A SAT-based Autonomous Strategy for Security Vulnerability Management

Martín Barrère, Rémi Badonnel and Olivier Festor

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

***Abstract*—Computer and network systems are consistently exposed to security threats, making their management even more complex. The management of known vulnerabilities plays a crucial role for ensuring their safe configurations and preventing security attacks. However, this activity should not generate new vulnerable states. In this paper we present a novel approach for autonomously assessing and remediating vulnerabilities. We describe a detailed mathematical model that supports this activity and we formalize the remediation decision process as a SAT problem. We present a framework that is able to assess OVAL vulnerability descriptions and perform corrective actions by using XCCDF-based descriptions of future machine states and the NETCONF protocol. We also provide details of our implementation and evaluate its feasibility through a comprehensive set of experiments.**

I. Introduction

The growing development of computer systems as well as convoluted and heterogeneous networks dramatically increases the complexity of their management. Autonomic computing contributes to address the growing complexity of network management by defining a strong basis for automated systems to manage themselves in an autonomous manner [1]. In that context, it is critical to ensure their correct and secure configurations. Vulnerability management, defined as the practice of (I) identifying, (II) classifying, (III) remediating and mitigating vulnerabilities, is a key activity to achieve at this point [2]. Several factors can create a vulnerable system such as software errors, contradictory autonomic functions and improper human administration activities. In this paper, we aim at providing an autonomic approach for identifying and remediating vulnerabilities based on high-level policies. These policies describe vulnerable machine states and potential corrective activities to ensure safe configurations and prevent security attacks.

The vulnerability remediation activity constitutes itself as a hard and challenging task. From a proactive perspective, it should be able to decide which potential states could be dangerous for the security of the system. In the same manner, but under a reactive perspective, effective vulnerable states should be rapidly eradicated to avoid potential attacks that could compromise the system. Finding those changes that can ensure the security of the system is also a complex activity. One single change may impact or activate other vulnerable states that were not present before the change. The same effect could occur over other system policies, in this work however, we only deal with security configuration vulnerabilities. In that context, looking for correct changes that together can provide a safe system configuration is an explosive combinatorial activity classified as an NP-complete problem [3].

We propose in this paper a novel approach for autonomously securing and remediating known vulnerabilities, formalizing the change decision problem as a satisfiability or SAT problem [4]. Given a boolean expression, the SAT problem consists of finding an assignment for variables such that the formula evaluates to $true$. By specifying our vulnerability knowledge source as a propositional logical formula, we fix those system properties that we cannot change and free those variables for which changes are available. We use a SAT solving engine to determine which changes have to be made to secure the system. In order to provide proactive and reactive solutions, we propose the concept of a future state. This describes how a system will look after applying a specific change. These descriptions can be used for analyzing the security impact of changes without actually changing the system. When this information is not available, we use the NETCONF protocol [5] and its notion of candidate state where changes can be applied, analyzed and rolled back if necessary.

Our main contributions are: (1) a complete mathematical model for describing vulnerability assessment and corrective activities, (2) extensions to the XCCDF[1] and OVAL[2] standard languages to specify remediation actions and future states, (3) a robust architecture as well as a detailed vulnerability management strategy for autonomously assessing and remediating computer system vulnerabilities, and (4) a comprehensive experimental analysis of the SAT solving problem, technical representations and behavioral aspects of the NETCONF protocol over IOS Cisco devices.

The remainder of this paper is organized as follows. Section II describes existing work and their limits. Section III presents the mathematical model for analyzing and remediating vulnerabilities. Section IV depicts the use and extension of languages for specifying future states. Section V illustrates our framework describing its architecture and the strategy for performing assessment and remediation activities. Section VI describes our implementation prototype as well as an extensive set of experiments performed to validate our solution. Section VII presents conclusions and points out future work.

[1]eXtensible Configuration Checklist Description Format [6]
[2]Open Vulnerability and Assessment Language [7]

978-1-4799-0913-1/14/$31.00

II. Related work

Managing large-scale networks is a complex task. Both humans and automated entities make errors when configuring them, potentially increasing their own security exposure. Under this perspective, vulnerability management constitutes a crucial activity. The CVE[3] language [8], introduced by the MITRE Corporation [9], is an effort for standardizing the enumeration of known information security vulnerabilities. Nevertheless, it only provides means for informing about their existence and not for their assessment. In order to cope with these problems, MITRE has developed the OVAL language [7] as an effort to standardize the process by which the state of a computer system can be assessed and reported. OVAL is an XML-based language that allows the expression of specific machine states such as vulnerabilities, configuration settings, patch states. Real analysis is performed by OVAL interpreters such as Ovaldi [7] and XOvaldi [10]. In order to provide an automated and comprehensive security model, NIST [11] has introduced the SCAP[4] protocol [12]. The SCAP protocol includes the OVAL specification but also XCCDF [6] and CVSS[5] [13]. XCCDF is a language conceived as a means for bringing a system into compliance through the remediation of identified vulnerabilities or misconfigurations. CVSS on the other hand provides an open and standardized method for rating IT vulnerabilities.

These technologies have already been used for measuring network security risks [14]. The OVAL language has been utilized for performing vulnerability assessment activities in large scale networks [15], [16]. However, the vulnerability management process also involves remediation activities when vulnerabilities are found. Therefore, change management techniques are also required for ensuring coherent automated security processes [17], [18]. In this paper, we do not deal with attack graphs and multi-step attacks. However, insightful research work on network security assessment and remediation techniques using attack graphs has been previously reported in [19], [20]. Bayesian attack graphs have been also used for assisting administrators on mitigation plans [21]. Our approach is indeed complementary. While attack graphs represent all paths that allow an attacker to compromise a system, our work deals with finding corrective actions that take a system to a non-vulnerable state.

Under an autonomic perspective, automated techniques for assessing change associated risks as proposed in [22] and [23] are important because they provide a key support for the change management process, particularly for taking decisions about effective change implementations. Even though their analysis are usually focused on the operational impact rather than security concerns, such previous works highlight key challenges that must be taken into account when vulnerability management activities are performed.

[3]Common Vulnerabilities and Exposures
[4]Security Content Automation Protocol
[5]Common Vulnerability Scoring System

In order to deal with network management operations and changes, IETF [24] has developed NETCONF [5], a network configuration protocol that provides mechanisms to install, manipulate and delete the configuration of network devices. The NETCONF protocol specification is a standard, though its deployment, as well as complete vendors implementations, seem to be still in an early stage. However, very interesting works have already been presented showing evaluations of its maturity as well as diverse technical aspects [25], [26]. To the best of our knowledge, the integration of change management techniques into the vulnerability management plane constitutes a novel approach that may positively contribute to the overall security of current and future computer systems.

III. Vulnerability and Remediation Modeling

In this section we detail our mathematical model for performing vulnerability assessment and remediation activities. We also discuss the exponential nature of finding appropriate changes for securing a vulnerable system and we propose a SAT-based approach for dealing with this issue.

A. Describing vulnerabilities

Each time a security analysis is made, vulnerability descriptions are analyzed in order to detect security weaknesses on target devices. In this work, we use the OVAL standard language for describing vulnerabilities. Vulnerabilities are represented by OVAL definitions. Each OVAL definition logically combines OVAL tests that represent atomic property checks or evaluations over the target device. Each OVAL test can be referenced by different OVAL definitions. Each test contains an OVAL object that describes the component to be analyzed, and an OVAL state that describes the properties expected to be observed on the specified component. The test result will be $true$ if the component actually exhibits the specified state, and $false$ otherwise. These concepts are formally represented by the following definitions:

- $H = \{h_1, h_2, \dots\}$ denotes the set of devices or systems 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 $p_i(h), h \in H$. Such predicates are used for both specifying required properties to be observed for a vulnerability to be present as well as properties the device already possesses.
- $S = \{s_1, s_2, \dots\}$ denotes the set of device states where a state s_i is used for describing in a compact manner a set of properties required to be observed over a network device as well as for describing existing specific network devices states. The set S is inductively defined as follows:
  - i if $p_i \in P$, then $p_i \in S \qquad (i \in \mathbb{N})$
  - ii if $\alpha, \beta \in S$, then $(\alpha \diamond \beta) \in S \qquad \diamond \in \{\wedge, \vee\}$
  - iii if $\alpha \in S$, then $(\neg\alpha) \in S$.
- $state : H \rightarrow S \equiv$ function that takes a device $h \in H$ as input and returns its current state $s \in S$.

In the OVAL context, an OVAL definition representing a vulnerability v is actually a specific machine state $s \in S$

where the involved properties $p_i \in P$ are evaluated by OVAL tests. We therefore consider the set of known vulnerability descriptions constituting our knowledge source as $V = \{v_1, v_2, \ldots, v_m\}$. As each vulnerability $v_i \in V$ can be specified as a logical formula, we can describe the whole vulnerability dataset as a disjunction of formulas as follows:

$$\phi = v_1 \vee v_2 \ldots \vee v_m = \bigvee(v_i) \qquad v_i \in V \tag{1}$$

Considering this definition, we specify an evaluation function $\Phi : S \rightarrow Boolean$ that classifies a system $h \in H$ as vulnerable under V if and only if the assessment of ϕ over the state of h is $true$, i.e., $\Phi(state(h)) = true$. From a logical point of view, ϕ is a logical consequence of those formulas constituting the state of the device h, i.e., $state(h) \vDash \phi$.

B. Specifying corrective changes

In order to remediate configuration vulnerabilities, corrective changes must be performed on the target system. However, a change aimed at solving a specific vulnerability may introduce or activate other vulnerabilities. If this effect is not properly managed, this process would still expose the system to security threats. A system could be re-assessed after a change is made, and undo such modification if other vulnerabilities arise. Nevertheless, this approach does not take the big picture into account. Introduced vulnerabilities may also have potential fixes that would lead the system into a secure state. In that context, we consider the available information about known vulnerabilities and corrective tasks, as a whole. This potentially allows us to find a sequence of changes or fixes such that the final machine state is secure. Even though intermediate states in the sequence are vulnerable.

During the search of such sequence, changes could be applied on the target system and immediately assessed in a backtracking fashion. However, the ability to project the consequences of a vulnerability fix or change, i.e., how the affected part of the system will look after applying such change, allows us to analyze change sequences without actually changing the target system. This is one of the cornerstones of our approach. In order to formalize this concept, our remediation model involves the following definitions:

- $C = \{c_1, c_2, \ldots\}$ denotes the set of changes or corrective actions applicable over network devices.
- $change : H \times C \rightarrow H \equiv$ function that takes a device $h \in H$ as input and returns the same device h after performing a change $c \in C$ that produces an observable change on its state. The following property holds in the considered model: $state(h) \neq state(change(h, c)), \quad \forall h \in H, \forall c \in C$.
- $future : C \rightarrow S \equiv$ function that takes a change $c \in C$ as input and returns a state $s \in S$ that projects the affected characteristics of a system after applying the change c.
- $\Pi : S \times S \rightarrow S \equiv$ function that takes a projected system state $s_1 \in S$ and a machine system state $s_2 \in S$ as input and returns s_2 updated with the properties of s_1.

In order to analyze the impact of a change c_i over a system h, the $future$ and projection Π functions are combined with the Φ formula to check present vulnerabilities as follows:

$$\Phi(\Pi(future(c_i), state(h)) \qquad c_i \in C, h \in H \tag{2}$$

From a technical point of view, the $future$ function can be intuitively understood as observing the resulting state of a change over a rollback-capable system. However, this can be also achieved by considering a specification mechanism for describing those system properties that will be modified once the change is applied. Our approach considers both techniques, which are discussed in the following sections. In light of these definitions, we define a sequence of changes ω as follows:

$$\omega = c_1 \circ c_2 \circ \ldots \circ c_n \qquad c_i \in C \tag{3}$$

We say that ω constitutes a secure sequence of changes for a system $h \in H$ if and only if $\Phi(\omega(h)) = false$. Finding such a sequence for different system states and contexts constitutes an NP-complete problem that we aim at tackling in this paper.

C. Addressing complexity of change sequence analysis

Each time a single change is made to fix a specific vulnerability, some system properties are naturally modified and therefore, the state for the next corrective change in the sequence is modified. In that context, the order and the combination of distinct changes for each vulnerability induce several different possible combinations. This issue falls into a family of problems called NP-complete where no solution in polynomial time is known. Within our approach, we have encoded this change decision problem as a boolean satisfiability problem (SAT), which is known to be an NP-complete problem [3]. Indeed, each vulnerability is represented as a boolean formula that is directly taken from OVAL XML descriptions. The translation from XML into SAT is therefore linear. Vulnerability formulas are combined with ORs and hence, the system is vulnerable if there exists an assignment such that makes the whole formula *true*. Negated, it expresses that the system is secured (with respect to the known-vulnerability database). Then, we reduce the boolean formula to those properties for which change actions exist, and use a SAT solver to find an assignment. In this section, we first present an illustrative example that shows the exponential nature of finding a suitable change sequence ω and then we formalize our approach as a SAT problem.

Let $h \in H$ be a target device where $s \in S$ constitutes its current state as follows: $s = state(h) = \{p_1, p_2, p_3, p_4\}$, meaning that properties p_1, p_2, p_3 and p_4 are present on the system. Let us also consider a vulnerability database V and a vulnerability fix database C_V as follows:

$$\begin{aligned} V \equiv \{&v_1 = p_1 \wedge p_2 \wedge p_3, \;\; v_2 = (\neg p_1 \vee \neg p_2) \wedge p4, \\ &v_3 = \neg p_1 \wedge p_3 \wedge \neg p_4\} \qquad p_i \in P, v_i \in V \end{aligned} \tag{4}$$

$$\begin{aligned} C_V \equiv \{&c_{1a} \rightarrowtail \neg p_1, \; c_{1b} \rightarrowtail \neg p_2, \; c_2 \rightarrowtail \neg p_4\} \\ &p_i \in P, c_i \in C \end{aligned} \tag{5}$$

This example is based on three real Cisco IOS vulnerabilities identified by *CVE-2008-3812*, *CVE-2008-3798* and *CVE-2008-3821* respectively [27]. Within our model, properties are mapped to the following propositions:

- $\diamond$ $p_1 \equiv$ IOS firewall is enabled.
- $\diamond$ $p_2 \equiv$ Deep Packet Inspection (DPI) is enabled.
- $\diamond$ $p_3 \equiv$ HTTP server is enabled.
- $\diamond$ $p_4 \equiv$ SSL/TLS is enabled.

As explained before, vulnerabilities are represented as specific machine states that are specified by logical formulas. For instance, vulnerability v_1 requires p_1, p_2 and p_3 to be active for the vulnerability to be present. Within the set of available changes C_V, c_{1a} and c_{1b} are alternatives changes for fixing vulnerability v_1 while c_2 constitutes the only remediation action for vulnerability v_2. No fix action is available for vulnerability v_3. In this scenario, it can be observed that the vulnerability v_1 is a semantic consequence of the properties present in the system, which are compactly represented as s. This means that v_1 is logically $true$ under these hypothesis. This fact is represented by a node labeled v_1 in the graph illustrated in Fig. 1. Beginning at this node, a search for a secure change sequence is launched. Two alternative changes are available for fixing the vulnerability v_1. Change c_{1a} deactivates the property p_1, changing the system state to $s = \{\neg p_1, p_2, p_3, p_4\}$. Under these conditions, v_2 becomes present in the system. However, a fix for v_2 exists so the change c_2 is applied. Such modification brings the state of the system to $s = \{\neg p_1, p_2, p_3, \neg p_4\}$ activating the vulnerability v_3. As no remediation action is available for v_3, this change sequence is considered as invalid. Backtracking to the beginning, fix c_{1b} is applied activating again the vulnerability v_2. Once again, change c_2 is applied but this time, the combination of remediation actions leaves the system as $s = \{p_1, \neg p_2, p_3, \neg p_4\}$ successfully eradicating all vulnerabilities.

This example aims at showing the combinatorial nature of the problem, which leads to find other solutions rather than classic ones such as backtracking. Naive approaches for assessing all possible combinations cannot provide solutions in polynomial time. It is important to highlight that even though the whole process is described in sequence, we are actually interested in the set of detected applicable changes at the end of such process. SAT solutions indicate changes to be performed at once.

This problem constitutes a decision problem that relies on changes being applied to ensure a secure system state. In our model, we have a boolean expression ϕ that indicates the vulnerable nature of a system when it is evaluated as $true$. Considering $\psi = \neg\phi$, we can say that a target system h is secure, or not vulnerable, when ψ is $true$. Our problem therefore consists of finding such a propositional assignment that makes the ψ formula $true$. In computational complexity theory, this is known as a satisfiability or SAT problem. Given the current state of a target system, ψ can be instantiated and evaluated, and changes can be understood as actions that can assign a specific value to the properties involved in the formula. Considering the proposed example, the ψ formula states that none of the known vulnerabilities in V can occur:

$$\psi = \neg(p_1 \wedge p_2 \wedge p_3) \wedge \neg((\neg p_1 \vee \neg p_2) \wedge p_4) \wedge \neg(\neg p_1 \wedge p_3 \wedge \neg p_4) \quad (6)$$

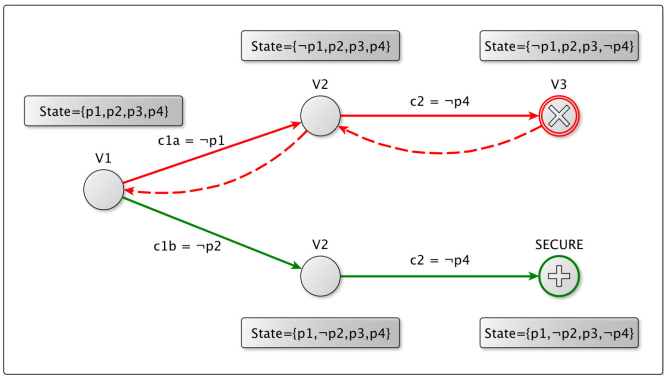

Fig. 1: Change sequence search example

Because we usually only know a small set of actions to remediate vulnerabilities, not every property is likely to be changed. In that context, we need to find solutions for ψ respecting those property values that are not changeable, i.e., there are no available actions for modifying their states. This in turn reduces the search space. Within our example, property p_3 is not changeable, and therefore, it must take its current system value, $true$, giving the following expression:

$$\psi = p_1 \wedge \neg p_2 \wedge \neg p_4 \quad (7)$$

The solution in this case is trivial. It states that our target system can be classified as secure only if those properties not matching the current state are changed, i.e., p_2 and p_4. Therefore, changes c_{1b} and c_2 must be applied, deactivating or updating the DPI engine as well as the SSL/TLS service. In the worst case, these changes could consist in deactivating or uninstalling the services themselves, nevertheless, the idea is that changes might generally patch or update to newer versions that do not present the characteristics involved in the vulnerabilities. The proposed example constitutes a simple scenario aimed at showing the insight of our general approach. However, versions can also be modeled as properties, thus enriching the expressiveness of our vulnerability descriptions and the accuracy of our solutions.

IV. The X2CCDF specification language

We have introduced the X2CCDF language, built on top of the XCCDF language [6], in order to express the future state of target systems after applying vulnerability remediation actions. In this section we present the core building blocks of X2CCDF and explain its use in the context of change analysis.

Specifying corrective changes for remediating known vulnerabilities in such a way machines can interpret them is crucial to achieve higher levels of security automation. The XCCDF language provides great support for this point by allowing referenced vulnerability descriptions expressed with the OVAL language and linking them to rules that can be applied to correct the specified security weaknesses. Nevertheless, applying changes blindly, without actually analyzing the impact of such changes, does not ensure a secure corrective process. In that context, we introduce the idea of future or post-action states. Future states are intended to describe how the system will look like after applying a specific change. They

do not describe the entire system but only the components affected by changes. In light of this and being designed for describing computer machine states, the OVAL language suitably fits for representing future machine states. Its interpretation however changes, i.e., in the general case, data is usually collected and compared against OVAL vulnerability descriptions [7]. Within our approach, collected data is mixed with OVAL-based future states descriptions and compared against OVAL vulnerability descriptions.

The ability to express this concept in a machine-readable manner provides new capabilities for analyzing different ways of modifying and correcting computer systems without actually changing them. The main objective in describing future states within X2CCDF is to complement management protocols such as NETCONF [5] by allowing the projection of changes using the current system state combined with future states of known remediation actions. While XCCDF provides means for specifying remediation actions when specific states are detected, X2CCDF extends its capabilities by specifying also the consequences of such actions when performed by means of the OVAL language.

Listing 1 presents an illustrative example where X2CCDF is used for specifying the two alternative actions for vulnerability v_1 as described in Section III-C. For the sake of clarity, we have omitted some XCCDF components that should be present in valid instances. Within this example, only one XCCDF group of management rules is defined (lines 3-5). The only referenced rule *v1-treatment* is declared below (lines 6-14). X2CCDF extends XCCDF by considering a new building block named *complex-Rule* under the *x2ccdf* namespace. This extension, structurally similar to the one proposed in [28] but semantically different, permits the specification of a boolean expression involving alternative

```
1. <cdf:Benchmark id="X2CCDF-test-1" xmlns:x2ccdf="..." xmlns:cdf="..."...>

2.  <cdf:title> X2CCDF example </cdf:title>
3.  <cdf:Group id="vulnerability-treatment-with-future-state" selected="1">
4.     <cdf:requires idref="v1-treatment"/>
5.  </cdf:Group>

6.  <x2ccdf:complex-Rule id="v1-treatment" selected="1" check="1">
7.    <x2ccdf:check system="http://oval.mitre.org/XMLSchema/oval">
8.      <x2ccdf:check-content-ref href="iosDefns.xml" name="oval:mitre:def:5302"
       />
9.    </x2ccdf:check>
10.   <x2ccdf:criteria operator="OR">
11.     <x2ccdf:criterion idref="v1-fix-1a" check="1"/>
12.     <x2ccdf:criterion idref="v1-fix-1b" check="1"/>
13.   </x2ccdf:criteria>
14. </x2ccdf:complex-Rule>

15. <cdf:Rule id="v1-fix-1a" selected="1">
16.   <cdf:fix> ./disableFirewall.sh </cdf:fix>
17.   <x2ccdf:check system="http://oval.mitre.org/XMLSchema/oval">
18.     <x2ccdf:future-content-ref href="iosFuture.xml" name="x2ccdf:inria:def:1
      "/>
19.   </x2ccdf:check>
20. </cdf:Rule>

21. <cdf:Rule id="v1-fix-1b" selected="1">
22.   <cdf:fix> ./disableDPIEngine.sh </cdf:fix>
23.   <x2ccdf:check system="http://oval.mitre.org/XMLSchema/oval">
24.     <x2ccdf:future-content-ref href="iosFuture.xml" name="x2ccdf:inria:def:2
      "/>
25.   </x2ccdf:check>
26. </cdf:Rule>

27. </cdf:Benchmark>
```

Listing 1: X2CCDF example

actions (lines 10-13) that can change different properties of a specific vulnerability (lines 7-9). Corrective changes from the model, c_{1a} (lines 15-20) and c_{1b} (lines 21-26), are described using standard XCCDF rules. However, the semantics of these rules express the future or post-action state of each change. While the particular actions to perform are specified inside the *fix* tag (lines 16 and 22 respectively), the *check* tag under the *x2ccdf* namespace serves as a semantic indicator for automated interpreters. It is a common practice to use scripts in XCCDF. It should be noticed that the example in Section III-C only deals with atomic changes and that is why the *OR* operator appears in Listing 1. However, the logical composition of changes constitutes an important issue to address that X2CCDF already supports using the *AND*, *OR* and *NOT* operators. This point has been already scheduled for future work.

In order to describe future or post-action states, we also use the OVAL language. However, its interpretation is different since we are comparing OVAL states against OVAL states, and not specific collected information (OVAL system characteristics) against OVAL states. The main idea is as follows. Each change is represented as a pair of OVAL object and OVAL state. The OVAL object represents the component over which the specific change is applied. The OVAL state represents the characteristics the object will present after applying this specific change. Finally, the impact of a change can be analyzed by looking for vulnerabilities which involve OVAL tests using the same OVAL object as the specified change. Affected OVAL tests are evaluated by comparing their OVAL states against the future OVAL-based state involved in the change. Other OVAL tests involved in the affected vulnerabilities are evaluated following the standard process, i.e., OVAL system characteristics against OVAL states.

While specifying the modifications that a change will have on a target system, the attributes inside an OVAL state are used to express the characteristics that the OVAL object will present. For instance, if a change is designed to change the version of a Cisco IOS system, the OVAL object will be the *version_object* while the *version_state* will contain the new version value, e.g., 12.4. Comparing instances of OVAL simple datatypes, such as integers and booleans, does not present difficulties. This can be done in the same way OVAL characteristics are compared against OVAL states. However, in future states, regular expressions can be utilized for specifying certain values inside information blocks that are a priori unknown. An example of this could be to look for a particular configuration line inside the running configuration file of a Cisco device. In that case, we need to potentially compare a regular expression against another regular expressions within OVAL states. According to the principles established in automata theory, the intersection of two regular languages e_1 and e_2 is a regular language e_3 [29]. Therefore, we can compute whether $e_1 \subseteq e_2$ by verifying if $e_1 = e_1 \cap e_2$. By operating over these regular expressions, we can say if a projected state might match the expressions present in the vulnerability descriptions. Within our experiments, we have used the Greenery tool to support these operations on regular expressions [30].

V. VMANS, AN AUTONOMOUS VULNERABILITY MANAGEMENT FRAMEWORK

VMANS is an autonomous framework designed for assessing and remediating configuration vulnerabilities over computer systems. In this section we explain its architecture as well as the underlying strategy in charge of orchestrating the overall vulnerability management process.

A. Architecture overview

The proposed architecture, illustrated in Fig. 2, comprises two independent processes, namely, one process for maintaining logical representations of OVAL vulnerabilities descriptions up-to-date, and a second process for performing vulnerability management activities. The first process is in charge of monitoring the OVAL vulnerability descriptions database (step I) and converting new vulnerability descriptions into equivalent boolean expressions when they become available (step II). Independently, a second process is in charge of dealing with vulnerabilities, which is orchestrated by the vulnerability manager component. At step 1, it communicates with the OVAL analyzer in order to launch the assessment process. The analyzer consumes OVAL vulnerability descriptions from the repository at step 2 and collects the required data from those devices under control at step 3. Once the assessment is performed, the analyzer sends the results back to the vulnerability manager. If the system is found to be vulnerable, the vulnerability manager analyzes the available remediation descriptions at step 4 and correlates them with the properties that can be changed in the target system. Considering the current system state and the available changeable properties, the SAT solver engine is used at step 5 to decide

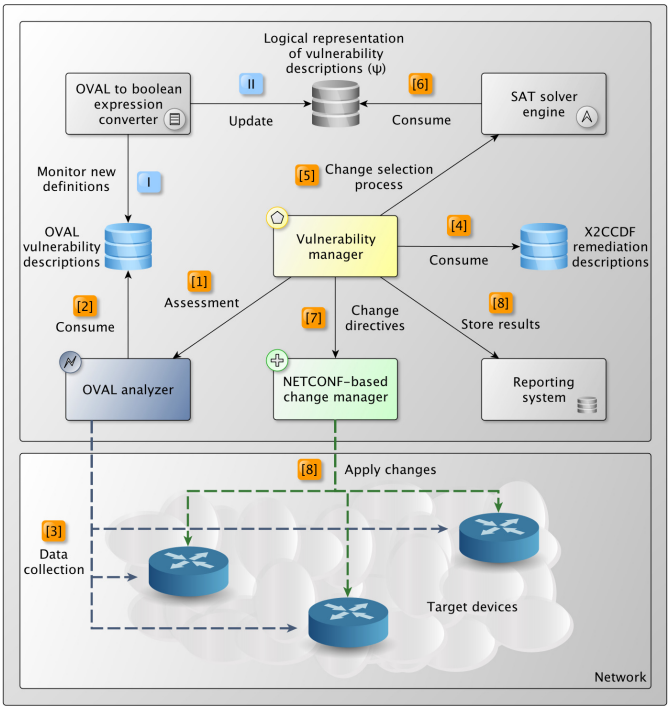

Fig. 2: VMANS high-level architecture

which changes must be applied in order to secure the system. At step 6, the SAT solver uses a logical representation ψ, such as the one illustrated in Eq. 7, specifying that none of the vulnerabilities can occur. A solution provided by the SAT solver indicates which properties must be changed in the system to present a secure state. The vulnerability manager interprets this information and sends specific directives to the NETCONF-based change manager subsystem at step 7 in order to effectuate these changes. Finally, the NETCONF protocol is used at step 8 to communicate and perform the specified changes on the target system.

B. Vulnerability management strategy

In order to autonomously deal with vulnerabilities, the proposed strategy illustrated in Fig. 3 is a closed control loop where three classes of events may potentially trigger vulnerability management activities. These activities can happen when new vulnerability or remediation descriptions become available and when the system presents changes that may compromise its security. In that context, an event monitoring component is in charge of observing these events and triggering the vulnerability management process when required. Once this process has been launched, vulnerabilities affected by the event that has triggered the process are computed (step 1). Depending on the case, these vulnerabilities can be recently added vulnerability descriptions, or vulnerabilities referenced by new remediation descriptions, or known vulnerabilities involving components that have been changed on the target system. Afterwards, these vulnerability descriptions are evaluated on the system (steps 2-4). If the system is not found to be vulnerable, the process ends and returns to the initial monitoring state. If it is vulnerable, then available remediation descriptions are consumed (step 5). If no remediation is available for treating the security issues, then a system warning is produced (step 6) and an analysis report is created and stored (step 7) ending the process and returning to the initial state. Otherwise, remediation descriptions are analyzed (step 8). In the case every remediation description provides a specification of the future state after applying the involved changes, the process continues with the change selection process (step 9). Considering the current properties present in the target system and the properties that can potentially change by applying the available vulnerability treatments, a SAT solver is used to provide a logical assignment of every property related to any vulnerability to ensure a secure system state. Once the changes have been identified, they are applied within a NETCONF session (steps 10-13) and an analysis report is generated (step 7) going back to the initial state. When a remediation description does not specify a future state, its impact is empirically evaluated by applying the involved changes on a candidate state of the target system (steps 14-19). To do so, the candidate state feature included in the NETCONF specification is considered. For each remediation description under these circumstances, involved changes are applied (step 17), modified properties are col-

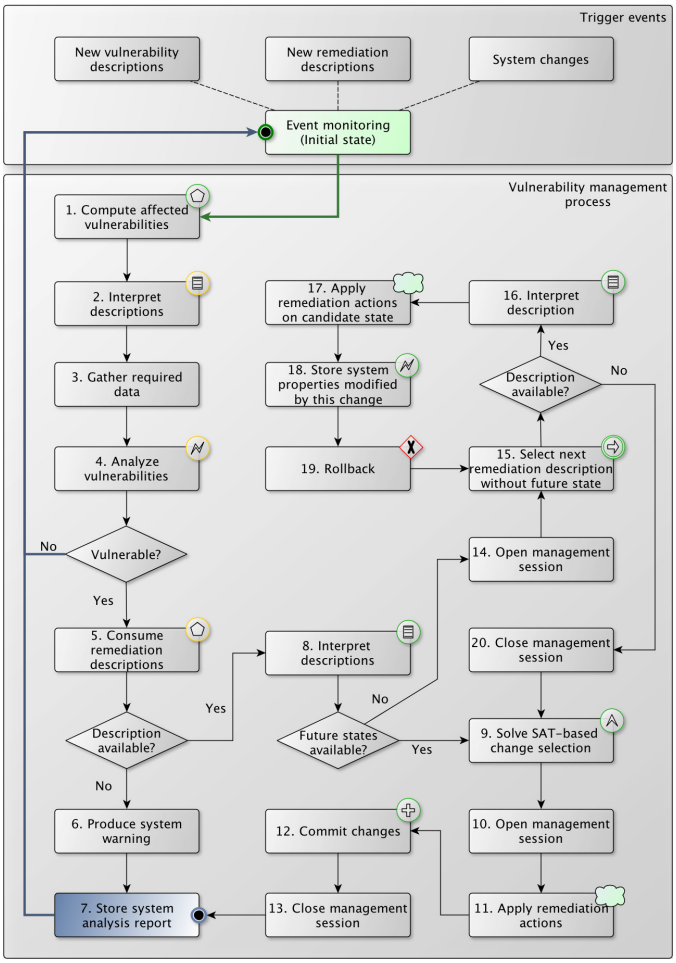

Fig. 3: VMANS control loop

lected and stored (step 18) and finally modifications are rolled back (step 19). When the loop is finished, the NETCONF session is closed (step 20) and the process continues normally with the change selection stage (step 9) as described before.

VI. Performance Evaluation

In order to provide a computable infrastructure to the proposed approach, we have developed an implementation prototype that integrates the building blocks presented in the VMANS framework. We have also performed a deep behavioral analysis using the Cisco IOS platform as a case study. In this section we detail the implementation prototype, the experiments and the obtained results.

All the architectural components described in Fig. 2 have been implemented in Java 1.6 SE. The OVAL analyzer is an extension of XOvaldi [10] for Cisco IOS. The model of Cisco routers used is c3725 with IOS version 12.4. They have been emulated using GNS3 [31] over a regular laptop (2 Ghz Intel Core i7 with 8GB RAM). OVAL vulnerability descriptions have been taken from the public OVAL repository [7]. We have used the SAT solver engine provided by the *Aima* project [32]. Operations between regular expressions for analyzing future states are performed with the Greenery tool [30]. We have used and slightly modified the Netconf4J project library [33] to communicate with Cisco routers via NETCONF.

In order to analyze the scalability of our framework, we have performed several experiments involving vulnerability representations as boolean expressions, SAT solving analysis time and behavioral aspects of the NETCONF protocol over Cisco. Our first experiment, illustrated in Fig. 4, shows the behavior of VMANS while dealing with vulnerability logical representations. In the general case, SAT solvers consume boolean expressions in conjunctive normal form (CNF). If the input formula is not in CNF, SAT solvers transform it internally. In that context, we have measured the time required to load standard logical representations into memory (red solid line) as well as their transformation to CNF (blue dashed line). We have repeated this measurement while varying the amount of vulnerability descriptions. When all the OVAL descriptions for IOS are considered (around 140), their representations are loaded in 53 milliseconds while their transformation to CNF takes 7.5 seconds approximately. We have observed a stable behavior for both activities in the general case as shown by the first derivatives depicted in the inner graph of the figure.

One of the critical points in the vulnerability remediation process is the change selection activity. We have analyzed the SAT solving time for different scenarios as shown in Fig. 5. We have evaluated the same system with one, three, five and ten active vulnerabilities each time, while varying the amount of vulnerability descriptions in the database. In addition, a set of available changes has been provided to the framework to detect which corrective actions must be performed. In all cases, we have observed a linear behavior as illustrated in the inner graph of the figure, taking around 2 seconds in average to provide the answers for the whole dataset. Often, the SAT solving time depends on the nature of the equations being solved. The observed behavior is partially supported by the fact that the sets of properties (OVAL tests) involved in the IOS vulnerability descriptions are mostly disjoint. Thus, the SAT process is faster because each part of the formula does not impact on the other clauses. In addition, we have observed two

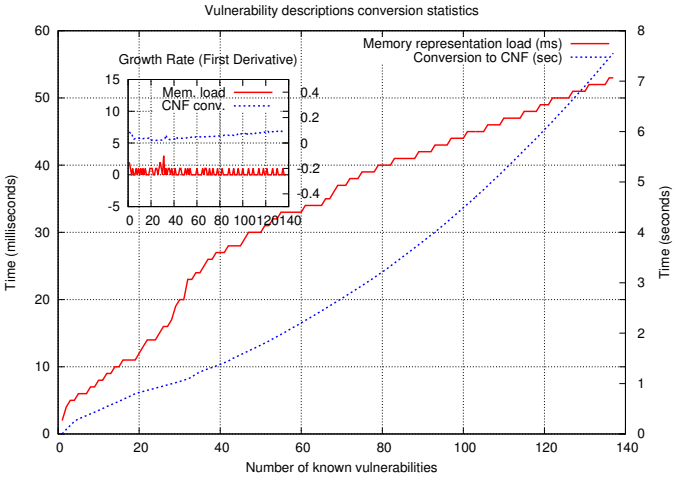

Fig. 4: Vulnerability conversion statistics

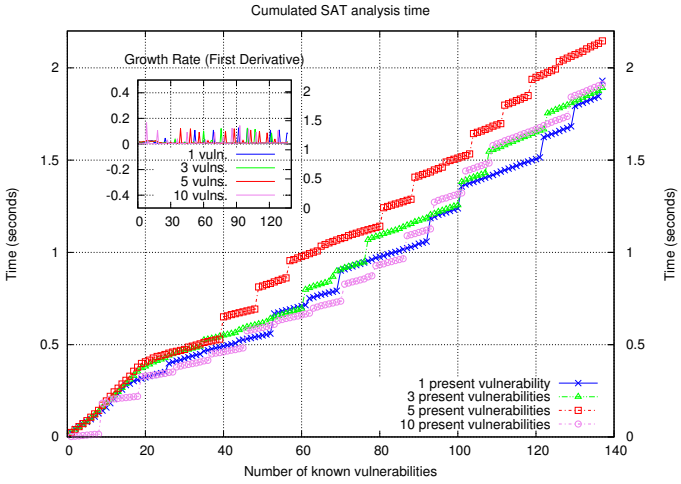

Fig. 5: SAT solving analysis time for change detection

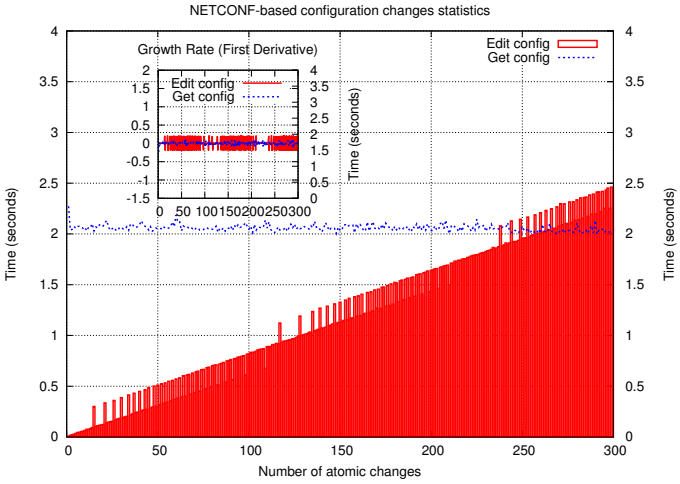

Fig. 6: NETCONF-Cisco statistics

interesting phenomena. The first one is that the SAT solving time (around 2 seconds), which includes a CNF transformation process, is faster than the CNF transformation time depicted in Fig. 4 (around 7.5 seconds). After investigating this behavior, we have realized that only changeable properties participate actively in the SAT solving process. The remaining properties take their truth values from the system, so they are fixed and ignored, making the internal CNF transformation faster. The second phenomenon is that the curve depicting the system with ten present vulnerabilities (violet dashed lines with rounded points) has lower values that the one with five vulnerabilities (red dashed line with square points). Sometimes, more available changes facilitates the search of the SAT solver though this fact also leaves more free assignable variables increasing the search space. Even though this depends on the mechanisms used by the SAT solver, the general behavior for remediating IOS vulnerabilities has been observed to be linearly stable.

Finally, we also have measured the time required to query and perform atomic changes over Cisco IOS via NETCONF. A complete NETCONF session for getting the current configuration in our scenario, including network delays, takes around 2 seconds in average (blue dashed line). NETCONF also allows us to perform a set of various changes in one single session. We have varied the size of this change set from 1 to 300 as shown in Fig. 6 (red bars). We have observed that the time grows linearly, as shown in the inner graph, and that it only requires about 2.5 seconds for performing 300 changes. Considering the overall behavior, these sets of experiments have shown the scalability of the proposed strategy in our context, in terms of representation conversion time, SAT solving time and NETCONF performance.

VII. Conclusions and Future Work

Vulnerability management constitutes a key activity for ensuring safe configurations and preventing security attacks. This activity should be self-corrective, i.e., it should not generate new vulnerable states. In this paper we have proposed a novel approach for dealing with vulnerability management activities in an autonomous manner. We perform vulnerability assessment activities taking advantage of the OVAL language. In order to detect which corrective actions must be performed to secure vulnerable systems, we consider the change decision process as a boolean satisfiability (SAT) problem. We have proposed the X2CCDF language, built on top of XCCDF and OVAL, that allows us to express the impact of these changes over target systems. Based on these descriptions, we use a SAT solving engine to successfully identify which changes must be applied. We have proposed an autonomous framework called VMANS that consolidates these activities using the NETCONF protocol. Finally, we have conducted several experiments over the Cisco IOS platform whose results validate the feasibility and scalability of the proposed approach.

In this work, we have considered remediation actions as atomic changes. However, some corrective activities may consider several changes at once to eradicate vulnerabilities. For future work, we plan to further analyze SAT solving techniques as well as sophisticated mechanisms for considering interactions, coherence and consistency between corrective changes in an unified manner. SAT solving techniques are also affected by the nature of the logical formulas they have to solve. In that context, we aim at investigating other computing platforms as well as distributed vulnerabilities involving multiple devices at once to further validate our approach. Finally, protocols such as NETCONF are able to manage changes in a try-rollback manner by using candidate configurations. However, they are not yet widely deployed and standardized. Even though some equipments implement NETCONF agents such as some versions of Cisco IOS, most of them do not cover the complete specification. In light of this we have planned to analyze diverse mechanisms to tackle this technical reality in order to provide more robust strategies for assessing and remediating vulnerabilities.

Acknowledgements

This work has been partially supported by the EU FP7 Univerself Project, FI-WARE PPP, and the Flamingo Network of Excellence.

REFERENCES

[1] J. Kephart and D. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, pp. 41–50, Jan. 2003.

[2] P. Foreman, *Vulnerability Management*. Taylor & Francis Group, 2010.

[3] S. A. Cook, "The Complexity of Theorem-Proving Procedures," in *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, (New York, NY, USA), pp. 151–158, ACM, 1971.

[4] M. Prasad, A. Biere, and A. Gupta, "A Survey of Recent Advances in SAT-Based Formal Verification," *STTT*, vol. 7, no. 2, pp. 156–173, 2005.

[5] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "RFC 6241, Network Configuration Protocol (NETCONF)," June 2011.

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

[7] "The OVAL Language." http://oval.mitre.org/. Cited August 2013.

[8] "CVE, Common Vulnerabilities and Exposures." http://cve.mitre.org/. Cited August 2013.

[9] "MITRE Corporation." http://www.mitre.org/. Cited August 2013.

[10] M. Barrère, G. Betarte, and M. Rodríguez, "Towards Machine-assisted Formal Procedures for the Collection of Digital Evidence," in *Proceedings of the 9th Annual International Conference on Privacy, Security and Trust (PST'11)*, pp. 32 –35, July 2011.

[11] "NIST, National Institute of Standards and Technology." http://www.nist.gov/. Cited August 2013.

[12] J. Banghart and C. Johnson, "The Technical Specification for the Security Content Automation Protocol (SCAP)," *NIST*, 2009.

[13] "CVSS, Common Vulnerability Scoring System." http://www.first.org/cvss/. Cited August 2013.

[14] M. S. Ahmed, E. Al-Shaer, M. M. Taibah, M. Abedin, and L. Khan, "Towards Autonomic Risk-aware Security Configuration," *In Proceedings of the IEEE Network Operations and Management Symposium (NOMS'08)*, pp. 722–725, Apr. 2008.

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

[16] X. Ou, S. Govindavajhala, and A. W. Appel, "MulVAL: A Logic-based Network Security Analyzer," *on USENIX Security*, 2005.

[17] 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)*, pp. 274–283, May 2007.

[18] M. Chiarini and A. Couch, "Dynamic Dependencies and Performance Improvement," in *Proceedings of the 22nd Conference on Large Installation System Administration Conference*, pp. 9–21, USENIX, 2008.

[19] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated Generation and Analysis of Attack Graphs," in *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, SP '02, (Washington, DC, USA), pp. 273–, IEEE Computer Society, 2002.

[20] M. Albanese, S. Jajodia, and S. Noel, "Time-Efficient and Cost-Effective Network Hardening Using Attack Graphs," in *DSN* (R. S. Swarz, P. Koopman, and M. Cukier, eds.), pp. 1–12, IEEE Computer Society, 2012.

[21] N. Poolsappasit, R. Dewri, and I. Ray, "Dynamic Security Risk Management Using Bayesian Attack Graphs," *IEEE Transanctions on Dependable and Secure Computing*, vol. 9, pp. 61–74, Jan. 2012.

[22] 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*, vol. 5, pp. 178–187, Sept. 2008.

[23] 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)*, pp. 71–84, 2009.

[24] "The Internet Engineering Task Force (IETF)." http://www.ietf.org/. Cited August 2013.

[25] H. M. Tran, I. Tumar, and J. Schönwälder, "NETCONF Interoperability Testing," in *Proceedings of the Third International Conference on Autonomous Infrastructure, Management and Security (AIMS'09)*, pp. 83–94, 2009.

[26] S. Wallin and C. Wikström, "Automating Network and Service Configuration using NETCONF and YANG," in *Proceedings of the 25th International Conference on Large Installation System Administration*, LISA'11, (Berkeley, CA, USA), pp. 22–22, USENIX Association, 2011.

[27] "NVD, National Vulnerability Database." http://nvd.nist.gov/. Cited August 2013.

[28] M. Barrère, R. Badonnel, and O. Festor, "Collaborative Remediation of Configuration Vulnerabilities in Autonomic Networks and Systems," in *Proceedings of the 8th IEEE International Conference on Network and Service Management (CNSM'12)*, IEEE, October 2012.

[29] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006.

[30] "The Greenery Project." http://qntm.org/greenery. Cited August 2013.

[31] "GNS3." http://www.gns3.net/. Cited August 2013.

[32] "AIMA." https://code.google.com/p/aima-java/. Cited August 2013.

[33] "Netconf4J." https://github.com/dana-i2cat/netconf4j. Cited August 2013.