A Coloured Petri Nets Based Attack Tolerance Framework

Wenbo Zhou, Philippe Dague, Lei Liu, Lina Ye, Fatiha Zaïdi

▶ **To cite this version:**

Wenbo Zhou, Philippe Dague, Lei Liu, Lina Ye, Fatiha Zaïdi. A Coloured Petri Nets Based Attack Tolerance Framework. 27th Asia-Pacific Software Engineering Conference (APSEC 2020), IEEE, Dec 2020, Singapore, Singapore. hal-03133790

HAL Id: hal-03133790
https://hal.science/hal-03133790

Submitted on 7 Feb 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Coloured Petri Nets Based Attack Tolerance Framework

Wenbo Zhou[1, 2, 3], Philippe Dague[3, 4], Lei Liu[1, *], Lina Ye[3, 4, 5], Fatiha Zaïdi[3]
[1]College of Computer Science and Technology, Jilin University, Changchun, China
[2]Key Laboratory of Symbolic Computation and Knowledge Engineering, Ministry of Education, Changchun, China
[3]Laboratoire de Recherche en Informatique, Université Paris-Saclay, Orsay, France
[4]Laboratoire Spécification et Vérification, ENS Paris-Saclay and Inria, Gif-sur-Yvette, France
[5]CentraleSupélec, Université Paris-Saclay, Orsay, France
zhouwb17@mails.jlu.edu.cn, philippe.dague@lri.fr, liulei@jlu.edu.cn, lina.ye@lri.fr, fatiha.zaidi@lri.fr

***Abstract*—Web services provide a general basis of convenient access and operation for cloud applications. However, such services become very vulnerable when being attacked, especially in the situation where service continuity is one of the most important requirements. This issue highlights the necessity to apply reliable and formal methods to attack tolerance in Web services. In this paper, we propose a Coloured Petri Nets based method for attack tolerance by modelling and analysing basic behaviours of attack-network interaction, attack detectors and their tolerance solutions. Furthermore, complex attacks can be analysed and tolerance solutions deployed by identifying these basic attack-network interactions and composing their solutions. The validity of our method is demonstrated through a case study on attack tolerance in cloud-based medical information storage.**

***Keywords*-attack tolerance; coloured Petri nets; Web services; formal methods; cloud security**

I. Introduction

Service-oriented architecture supports a lot of modern software systems, which advocates "Everything as a Service". This thought is absorbed by cloud computing and developed as its typical service models, i.e., Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). Cloud service providers offer multiple resources, e.g., VM (Virtual Machine), to clients on demand through the Internet. However, since clients' information is centrally stored and managed in cloud services, it will cause terrible loss once such services are attacked successfully.

Attacks bring great threats to cloud applications and hinder further development of cloud services. How to deal with attacks towards cloud services becomes an important issue. Recently, attack tolerance has been highlighted as a significant property, which allows systems to continue when attacks occur [1]–[5]. However, there are very few studies towards attack tolerance for cloud services, especially researches combining formal methods into attack tolerance [2]. Moreover, most of the existing work focuses on applications. There is a lack of research on formal foundation for better understanding the essence of attacks and their solutions.

To further tolerate attacks in a reliable way, applying formal methods is a good choice, which provides strong guarantee for systems security. As one of the most typical formal models, Coloured Petri Nets (CPNs) provide an elegant and practical mathematical formalism by combining Petri nets with a programming language to obtain a scalable modelling language for concurrent systems [6]. CPNs have been used to model and analyse security aspects of many critical systems [7]–[10]. Specially, useful related tools provide strong support for verification and validation of systems models. Considering good visualization for better comprehension, concurrency description, executable simulation and wide applications, we choose CPNs as the formal model to analyse the attack tolerance issue.

In this paper, inspired from formal constructions, we define some meta operations for attack tolerance, modelled and analyzed with CPNs. First, we model the basic behaviours of attack-network interaction as CPN-based patterns. Then, we provide CPN-based detection mechanisms and investigate some CPN-based tolerance solutions according to this detection. Finally, we evaluate the effectiveness of our approach by a case study. Note that our framework is generic to services in general, but as cloud services are among the most well-developed and complex services, we explain the approach in this scenario. Our contributions are as follows.

- We explore a semi-formal framework for attack tolerance, specifically using CPNs to design and deploy a tolerance solution according to meta attack operations.
- We classify the basic attack-network interactions behaviours into four patterns, i.e., blocking, leakage, stealing and injection. Meanwhile, we analyze their detection conditions and construct their corresponding tolerance solutions, including bypassing, encryption and compensation.
- We evaluate our method through a case study about attack tolerance in cloud-based medical information storage, and demonstrate its effectiveness by means of CPN Tools.

This paper is organised as follows. Sect. 2 briefly introduces Coloured Petri Nets, which are used to model Origin-Attack-Tolerance Nets, in brief OAT Nets. Sect. 3 presents our CPNs based methods, including basic interaction attack patterns, attack detectors and their tolerance solutions. Sect. 4 discusses

* Correspondence to: Lei Liu ⟨liulei@jlu.edu.cn⟩

a case study about attack tolerance in cloud-based medical information storage with the help of CPN Tools. Sect. 5 introduces related work. Finally, Sect. 6 concludes this paper and outlines future work.

II. Preliminaries

Our method is mainly described using CPNs. If CPNs have the same expressive power as Petri nets, in practice they offer much more modelling power with better structuring facilities, e.g., types and modules. A typical definition of CPNs is as follows [11].

Definition 1 (Coloured Petri Net): A Coloured Petri Net is a tuple $CPN = (P, T, A, \Sigma, V, C, G, E, I)$, where

- P is a finite set of places;
- T is a finite set of transitions and $P \cap T = \emptyset$;
- $A \subseteq (P \times T) \cup (T \times P)$ is a set of arcs;
- Σ is a finite set of types, each one being a non-empty "colour set";
- V is a finite set of typed variables, i.e., $type(v) \in \Sigma$ for all $v \in V$;
- $C : P \to \Sigma$ is a colouring function that associates a type to each place;
- $G : T \to Expr_V$ is a labelling function that associates expressions (called guards) with free variables from V to transitions such that $type(G(t)) = Bool$;
- $E : A \to Expr_V$ is a labelling function that associates expressions with free variables from V to arcs such that $type(E(a)) = C(p)_{MS}$, where p is the place connected to the arc a and $C(p)_{MS}$ is a multiset type;
- $I : P \to Expr_\emptyset$ is an initialisation function that associates an expression without free variables to every place such that $type(I(p)) = C(p)_{MS}$.

Definition 2 (Marking): Given a CPN, a marking is a function M mapping P to multisets of tokens, such that $type(M(p)) = C(p)_{MS}$. The initial marking is given by I.

Definition 3 (Enabled): Given a marking M, $t \in T$, its preset $pre(t) = \{p|(p,t) \in A\}$ (resp., its postset $post(t) = \{p|(t,p) \in A\}$) and a binding b of its variables ($\langle b \rangle$: evaluation in b), (t, b) is enabled for M if $M(p)$ satisfies (i.e., is greater than or equal to) $E(p,t)\langle b \rangle$ for all places $p \in pre(t)$ and $G(t)\langle b \rangle = true$.

Definition 4 (Firing): Given $t \in T$ and a binding b, if (t, b) is enabled for M, t may fire and yield a new marking M', defined, for all $p \in pre(t) \cup post(t)$, by $M'(p) = M(p)$ -- $E(p,t)\langle b \rangle$ ++ $E(t,p)\langle b \rangle$ where $E(p,t)\ \langle b \rangle = 0$ (resp., $E(t,p)\langle b \rangle = 0$) for every $p \notin pre(t)$ (resp., $post(t)$). Here, -- (resp., ++) indicates the subtraction (resp. the sum) of multisets [11].

Fig. 1 is an example of firing. In the marking before firing t, the place p_1 is set with "2`1 ++ 3`4 ++ 1`5" ($p_2$ with "1`3"), which is a multiset with two occurrences of the value 1, three occurrences of the value 4 and one occurrence of the value 5. If the value "5" is selected randomly to bind to the variable a, then the guard "$a > 2$" is satisfied, and t is enabled and can be fired. After the firing, t produces 1`6 to p_2 according to the expression $E(t, p_2)$. We refer to [11] for precise definitions

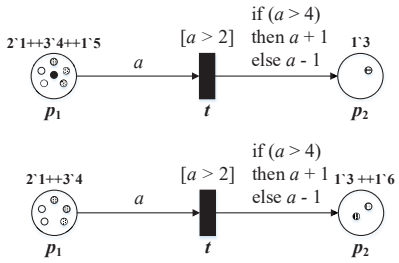

Fig. 1. An example before (top) and after (bottom) firing

of the semantics and show how CPN models are executable and good at modelling behaviours of concurrent systems. This enlightens their use for analysis of security design aspects in concurrent systems, e.g., attack tolerance.

III. Method Overview

In our model-based method, we try to discover the essence of attacks by means of semi-formal construction. This is done by modelling some basic behaviours of attack-network interaction together with their detectors and tolerance solutions, which takes advantages of CPN theory, improving thus the ability for the network to tolerate an attack. To the best of our knowledge, this work is the first that attempts to apply CPN to attack tolerance problem, thus leading to better understanding and capturing it through the formal constructions. On the one hand, we propose some basic attack-network interaction patterns modelled with CPN, which will be helpful for constructing more complex models of attack tolerance. On the other hand, since the deployment of attack tolerance solutions is always time- and resource-consuming, simulations with the help of CPN Tools provide early access to effects of solutions. These executable models are also useful for further analysis or verification of these solutions (e.g., checking deadlock or reachability), since many tools support such functions. The major steps of our whole method are as follows.

1) An original net (O net) equipped with detectors (monitoring functions), modelled as a CPN, should be simulated with CPN Tools.
2) A predefined attack, modelled as a CPN (A net), is injected to O net to constitute the attacked net (OA net), whose simulation continues.
3) The attack is detected by the *detectors* and then identified according to our pre-built library of *basic attack-network interaction patterns*.
4) From this identification, a tolerance solution for this attack is generated as a CPN (T net) by composition from our pre-built library of *basic tolerance solutions*.
5) The simulation is suspended, the tolerance solution is deployed by composing T net with OA net to construct attack tolerant net (OAT net), whose simulation resumes showing that it tolerates the injected attack.

In real-world practice, a tolerance solution should be deployed immediately when an attack is detected. In this paper, we use CPN Tools to simulate the framework, where it is not allowed to change places and transitions dynamically during

continuous multiple-steps simulation. To mimic the real situation, we use the "monitors" function of CPN Tools to monitor the simulation. When an attack is detected, the monitors would suspend the simulation. Then, a tolerance solution is deployed. This way does not interfere the correctness of our method as the "suspension" does not change any other parts except the deployment of a tolerance solution. It is the same as deploying a tolerance solution immediately when a detector detects an attack. After deploying the solution, the simulation continues from the pausing state (i.e., marking) to ensure the continuity. Note that we assume that a new deployed tolerance solution cannot be attacked. If the solution is allowed to be attacked again, it leads to an infinite meta-hierarchy problem, which is out of the scope of this paper. Next we illustrate three important concepts, i.e., basic attack-network interaction patterns, attack detectors and basic tolerance solutions.

A. Basic Attack-network Interaction Patterns

Basic attack-network interaction patterns represent malicious actions, such as unauthorisedly collecting information, illegally changing entities, and so on. Here, we classify basic units, each consisting of one place, one transition and one arc. This classification is neither empirical or experimental but considers exhaustively all the possible constructions of basic interaction units between the O and A nets. As shown in Table I, there are totally 16 types of basic units (according to whether its place (resp., transition or arc) belongs to O or A net and arc direction is from place to transition or from transition to place). Since this paper focuses on the interaction between O net and A net, four basic interaction attack patterns, i.e., blocking, leakage, stealing and injection (No. 2, 3, 6, 7), are kept for study, which can be combined to constitute more complex attacks. In Table I, the other cases are internal behaviours of O net (No. 9, 13) or of A net (No. 4, 8) and meaningless (No. 1, 5, 10-12, 14-16). For No. 1, 5, 12, 16, it is abnormal that both place and transition belong to O net (resp. A net) while arc belongs to A net (resp. O net). For No. 10, 11, 14, 15, it is also senseless that O net leaves an arc linked to a place or transition of A net. So, these cases can be excluded.

In the table and all the following figures, O net is mainly depicted in black with detectors in grey, A net in red, and T net in blue or other colours if needed. Places or transitions shared by two or three of those nets are depicted with the two or three corresponding colours.

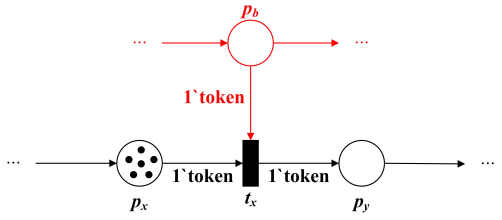

Fig. 2. Blocking

Blocking is a behaviour that blocks the process execution and controls when it can be continued. Fig. 2 presents an example of the blocking pattern, where the transition t_x is not enabled since there is no token in p_b. Consequently, more and more tokens arrive at p_x as shown in Fig. 2 and the marking of p_y remains unchanged for a long time. Moreover, an attacker can even control when to stop blocking, and make the process run again like nothing had happened.

Leakage refers to a behaviour that discloses some important information to attackers unintentionally, which is however not detected by the original process. As shown in Fig. 3, the token o_x has been delivered to both p_y and p_l, the latter belonging to an attacker.

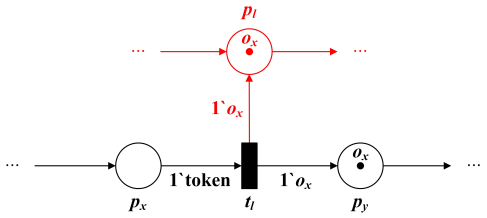

Fig. 3. Leakage

Stealing means that attackers take data, VM or other entities away without a permission of the system and keep it. In Fig. 4, the transition t_s intends to steal the token o_x from the place p_x. Note that there are two differences between stealing and leakage: (1) for "stealing", an attacker needs to fire a transition to obtain the concerned token, while for "leakage", tokens flow to an attacker's place without any initiative action of the attacker; (2) stealing implies that tokens are removed from the place where they should be, however, leakage never changes related tokens in O net.

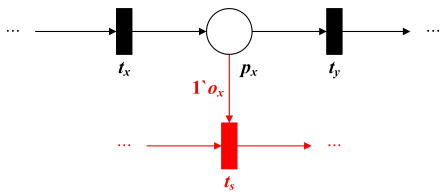

Fig. 4. Stealing

Injection is a way used by attackers to throw some modified data or codes, usually malicious, into a system, that can facilitate the coordination of attackers to extract information from or interfere with the system. A successful injection, which often needs the help of other basic attack-network interaction patterns, should avoid the attention of a normal process. Normally, an attacker first collects enough information, based on which a malicious entity is then constructed and finally injected back to the original system, as shown in Fig. 5.

Furthermore, compound attack patterns can be derived from the compositions of basic attack-network interaction patterns. For example, by combining stealing with injection, we can obtain two new attack patterns, named copy and modification, as shown in Fig. 6. The "copy" pattern takes a token away from the original process, copies it and puts it back to the

TABLE I
TYPES OF BASIC UNITS

No	CPN	Comment	No	CPN	Comment	No	CPN	Comment	No	CPN	Comment
1		/ [a]	5		/	9		IBON [b]	13		IBON
2		Blocking	6		Leakage	10		/	14		/
3		Stealing	7		Injection	11		/	15		/
4		IBAN [c]	8		IBAN	12		/	16		/

[a] Meaningless; [b] Internal Behaviour of Original Net; [c] Internal Behaviour of Attack Net

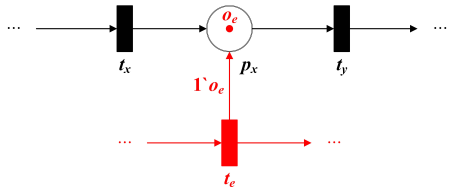

Fig. 5. Injection

original process. Different from "copy", the "modification" pattern puts a modified (usually malicious) token back to the original process. Note that these two attack patterns are quite simple compound attacks, and more complex ones can be constructed in a similar way.

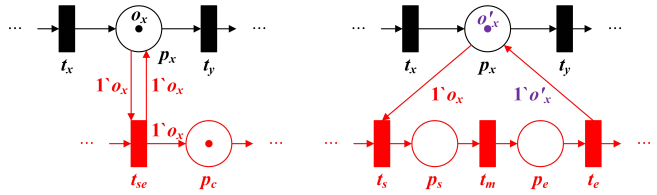

Fig. 6. Copy (left); Modification (right)

B. Attack Detectors

In this subsection, we present examples of monitoring functions for O net, in order to detect some of the attacks and identify their corresponding patterns. These detectors are designed and efficiently installed in O net.

When a transition is blocked by an attacker, the blocking can be detected by checking that at least one of two conditions is satisfied. The first is that the number of tokens in a place exceeds a fixed threshold, this accumulation being possibly a symptom that the transition after this place is blocked. As shown in Fig. 7, when the number of tokens in p_x exceeds 5, transition t_z will be fired and put a token to p_y. The second is that the marking of some place does not change for a predefined maximal time, indicating an interruption of the data flow. The limitation of our detection is that a blocking with a number of tokens or a time period below the pre-defined thresholds cannot be detected. Short-time temporary blocking (below the pre-defined threshold) is difficult to detect without more knowledge about an attacker and this is out of the modelling scope in this paper.

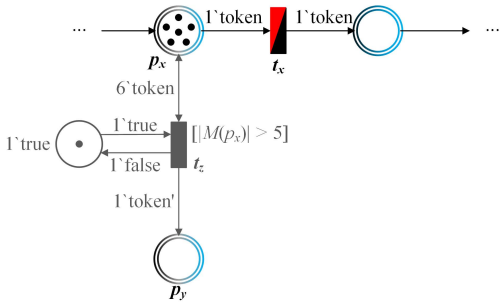

Fig. 7. Detector for blocking

Leakage is very difficult or even impossible to detect without explicit signals. So, for leakage, we can only take actions, e.g., encryption, to prevent attackers from using it.

For stealing, the detector determines whether some entities have been lost. As shown in the example of Fig. 8, the token o_{ex} is backed up to the place $p_{duplicate}$. Then, the transition $t_{compensate}$ judges whether o_{ex} has been lost according to $p_{duplicate}$, p_x and p_y. We use inhibitor arcs to indicate that o_{ex} belongs to neither p_x nor p_y. Notice that we detect stealing in the interval between p_x and p_y in this example, where the marking of p_x or p_y is equal to or less than one. More complex situations (e.g., multiple tokens) are similar to this example but with more control structures.

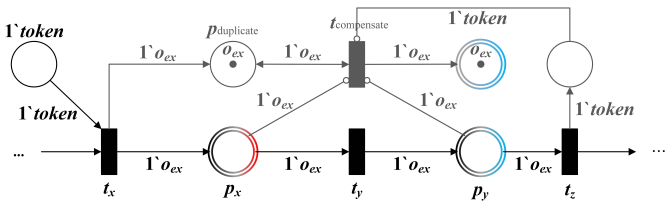

Fig. 8. Detector for stealing

When it comes to injection, detectors can check whether there is an intrusive or modified entity. As shown in Fig. 9, the place $p_{library}$ stores some necessary information for checking. When a token arrives at t_y, it will be checked with the

information from the library, which is denoted as o_{lib}. A bad token will be removed from O net, and put into $p_{isolation}$ for further analysis. The place p_{check_ctrl} is used to guarantee that every token will be checked and the place p_{ctrl} makes sure that there will be only one token in the place p_x. In [4], some parts of software are assigned "hash codes", whose modifications are signals of malicious actions. Actually, checking whether hash codes have been modified can be considered as one of specific implementations of our checking detector.

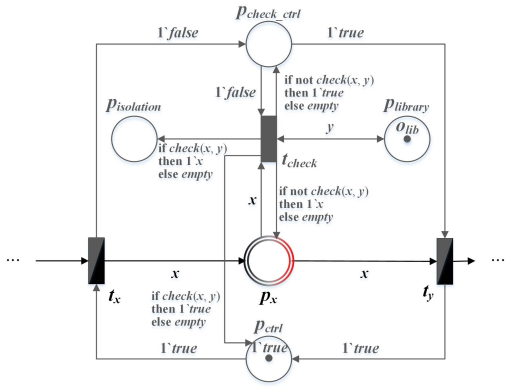

Fig. 9. Detector for injection

It is worth noting that, for stealing and injection, an idea for more general detection process would be to use place invariants [12], which depend only on the topological structure of the Petri net and are independent of any dynamic process. In absence of attack in the O net, place invariants should always be satisfied; however, once some place invariants become false, there must be something wrong. Then, one way to locate the "error" place might be using the intersection of places related to multiple unsatisfied places invariants to reduce the searching scope. We would explore more about the role of place invariants and how to locate and tolerate attacks by using them in the future.

C. Basic Tolerance Solutions

In this subsection, we propose tolerance solutions for the above basic attack-network interaction patterns. They are deployed (in blue) according to the locations in the O net associated to the detectors (already in blue), in order to restore the normal function of an original system.

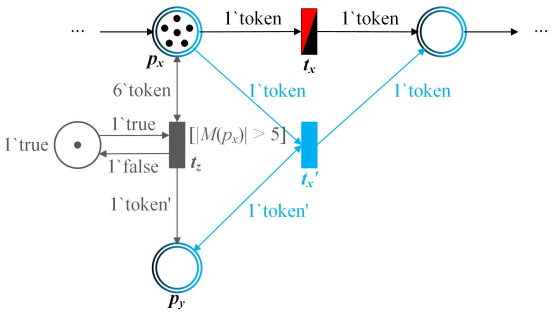

Fig. 10. Bypassing

Bypassing provides a good way to deal with blocking. Once the blocking detection condition is met, a bypassing path should be built and triggered. Fig. 10 presents an example for the number of tokens in a place exceeding a fixed threshold. In this case, the place p_y obtains a token. Then, the bypassing path is activated by triggering t'_x, and the blocking problem is solved. Moreover, this solution can also be used to mitigate overloading besides blocking.

Encryption is a process that encodes critical information to protect it from being illegally used by unauthorised users. In Fig. 11, a leakage occurs in t_l. If the token o_x has been encrypted as o_{ex} and the wiretapper is unable to decode it, the solution is successful. On the contrary, if the wiretapper can decode what he wants, the encryption solution fails. Actually, this is a very difficult problem, because we can never know the ability of an attacker. Since we cannot predict the ability of attackers to decode information, the only thing we can do is the prevention with best efforts.

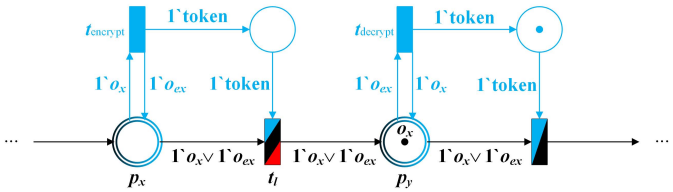

Fig. 11. Encryption

Compensation solution makes up for lost entities. To tolerate stealing, two things need to be done: (1) encrypting information to make attackers unable to use it, described as above; (2) compensating the lost entities in the original process, according to the backup of any token found to be lost by the detector. In the example of Fig. 12, only one token has to be compensated. In practice, we may need to deal with a set of lost tokens, and the implementations of control parts are different according to different cases.

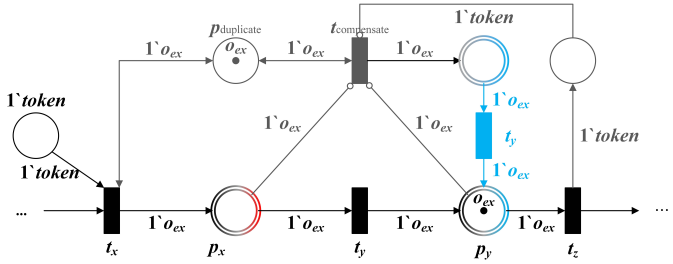

Fig. 12. Compensation

The compensation solution can also handle injection. Detectors monitor whether entities are intrusive or modified. If an entity is intrusive, i.e., the entity does not belong to the O net, we only need to remove the injected token. If an entity is illegally modified, besides removing the modified entity, a normal entity should be compensated in the original process.

These solutions correspond to the basic attack-network interaction patterns. If a complex attack is composed of multiple

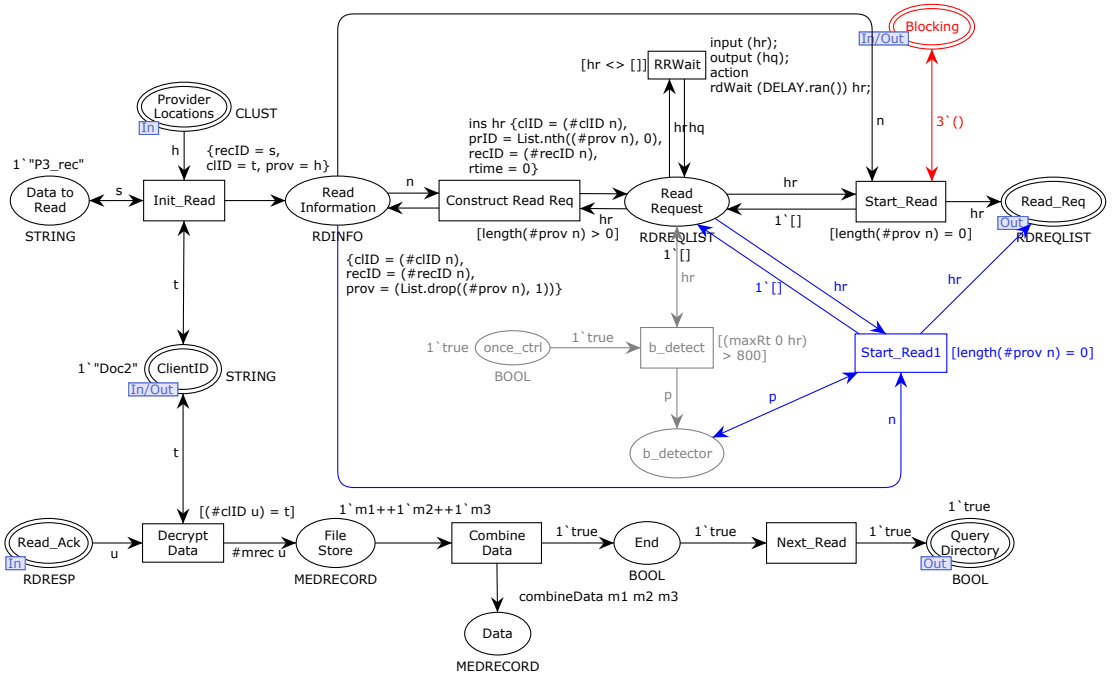

Fig. 13. Doctor reading

such patterns, we can construct its tolerance solution by combining the related solutions together, which is illustrated by the case study in the next section. Note that our framework models these attacks and solutions with the view of a relatively low-cost modelling. Though we deploy one single tolerance solution in response to one given attack, the library of generic attack patterns / tolerance solutions remains small. The large one is the potential combination of all possible deployment instances, but we do not compute it in advance. After the detectors detect and identify the attacks, the tolerance solution is instantiated with the precise locations in the network, and thus deployed on the fly.

IV. Case Study

To show the effectiveness of our framework, we apply our approach to a case study about cloud-based medical information storage originally presented in [13], [14] and modelled with CPN Tools, a tool for editing, simulating and analysing CPNs [15]. In this system, patients and doctors read or write medical records stored through cloud-based information storage. Note that [13], [14] focus on security and fault tolerance, while we introduce basic attack-network interactions and illustrate our tolerant solutions. The major differences are: (1) Their example only involves one patient and one doctor, while we adapt it to a scenario with three patients and two doctors, extendable to more patients and doctors in a straightforward way. (2) We replace fault tolerance parts by integrating the attack tolerance framework. The whole .cpn file can be found at the site "https://github.com/TURTING-BO/CPNs-Attack-Tolerance".

In this case study, there are mainly four modules including directory, patient, doctor and cloud. The directory module receives query requests before responding. The patient or

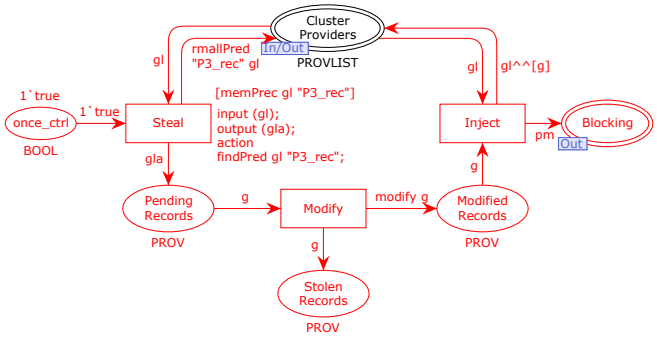

Fig. 14. Attack

doctor modules read or write records, where a patient can only read his/her own information but a doctor can both read and write records of his/her patients. The cloud module stores records of patients and manages reading and writing operations. When a patient needs to read records, he/she first sends a query request to the directory module. After receiving the request, the directory module responds to the patient with a provider list, indicating where his/her records are stored. Then, the patient constructs a read request according to the provider list and sends it to the cloud module. The cloud module supervises the reading process and returns the records if successful. Finally, the patient receives his/her records. Note that a patient's record is distributed in three providers to improve security. The read and write operations of doctors are similar to the read operation of patients.

In this medical storage system, the basic attack-network interactions may violate the following security requirement aspects.

- Blocking: medical records must be guaranteed to be available to all doctors at any time to save the patients;

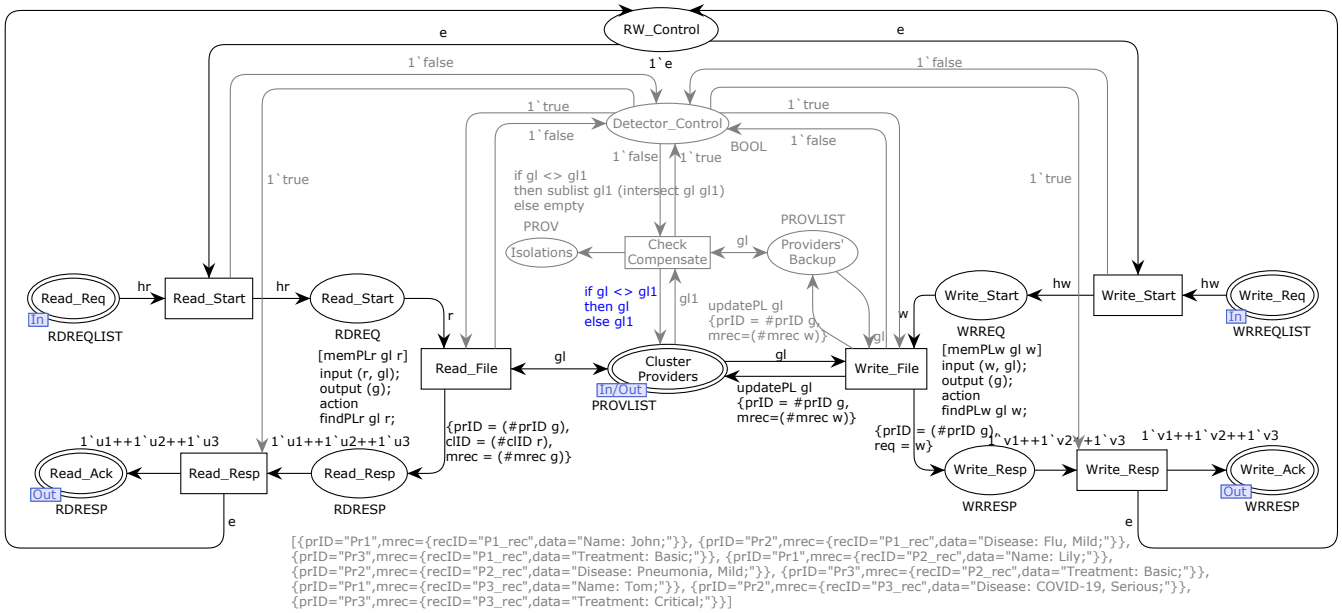

Fig. 15. Cloud

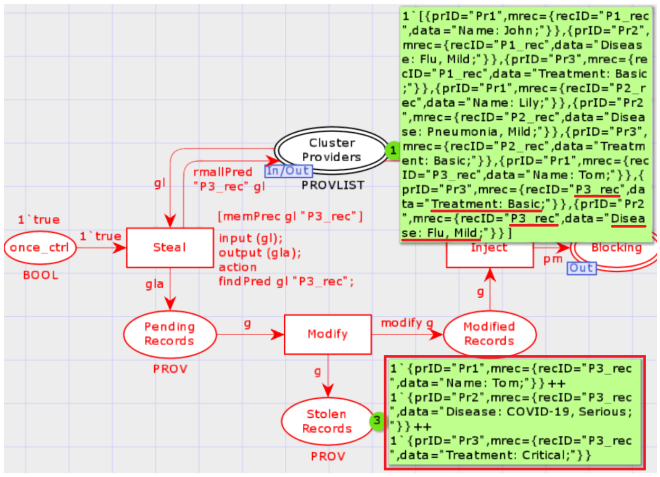

Fig. 16. Attack result

- Leakage: patients' records cannot be leaked to a third part in order to protect the privacy;
- Stealing: patients' records cannot be lost as the history of health status is critical to the diagnosis and treatment;
- Injection: insider attacks (e.g., from employees of cloud provider) are not allowed to modify records;

Now consider three patients A, B, C and two doctors X, Y. The medical record of the patient C is {recID = "P3_rec", data = "Name: Tom; Disease: COVID-19, Serious; Treatment: Critical;"}. The doctor Y needs to read the record of C and determine the therapy. Consider an attack (Fig. 14) aiming at making the doctor Y read a wrong record {recID = "P3_rec", data = "Name: Tom; Disease: Flu, Mild; Treatment: Basic;"}, leading to incorrect diagnosis and treatment of the patient C, more seriously, losing C's life with no timely treatment! This compound attack might consist in blocking (Fig. 13), stealing and injection (Fig. 15). The attacker carries out the following steps: First, he blocks the doctor Y to read the record of the patient C; then, during the blocking time, he steals the record of C; next, according to the stolen one, he modifies the record and injects it back to the O net; finally, he cancels blocking, which allows the doctor Y to read an incorrect record.

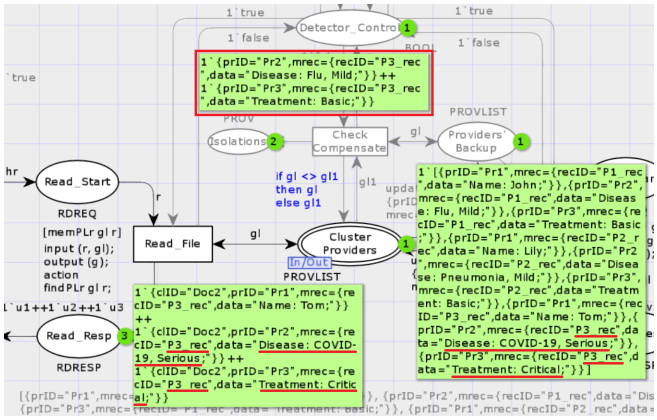

Fig. 17. Checking and compensation result

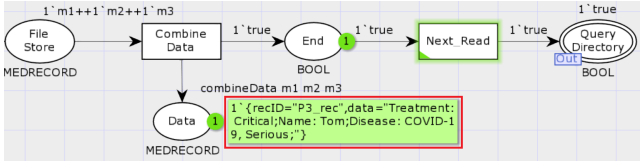

Fig. 18. Doctor Y reading result

Fig. 16, Fig. 17 and Fig. 18 present the result of our solution to the attack, where the doctor Y can read the correct record of the patient C. The green boxes show the current markings of corresponding places and the red lines highlight points for the following explanations. In Fig. 16, the attacker successfully modify C's record from "Disease: COVID-19, Serious; Treatment: Critical;" to "Disease: Flu, Mild; Treatment: Basic;". In Fig. 17, our detector found

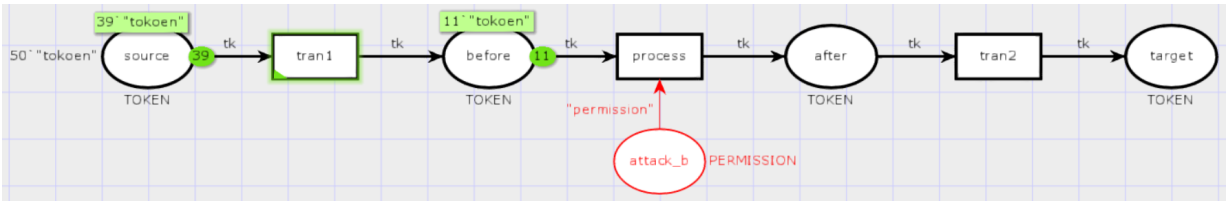

Fig. 19. Simple original net with a blocking attack

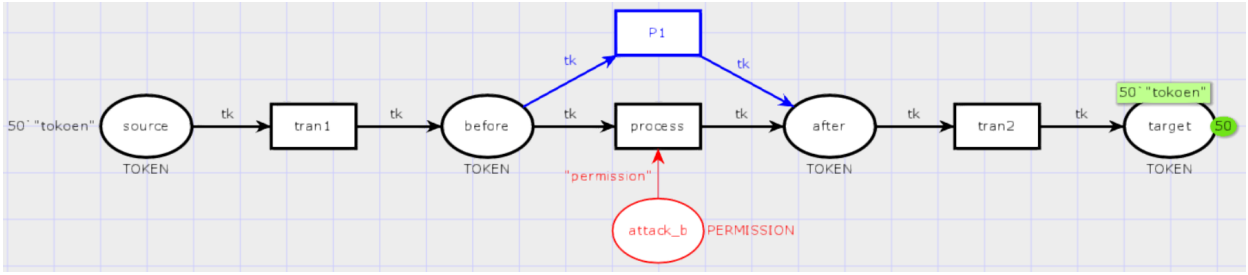

Fig. 20. Simple original net with a tolerance solution

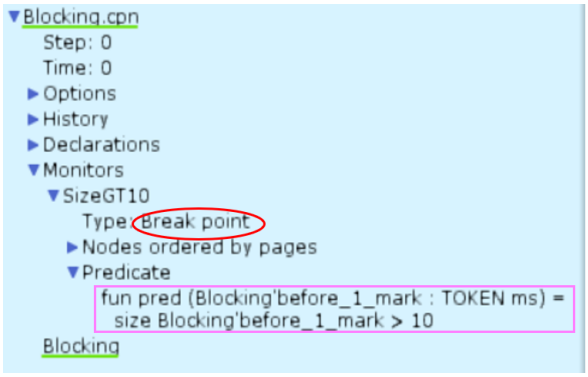

Fig. 21. Breakpoint monitor

```
921            <text tool="CPN Tools"
922                  version="4.0.1">tk</text>
923          </annot>
924        </arc>
925
926    <!--New Added: Bypassing Solution, Transition and Arcs-->
927
928        <trans id="ID1500000000"
929               explicit="false">
930          <posattr x="42.000000"
931                   y="336.000000"/>
932          <fillattr colour="White"
933                    pattern=""
934                    filled="false"/>
935          <lineattr colour="Blue"
936                    thick="3"
937                    type="solid"/>
```

Fig. 22. Adding a bypassing tolerance solution to the .cpn file

this change and replaced the wrong record with the correct one, with the cooperation of compensation solution shown in Fig. 15. Finally, according to Fig. 18, the doctor Y can read the correct record of C and decide a proper therapy. The composition of bypassing and compensation solutions can effectively tolerate the above attack. The current limitation is that the detectors and the tolerance solutions are not deployed automatically and can only be applied to models during the design stage instead of runtime.

A. One More Discussion: A Blocking Example

Now, for one step forward in the automation of our framework, we explore how to detect a blocking attack and deploy its tolerance solution by modifying the .cpn file, which is the format used by the tool CPN. Here, due to lack of space, we just show a very simple version of the bypassing solution by omitting some optimised control places and transitions.

As shown at Fig. 19, the blocking attack called “attack_b” leads to the accumulation of tokens in the place “before”. In order to detect the blocking and stop the simulation, the mechanism in this tool called “Breakpoint monitor” is adopted that can stop the simulation under some conditions [15]. In Fig. 21, we set a Breakpoint monitor for the place “before”, which will stop the simulation when the number of tokens in “before” exceeds 10. Once the blocking attack is detected, the simulation stops and the bypassing solution is deployed by modifying the corresponding .cpn file based on the information from the blocked transition. In the pink rectangle of Fig. 22, we add some .xml descriptions (i.e., one transition and two arcs of a bypassing path) to the original .cpn file. Fig. 20 presents the O net with a new added tolerance solution. As we can see, all the tokens arrive to “target” through the bypassing path. Thus the tolerance is successful.

This very useful step provides the possibility of implementing automatic tools for attack tolerance design. In the future, we will build two CPN libraries of basic attack-network interaction patterns and basic tolerance solutions, respectively. The goal is to automatically detect an attack, identify it (as a basic one from the library or a compound one made up of several basic ones), find the generic solution (from the library of basic solutions directly or by composition) and deploy this solution, i.e., inserting its code in the initial code at the proper locations. In this part, we only illustrated the idea with the simplest basic attack-network interaction pattern (i.e., blocking), and the insertion of the code is currently done manually but can be implemented as a automatical one.

V. Related Work

A. Modelling and Analysis of Web Services with CPNs

Coloured Petri Nets is a graphical language for constructing models of concurrent systems and analysing their properties [6], [11], [16], and has been widely used to model and analyse Web services [17]–[24].

In [17], a modelling style for representing interaction flows in Web interfaces, called wiCPN (Web Interaction Modeling Using Coloured Petri Nets), was explored, which is able to refine the model components and represent interaction flows. In [18], a timed CPN model was presented to evaluate the service composition in multicloud environments while minimising the number of clouds involved in serving a composite service request. In [19], taking Fuzzy CPNs as the automatic combination technology for OWL-S Web Services in Supercomputing Cloud Platform, algorithms were designed to build and simplify the Fuzzy CPN dependency relation graph. Model-to-model transformation technologies are exploited to convert other models to CPNs since, after transformation, existing tools can provide powerful support for analysis and verification of CPNs. In [20], a UML profile for the publish/subscribe paradigm was proposed, together with a model-to-model transformation from UML to CPNs, and, with the help of CPNs Tools, design errors cloud be detected and fixed. In [21], a reliability markup language for DRBD (dynamic reliability block diagram) models was presented, as well as an algorithm that automatically converts DRBD model to CPN, and a case study for verification of DRBD model was illustrated with CPN Tools. Specially, some CPN based diagnosis analysis approaches were also proposed [22]–[24], e.g., in [22], the authors used spiking neural P systems with coloured spikes to model the fault of available service, component, and connector in the service composition for fault location and handling. The above methods efficiently deal with different aspects of Web services, including service composition modelling, model-to-model transformation and fault diagnosis. With a different motivation, we try to take one of the first steps of applying CPNs to attack tolerance.

B. Attack Tolerance for Web Services in Cloud

Attack tolerance turns out to be extremely important to improve reliability and security of cloud services. Many projects are devoted to dealing with this problem [25]–[27]. A representative one is the H2020 CLARUS project which summarised the main cloud vulnerabilities and the solutions proposed, including intrusion detection and attack-tolerance, and then, by leveraging diversity [28], further contributed to this field by several novel and effective attack-tolerant approaches. In [2], an attack-tolerant architecture and framework for Web services was proposed, where variants in the attack-tolerant library substitute the original implementation periodically or according to online monitoring. In [3], a state of the art of attack tolerance was presented, and how web services can be tolerant through diversification was explained. In [29] software diversity was explored as a defense against side-channel attacks by dynamically and systematically randomising the control flow of programs, which reduces side-channel information leakage significantly.

Software Reflection is also an efficient technique for attack tolerance. In [4], a new approach was proposed, exploring its usage as a mean of detection and mitigation of insider attacks. The authors of [5] considered any application deployed in the cloud as a choreography of services and extended a formal framework for choreography verification by incorporating detection and remediation strategies using Software Reflection. These provide good solutions for attack tolerance. However, our goal is different. We want to design an abstract generic framework for detection and tolerance of attacks, which it will be possible to instantiate and (more or less automatically) deploy for a particular implementation.

VI. Conclusion

In this paper, we have proposed a model-based framework for attack tolerance using CPNs. We have extracted four meta attack operations modelled as basic attack-network interaction patterns and proposed the corresponding detectors and their tolerance solutions. To further validate our method and illustrate the way towards its automation, a case study was simulated with CPN Tools. Note that it is the first step to understand attack tolerance by modelling with CPNs and its extensibility in practice remains to explore. Our patterns of attack and tolerance are general, and may be instantiated using other formal methods that can be translated from/to PNs, e.g., membrane systems [30], [31].

Future work includes (1) investigating composition rules for the library of generic basic behaviours of attack-network interaction to interpret more complex attack patterns, (2) designing automatic tools for identifying such complex attacks, (3) building corresponding tolerance solutions by composition from the library of generic basic tolerance solutions, (4) deploying these instantiated solutions at the proper locations (which have to be previously identified) of the initial net. In addition, we would explore how to use the net invariants to detect and locate attacks, and more generally the analysis methods provided by Petri nets to prove some functional properties about the attacked net equipped with the deployed tolerance solution w.r.t. the original net (such as reachability, trace inclusion, equivalence, etc.), as well as model-based diagnosis combining previous work [32], [33]. In addition, generating formal models (e.g., under our framework) from the basic threat models (e.g., the STRIDE framework [34]), simulating actual complex attacks and conducting more experiments is part of our perspectives, including exploring the use of this approach for users or developers to identify and mitigate threats from the perspective of experiments, as well as validating further recurring attacks and corresponding solutions [35].

Acknowledgment

This work is supported by the China Scholarship Council, the Graduate Innovation Fund of Jilin University under

Grant No. 101832018C025 and the Natural Science Research Foundation of Jilin Province of China under Grant No. 20180101053JC.

REFERENCES

[1] R. Constable, M. B. Mark, and V. R. Robbert, "Investigating correct-by-construction attack-tolerant systems," Department of Computer Science, Cornell University, Tech. Rep., Mar. 2011.

[2] G. Ouffoué, F. Zaïdi, A. R. Cavalli, and M. Lallali, "An attack-tolerant framework for web services," in *2017 IEEE International Conference on Services Computing (SCC)*, Honolulu, HI, USA, Jun. 2017, pp. 503–506.

[3] ——, "How web services can be tolerant to intruders through diversification," in *2017 IEEE International Conference on Web Services (ICWS)*, Honolulu, HI, USA, Jun. 2017, pp. 436–443.

[4] A. R. Cavalli, A. M. Ortiz, G. Ouffoué, C. A. Sanchez, and F. Zaïdi, "Design of a secure shield for internet and web-based services using software reflection," in *International Conference on Web Services*, Seattle, WA, USA, Jun. 2018, pp. 472–486.

[5] G. Ouffoué, F. Zaïdi, and A. R. Cavalli, "Attack tolerance for services-based applications in the cloud," in *IFIP International Conference on Testing Software and Systems*, Paris, France, Oct. 2019, pp. 242–258.

[6] K. Jensen and L. M. Kristensen, "Colored Petri Nets: a graphical language for formal modeling and validation of concurrent systems," *Communications of the ACM*, vol. 58, no. 6, pp. 61–70, 2015.

[7] S. Jaidka, S. Reeves, and J. Bowen, "A coloured petri net approach to model and analyze safety-critical interactive systems," in *26th Asia-Pacific Software Engineering Conference*, Putrajaya, Malaysia, Dec. 2019, pp. 347–354.

[8] R. Liu, J. G. Delgado-Frias, D. Boyce, Y. Qian, and R. Khanna, "Online firmware functional validation scheme using colored petri net model," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 7, pp. 1532–1545, 2019.

[9] D. A. Zaitsev, T. R. Shmeleva, W. Retschitzegger, and B. Pröll, "Security of grid structures under disguised traffic attacks," *Cluster Computing*, vol. 19, no. 3, pp. 1183–1200, 2016.

[10] H. B. Attia, L. Kahloul, S. Benhazrallah, and S. Bourekkache, "Using hierarchical timed coloured petri nets in the formal study of TRBAC security policies," *International Journal of Information Security*, vol. 19, no. 2, pp. 163–187, 2020.

[11] K. Jensen and L. M. Kristensen, *Coloured Petri Nets: modelling and validation of concurrent systems*. Heidelberg, Berlin, Germany: Springer, 2009.

[12] K. Yamalidou, J. Moody, M. Lemmon, and P. Antsaklis, "Feedback control of petri nets based on place invariants," *Automatica*, vol. 32, no. 1, pp. 15–28, 1996.

[13] D. F. Fitch and H. Xu, "A petri net model for secure and fault-tolerant cloud-based information storage," in *24th International Conference on Software Engineering and Knowledge Engineering*, Redwood City, San Francisco Bay, USA, Jul. 2012, pp. 333–339.

[14] ——, "A raid-based secure and fault-tolerant model for cloud information storage," *International Journal of Software Engineering and Knowledge Engineering*, vol. 23, no. 5, pp. 627–654, 2013.

[15] CPN Tools: A tool for editing, simulating, and analyzing Colored Petri Nets. [Online]. Available: http://cpntools.org/

[17] T. Brant-Ribeiro, R. D. Araujo, I. Mendonça, M. S. Soares, and R. G. Cattelan, "Interactive web interfaces modeling, simulation and analysis using Colored Petri Nets," *Software and Systems Modeling*, vol. 18, no. 1, pp. 721–737, 2019.

[16] K. Jensen, L. M. Kristensen, and L. Wells, "Coloured Petri Nets and CPN tools for modelling and validation of concurrent systems," *International Journal on Software Tools for Technology Transfe*, vol. 9, no. 3-4, pp. 213–254, 2007.

[18] R. Entezari-Maleki, S. E. Etesami, N. Ghorbani, A. A. Niaki, L. Sousa, and A. Movaghar, "Modeling and evaluation of service composition in commercial multiclouds using timed colored Petri nets," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, no. 3, pp. 947–961, 2020.

[19] Z. Deng, J. Zhang, and T. He, "Automatic combination technology of fuzzy CPN for OWL-S web services in supercomputing cloud platform," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 31, no. 7, pp. 1 759 010:1–1 759 010:27, 2017.

[20] A. Gómez, R. J. Rodríguez, M. Cambronero, and V. Valero, "Profiling the publish/subscribe paradigm for automated analysis using colored Petri nets," *Software and Systems Modeling*, vol. 18, no. 5, pp. 2973–3003, 2019.

[21] R. Robidoux, H. Xu, L. Xing, and M. Zhou, "Automated modeling of dynamic reliability block diagrams using colored petri nets," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 40, no. 2, pp. 337–351, 2010.

[22] S. Pang, M. Wang, S. Qiao, X. Wang, and H. Chen, "Fault diagnosis for service composition by spiking neural p systems with colored spikes," *Chinese Journal of Electronics*, vol. 28, no. 5, pp. 1033–1040, 2019.

[23] G. P. Bhandari and Ratneshwer, "Dependency-based fault diagnosis approach for SOA-based systems using Colored Petri Nets," *Journal of King Saud University-Computer and Information Sciences*, pp. 2973–3003, 2018. [Online]. Available: https://doi.org/10.1016/j.jksuci.2018.12.002

[24] G. P. Bhandari, G. Ratneshwer, and S. K. Upadhyay, "Colored Petri nets based fault diagnosis in service oriented architecture," *International Journal of Web Services Research*, vol. 15, no. 4, pp. 1–28, 2018.

[25] MUSA H2020 project. [Online]. Available: http://musa-project.eu/

[26] Seaclouds project. [Online]. Available: http://www.seaclouds-project.eu/

[27] CloudWATCH2. [Online]. Available: https://www.cloudwatchhub.eu/

[28] B. Baudry and M. Monperrus, "The multiple facets of software diversity: Recent developments in year 2000 and beyond," *ACM Computing Surveys*, vol. 48, no. 1, pp. 16:1–16:26, 2015.

[29] S. Crane, A. Homescu, S. Brunthaler, P. Larsen, and M. Franz, "Thwarting cache side-channel attacks through dynamic software diversity," in *22nd Annual Network and Distributed System Security Symposium*, San Diego, California, USA, Feb. 2015, pp. 1–14.

[30] J. Kleijn, M. Koutny, and G. Rozenberg, "Towards a Petri net semantics for membrane systems," in *International Workshop on Membrane Computing*, Vienna, Austria, Jul. 2005, pp. 292–309.

[31] F. Liu and M. Heiner, "Modeling membrane systems using colored stochastic Petri nets," *Natural Computing*, vol. 12, no. 4, pp. 617–629, 2013.

[32] Y. Li, T. Melliti, and P. Dague, "Modeling BPEL web services for diagnosis: Towards self-healing web services," in *3rd International Conference on Web Information Systems and Technologies - Internet Technology*, Barcelona, Spain, Mar. 2007, pp. 134–140.

[33] Y. Li and O. Boucelma, "A CPN provenance model of workflow: towards diagnosis in the cloud," in *15th East-European Conference on Advances in Databases and Information Systems*, Vienna, Austria, Sep. 2011, pp. 55–64.

[34] STRIDE chart. [Online]. Available: https://www.microsoft.com/security/blog/2007/09/11/stride-chart/

[35] J. Dong, T. Peng, and Y. Zhao, "Automated verification of security pattern compositions," *Information and Software Technology*, vol. 52, no. 3, pp. 274–295, 2010.