HAL
open science

Collaborative Remediation of Configuration Vulnerabilities in Autonomic Networks and Systems

Martín Barrère, Rémi Badonnel, Olivier Festor

To cite this version:

Martín Barrère, Rémi Badonnel, Olivier Festor. Collaborative Remediation of Configuration Vulnerabilities in Autonomic Networks and Systems. 8th IEEE International Conference on Network and Service Management (CNSM'12), IEEE, Oct 2012, Las Vegas, United States. hal-00747646

HAL Id: hal-00747646

https://inria.hal.science/hal-00747646v1

Submitted on 2 Nov 2012

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.

Collaborative Remediation of Configuration Vulnerabilities in Autonomic Networks and Systems

Martín Barrère, Rémi Badonnel and Olivier Festor

INRIA Nancy Grand Est - LORIA, France
Email: {barrere, badonnel, festor}@inria.fr

***Abstract*—Autonomic computing has become an important paradigm for dealing with large scale network management. However, changes operated by administrators and self-governed entities may generate vulnerable configurations increasing the exposure to security attacks. In this paper, we propose a novel approach for supporting collaborative treatments in order to remediate known security vulnerabilities in autonomic networks and systems. We put forward a mathematical formulation of vulnerability treatments as well as an XCCDF-based language for specifying them in a machine-readable manner. We describe a collaborative framework for performing these treatments taking advantage of optimized algorithms, and evaluate its performance in order to show the feasibility of our solution.**

I. Introduction

Autonomic computing contributes to address the growing complexity of network management by defining a strong basis for automated systems capable of managing themselves in an autonomous manner [17], [16]. Nevertheless, when self-management operations are performed in order to obey high-level policies, operated changes may lead to vulnerable states increasing the exposure of the environment. In addition, human errors occur when systems administration tasks are executed, therefore vulnerability management activities, namely, identification, classification and remediation of vulnerabilities, are required in order to ensure safe configurations. We have already shown in [9] that it is possible to specify and assess distributed vulnerabilities involving several devices simultaneously, and also covering device-based vulnerabilities as a particular case. In this paper we propose a collaborative strategy for remediating both distributed and device-based vulnerabilities.

The scenario presented in Fig. 1 shows a typical example [23] where two devices, a SIP[1] server with no flooding protection and a local DNS[2] server with external unknown name resolution, constitute a distributed vulnerable state. In this situation, an attacker can perform a denial of service attack by flooding the SIP server with unresolvable domains that must be solved by a local DNS server. The local DNS server in turn is configured for solving unknown domains querying external servers, thus increasing the number of waiting requests as well as the response time for each SIP request. If at least one of the servers is not present or is not compliant with the required specific state, the distributed vulnerability has no place in the environment. In order to correct such security problem, different remediation tasks could be performed in the SIP server or in the DNS server. A key challenge is to define a strategy for determining how and by which devices the distributed vulnerability can be remediated.

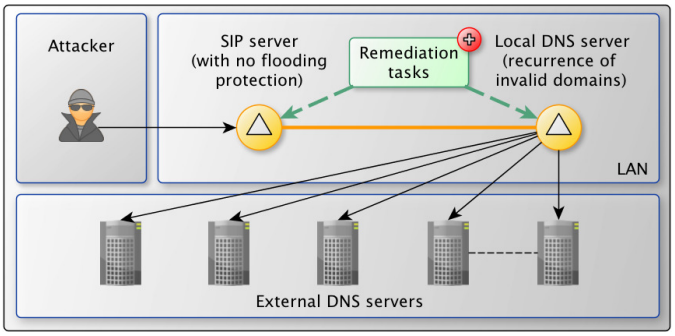

Fig. 1: Distributed vulnerability scenario [9]

We therefore introduce in this paper a collaborative approach for describing and performing treatments of configuration vulnerabilities in autonomic networks and systems. These correction advisories are taken into account by our framework in order to remediate vulnerable states found across the network and thus, reducing the exposure to security attacks. Our main contributions are: (1) a mathematical approach and an XCCDF[3]-based specification language for describing configuration vulnerability treatments, (2) a deployable infrastructure based on the Cfengine system [1], capable of enforcing the application and reporting of configuration vulnerability treatments as security policies, (3) optimized algorithms and their evaluation for performing remediation activities in a collaborative manner over the network.

The remainder of this paper is organized as follows. Section II describes existing work and their limits. Section III presents the proposed approach for modeling configuration vulnerability treatments in autonomic networks and systems. Section IV introduces DXCCDF, a language for specifying distributed vulnerability treatments. The architecture of our framework as well as the proposed remediation algorithms are described in Section V. Section VI provides an evaluation of our solution through a comprehensive set of experiments. Section VII presents conclusions and future work.

[1]Session Initiation Protocol
[2]Domain Name System

[3]eXtensible Configuration Checklist Description Format [24]

978-3-901882-48-7 © 2012 IFIP

II. Related work

Vulnerability management constitutes a crucial activity for maintaining safe configurations in autonomic networks and systems. Commonly, vulnerabilities can take the form of software flaws or misconfiguration errors and they can be usually corrected by means of different methods such as applying software patches, adjusting configuration settings or removing the affected software [18]. However, when corrective actions are performed changes are introduced in the environment thus change management mechanisms such as those proposed in [13], [14] must be taken into account. Risk assessment methods are also important as they provide a strong basis for analyzing the impact of remediation activities within the vulnerability treatment process [20], [19], [22]. It is crucial to ensure safe changes not only from an operational viewpoint but from a security perspective too. While several works have been focused on vulnerability management such as [21], just a few works address this topic into autonomic environments, mainly focused on the vulnerability assessment activity [9], [8], [12].

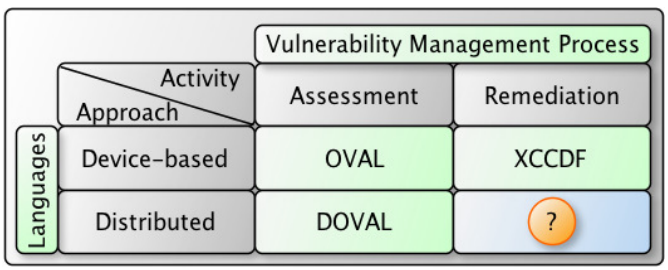

Fig. 2: Languages involved in the vulnerability management process

Several languages have been proposed for supporting the vulnerability management process as depicted in Fig. 2. De facto standards such as OVAL[4] [7] and XCCDF provide means for dealing with device-based vulnerabilities. OVAL is an XML-based language supported by MITRE Corporation [5] intended to standardize how to assess and report upon the machine state of computer systems, with a particular focus on vulnerability descriptions. The XCCDF language maintained by NIST [6] provides standard means for specifying security checklists under both technical and non-technical perspectives. In order to also cover distributed scenarios, we have proposed DOVAL[5] in our previous work [9], a language built on top of OVAL for describing and assessing distributed vulnerabilities. However, there is a lack for supporting collaborative treatments capable of mitigating and remediating distributed vulnerabilities. In this paper, we propose the DXCCDF[6] language, built on top of XCCDF, as a means for specifying distributed security treatments in a generalized manner for both distributed and device-based vulnerabilities.

III. Modeling vulnerability treatments

In this section we formalize configuration vulnerability treatments by extending and enhancing our previous mathematical model [9]. A distributed vulnerability is defined as a set of conditions over two or more network devices that if observed simultaneously, a potential threat is present on that network. It is important to remark that the required conditions to be observed over a specific device do not necessarily constitutes a complete device-based vulnerability description.

A. Describing distributed vulnerabilities

In order to specify a distributed vulnerability as well as the network over which this vulnerability will be assessed, the model defines a set of *core definitions* as follows:

- $H = \{h_1, h_2, \dots\}$ denotes the set of devices in the network (e.g. hosts, routers).
- $P = \{p_1, p_2, \dots\}$ denotes the set of device properties in the form of unary predicates $p_i(h), h \in H$.
- $S = \{s_1, s_2, \dots\}$ denotes the set of device states where a state s_i describes a set of properties required to be observed over a network device (called *role*). The set S is inductively defined as follows:
  - i if $p_i \in P$, then $p_i \in S \quad (i \in \mathbb{N})$
  - ii if $\alpha, \beta \in S$, then $(\alpha \diamond \beta) \in S \quad \diamond \in \{\wedge, \vee\}$
  - iii if $\alpha \in S$, then $(\neg\alpha) \in S$.
- $R = \{r_1, r_2, \dots\}$ denotes the set of relationships between network devices such as reachability and service provisioning. The relationships are modelled as n-ary predicates of the form $r_i(h_i, ..., h_j)$.

Based on the previous definitions, a distributed vulnerability DV is defined as the compliant projection of the pattern (P^H, P^R) over the network (H, R) where the constructs P^H and P^R are defined as follows:

- $P^H = \{s_1, ..., s_n\}$ denotes the set of machine states or roles required to be observed on specific network devices.
- $P^R = \{r_1, ..., r_v\}$ denotes the set of relationships between those devices matching the required roles.

Under a logical perspective, a compliant projection of the pattern (P^H, P^R) over the network (H, R) makes the following sentence to be true: $\exists(h_1, ..., h_n)(s_1(h_1) \wedge ... \wedge s_n(h_n) \wedge r_1(h_i, ..., h_j) \wedge ... \wedge r_v(h_k, ..., h_l))$. We specify the previous sentence in short by considering the predicate $DV(H, R)$ that expresses the evaluation of a distributed vulnerability DV based on the pattern (P^H, P^R) over a generic network (H, R). It is important to notice that the model allows to specify a device-based vulnerability by just defining $P^H = \{s_1\}$ and $P^R = \{\}$, or a sequence of vulnerabilities spread across the network by considering $P^H = \{s_1, ..., s_n\}$ and $P^R = \{\}$.

B. Specifying distributed treatments

Based on this modeling, we consider a distributed treatment DT as a body of tasks performed over a set of network devices that introduces configuration changes in order to eliminate the security weakness described by a specific distributed vulnerability DV. In order to formally define what a distributed treatment is, we extend the description model explained in Section III-A by defining the following domains:

[4]Open Vulnerability and Assessment Language
[5]Distributed OVAL
[6]Distributed XCCDF

- $A = \{a_1, a_2, \dots\}$ denotes the set of actions applicable over network devices.
- $T = \{t_1, t_2, \dots\}$ denotes the set of tasks applicable over network devices. A task t_i is a logical combination of actions and its logical value is computed based on the successful application of each action. The set T is inductively defined as follows:

  i if $a_i \in A$, then $a_i \in T \qquad (i \in \mathbb{N})$

  ii if $\alpha, \beta \in T$, then $(\alpha \diamond \beta) \in T \qquad \diamond \in \{\wedge, \vee\}$

In order to define the application of remediation tasks over the network, we specify the following set of *core functions*:

- $state_H : H \to S \equiv$ function that takes a device $h \in H$ as input and returns its current state $s \in S$.
- $state_R : R \to 2^S \equiv$ function that takes a network relationship $r \in R$ as input and returns a set with the current state $s_i \in S$ of each involved network device $h_i \in H$ in the relationship.
- $action : H \times A \to H \equiv$ function that takes a device $h \in H$ as input and returns the same device h after performing an action $a \in A$.
- $task_H : H \times T \to H \equiv$ function that takes a device $h \in H$ as input and returns the same device h after performing a task $t \in T$ that produces an observable change on its state. This means that at least one action $a_i \in A$ must introduce a change that cannot be rolled back by any other action in the task nor a combination of them. The following property holds in the considered model: $state_H(h) \neq state_H(task_H(h, t)), \quad \forall t \in T, \forall h \in H.$
- $task_R : R \times T \to R \equiv$ function that takes a network relationship $r \in R$ as input and returns the same network relationship r after performing a task $t \in T$ over its member devices. Based on the definition of T, is can be noticed that the task t will produce an observable change on its state and that the following property also holds: $state_R(r) \neq state_R(task_R(r, t)), \quad \forall t \in T, \forall r \in R.$
- $T^H = \{t_1^H, ..., T_n^H\}$ denotes the body of available tasks for performing over network devices where each task t_i^H is semantically related to a specific state s_i. Usually, the following equation can hold $|T^H| < |P^H|$, meaning that treatment tasks are not always available for correcting certain device states.
- $T_R = \{t_1^R, ..., t_v^R\}$ denotes the body of available tasks for performing over network relationships where each task t_i^R is semantically related to a specific relationship r_i. Usually, the following equation can hold $|T^R| < |P^R|$, meaning that treatment tasks are not always available for correcting certain network relationships.

We therefore define a distributed treatment DT as the compliant application of (T^H, T^R) over the network (H, R) that eliminates every possible matching projection of the pattern (P^H, P^R) over (H, R). Under a logical perspective, this is defined as the disjunction of task applications over each potential combination of devices and network relationships (H', R') performing the roles required by the distributed vulnerability DV as follows:

$DT(H, R) = \Pi(T^H, T^R) = task_H(h_1, t_1^H) \vee \dots \vee task_H(h_n, t_n^H) \vee task_R(r_1, t_1^R) \vee \dots \vee task_R(r_v, t_v^R)$

$\forall H' = \{h_1, ..., h_n\} \subseteq H, R' = \{r_1, ..., r_v\} \subseteq R$ such that $DV(H', R')$ holds.

Changes done for correcting different instances (H', R') of the distributed vulnerability must not shadow performed remediations for other observed vulnerable instances of DV, thus $\neg DV(H, R)$ must hold after the DT application.

IV. DXCCDF, A DISTRIBUTED VULNERABILITY REMEDIATION LANGUAGE

In order to capture the previous mathematical constructions, we have conceived the DXCCDF language, built on top of XCCDF, as a means for expressing vulnerability treatments in a machine-readable manner. XCCDF rules allow to specify remediation information that can be used by automated systems to perform corrective actions when specific states are detected. These states can be specified by languages such as OVAL and DOVAL. DXCCDF extends XCCDF by considering a new building block named *complex-Rule* under the *dxccdf* namespace. This extension provides the ability to specify a boolean expression involving all the potential tasks that can be performed for remediating a specific machine state. Fig. 3 depicts the mapping between the main components involved in the mathematical model and their representatives constructs within the DXCCDF language. An action a_i can be understood as a simple operation, i.e. a shell command, that is performed for changing a system property p_i. This property can be checked and remediated using an XCCDF rule. A task t_i is a combination of actions in the form of a boolean expression intended to correct a specific state s_i. Tasks are represented by means of DXCCDF complex rules. A distributed treatment DT is also represented using DXCCDF complex rules and they are finally put together into XCCDF groups.

The model and the DXCCDF language			
Model block	**Insight**	**Applies to**	**Expressed with**
Action a_i	chmod 644 passwd	Property p_i	XCCDF rule
Task t_i	$a_1 \vee (a_2 \wedge a_3)$	State s_i	DXCCDF complex rule
Distributed treatment DT	$t_1 \vee t_2 \vee \dots \vee t_k \vee \dots$	Distributed vulnerability	DXCCDF complex rule
Distributed treatments	$\{DT_1, DT_2, \dots\}$	$\{DV_1, DV_2, \dots\}$	XCCDF group of complex rules

Fig. 3: Mapping the model into the DXCCDF language

We now put forward an illustrative DXCCDF example illustrated in Listing 1 where a distributed treatment is specified in order to provide directives for remediating the distributed vulnerability previously depicted in Fig. 1. For the sake of clarity, we have omitted some XCCDF components that should be present in valid instances. Within this document, the group for vulnerability treatments is selected for evaluation containing only one treatment. The construct DXCCDF complex rule represents the distributed treatment itself. A DXCCDF complex rule allows to refer a check system for assessing

```
<cdf:Benchmark id="sip-dos-test-1" xmlns:dxccdf="..." xmlns:cdf="..."...>

  <cdf:title> DXCCDF example </cdf:title>
  <cdf:Group id="vulnerability-treatments" selected="1">
     <cdf:requires idref="dv1-treatment"/>
  </cdf:Group>

  <dxccdf:complex-Rule id="dv1-treatment" selected="1" check="1">
    <dxccdf:check system="http://doval.inria.fr/XMLSchema/doval">
      <dxccdf:check-content-ref href="dvDefns.xml" name="doval:inria:def:1"/>
    </dxccdf:check>
    <dxccdf:criteria operator="OR">
      <dxccdf:criteria operator="OR">
        <dxccdf:criterion idref="flooding-protection-yum" check="0"/>
        <dxccdf:criterion idref="flooding-protection-custom" check="0"/>
      </dxccdf:criteria>
      <dxccdf:criterion idref="stop-bind-daemon" check="0"/>
    </dxccdf:criteria>
  </dxccdf:complex-Rule>

  <cdf:Rule id="flooding-protection-yum" selected="1">
    <cdf:fix> yum install asterisk-sip-dos-patch </cdf:fix>
    <cdf:check system="http://oval.mitre.org/XMLSchema/oval">
      <cdf:check-content-ref href="sipDefns.xml" name="oval:mitre:def:1002"/>
    </cdf:check>
  </cdf:Rule>

  <cdf:Rule id="flooding-protection-custom" selected="1">
    <cdf:fix>
      wget http://doval.inria.fr/fixes/sip/asterisk-sip-dos-patch-r0.2.rpm
      rpm -Uvh asterisk-sip-dos-patch-r0.2.rpm
    </cdf:fix>
    <cdf:check system="http://oval.mitre.org/XMLSchema/oval">
      <cdf:check-content-ref href="sipDefns.xml" name="oval:mitre:def:1002"/>
    </cdf:check>
  </cdf:Rule>

  <cdf:Rule id="stop-bind-daemon" selected="1">
    <cdf:fix> service named stop </cdf:fix>
    <cdf:check system="http://oval.mitre.org/XMLSchema/oval">
      <cdf:check-content-ref href="unixDefns.xml" name="oval:mitre:def:2000"/>
    </cdf:check>
  </cdf:Rule>

</cdf:Benchmark>
```

Listing 1: DXCCDF vulnerability treatment

the distributed vulnerability under analysis by means of a DOVAL reference, and also to specify a logical criterion involving both XCCDF rules and DXCCDF complex rules. In the proposed scenario there exist two involved roles, the SIP server and the DNS server. Within the DXCCDF example, one remediation task has been defined for each role, namely, T^{SIP} and T^{DNS}. The task T^{SIP} is in turn a non-atomic task since two corrective tasks, namely, T^{SIP}_{yum} and T^{SIP}_{web}, can be alternatively performed. Thus, the distributed treatment expressed by the DXCCDF complex rule corresponds to the logical expression $(T^{SIP}_{yum} \vee T^{SIP}_{web} \vee T^{DNS})$ where each one of the referenced standard XCCDF rules are defined in the final part of the document. These XCCDF rules define the actual corrective actions to perform over the involved devices and they are orchestrated by the DXCCDF complex rule in order to remediate the specific distributed vulnerability.

V. A Framework for Collaboratively Treating Distributed Vulnerabilities

The DXCCDF language provides the capability of describing remediation activities that can be consumed by autonomic entities in order to secure the environment. In this section we propose a framework for supporting collaborative treatments specified in DXCCDF considering the main building blocks of our previous DOVAL-based assessment framework.

A. Architecture overview

In order to remediate a distributed vulnerability several tasks may be performed on different devices. At the moment of the analysis however, some of the involved devices may present particular states that do not allow them or make it more expensive to perform specific corrective actions than other involved devices. Factors such as availability, capability, or even policy consistency must be considered during the remediation process. We refer to this spectrum of factors as the *cost* of the node for performing a corrective task. Potential mechanisms for actually computing task costs are beyond the scope of this paper and they may involve several activities such as risk assessment and change management techniques. The main process for detecting and remediating distributed vulnerabilities considers these costs as illustrated in Fig. 4. Repositories of vulnerability descriptions as well as treatments specifications are available constituting the knowledge source of the network. At Step 1, a vulnerability description is consumed and assessed over the network as explained in [9]. If there is one or more pattern matching instances over the network, a treatment analysis is launched at Step 2.1 and its corresponding distributed treatment is consumed from the treatments repository at Step 2.2. Based on the available tasks for correcting the security vulnerability, devices are analyzed across the network in order to find a node for performing a

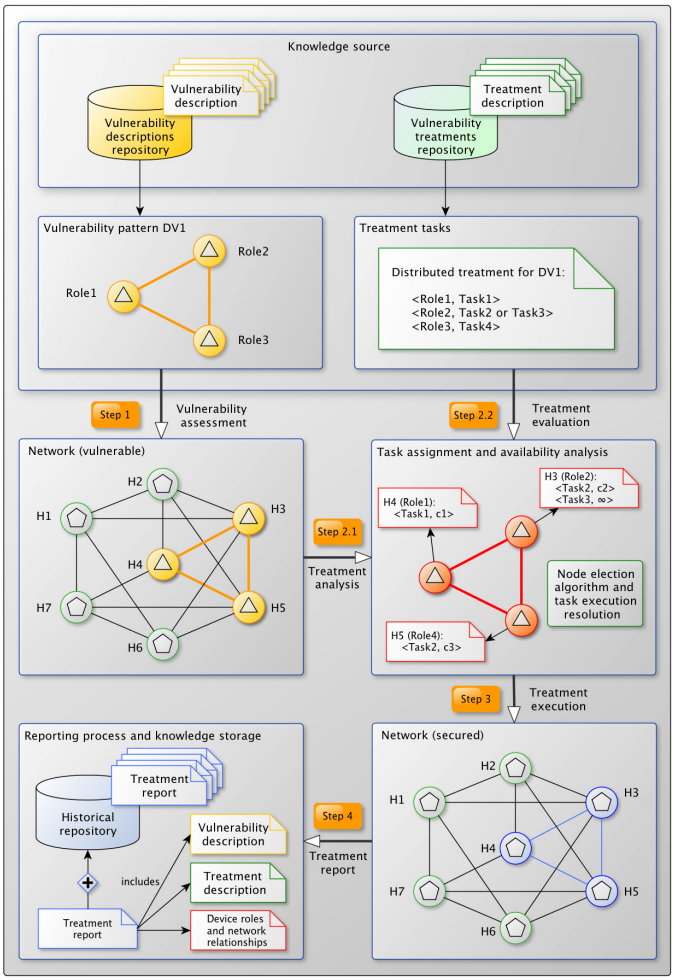

Fig. 4: Collaborative treatment - High level operation

remediation task. Once the treatment execution has been done and the network has been secured at Step 3, a treatment report is generated at Step 4 involving the vulnerability description, the treatment description used for remediating the vulnerability as well as the information gathered from the network in order to perform the corrective activity. Generated reports are stored in a historical database providing the ability to consider past experiences in future treatments.

B. Assessment and treatment strategies

We consider in our approach that treatment descriptions may be available later in time after vulnerabilities have been discovered. In light of this, we put forward a treatment strategy that can be applied after performing assessment activities and that can also be integrated to the assessment process if treatments are available at that time.

In order to present the proposed treatment strategy, we introduce a situation where the distributed vulnerability described in Fig. 1 is present in the form of many instances. Within the example illustrated in Fig. 5, a network with a Cfengine server and five devices is considered. Devices $h1$, $h2$ and $h3$ are involved in the instances of the distributed vulnerability described by a DOVAL document. Once the assessment has been performed at Step 1, a DOVAL report identifying detected vulnerability instances is generated. In order to collect required information, a spanning tree is generated at Step 2 by means of Cfengine's functionalities [10] as used in [9]. The Cfengine server traverses the network starting at the root of the spanning tree where each node will inform on how it can collaborate in the corrective tasks and at what cost, as depicted

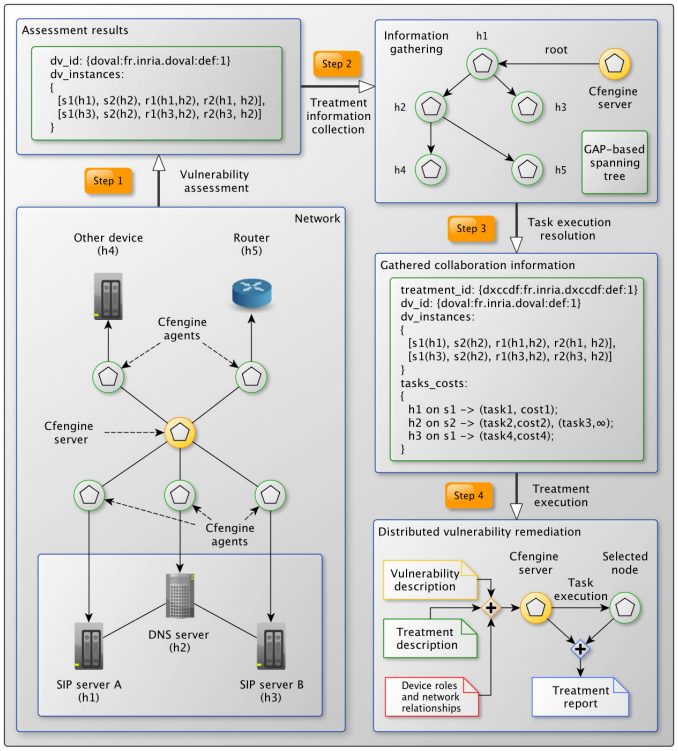

Fig. 5: Vulnerability treatment scenario

in Algorithm 1 that we explain later. At Step 3, a report is generated including the cost of each node for correcting the roles that each one is performing. Based on this information, the Cfengine server selects a node for applying corrective actions at Step 4 and a report indicating the results of the remediation activity is generated.

Input: TreeNode h, DOVAL document dv, DXCCDF document dt, CostTable ct.
Output: Table with costs for each node on each role.

1 **if** *currentNode h is alive* **then**
2 $\quad roleList \leftarrow findRolesForDevice(h, dv)$;
3 $\quad$ **foreach** *role* $s \in roleList$ **do**
4 $\quad\quad taskList \leftarrow findTasksForRole(s, dt)$;
5 $\quad\quad$ **foreach** *task* $t \in taskList$ **do**
6 $\quad\quad\quad cost \leftarrow queryNodeForTaskCost(h, t)$;
7 $\quad\quad\quad updateCostTable(ct, h, s, cost)$;
8 $\quad\quad$ **end**
9 $\quad$ **end**
10 $\quad gatherTasksCosts(leftTreeNode(h), dv, dt, ct)$;
11 $\quad gatherTasksCosts(rigthTreeNode(h), dv, dt, ct)$;
12 **end**

Algorithm 1: gatherTasksCosts (recursive)

In the proposed approach, a spanning tree is built in order to explore the network. Algorithm 1 presents the activity performed at each active node of the tree (line 1) for gathering and analyzing devices information. Each node h reads the list of roles involved in the vulnerability instances given in the DOVAL document dv, identifies itself in the list (lines 2-3), and for each task t found in the DXCCDF document dt applicable on each specific role s that h plays (lines 4-5), the task cost is computed by the node itself and attached to the general cost table ct (lines 6-7). The traversal continues on the left and right sub-trees (lines 10-11) until the whole spanning tree has been explored. This strategy can be easily integrated if treatments are available during the assessment process. By computing task costs at the same time network devices are evaluated looking for specific states, the Cfengine server will have all the required information for deciding which nodes will execute remediation tasks. Already scheduled as future work, a decentralized distributed strategy would allow any agent to start a vulnerability treatment though distributed algorithms such as leader election algorithms [15], [11], are necessary for deciding which node will execute corrective tasks.

VI. Performance Evaluation

Within the proposed framework, the complexity of the process is dominated by the number of nodes in the network and Algorithm 1 is $O(n)$. This means that the treatment process can be coupled with the assessment process without increasing its growth rate. Given a distributed vulnerability, the total assessment time under both a sequential (T_S^A) and a parallel (T_P^A) approach has been mathematically defined in [9]. In order to evaluate the whole time process involving assessment and treatment activities ($A + T$), we have extended these definitions by considering two additional parameters, namely,

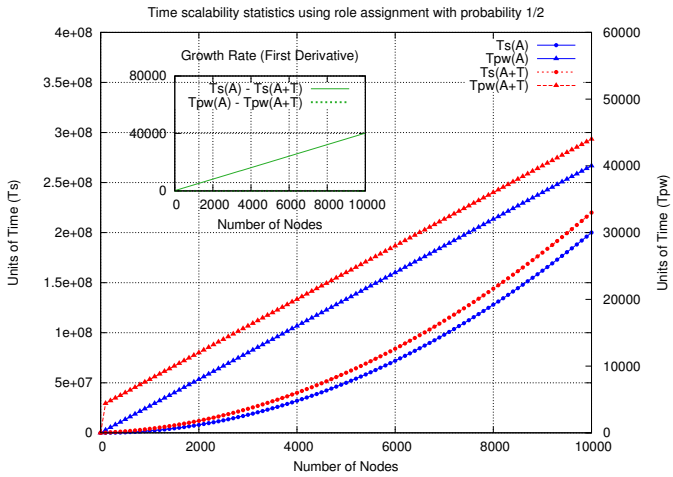

Fig. 6: Time statistics for vulnerability assessment and treatment activities

λ that denotes the average number of available tasks for each role s_j, and w denoting the average time for computing the cost of a single task. The new equations are defined as follows:

$$T_S^{A+T} = T_S^A + (n * k * P(A) * \lambda * w) \quad (1)$$

$$T_{Pw}^{A+T} = T_{Pw}^A + (k * \lambda * w) \quad (2)$$

Using a sequential approach (T_S^{A+T}), the total time is given by the time required for assessing the distributed vulnerability plus the time needed for each node (n nodes in the network) to assess its k potential roles. The probability for a node to play a certain role s_j is expressed by $P(A)$. For those roles present in the node, the number of tasks λ multiplied by the average time each task takes w is considered. Under a parallel approach (T_{Pw}^{A+T}) and considering the worst case, the total time is given by the time required for assessing the distributed vulnerability in the worst case plus the time needed for a node to compute the cost of every available task (λ tasks) for every role s_j considering an average time w for each task. In order to prove the scalability of the proposed approach, we have performed several analytical experiments that combine the assessment and treatment processes at the same time as shown in Fig. 6. Both solid blue lines with rounded and triangular points represent the time growth when the number of network devices is increased and only the assessment process is performed under a sequential ($T_S(A)$) and a parallel ($T_{Pw}(A)$) approach. Dashed red lines show the time behavior when assessment and treatment activities are performed at the same time ($A+T$). $T_S(A+T)$ illustrated by the dashed red line with rounded points shows the same growth rate than $T_S(A)$. The same phenomenon can be observed when a parallel approach is taken as depicted by $T_{Pw}(A)$ and $T_{Pw}(A+T)$. Within our experiments we have overvalued the parameter w in order to obtain a visible distance between curves and be able to notice the same growth behavior. First derivatives drawn in the inner graph confirms the same growth rates for both sequential (green solid line) and parallel (green dashed line) approaches, considering only the assessment (A), and both the assessment plus treatment ($A+T$) activities.

In order to provide a technical and deployable infrastructure for performing treatment activities, we are currently developing Scapalyzer, a Java-based [4] system capable of consuming OVAL and DOVAL vulnerability descriptions as well as XCCDF and DXCCDF treatment descriptions and producing the appropriate assessment and corrective Cfengine policy rules. Scapalyzer uses the services provided by our previous system [8], [9], and it is currently focused on the Cisco IOS platform [2] though more platforms can be also supported by means of its plugin-based architecture. Moreover, Scapalyzer uses the JAXB framework [3] for managing XML related issues, thus enabling our system to seamlessly evolve with new versions of vulnerability and remediation description languages by automatically regenerating its internal data model.

VII. Conclusions and Future Work

In this paper we propose a collaborative approach for integrating remediation descriptions into the management plane of autonomic networks and systems. We have put forward a mathematical model for defining vulnerability treatments that provides a robust foundation for its technical implantation. Indeed, an XML-based language called DXCCDF has been designed for capturing these mathematical constructs, providing the ability of describing remediation activities in a machine-readable manner. Our Cfengine-based framework takes into account these vulnerability treatment descriptions when unsafe vulnerable states are detected during the vulnerability assessment process in order to eradicate such security weaknesses. We have performed an analytical performance evaluation of our approach obtaining successful linear costs when it is integrated into the vulnerability management process. We have also introduced Scapalyzer, an under-development prototype implementation for translating DXXCDF remediation descriptions into Cfengine policy rules.

Treating and remediating vulnerabilities opens a wide spectrum of challenges that must be addressed in order to fully integrate the vulnerability management process into the management plane of self-governing environments. The proposed framework structures the remediation activity, however metrics are required for quantifying the costs of corrective remediation tasks with respect to device capacities, service dependencies and policy consistency. Sometimes the same corrective activity may solve several vulnerable states at once, therefore optimized strategies must be specified for minimizing these remediation costs. Finally, centralized approaches may generate bottleneck issues implying poor performance in certain situations. Decentralized distributed vulnerability management strategies are required for tackling this issue by balancing the workload in the network. However, the support algorithms pose hard challenges in terms of correctness and efficiency thus appropriate formal proofs should be considered as well.

Acknowledgements

This work was partially supported by the EU FP7 Univerself Project and the FI-WARE PPP.

REFERENCES

[1] Cfengine. http://www.cfengine.org/. Last visited on April 8, 2012.

[2] Cisco IOS. http://www.cisco.com/. Last visited on April 8, 2012.

[3] Java Architecture for XML Binding. http://java.sun.com/developer/technicalArticles/WebServices/jaxb/. Last visited on March 10, 2012.

[4] Java technology. http://www.oracle.com/technetwork/java/. Last visited on April 8, 2012.

[5] MITRE Corporation. http://www.mitre.org/. Last visited on March 10, 2012.

[6] NIST, National Institute of Standards and Technology. http://www.nist.gov/. Last visited on March 10, 2012.

[7] The OVAL Language. http://oval.mitre.org/. Last visited on March 10, 2012.

[8] M. Barrère, R. Badonnel, and O. Festor. Supporting Vulnerability Awareness in Autonomic Networks and Systems with OVAL. *Proceedings of the 7th IEEE International Conference on Network and Service Management (CNSM'11)*, October 2011.

[9] M. Barrère, R. Badonnel, and O. Festor. Towards the Assessment of Distributed Vulnerabilities in Autonomic Networks and Systems. *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS'12)*, April 2012.

[10] M. Burgess, M. Disney, and R. Stadler. Network Patterns in Cfengine and Scalable Data Aggregation. In *Proceedings of the 21st conference on Large Installation System Administration Conference*, pages 22:1–22:15, Berkeley, CA, USA, 2007. USENIX Association.

[11] M. Castillo, F. Farina, A. Cordoba, and J. Villadangos. A Modified O(n) Leader Election Algorithm for Complete Networks. In *Proceedings of the 15th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, PDP '07, pages 189–198, Washington, DC, USA, 2007. IEEE Computer Society.

[12] F. Chiang, J. Agbinya, and R. Braun. Risk and Vulnerability Assessment of Secure Autonomic Communication Networks. *The 2nd International Conference on Wireless Broadband and Ultra Wideband Communications (AusWireless 2007)*, (AusWireless):40–40, August 2007.

[13] M. Chiarini and A. Couch. Dynamic Dependencies and Performance Improvement. In *Proceedings of the 22nd conference on Large Installation System Administration Conference*, pages 9–21. USENIX, 2008.

[14] Y. Diao, A. Keller, S. Parekh, and V. V. Marinov. Predicting Labor Cost through IT Management Complexity Metrics. *Proceedings of the 10th IFIP/IEEE International Symposium on Integrated Network Management (IM'07)*, (1):274–283, May 2007.

[15] H. Garcia-Molina. Elections in a Distributed Computing System. *IEEE Trans. Comput.*, 31(1):48–59, January 1982.

[16] IBM. An Architectural Blueprint for Autonomic Computing. *IBM*, 2006.

[17] J.O. Kephart and D.M. Chess. The Vision of Autonomic Computing. *Computer*, 36(1):41–50, January 2003.

[18] P. Mell, T. Bergeron, and D. Henning. Creating a Patch and Vulnerability Management Program. *NIST*, November 2005.

[19] J. Sauve, R. Santos, R. Reboucas, A. Moura, and C. Bartolini. Change Priority Determination in IT Service Management Based on Risk Exposure. *IEEE Transactions on Network and Service Management*, 5(3):178–187, September 2008.

[20] T. Setzer, K. Bhattacharya, and H. Ludwig. Decision Support for Service Transition Management - Enforce Change Scheduling by Performing Change Risk and Business Impact Analysis. *Proceedings of the IEEE Network Operations and Management Symposium (NOMS'08)*, pages 200–207, April 2008.

[21] J. A. Wang and M. Guo. OVM: An Ontology for Vulnerability Management. In *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies (CSIIRW'09)*, pages 34:1–34:4, New York, NY, USA, 2009. ACM.

[22] J. A. Wickboldt, L. A. Bianchin, and R. C. Lunardi. Improving IT Change Management Processes with Automated Risk Assessment. *Proceedings of IEEE International Workshop on Distributed Systems: Operations and Management (DSOM'09)*, pages 71–84, 2009.

[23] G. Zhang, S. Ehlert, T. Magedanz, and D. Sisalem. Denial of Service Attack and Prevention on SIP VoIP Infrastructures using DNS Flooding. In *Proceedings of the 1st International Conference on Principles, Systems and Applications of IP Telecommunications (IPTComm'07)*, pages 57–66, New York, NY, USA, 2007. ACM.

[24] N. Ziring and S. D. Quinn. Specification for the Extensible Configuration Checklist Description Format (XCCDF). *NIST*, March 2012.