enhance defenses. In the scope of our framework, for example, the known attack patterns in CAPEC may help to identify easily and quickly the weaknesses as well as their possible exploitation when analyzing the risk and vulnerability of target systems. Penetration Testing Execution Standard (PTES) [34] defines a methodology based on penetration testing to check the robustness of a given system or application. There are many certifications designed for penetration testing: EC-Certified Ethical Hacker, GIAC Web Application Penetration Test (GWAPT), Certified Penetration Tester, etc. In the same way, the OWASP [35] Benchmark Project attempts to establish a

universal security benchmark by providing a suite of thousands of small Java programs containing security threats. In addition, our approach could be part of a global framework such as the NIST Framework for Improving Critical Infrastructure Cybersecurity (NIST Cybersecurity Framework or CSF) (https://www.nist.gov/cyberframework). Indeed, this framework leverages a risk-based approach and the core part of this framework is divided into five functions: identify, protect, detect, respond, and recover. These are similar to the five steps of our approach. In our research work, we do not consider these methodologies but we recognize that they are complementary to our approach, in particular to test tolerance to attacks.

3.3. Threats Modeling

The first step to avoid or repel the different threats that can affect an asset is to model them by identifying: affected modules/components, actions/behavior to trigger the threat, and potential objective of the threat. The formal model of a threat helps to understand the operation of the attacks and allows the creation of security mechanisms to protect, not only the assets, but also the software mechanisms that support them. Once the threats are modeled, we can identify the vulnerabilities that can affect the system and define monitoring and remediation mechanisms to minimize the damages that might occur. Again, consider the access control example. An access control process has two main steps: authentication and authorization. The latter usually comes after the former in a normal workflow. The authentication step is the more critical part of the access control process. The following description illustrates this assertion: Indeed, the attacker that may be inside the organization already knows or can easily find weak points in the organization's defenses (inadequate security controls, failure of the principle of least privilege, software vulnerabilities, etc.). He can then attempt a privilege escalation to gain more permissions, to overcome an operating system's permission, and to impersonate the root user so that he can create the fake user with root privilege and grant himself all the necessary privileges for further attacks or directly steal sensitive information with the administrator's capabilities.

3.4. System Security Monitoring

The monitoring mechanism we propose allows constantly monitoring activities or events occurring in the network, in the applications, and in the systems. This information will be analyzed in near real-time to early detect any potential issue that may compromise the security or data privacy. If any anomalous situation is detected, the monitoring module will trigger a series of remediation mechanisms (countermeasures) oriented to notify, repel, or mitigate attacks and their effects.

3.5. Remediation

Once the risks of any system are established and the means of detection identified, it is essential to think about how to set up mechanisms that will allow to complete the risk-based monitoring loop i.e., to tolerate and mitigate the effects of the potential detected attacks. An efficient remediation technique should thwart as many attacks as possible. We explain below the proposed new approaches. They are based on diversity and meta-programming methods called software reflection.

3.5.1. Diversity-Based Attack Tolerance

Recall that diversity is the quality or state of having many different forms, types, ideas, etc. As our work targets attacks tolerance, we concentrate on the use of diversity as a mean for achieving it. At runtime, in the case an attack has been detected, the implementation of the running software is dynamically replaced by an implementation which is more robust. This idea is implemented through two complementary approaches. First, we present model-oriented diversity. This contribution is based on formal models. Then, we present the second approach, implementation-oriented diversity that reduces the shortcomings of the first approach and extends it. This second approach leverages Software Product lines (SPL) for devising a fine-grained attack tolerance system.

Model-based diversity for attack tolerance. It consists in investigating attack-tolerance at the design and specification phase. This model-based approach tries to obtain a balance between security and a good quality of experience. One can argue that the more secure model has to be implemented first. In that case, the user experience is lowered. For many applications, it is clear that the choice is not the more constrained model [36]. The centerpiece of that approach is the usage of monitoring methods and formal models:

- A running system is monitored to observe its run-time behavior with our Monitoring Tool (MMT). A formal model of the module that is susceptible to be suffering an attack is designed. This model is expressed as a Finite State Machine (FSM). The monitored values are abstracted and related to security properties we defined. These properties are written in linear temporal logic (LTL).
- From this first model, other modified models are obtained. They have the same functionality but can have more mechanisms to impeach attacks; these models are more secure and robust.
- Associated to each model, implementations are produced.
- Violations of the properties described are thus detected by the monitoring tool. This detection triggers the adaptation process. The model is replaced with another model that is more robust and the implementation as well.

This approach is illustrated in Figure 2. The principal difficulty of this method lies in the derivation of models. How should the models and implementations from the first model and implementation be derived? **Implementation-based diversity for attack tolerance.**

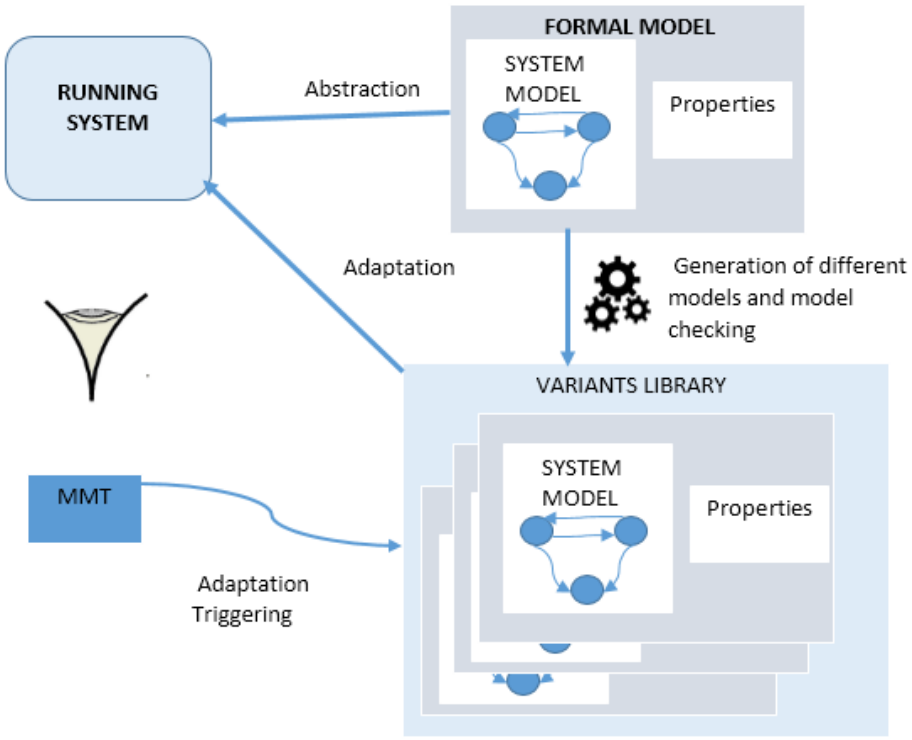


Figure 2. Model-based diversity overview.

This approach aims at extending and solving the issues raised by the former approach by leveraging diversity [37]. The idea is still the same as in the previous approach but here there is only one model and several implementations. To this end, we base our work on the concept of diversification. The more diversification there is, the more security is ensured. The model chosen is Feature Model (FM). A FM is used in the area of software product lines to model a particular product

line. It specifies the common as well as the variable features of a family. A variable feature is called variability. Three patterns of variability were used in the FM: encoding style (document and RPC), encoding (literal and encoded) types, and languages (C and C ++). After implementing the WSDL of the diversified service, the skeletons of these services are generated. Then, the code of the latter are obfuscated by adding instructions that modify their normal control flow, just to have a source level diversification. Finally, a new layer of diversification at the binary level is added. This ensures that the implementations are not vulnerable to the same attacks leveraging the computation flow (code reuse attack). The services are then highly diversified and redundant. There are in total three levels of diversification.

To ensure the continuous availability of our system, it is configured in two ways: normal mode and attack mode. In normal mode, time is divided into *epochs*. In each *epoch*, only a unique variant is chosen. When the *epoch* of time elapses, another implementation is deployed to ensure continuity of the service. In abnormal mode, it is the case where the defense mechanism has successfully detected an attack. The system reacts by switching to another more resistant implementation before even the *epoch* has elapsed. The main design of the solution is depicted in Figure 3.

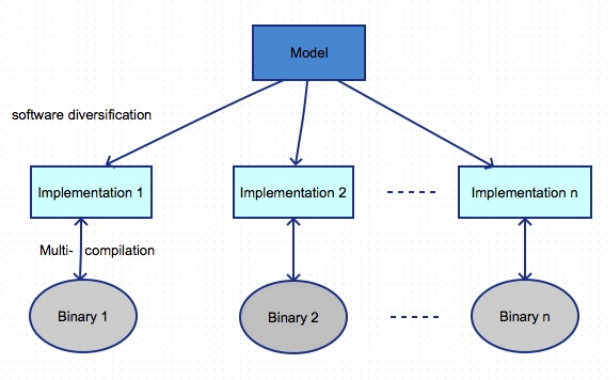


Figure 3. Layered implementation-based diversity overview.

Let us take another example of an organization. The asset in this case is the intellectual property of the company or the organization. This company has deployed an information system through which employees can communicate and exchange information. This information system may leverage web services. This information system is also connected to the Internet to allow the company to communicate with the outside. We assume that there is an unsuspecting employee in this company. This employee receives a malicious email containing malicious content and clicks on it. The threat to the company and its information system is the malicious email (sent by the hacker). The naive employee is therefore a vulnerability for the system. The hacker could therefore take advantage of this vulnerability. This vulnerability, once successful, may pose a risk for the company and its asset.

If the company anticipated this risk by ensuring remediation by construction that consists in diversifying the different parts of its information system and set up monitoring points at the level of the network and the applications, the risk is then mitigated, the asset is protected, and the continuity of the service is ensured.

3.5.2. Reflection-Based Attack Tolerance

The aim of this technique is to approach attack tolerance in a manner different from the methods mentioned above [38]. In fact, the latter has the ability to detect attacks coming from the outside (DDoS, Brute-force, etc.). In addition, their tolerance features are designed before the deployment of the application(e.g., diversification of web services). That is why we consider finding a solution that would tolerate internal attacks. Meta-programming techniques in particular software reflection is investigated. Reflection is the possibility for a program of monitoring and/or modifying its behavior dynamically.

The basic idea is therefore the following. It is assumed that the software of the client is located in a safe environment. Some potential attacks that can take place are internal ones, i.e. coming from internal hackers. The goal of the intruder is to usurp the actions, i.e., to modify the methods of the API of the platform. By reflection, all the hash of the source code of any methods of the API are processed (Figure 4). Any deviation at runtime of that hash value means the presence of a misbehavior. Such misbehavior could be an insider attack or a virus attack. Information such as date, hour, operation, hash, and host are stored in the log file. Any request has then two traces in the logs: outbound (request) and inbound (response). Let us describe the situation when an attack occurs, i.e., someone has modified the API and overridden one or several methods. First, this is the case where we see some information of the methods but there is no Hash. It is also possible to get only one hash for the outbound operation and nothing for the inbound operation. If there is an attack, the hashes of both Outbound and Inbound could not correspond in the log files. Finally, we can get some inconsistencies in the logs: timestamps incoherence, method inconsistencies (answer before request), or combination of inconsistencies.

```
def show_stack():
    stack = inspect.stack()
    ''' Inspect the stack '''

    for s in stack:
        a=inspect.getsource(s[0])
        ''' Get the source '''

        m=hashlib.md5()
        ''' hash that source code'''
```

Figure 4. An example of using reflection in python language.

For the monitoring part of the framework, the programs are checked at runtime using reflection, as mentioned above. For detecting attacks, logs located on the two endpoints, the premises and server, are leveraged. We developed a new plugin for this kind of detection in the monitoring tool MMT. Some security policies (rules) are then applied. If the threat detected is a modification of one of the methods of the API, the system reacts using reflection by replacing this method with the original method in the API. If the attack is not known in the vulnerability DB, the system checks its own DB (M-DB). If the attack exists, countermeasures are launched, else the hash is stored in the M-DB.

4. Attack Tolerance for Cloud Applications Based on Web Services

This section presents how we instantiate the risk-based methodology for cloud applications based on web services. As the first stages of the risk-based monitoring loop are specific to the type of the application, we focus on the last two phases of that loop: monitoring and remediation. Two main approaches of remediation can be described:

- Anticipate the ability to tolerate attacks. This consists in introducing mechanisms allowing the tolerance to the attacks during the modeling of the system. The system is called tolerant-by-design or offline tolerant to attacks.
- Considering tolerance by a constant monitoring. In this type of approach, the tolerance capacity is entirely managed by the monitoring tool. The system is actively monitored for detecting malicious behaviors. The system is called online tolerant to attacks.

We believe that, to have an effective attack tolerance (offline and online), it is appropriate to use these two approaches in a complementary way. We therefore propose both online and offline attack tolerance. The resulting framework consists of two main parts (Figure 5). The first part presents how we model web service applications deployed in the cloud to make them attack tolerant. The second part presents how we monitor the system to detect the attacks and how we mitigate these attacks.

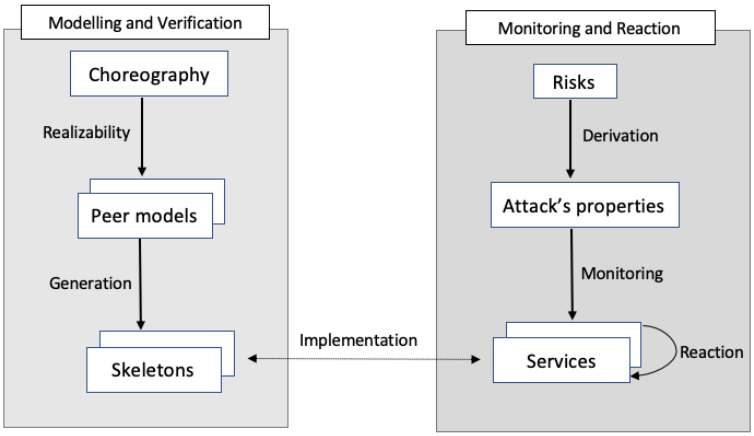


Figure 5. Architecture and components of the framework.

4.1. Modeling of Cloud Applications

Buyya et al. [39] gave a definition of the cloud: "A Cloud is a type of parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service level agreements established through negotiation between the service provider and consumer".

Different cloud models and architectures of different existing providers are presented in [40]. In [41], the authors aimed at defining the steps involved in the requirements engineering phase for cloud application adoption by providing systematic guidance for an organization to evaluate its choice and risks when moving and adopting a cloud. A cloud application can be deployed as a composed collection of services. According to Cloud Technology Partners (CTP) [42], the benefits of this composition deployment are manifold. First, since the services are loosely coupled, it is easier to track and maintain the application. Additional benefits may include re-usability. Applications can been broken up into hundreds of underlying services that have value when used by other applications. To fully benefit from these advantage, we consider our applications as a composition of web services deployed in the cloud.

In addition, in [43], cloud applications are considered as distributed applications composed of several virtual machines running a set of interconnected execution units called software elements. It should be noted that there are two main models of deployment of distributed applications in the cloud [43]:

- **Infrastructure-oriented solutions** envision the deployment of an application in the cloud through the implementation of a set of virtualized hardware resources. They come in the form of a public or private cloud. This type of cloud application can benefit from several services provided in the cloud such as database storage, virtual machine cloning, or memory ballooning.
- **Application-oriented solutions** aim at combining the service-oriented approach, in which a distributed application is defined as a composition of high-level services, and the infrastructure-oriented approach, which explains how an application breaks down within a set of virtual resources. Application-oriented solutions thus offer a high degree of parameterization for the user to define the application to be implemented.

Generally service compositions are classified into two styles: orchestrations and choreographies. Orchestration always represents control from one participant's perspective, called the orchestrator. Unlike the orchestration, there is no privileged entities in the choreography. Furtado *et al.* [44] argued that web services composition, in particular choreography, is a suitable solution used to build application and systems in the cloud. They built a middleware solution that is capable of automatically deploying and executing web services in the cloud.

We agree with them that choreography is a good approach for deploying cloud applications based on web services. The applications in the cloud are deployed as service choreographies that integrate attack tolerance features. However, before and when deploying such choreography, one should ensure that this choreography is realizable. Realizability, a fundamental issue of choreography, is whether a choreography specification can correctly be implemented. In a top-down service choreography approach, the realizability issue results in verifying whether a choreography model can be correctly projected onto role models that will be then implemented by services. For this goal, we leverage SChorA [45], a verification and testing framework for choreographies.

4.1.1. SChorA Framework

SChorA (http://SChorA.lri.fr) was proposed by Nguyen [45]. This framework aims to solve the key issues in choreography-based top-down development: (i) realizability, whether a choreography is realizable, i.e., ensuring that a choreography can be practically implemented; and (ii) projection, the ability to derive local models of a global choreography on peers. To easily express the choreographies, a formal language, *ChorD*, which is an extension of the *Chor* language [46] with data, has been proposed. *Chor* language is expressive and abstract enough to enable one to specify collaborations but lacks data support, which *ChorD* covers.

The basic event in choreography is an interaction. An interaction represents a communication between two roles. There are two kinds of interactions: free interactions and bound interactions. A free interaction represents a communication of value of variable x realized through an operation o from role a to b is denoted by $o^{[a,b]}.x$, while the bound one is denoted $o^{[a,b]}.\langle x \rangle$. In free interaction, the data exchange must be known before the interaction may occur. In bound interaction, the data exchange is bounded at the moment the interaction occurs

ChorD is described as:

$$ChorD ::= 1 | \alpha | A; A | A + A | A \| A | A[> A | [\phi] \triangleright A | [\phi] \star A$$

A basic activity is either an inaction (*1*) or a standard basic event (α), presented above. There are structuring operators that can be used to specify composite activities such as sequencing (;), non-deterministic choice (+), parallel activities ($\|$), and interruption ([>).

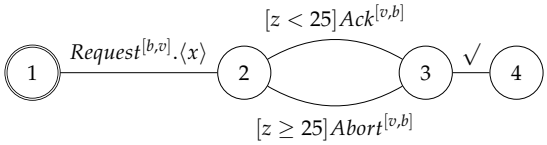
One should note that we distinguish the global specification of the choreography called *global model* and the specification of this choreography on the different roles termed *role model*. In *role models*, events are modeled as sending (!) or reception (?). For example, let us express a simple online shopping choreography between two roles: b (buyer) and v (vendor). The buyer first requests an article by providing an amount to be bought. If the amount is greater than 25, then the vendor aborts this transaction. Otherwise, a confirmation will be issued from the vendor to the buyer. This can be described as follows:

$$C : Request^{[b,v]}.\langle x \rangle; ([x < 25] \triangleright Ack^{[v,b]} + [x \geq 25] \triangleright Abort^{[v,b]})$$

$$\text{For the Buyer: } Request^{[b,v]}!.\langle y \rangle; (Ack^{[v,b]}? + Abort^{[v,b]}?)$$

$$\text{For the Vendor: } Request^{[b,v]}?.\langle z \rangle; ([z < 25] \triangleright Ack^{[v,b]}! + [z \geq 25] \triangleright Abort^{[v,b]}!)$$

In fact, *ChorD* is a process algebra and its semantics is given by Symbolic Transition Graphs (STGs) [47]. An STG is a transition system. Each transition of STG is labeled by a guard ϕ and a basic event α. The guard ϕ is a boolean equation which has to hold for the transition to take place. A symbolic transition from state s to state t with a guard ϕ and an event α is denoted as $s \xrightarrow{[\phi]\alpha} t$. $\surd$ is added to denote activity termination. The representation of the simple shopping choreography using STG is the following:



For the realizability and projection issues, STG are also used. By their formal richness, STGs are perfectly suited for verification of choreographies. STGs can be expanded to describe more operations since they support data, guard and free/bound variables. From the representation of the choreography as STG, if there are non-realizable parts, some additional interactions are incorporated to the graph to allow all the transitions to be realizable. Once the realizability is verified (i.e., it can be directly implemented or some additional interactions are added), they are projected on the different roles or peers. By doing so, we are sure that our local models projected can actually be implemented concretely. In our case, local models are implemented as a web service. Once this step is over, it is useful to implement these projected models on the peers. The process algebra, which represents the choreography model, follows the requirements of the specification. We follow with this approach a software development process. Interested readers are invited to refer to [45] for more details of the translation from process algebra descriptions to STGA automaton. The next section presents how in this framework skeletons of the services are made possible.

4.1.2. Code Generation

In top-down software development approaches, an important part of the process is to reduce the costs of development by promoting modularity, reusability, and code generation. This is especially true in modeling and designing choreographies. It is therefore essential to have automatic mechanisms that perform code generation. This is why we propose a code generation strategy in this framework.

For this aim, we can leverage frameworks or tools in the literature [48] that take as input STGs and produce source code. Moreover, we propose a new Domain Specific Language (DSL) for our choreography called *ChorGen*.

The *ChorGen* language has the following grammar:

```
Model:
  (choreographies+= Choreography)+;

Choreography:
  'choreography' name=ID '{'
    (roles+=Roles)*
  '}'
;
Roles:
  'role' name=ID '{'
    operations+=Operation
  '}'
;
Operation:
  'operations' '{'
     (methods+=Function)*
  '}'
;
Function:
  name=ID '('(params+=Param)* ')'
;
Param:
  name=ID type=ID ',' |name =ID type=ID
;
```

This means that a choreography contains several roles that expose some operations to interact with the other roles. An example of a well defined implementation is presented in Figure 6.

```
choreography chor{
    role client{
      operations {
        send(IP String)
        receive(Data string)
    }
    }
    role server{
      operations {
        verify(IP String)
        ack(IP string)
    }
    }
}
```

Figure 6. Example of definition of a choreography with 2 roles.

We use model-driven engineering technologies Xtext (https://www.eclipse.org/Xtext/) for the semantics and Xtend (https://www.eclipse.org/xtend/) for code generation to the target languages: WSDL and Python.

The advantage of doing such code generation is the reduction of the development costs and efforts. This allows us to be more efficient when implementing the services. This is useful, for example, for choreographies containing a very large number of peers. Another advantage is that, since all interactions are taken into account, we are sure that developers will not forget to implement them since their signatures are available.

Moreover, it should be noted that a single implementation is not sufficient to have a complete tolerance. It is admitted that more diversification implies more security.

4.2. Deployment, Monitoring and Reaction

4.2.1. Choreography Implementation and Deployment

We leverage diversity as well. For implementing choreography, we have the choice between two methods. The first is to diversify for example at the level of programming languages, i.e., having the implementations in different languages. This method has the advantage of generating few dependency between the variants of these services. However, this can create significant costs and workload for developers and can increase the time-to-market. The developer may not be able to master other languages. Moreover, since one of the members of the service choreography can be changed on the fly while the others remain intact, there could be some inconsistencies between the communication between these services. Indeed, although based on the remote procedure call (RPC), the ways to deploy web services are not the same.

The second way is to consider only one target programming language but have diversified implementation. The variants differ for example at the control flow (AST) level and use different data structures. The advantage of such an approach is that it is more flexible. The disadvantage is that we have a low diversity rate. However, this rate may be improved by using different OS during the deployment of the services. For such reasons, we choose the second method. Thus, from the global choreography, the local models are projected taking into account the interactions added to the specifications. This is the case for example when, in the verification phase, interactions are added to the models to make the choreography realizable. After, there is a generation of the skeletons of the services that implement the choreography. In particular, we generate the WSDL files (interface file of the web services) as well as the skeletons of the implementations of these services in Python.

Recall that it has been claimed that more diversity implies more security. In accordance with our attack tolerance methodology, one implementation at a time is chosen. The others are started but inactive.

When the current implementation is attacked, it is replaced by one of the variants. We assume that the communication channels between the choreography members are reliable. Thus, only the different roles can emit false messages when they are attacked or compromised. Each member of the choreography is deployed in a container on a public cloud platform. They are observation probes (Algorithm 1) available for each role in the choreography. These agents are in charge of monitoring and detecting attacks. They will also be given the ability to implement the replacement of the container when the attack is detected.

These agents do not normally interact directly with the members of the choreography. They are therefore not visible from the outside and are located in a safe environment. They are considered safe and they do not crash.

When a member is attacked, with respect to the rules described (in the formalism of the monitoring tool), the monitoring agent interrupts the connection of the attacked container while saving the last non executed requests. It replaces the current damaged container with another container (Lines 5 and 6 in Algorithm 1). Afterwards, he notifies the other members of the choreography and send them the information about the new active implementation (Line 7). He expects to receive acknowledgments from the other members during a certain period of time (Lines 9–13). For example, it sends the new IP address of the new deployed member to the other members of the choreography. Acknowledgments should be received with the correct hash of the IP address to ensure that members have obtained the correct address. After the end of the timer, if he does not receive the acknowledgments from all members, he sends again the message to the members (Lines 15–19). Otherwise, if there is no detection of attack, the manager keeps working. If he receives a message from a new member (Line 23), he answers back (Line 24).

In the same way, in Algorithm 2, we describe how the members of the choreography act. They are initially in the inactive state. Then, one of the members is chosen after a detection of attack or misbehavior. This member becomes active (Line 6, Algorithm 1), launches the list of remaining non executed instructions he receives from the manager (Line 4, Algorithm 2), and continues to perform his normal tasks (Line 7, Algorithm 2). We see the value of using these applications in cloud computing because you can deploy/disconnect containers quickly.

The next step is to ensure that, when these services are deployed, there are no other misbehaviors such as attacks or viruses. The following section then presents the monitoring strategy.

Besides, it is undeniable that to better tolerate attacks, it is necessary to detect them. Our approach of monitoring is also based on reflection, as presented in Section 3.

Software reflection makes it possible to dynamically retrieve the code and even the execution trace of a method, a class, or a module. One can also modify the class at run time. In Python language for example the *inspect* module provides functions for learning about modules, classes, instances, functions, and methods. Using reflection, all the hashes of the source code of any methods of the system are processed [38]. In fact, hash functions by their robustness are used to ensure the integrity of messages or transactions in distributed systems. This is the case in modern protocols, for example ssh and bitcoin. As such, the detection of attacks leveraging hash functions is legitimate.

Table 2 shows a situation where there is no attack. The hashes for the outbound and inbound requests are the same.

Algorithm 1 Observer or manager execution

```
1:  detected ← false
2:  while true do
3:      if detected then
4:          setLocalCountermeasures()
5:          chosenVariant ← random(1, n_variants)
6:          chosenVariant.state ← active
7:          notifyAllMembers(IP)
8:          nextInstruction ← buffer.pop()
9:          setTimer(Timer)
10:
11:         while Timer do
12:             buffer ← buffer ⋃ ack()
13:         end while
14:
15:         for i=0 to N do
16:             if !ack[i] then
17:                 notifyMember(IP,i)
18:             end if
19:         end for
20:         bind(chosenVariant)
21:         sendInstr(chosenVariant, nextInstruction)
22:     else
23:         if receiveChange[IP,i] then
24:             sendAck(mac(IP),i)
25:         end if
26:     end if
27: end while
```

Algorithm 2 Choreography member execution.

```
1:  while True do
2:      if receiveActivation then
3:          if receive(instruction) then
4:              execute(instruction)
5:          end if
6:
7:          continueExecution()
8:      end if
9:  end while
```

Table 2. Normal entries in the log of the client application.

Date	Hour	Method	Hash	Host
30 May 2020	11:00:00 AM	! updateFunction (Outbound)	365db5224a4...	a
30 May 2020	11:15:00 AM	? updateFunction (Inbound)	365db5224a4...	b

If an attack occurrs, the hashes of both outbound and inbound cannot correspond in the log files. This is the case depicted in Table 3. One can also get some other inconsistencies in the logs: hashes not equal, timestamps incoherence, method inconsistencies (answer before request), or combination of inconsistencies.

Table 3. Bad entries in the log of the client application.

Date	Hour	Method	Hash	Host
30 May 2020	11:00:00 AM	! updateFunction (Outbound)	365db5224a4...	a
30 May 2020	11:15:00 AM	? updateFunction (Inbound)	365db4567ab...	b

4.2.2. Montimage Monitoring Tool

For detecting attacks, we developed a new plugin for this kind of detection in the Montimage Monitoring Tool (https://montimage.com/products/MMT_DPI.html) (MMT) [49]. MMT is a solution

for monitoring networks and applications. It consists of three main complementary modules, as shown in Figure 7.

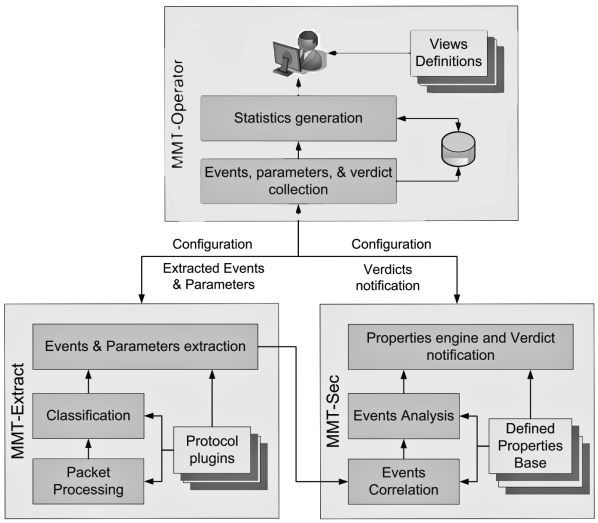


Figure 7. Component Architecture of Montimage Monitoring Tool.

- MMT-Extract is the core packet processing module. It is a C library. It captures and analyzes network traffic using Deep Packet/Flow Inspection (DPI/DFI) techniques in order to identify network and application based events by analyzing: protocols' fields values, network and application Quality of Service (QoS) parameters, and key performance indicators. In a similar way, it also allows to analyze any structured information generated by applications, such as, execution traces, logged messages. MMT-Extract incorporates a plugin architecture for the addition of new network protocols or application messages. It disposes of a public API for easily integrating into third party probes.
- MMT-Security is a security analysis engine written in C to analyze MMT-Security properties. The properties are detailed in the next subsection. MMT-Security analyzes and correlates network and application events to detect operational and security incidents. For each occurrence of a security property, it allows detecting whether it was respected or violated. If the property contains predefined embedded functions, MMT-Security triggers these functions, thus it allows users to perform their counter measures corresponding to the detection results.
- MMT-Operator is a visualization web application. It allows collecting and aggregating network statistics and security incidents to present them via a graphical user interface. It is conceived to be customizable, i.e., the user can define new views or customize one from a large list of predefined views. With its generic connector, it can be easily integrated with third party traffic probes.

4.2.3. Attack Detection

MMT's security properties are written in XML format. This has the advantage of simple and straightforward structure verification and processing by the tool. Any security property is written in XML. Each property begins with a <property> tag and ends with </property>. A MMT-Security property is an IF-THEN expression that describes constraints on network events captured in a trace T = $\{p_1, ..., p_m\}$. It has the following syntax:

$$e_1 \xrightarrow{W,n,t} e_2$$

W $\in$ { BEFORE, AFTER }, n $\in \mathbb{N}$, t $\in \mathbb{R}_{>0}$ and e_1 and e_2 two events. This property expresses that, if the event e_1 is satisfied (by one or several packets p_i , i $\in \{1, ..., m\}$, then event e_2 must be satisfied (by another set of packets p_j , j $\in \{1, ..., m\}$) before or after (depending on the W value) at most n packets

and t units of time. e_1 is called triggering context and e_2 is called clause verdict. When monitoring a system to detect attacks, the non respect of the MMT-Security property indicates the detection of an abnormal behavior that might imply the occurrence of an attack. For example, if we consider a vote system (our use case deeply presented in Section 5), a rule in the MMT formalism is the following.

Figure 8 describes a property for detecting the insider attack according to the formalism of MMT (Section 4). This means that in the log file any vote request should have hashes for its operations (outbound and inbound) in the log files and these hashes must correspond; otherwise, an attack is triggered. Event 1 (e_1) expresses the reception of the inbound operation in the log file with a hash. If event 2 (e_2), the reception of the corresponding inbound operation in the log appears, there is a comparison between the hash collected of that event and the hash obtained in the previous event e_1 (the built-in C function *strcmp* is used for the comparison). If the hashes correspond, the system is attack free. Otherwise, an alert is triggered.

```
<beginning>
  <property value="THEN" delay_units="s" property_id="10" type_property="ATTACK"
    description="Detection of the insider attack: ">
    <event value="COMPUTE" event_id="1" boolean_expression="((#strcmp(log.method, 'vote(inbound)') != 0)
      &amp;&amp; (#strcmp(log.hash,'')!= 0))"/>

    <event value="COMPUTE" event_id="2" boolean_expression="((#strcmp(log.hash, '')!= 0) &amp;&amp;
      (#strcmp(log.method, 'vote(outbound)') == 0) &amp;&amp; (#strcmp(log.hash, log.hash.1) != 0))"/>
  </property>
</beginning>
```

Figure 8. Security rule of the insider attack of the vote example.

Then, each module is monitored by extracting the program stack using reflection. If the attack exists, countermeasures are launched, else the hash is stored in the M-DB. For the mitigation of the attack, in the case of attacks the current source code is replaced with one of the other variants randomly. The hash is also adapted. The whole monitoring and detection is depicted in Figure 9. To detect virus attacks, we use the API VirusTotal site (https://www.virustotal.com/fr/). VirusTotal is a free service that analyzes suspicious and detects viruses or malware.

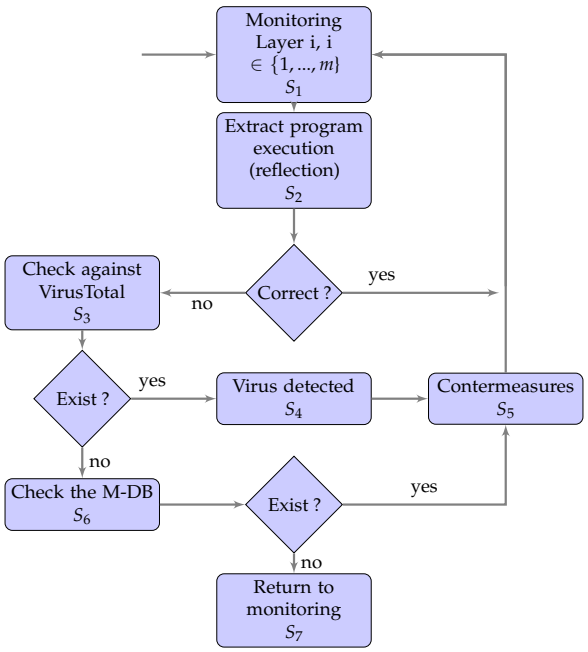


Figure 9. Detection and deaction model.

Here, when attacks are detected, we react differently. In Table 4, we define security levels of the attacks according to the criticality levels as well as the reaction we provide. When an attack occurs in one of these methods of a given API, the countermeasure corresponding to the criticality level of this API is triggered. The new corresponding reaction is returned to mitigate the effect of the attack. According to the security levels, if the threat detected was a modification of one of the methods of the API, the system reacts by switching to another implementation of the system as described in the deployment section. If the attack is a virus attack, in addition, the compromised implementation is then quarantined and destroyed.

Table 4. The security levels.

Criticality Level	Security Level	Description	Reaction
0	Normal	Correct functioning of the system. Here, events in the logs are noncritical.	Nothing to do.
1	Warning	These events indicate that a component is not in an ideal state and that other actions may cause a critical error.	We stimulate the layer or component in order to check its response to some predefined inputs. According to the responses obtained, we can locate and correct the misbehavior.
2	Attack	The events indicate that a component or a layer of the system has been affected and that layer failed or stopped responding due to an attack. The layer of the attack is then identified	The layer or component is automatically disconnected from the network and the other components/layers. It will be quarantined and we will remove the malicious code. Thus, the removal is a very effective mechanism for avoiding the production of more viruses. This is to prevent the virus from spreading to other unaffected layers. We replace the affected component/layer with new ones.
3	Critical Attack	The Events demand the immediate attention of the system administrator. They are generally directed at the global level. The events indicate that one or several components or layers of the system have been affected at the same time due to an attack. This also the case where any critical part (databases, storage) of the system has been accessed.	We react the same as in the case above. In the case a critical part such as the storage has been destroyed, there is a recovery step in which the data are restored with backup data.

However, it should be noted that classic hash functions can provide the same hash for two different strings of characters. In practice, the probability of this happening is small. As a result, a more robust hash is not foolproof either, but the probability of a collision is higher and/or the means of generating it are more complex and inaccessible.

In summary, the steps for implementing the approach are the following:

1. Specify the choreography in the ChorD language and verify the realizability, conformance, and projection of that choreography using SChorA conformance tool.
2. Generate the WSDL files and the skeletons of the peers and implementing them.
3. Specify some properties in order to check the conformance of the implementation *with respect to* the choreography and expressing them in the formalism of SChorA passive test engine. Specify the remaining properties using the formalism of MMT.
4. In line with our risk-based monitoring approach, check the assets of the choreography in order to anticipate potential vulnerabilities and attacks.
5. Specify some MMT properties for monitoring the peers and detecting attacks.
6. Deploy the peers on different cloud providers to enhance diversity.

Even though in this paper, we focus on web services, we believe that our approach may suit micro services. Micro services are distributed by nature and a choreography architecture is close to a micro service architecture, because choreographies can be decomposed into micro services that implement specific business functions and can be deployed into containers. Moreover, in [50], the authors claimed that a choreography of micro services is much faster than an orchestration of the same services. Finally, the approach proposed in this paper, which makes it possible to ensure that the choreographies are realizable and each member of the choreography is continuously monitored, will bring an additional trust on the micro services since security is a major concern of the micro service architecture.

5. Use Case: Vote Application

In this section, we present the implementation of our attack tolerance approach on a concrete case study. To illustrate its application, we propose an electronic voting choreography for the election of the president in a certain country. This application allows citizens to register on the electoral lists and to vote electronically. The application is formally described by the *VoteElecService* choreography. It is composed of three basic members: *Inscription*, *Vote*, and *Citizen*. In the following, we describe the main components of the attack tolerance framework associated to this use-case.

5.1. Identifying Assets and Attack Scenarios

In line with the approach proposed in this paper, the assets of the vote example are the votes of the citizens and the availability of the vote platform. Citizens of the country must be able to vote for their preferred candidate the election day at any time.

The main attacks we considered in the vote example are insider attacker or a not cautious user of the vote system [51–54]. In [53], the authors argued that in practice the code of a vote system could be leaked by dishonest insiders, or through a compromised developer workstation. This is also the case in [54], where the authors considered that ballots could be compromised before their delivery to voters, either by a dishonest insider who can alter the software or by remote attackers who may compromise the computers used to build or distribute the vote system.

5.2. System Security Monitoring and Reaction

5.2.1. Modeling and Verification

Let us first describe the choreography. Once registered, the citizen can vote electronically after verification of his registration by the member *Vote*. Subsequently, this service will provide the list of associated candidates and their identification number (1, 2, ...) in addition to the number *zero* that is associated with the blank ballot. A registered citizen will vote by selecting one or more voting numbers (including the blank ballot) and submitting his/her choices. The choreography can be expressed in *ChorD* as follows, in which the citizen member, the inscription member, the vote member, and the result member correspond to, respectively, c, i and v.

$$inscription^{[c,i]}.\langle info\rangle; voteRequest^{[c,v]}.\langle y\rangle; resultVerifInfo^{[v,i]}.\langle x\rangle;$$
$$([x=0] \triangleright rejection^{[v,c]} + [x!=0] \triangleright (confirmation^{[v,c]}; liste^{[v,c]}; vote^{[c,v]})).$$

The results of the formal analysis of the choreography are depicted in Figure 11.

As one can observe, the choreography is fully realizable without the need of adding new interactions. It is therefore generated on the different roles. We also describe some implementations models in order to check the conformance of the locals models, *with respect to* to the choreography (Figure 12). From these descriptions, we generate skeletons of the roles (Figure 13) that the developer should complete later.

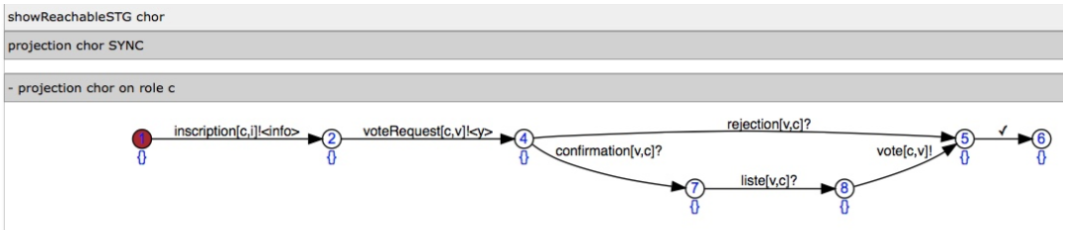


Figure 10. Projection of the choreography on the peers.

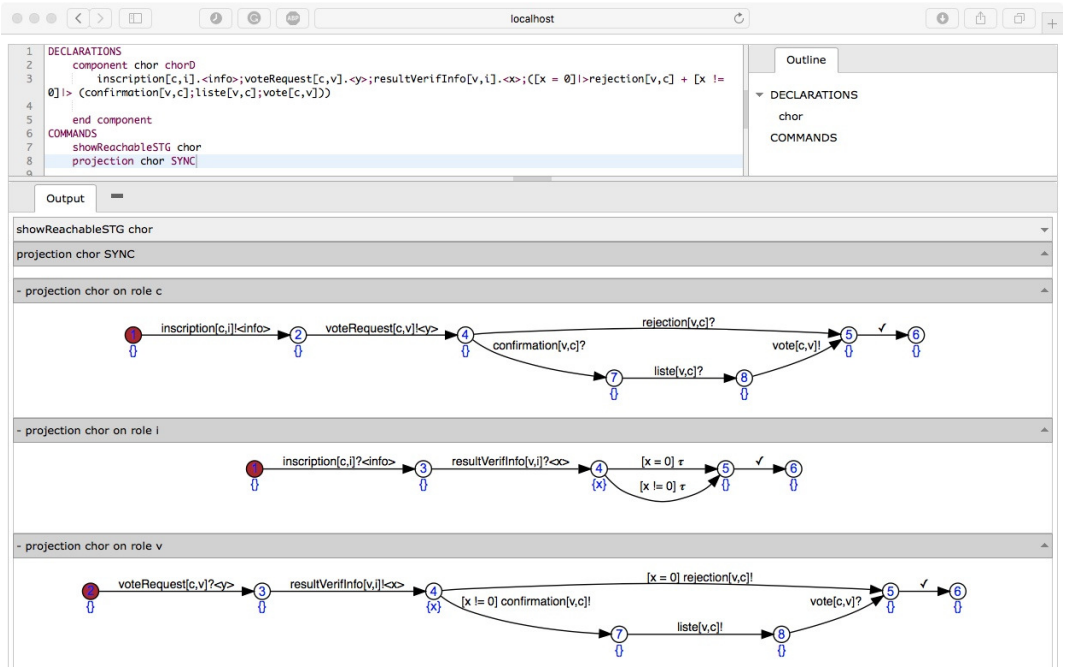


Figure 11. Verification of the vote choreography.

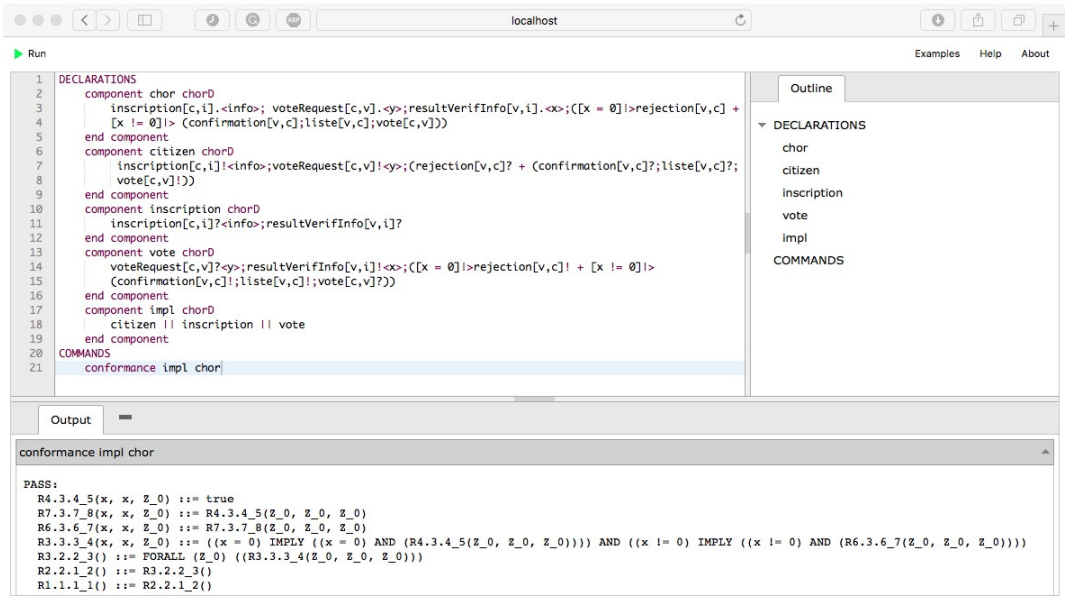


Figure 12. Conformance checking.

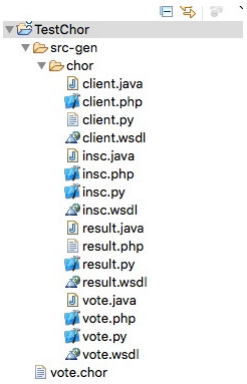


Figure 13. Generated skeletons.

```
<beginning>
        <property value="THEN" delay_units="s" delay_min="0+" delay_max="2"
                property_id="1" type_property="SECURITY" description="
                Any citizen should vote once ">
                <event value="COMPUTE" event_id="1" description="reception
                        of the vote for one citizen" boolean_expression="
                        ((#strcmp(log.method, 'vote') == 0))"/>
                <event value="COMPUTE" event_id="2" description="
                        reception of another vote for the same citizen in the log"
                        boolean_expression="(#strcmp(log.method, 'vote' == 0)
                        &amp;&amp; (#strcmp(log.host, log.host.1)!= 0))"/>
        </property>
</beginning>
```

Figure 14. Security rule of the unique vote: A person should vote once. Here, host refers to the concatenation of the citizen id number, the vote, and a string that means the result of the operation.

5.2.2. Deployment, Experiments, and Results

The services were deployed on the Amazon Web Services (AWS) cloud platform. We used a virtual machine for each member of the choreography.

For the monitoring, Figure 8 presents a property for detecting the insider attack. This means that in the log file any vote request should have hashes for its operations (outbound and inbound) in the log files and these hashes must correspond; otherwise, an attack is triggered.

The same choreography was deployed on a local test-bed consisting of three Dell machines having two micro processors, 3 GB RAM memory and all using the Ubuntu OS in its latest version. The same choreography was deployed on an Amazon cloud. The three different virtual machines also have 3 GB RAM memory and 2 vCPU. Some experiments were conducted to test the attack tolerance capability of our approach.

Experiment 1: Since the approach of the framework consists of modeling and deploying cloud-based applications as distributed service choreographies, we evaluated the latency of the service to respond to some amounts of requests on premises and on the cloud.

As shown in Table 5, the response times to requests are substantially equal. The slight difference can be explained by the latency of the network. One way to significantly reduce this latency is choosing to deploy virtual machines in the cloud in regions that are very nearby. Then, the flexibility and cost reduction offered by cloud computing make the overhead negligible. The following experiments were conducted for testing the detection capability of the framework. Here, we only focus on attacks consisting in modifying the source code of a method or a function deliberately (by a human being) or made by a virus. As explained above, it has been previously that this kind of attacks may appear in voting systems.

Table 5. Latency measurement.

Number of Requests	On Premises	In the Cloud
100	0.27 s	0.34
200	0.69 s	0.87
300	1.10 s	1.40
400	1.80 s	2.56
500	2.48 s	3.24
600	3.45 s	5.13
700	4.41 s	6.35
800	5.65 s	7.24
900	6.89 s	8.68
1000	8.41 s	9.17

Experiment 2: We evaluated the time elapsed to detect the insider attacks coming from both a modification of the source code and a virus. For this proof of concept, the virus (or malware) considered modifies the code of the vote application so that several votes for some candidates are counted as votes of the hacker's preferred candidate. A signature of this virus was incorporated to our virus database based on the virusshare's (https://virusshare.com) database.

The accuracy of the detection mechanism was discussed by Cavalli *et al.* [38]. In this section, we only evaluate the efficiency of the monitoring *with respect to* the two attacks above.

In Table 6, the modification seems to be easily detectable than the virus. This was predictable since the database of virus may contain a larger number of rows in comparison with the database containing the hashes of the methods. To have a fair detection time, one can have a unique database. The drawback of this solution is that we lose readability and flexibility. Along with the detection of attacks, the system reacts, as mentioned in Section 4. This reaction is transparent for the user. Although we can detect attacks with great granularity, it is also important to consider the impact of the monitoring mechanism. That is why we measured the overhead of the new monitoring method in the next experiment.

Table 6. Detection mean time.

Virus	Modification
0.053 s	0.031 s

Experiment 3: Evaluation of the impact of the monitoring mechanism.

As shown in Table 7, the overhead of the monitoring is not too significant. In future works, we will investigate how to reduce this overhead. The detection approach based on software reflection is suitable for the monitoring of cloud applications deployed as choreographies of services. To a certain extent, it can also be useful for detecting attacks such as buffer overflows and SQL injections. One limitation of such approach is the fact that this detection is only appropriate for attacks targeting the source code. However, the main limitation is that the approach cannot be applied in programs developed in programming languages that do not allow reflection or do not provide a powerful reflection API.

Table 7. Overhead of the monitoring mechanism.

Number of Requests	Without Monitoring (s)	With Monitoring (s)
100	0.38	0.46
200	0.88	0.91
300	1.39	1.4
400	2.50	2.57
500	3.26	3.27
600	5.22	5.30
700	6.31	6.35
800	7.21	7.45
900	8.89	9.12
1000	9.15	10.02

6. Conclusions and Perspectives

In this paper, we investigate the attack tolerance or resilience issue. We show that, to better tolerate and limit the impact of these attacks, the monitoring of the information systems is of paramount importance for any organization. However, the monitoring and detection of attacks require an awareness of the risks that the system might be exposed to. As such, we propose a risk-based monitoring approach. This methodology involves the following aspects: (i) assets identification to define what is necessary to protect; (ii) threats and vulnerability analysis to evaluate the potential flaws the system may suffer; (iii) risk analysis to categorize the threats that can exploit the system vulnerabilities; (iv) system monitoring to detect potential occurrences of attacks; and (v) remediation strategies to repel or mitigate the impact of the attacks. Leveraging this methodology, we develop a new attack tolerance framework based on formal monitoring techniques as well as diversification and reflection software techniques. We instantiate the risk-based methodology for services-based applications deployed in the cloud and propose an offline and online attack tolerance framework for web services-based application in the cloud. With this aim, we first express any application deployed in the cloud as a choreography of services, which must be continuously monitored and tested. Then, we extend a formal framework for choreography testing by incorporating the methods for detecting and mitigating attacks presented in the previous sections. Adding mechanisms of detection and reaction on the fly to these applications ensures optimal attack tolerance.

As such, the risk-based monitoring approach is compliant with cloud asset management strategies of companies. In fact, this approach delivers visibility, control of all the assets of a cloud, and it is a crucial first step towards a more secure cloud.

Now, let us discuss improvements and open directions. We define in this paper attack tolerance as the ability of a system to continue to function properly with minimal degradation of performance, despite intrusions. The aim is to detect the known and unknown attacks and if not possible to reduce their impact on the system. Although we obtained satisfactory results, we believe that we can improve the tolerance to attacks if we can somehow anticipate or predict these attacks. Thus, in addition to detection and remediation, it would be necessary to be able to predict and anticipate future attacks. We think that the following two axes would be interesting to investigate.

Diagnosticability and predictability [55]. Diagnosis consists in designing and implementing algorithms for verifying the formal properties of the system, ensuring that a model, which is known in advance of observable events, allows the detection and discrimination of a set of possible failures. Similarly, predictability is the ability to predict a future occurrence of a fault using the observable events preceding. We think that, if we can predict the occurrence of a fault, it would be interesting to prevent it from taking place and therefore to tolerate attacks effectively. However, an important step for using diagnosticability and predictability for attack tolerance will therefore be the formalization of faults that can occur from attacks.

Big data and machine learning. Recently, machine learning has emerged as a means to enhance security [56]. The authors reviewed the literature of machine learning (ML) and data mining (DM) methods for intrusion detection. This study evaluated the different existing algorithms. They pointed out that the most effective methods for cyber detection must be established and adapted to the specificity of the attacks. Furthermore, adding big data to machine learning [57] can improve cyber security. The introduction of Big Data processing led to a new era in the design and development of large-scale data processing systems. The idea is that data in raw format make it possible to create statistical baselines to identify normality. Subsequently, it is possible to instantly determine when the data deviate from this standard. These historical data also make it possible to create predictive and statistical models. While some supervised and unsupervised learning algorithms are already available for big data, there is much room for improvement. It has be recognized that the false-positive rate of machine learning algorithms is too high and the alerts generated are not always sufficiently interpretable to enable their exploitation. In summary, there is a research avenue for the application of such techniques for attack tolerance [57].

The result of using predictability and/or ML and big data in conjunction with our attack tolerance methods would be the design and the implementation of a framework for software systems that is attack tolerant in the sense that is has the possibility to continue to deliver their services even after a successful attack and is able to recover quickly and learn from the past.

Author Contributions: xxx.

Funding: xxx.

Conflicts of Interest: xxx.

Acknowledgments: Montimage's authors acknowledge the support of the European Commission (H2020 VeriDevOps project) under grant 957212.

References

1. Dhirani, L.L.; Newe, T.; Lewis, E.; Nizamani, S.A. Cloud computing and Internet of Things fusion: Cost issues. In Proceedings of the Eleventh International Conference on Sensing Technology, Sydney, Australia, 4–6 December 2017; IEEE: Piscataway, NJ, USA, 2017, pp. 1–6.
2. Younis, Y.A.; Kifayat, K.; Hussain, A. Preventing and Detecting Cache Side-Channel Attacks in Cloud Computing. In *Proceedings of the Second International Conference on Internet of Things, Data and Cloud Computing*; Association for Computing Machinery: New York, NY, USA, 2017.
3. Zhang, T.; Zhang, Y.; Lee, R.B. CloudRadar: A Real-Time Side-Channel Attack Detection System in Clouds. In *Research in Attacks, Intrusions, and Defenses*; Monrose, F., Dacier, M., Blanc, G., Garcia-Alfaro, J., Eds.; Springer International Publishing: Cham, Switzerland, 2016; pp. 118–140.
4. Tirumala, S.S.; Sathu, H.; Naidu, V. Analysis and Prevention of Account Hijacking Based INCIDENTS in Cloud Environment. In Proceedings of the 2015 International Conference on Information Technology (ICIT), Bhubaneswar, India, 21–23 December 2015; pp. 124–129, doi:10.1109/ICIT.2015.29.
5. Ouffoué, G.; Zaïdi, F.; Cavalli, A.R. Attack Tolerance for Services-Based Applications in the Cloud. In Proceedings of the IFIP International Conference on Testing Software and Systems-ICTSS, Paris, France, 15–17 October 2019; pp. 242–258.

6. Duan, Q.; Wang, S.; Ansari, N. Convergence of Networking and Cloud/Edge Computing: Status, Challenges, and Opportunities. *IEEE Netw.* **2020**, 34, 144–148, doi:10.1109/MNET.011.2000089.
7. Saha, G.K. Software based fault tolerance: A survey. *Ubiquity* **2006**, *2006*, 1:1.
8. Knight, J. *Fundamentals of Dependable Computing*; CRC Innovations in Software Engineering and Software Development: Boca Raton, FL, USA, 2012.
9. Zhao, W. Towards practical intrusion tolerant systems: A blueprint. In Proceedings of the Annual Workshop on Cyber Security and Information Intelligence Challenges, 2008; p. article No 19.
10. La, V.H. Security Monitoring for Network Protocols and Applications. Ph.D. Thesis, Université Paris-Saclay, Saint-Aubin, France, 2016.
11. Mishra, P.; Pilli, E.S.; Varadharajan, V.; Tupakula, U.K. Intrusion detection techniques in cloud environment: A survey. *J. Netw. Comput. Appl.* **2017**, *77*, 18–47.
12. Yan, Q.; Yu, F.R.; Gong, Q.; Li, J. Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) Attacks in Cloud Computing Environments: A Survey, Some Research Issues, and Challenges. *IEEE Commun. Surv. Tutorials* **2016**, *18*, 602–622.
13. Meixner, C.; Develder, C.; Tornatore, M.; Mukherjee, B. A Survey on Resiliency Techniques in Cloud Computing Infrastructures and Applications. *IEEE Commun. Surv. Tutorials* **2016**, *18*, 2244–2281.
14. Raj, S.; Varghese, G. Analysis of intrusion-tolerant architectures for Web Servers. In Proceedings of the 2011 International Conference on Emerging Trends in Electrical and Computer Technology, Nagercoil, India, 23–24 March 2011; pp. 998–1003.
15. Saidane, A.; Nicomette, V.; Deswarte, Y. The Design of a Generic Intrusion-Tolerant Architecture for Web Servers. *IEEE Trans. Dependable Secur. Comput.* **2009**, *6*, 45–58.
16. Constable, R.; Mark, M.B.; Robbert, V.R. *Investigating Correct-by-Construction Attack-Tolerant Systems*; Technical Report; Department of Computer Science, Cornell University: Ithaca, NY, USA, 2011.
17. Roy, A.; Kim, D.S.; Trivedi, K.S. Cyber security analysis using attack countermeasure trees. In Proceedings of the 6th Cyber Security and Information Intelligence Research Workshop, CSIIRW, Oak Ridge, TN, USA, 21–23 April 2010; Sheldon, F.T., Prowell, S.J., Abercrombie, R.K., Krings, A.W., Eds.; ACM: New York, NY, USA, 2010, p. 28.
18. Nicomette, V.; Powell, D.; Deswarte, Y.; Abghour, N.; Zanon, C. Intrusion-tolerant fine-grained authorization for Internet applications. *J. Syst. Archit.* **2011**, *57*, 441–451.
19. Nguyen, Q.L.; Sood, A. A Comparison of Intrusion-Tolerant System Architectures. *IEEE Secur. Priv.* **2011**, *9*, 24–31.
20. Hierons, R.M.; Bogdanov, K.; Bowen, J.P.; Cleaveland, R.; Derrick, J.; Dick, J.; Gheorghe, M.; Harman, M.; Kapoor, K.; Krause, P.J.; et al. Using formal specifications to support testing. *ACM Comput. Surv.* **2009**, *41*, 9:1–9:76.
21. Stankovic, V.; Strigini, L. *A Survey on Online Monitoring Approaches of Computer-Based Systems*; Technical Report; Centre for Software Reliability, City University London: London, UK, 2009.
22. Lee, S.; Levanti, K.; Kim, H.S. Network monitoring: Present and future. *Comput. Networks* **2014**, *65*, 84–98.
23. Cavalli, A.R.; Higashino, T.; Nú nez, M. A survey on formal active and passive testing with applications to the cloud. *Ann. Des Télécommun.* **2015**, *70*, 85–93.
24. Sánchez, C.; Schneider, G.; Ahrendt, W.; Bartocci, E.; Bianculli, D.; Colombo, C.; Falcone, Y.; Francalanza, A.; Krstic, S.; Lourenço, J.M.; et al. A survey of challenges for runtime verification from advanced application domains (beyond software). *Form. Methods Syst. Des.* **2019**, *54*, 279–335.
25. Nguyen, H.N.; Poizat, P.; Zaïdi, F. Passive Conformance Testing of Service Choreographies. In Proceedings of the SAC'12, Trento, Italy, 26–30 March 2012; pp. 1927–1934.
26. Merayo, M.G.; Hierons, R.M.; Nú nez, M. Passive Testing with Asynchronous Communications and Timestamps. *Distrib. Comput.* **2018**, *31*, 327–342, doi:10.1007/s00446-017-0308-0.
27. Merayo, M.G.; Hierons, R.M.; Nú nez, M. A Tool Supported Methodology to Passively Test Asynchronous Systems with Multiple Users. *Inf. Softw. Technol.* **2018**, *104*, 162–178.
28. Kuyoro, S.; Ibikunle, F.; Okolie, S. Security Issues in Web Services. *Int. J. Comput. Sci. Netw. Secur.* **2012**, *12*, 23–27.
29. Ficco, M.; Rak, M. Intrusion Tolerant Approach for Denial of Service Attacks to Web Services. In Proceedings of the 2011 First International Conference on Data Compression, Communications and Processing, Palinuro, Italy, 21–24 June 2011; IEEE Computer Society: Washington, DC, USA, 2011; CCP'11, pp. 285–292.

30. Singhal, A.; Winograd, T.; Scarfone, K. Recommendations of the National Institute of Standards and Technology. 2007. Available online: http://cyberwar.nl/d/20111220_draft-SP800-155_Dec2011_SCRUBBED.pdf (accessed on).
31. Sharma, R.; Sood, M.; Sharma, D. Modeling Cloud SaaS with SOA and MDA. In *Advances in Computing and Communications*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 511–518.
32. Sadegh, B.; Azgomi, M.A. A New Architecture for Intrusion-Tolerant Web Services Based on Design Diversity Techniques. 2015. Available online: https://www.sid.ir/FileServer/JE/5055520150405.pdf (accessed on).
33. CAPEC. Common Attack Pattern Enumeration and Classification. 2020. Available online: https://capec.mitre.org (accessed on 3 December 2020).
34. PTEST. Penetration Testing Execution Standard. 2020. Available online: https://www.cybersecurityeducationguides.org/what-is-the-ptes-penetration-testing-execution-standard/ (accessed on 3 December 2020).
35. OWASP. OWASP Benchmark. 2016. Available online: https://www.owasp.org/index.php/Benchmark (accessed on 3 December 2020).
36. Ouffoué, G.; Zaïdi, F.; Cavalli, A.R.; Lallali, M. Model-Based Attack Tolerance. In Proceedings of the 2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA), Taipei, Taiwan, 27–29 March 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 68–73.
37. Ouffoué, G.L.A.; Zaïdi, F.; Cavalli, A.R.; Lallali, M. Attack-Tolerant Framework for Web Services. In Proceedings of the 2017 IEEE International Conference on Services Computing (SCC), Honolulu, HI, USA, 25–30 June 2017; pp. 503–506, doi:10.1109/SCC.2017.75.
38. Cavalli, A.R.; Ortiz, A.M.; Ouffoué, G.; Sanchez, C.A.; Zaïdi, F. *Design of a Secure Shield for Internet and Web-Based Services Using Software Reflection*; *Web Services—ICWS 2018*; Springer International Publishing: Berlin/Heidelberg, Germany, 2018.
39. Buyya, R. Market-Oriented Cloud Computing: Vision, Hype, and Reality of Delivering Computing as the 5th Utility. In Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, Shanghai, China, 18–21 May 2009.
40. Dhirani, L.L.; Newe, T.; Nizamani, S. *Hybrid Multi-Cloud Demystifying SLAs for Smart City Enterprises Using IoT Applications*; IGI Global: 2020; pp. 52–67.
41. Zardari, S.; Bahsoon, R. Cloud Adoption: A Goal-Oriented Requirements Engineering Approach. In Proceedings of the SECLOUD, Waikiki, Honolulu, HI, USA, 22 May 2011; pp. 29–35.
42. Cloudtp. Cloud-Ready Application Development: Step-by-Step Guide. Available online: https://www.cloudtp.com/doppler/5-steps-building-cloud-ready-application-architecture/ (accessed on 2018).
43. Etchevers, X. Déploiement D'applications Patrimoniales en Environnements de Type Informatique dans le nuage. Ph.D. Thesis, Université de Grenoble, Grenoble, France, 2012.
44. Furtado, T.; Francesquini, E.; Lago, N.; Kon, F. A Middleware for Reflective Web Service Choreographies on the Cloud. In Proceedings of the 13th Workshop on Adaptive and Reflective Middleware, Bordeaux, France, 8–12 December 2014; ARM '14, pp. 9:1–9:6.
45. Nguyen, H.N. Une Approche Symbolique pour la Vérification et le Test des Chorégraphies de Services. Ph.D. Thesis, Université Paris-Sud, Orsay, France, 2013.
46. Qiu, Z.; Zhao, X.; Cai, C.; Yang, H. Towards the Theoretical Foundation of Choreography. In Proceedings of the WWW'07, 2007.
47. Hennessy, M.; Lin, H. Symbolic Bisimulations. *Theor. Comput. Sci.* **1995**, *138*, 353–389.
48. Pavel, S.; Noyé, J.; Poizat, P.; Royer, J.C. *A Java Implementation of a Component Model with Explicit Symbolic Protocols*; Software Composition; Springer: Berlin/Heidelberg, Germany, 2005.
49. Wehbi, B.; Montes De Oca, E.; Bourdellès, M. Events-based security monitoring using MMT tool. In Proceedings-IEEE 5th International Conference on Software Testing, Verification and Validation, ICST 2012, Montreal, QC, Canada, 17–21 April 2012; pp. 860–863.
50. Chaitanya K.R. Comparison of Event Choreography and Orchestration Techniques in Microservice Architecture. *Int. J. Adv. Comput. Sci. Appl.* **2018**, *9*, 18–22.
51. Estehghari, S.; Desmedt, Y. Exploiting the Client Vulnerabilities in Internet E-voting Systems: Hacking Helios 2.0 As an Example. In Proceedings of the 2010 International Conference on Electronic Voting Technology/Workshop on Trustworthy Elections, USENIX Association, 2010; pp. 1–9.

52. Beaucamps, P.; Reynaud, D.; Marion, J.; Filiol, E. On the Impact of Malware on Internet Voting. 1st Luxembourg Day on security and reliability, 2009.
53. Wolchok, S.; Wustrow, E.; Isabel, D.; Halderman, J.A. Attacking the Washington, D.C. Internet Voting System. In *Financial Cryptography and Data Security*; Keromytis, A.D., Ed.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 114–128.
54. Springall, D.; Finkenauer, T.; Durumeric, Z.; Kitcat, J.; Hursti, H.; MacAlpine, M.; Halderman, J.A. Security Analysis of the Estonian Internet Voting System. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, 3–7 November 2014; CCS '14, pp. 703–715.
55. Ibrahim, H. SAT-Based Diagnosability and Predictability Analysis in Centralized and Distributed Discrete Event Systems. Ph.D. Thesis, Université Paris-Saclay, Saint-Aubin, France, 2016.
56. Buczak, A.L.; Guven, E. A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Commun. Surv. Tutorials* **2016**, *18*, 1153–1176.
57. Suresh, A.T.; Yu, F.X.; McMahan, H.B.; Kumar, S. Distributed Mean Estimation with Limited Communication. Available online: http://proceedings.mlr.press/v70/suresh17a.html (accessed on).