Towards the Assessment of Distributed Vulnerabilities in Autonomic Networks and Systems

Martín Barrère, Rémi Badonnel, Olivier Festor

Towards the Assessment of Distributed 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*—Vulnerability management constitutes a crucial activity within autonomic networks and systems. Distributed vulnerabilities must be assessed over a consolidated view of the network in order to detect vulnerable states that may simultaneously involve two or more devices. In this work, we present a novel approach for describing and assessing distributed vulnerabilities in such self-governed environments. We put forward a mathematical construction for defining distributed vulnerabilities as well as an extension of the OVAL language called DOVAL for describing them. We then define a framework for assessing distributed vulnerabilities in autonomic environments that exploits the knowledge provided by such descriptions. We finally show the feasibility of our solution by analyzing the behavior of the proposed algorithms and strategies through a comprehensive set of experiments.**

I. Introduction

Autonomic computing provides a promising paradigm for dealing with large scale network management [19], [16]. In this context, autonomic networks and systems are responsible for their own management. They have to adapt their configurations with respect to their environment, to protect themselves against security attacks, to repair their own failures, and to optimize their various parameters. In order to achieve these objectives, autonomic related operations are performed modifying the environment. Such operations may lead to potential vulnerable states increasing the exposure to security threats. This article addresses the ability of autonomic networks and systems to deal with security issues, particularly, to detect distributed vulnerabilities and maintain safe configurations across the network.

As systems and technologies evolve, new space for vulnerabilities comes into scene. Autonomic networks must integrate support mechanisms for preventing vulnerabilities. Nowadays, networks are analyzed in order to detect vulnerabilities that may allow a malicious user to perform an attack. However, traditional mechanisms perform a global analysis by investigating each network element individually. Even though such approaches can detect sets of vulnerabilities that may allow an attacker to perform a multi-step attack, they do not provide the capability of detecting vulnerabilities that simultaneously involve two or more devices under specific conditions. The underlying problem relies in that each network device can individually present a secure state, but when combined across the network, a global vulnerable state may be produced.

978-1-4673-0269-2/12/$31.00

In order to cope with this problem, several issues must be attended. Formal mechanisms for describing distributed vulnerabilities are required. Moreover, using standard means for achieving such objective can promote the exchange of security knowledge among practitioners and organizations. Such descriptions in turn must be integrated into the management plane of autonomic networks and systems. Mechanisms for interpreting and assessing theses security advisories must be provided. In addition, optimized algorithms and strategies for collaboratively assessing the network should be developed.

In this paper, we present a framework for describing and assessing distributed vulnerabilities in autonomic networks and systems. We put forward mechanisms for specifying distributed vulnerable states that are taken into account by the proposed framework thus increasing the vulnerability awareness of such self-governed environments. We also perform an analytical and technical evaluation of the proposed approach in order to analyze and show the feasibility of our solution. Our main contributions are: (1) a mathematical approach and an OVAL-based specification language allowing to describe distributed vulnerabilities independently of the technology used to its assessment, (2) a deployable infrastructure based on the Cfengine system, capable of enforcing the assessment and reporting of distributed vulnerabilities as security policies, (3) optimized algorithms and strategies for evaluating such security threats in the underlying network.

The remainder of this paper is organized as follows. Section II describes existing work and their limits to address complex vulnerability management. Section III presents the proposed approach for specifying distributed vulnerabilities in autonomic networks and systems. The architecture of our framework as well as the proposed algorithms and strategies for the assessment of distributed vulnerabilities are described in Section IV. Section V provides an evaluation of our solution through a comprehensive set of experiments and the obtained results. Section VI presents conclusions and future work.

II. Related work

Vulnerability management refers to the cyclical practice of identifying, classifying, remediating and mitigating vulnerabilities [18]. The ability of identifying host-based and distributed vulnerabilities constitutes the first step for such process to be completely embedded into the management plane of autonomic networks and systems.

Currently, several network scanners exist for assessing vulnerabilities on a target network [5], [7]. Some of them use the functionalities provided by powerful port scanners such as Nmap [6]. However, these tools do not provide standard means for describing and exchanging the vulnerabilities they are able to assess. The OVAL[1] language [8], introduced by the MITRE corporation [4], is an information security community effort to standardize how to assess and report upon the machine state of computer systems. OVAL is an XML-based language that allows to express specific machine states such as vulnerabilities, configuration settings or patch states. Real analysis is performed by OVAL interpreters such as Ovaldi [9] and XOvaldi [13]. Related technologies such as the XCCDF[2] language [26], included in the SCAP[3] protocol [11], can be used for several purposes, including automating vulnerability checking, technical control compliance activities, and security measurement. XCCDF is a promising host-targeted language that can be used for expressing actions to take when a vulnerable condition is observed.

In our previous work, we proposed an OVAL-based approach for increasing vulnerability awareness in self-governed environments [12] that relies on the Cfengine tool [1], an autonomic maintenance system that provides support for automating the management of large-scale environments based on high-level policies. The proposed framework promotes the capitalization of external knowledge provided by OVAL repositories enabling autonomic environments to take into account such security advisories when self-management operations are performed. These operations in turn may perform changes on their environments producing unsafe states, thus change and risk management techniques must be considered [2], [17], [15].

Several approaches also consider the idea of modeling a network intrusion as a sequence of steps where each gained privilege by the attacker opens new intrusion capabilities in the network [24], [22], [23]. This concept provides robust foundations for attack graphs, a widely used approach for performing network vulnerability analysis as reported in [10], [21], [20]. However, these approaches do not consider global vulnerable states produced by a simultaneous combination of network devices under certain conditions, thus uniform means for describing distributed vulnerabilities, analyzing and detecting them are deeply required as they provide essential building blocks for dealing with the vulnerability management process within autonomic networks and systems.

III. Specification of distributed vulnerabilities

This work is focused on describing and assessing distributed vulnerabilities. However, during our research, a recurrent question used to come up about the actual definition of a distributed vulnerability. In this section, we introduce a mathematical definition for distributed vulnerabilities and we present DOVAL, a language for describing such vulnerabilities in a machine-readable manner.

[1]Open Vulnerability and Assessment Language
[2]eXtensible Configuration Checklist Description Format
[3]Security Content Automation Protocol

A. Distributed vulnerabilities

The concept of a distributed vulnerability may usually be understood as a set of individual vulnerabilities distributed in the network that potentially might allow a multi-step attack. We think that even though this definition provides a useful perspective to the topic in question, it does not offer a complete outlook of the problem. We consider a distributed vulnerability as a set of conditions over two or more network devices that if observed simultaneously, then a potential exploitable threat is present on the network under analysis. The main difference between considering a set of individual vulnerabilities over different network devices and the proposed definition is that the required conditions to be observed over one network device may not constitute a complete vulnerability description. We now present some required definitions in order to mathematically specify a distributed vulnerability:

- $H = \{h_1, ..., h_m\}$ denotes the set of devices in the network (e.g. hosts, routers).
- $R = \{(h_i, ..., h_j)*\}$ denotes the set of relationships between devices in the network (e.g. reachability, service provisioning).
- $P = \{p_1, ..., p_n\}$ denotes the set of device properties (unary predicates $p_i(h_j)$) required to be observed for the distributed vulnerability to be present in the network under analysis.
- $P^H = \{s_1, ..., s_k\}$ denotes the set of subsets of P where $s_j = \prod(P) = \{p_i*\}, i \in \{1, ..., n\}$. s_j denotes the set of properties, called *role*, required to be observed on one specific device.
- $P^R = \{r_1, ..., r_v\}$ denotes the set of relationships between devices (n-ary predicates $r_i(h_i, ..., h_j)$) required to be observed for the distributed vulnerability to be present in the network under analysis.

Based on the previous definitions, we define a distributed vulnerability DV as the compliant projection of the pattern (P^H, P^R) over the network (H, R) as illustrated in Figure 1.

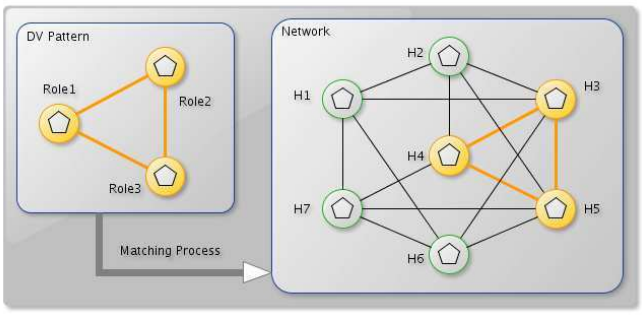

Fig. 1: Distributed vulnerability matching process

B. DOVAL, a distributed vulnerability description language

We have conceived the DOVAL language (Distributed OVAL) on top of OVAL as a means for expressing the proposed mathematical approach in a machine-readable manner.

The OVAL perspective can be seen as a host-based approach, capable of describing specific host states independently. DOVAL leverages the OVAL language by providing mechanisms for describing vulnerabilities that involve two or more network devices at the same time. While the universe of discourse in the OVAL language is composed by digital components (e.g. processes, files), DOVAL extends it by considering network devices as well. In addition, we extend the semantic of the language by allowing to express relationships between objects in order to describe conditions involving several devices simultaneously. In this manner we can for instance specify that a network is vulnerable, if a given traffic between specific processes and devices is allowed.

Within the DOVAL language, the required conditions over each involved device are described using standard OVAL definitions. An OVAL definition is intended to describe a specific machine state using a logical combination of tests that must be performed over a host. If such logical combination is observed, then the specified state is present on that host (e.g. vulnerability, specific configuration) [8]. Under a logical perspective, this combination can be understood as a first order formula where each test corresponds to an atomic unary predicate over that system [12]. DOVAL extends this concept by enabling the expression of predicates that involve more than one device, thus allowing the specification of required relationships over the network. Figure 2 depicts the main DOVAL constructs and provides a description of the intended purpose of each main building block.

The DOVAL language		
DOVAL constructs	**Description**	**First-order logic**
DOVAL document	Set of distributed vulnerabilities	Arrangement of compound logical formulas
DOVAL definition	Distributed vulnerability	Compound logical formula
DOVAL test	Assessment of a condition between several devices	Container of an atomic n-ary predicate
DOVAL object	Devices with specific conditions using OVAL definitions	Family of individuals in the discourse universe
DOVAL state	Network condition between devices characterized by DOVAL objects	Mathematical relationship specification

Fig. 2: DOVAL logical description

A DOVAL document is intended to meet the specification of several components required to describe a set of distributed vulnerabilities. Each distributed vulnerability is specified by a DOVAL definition, which provides the capability of expressing a logical formula that involves several DOVAL tests. Each DOVAL test in turn constitutes the container of an n-ary predicate over a set of network devices $(h_i, ..., h_j)$ and it is in charge of putting together the required devices and the states expected to be observed between them.

We consider a required network device as a device that meets certain conditions (s_j) and that is required to be present for the distributed vulnerability DV to be true in the network under analysis (H, R). Required network devices (P^H) are described by means of DOVAL objects. In order to express DOVAL objects, we take advantage of the very final objective of the OVAL language, this is to say, to express specific machine states. Thus, a DOVAL object is actually a set of references to OVAL definitions, where each one describes a required specific machine state (p_i) on that device. Finally, the expected relationships over the network (P^R) are expressed using DOVAL states. A DOVAL state (r_i) specifies properties between devices and the roles each device has within such relationship, and can be seen as an actual predicate itself.

There exist several situations where neither individual host assessments nor inference chains over individual exploitable vulnerabilities can expose potential security threats in a wide network. Mechanisms for globally specifying and evaluating network distributed vulnerabilities are essential.

C. A case study

We motivate the use of the DOVAL language by considering the scenario described in [25] and depicted in Figure 3 where two related hosts, a SIP[4] server and a DNS[5] server, each one with specific properties, constitute a potential exploitable network vulnerability.

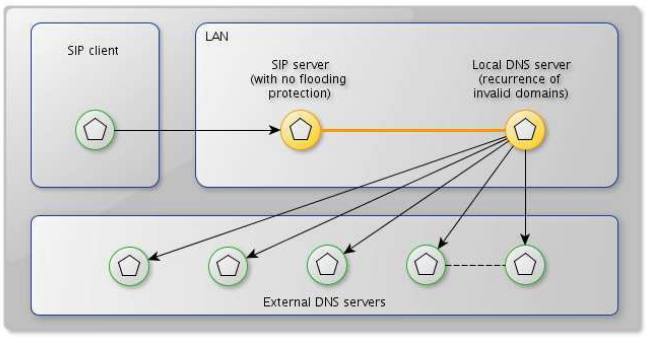

Fig. 3: Distributed vulnerability scenario

In this example, a denial of service (DoS) attack over the SIP server can be performed by flooding it with unresolvable domain names that must be solved by a local DNS server. The local DNS server in turn, is configured for requesting the resolution of unknown domains to external servers, increasing the number of waiting requests and therefore the response time for each SIP request. Under these configuration states, flooding a SIP server with such type of messages will prevent it to respond to legitimate requests. It is important to highlight that both servers and the relationship between them are required conditions for the distributed vulnerability to be present. If the DNS server is not present or it is not compliant with the required specific conditions, the SIP server would immediately respond to a SIP client that its SIP request has failed.

[4]Session Initiation Protocol
[5]Domain Name System

Even in such a situation, thousands of SIP requests may collapse the SIP server anyway, though it is a slightly different scenario that could be specified using standard OVAL definitions. On the other hand, if there is no SIP server, it is quite clear that the distributed vulnerability has no place in this environment. Such scenario can be specified using the DOVAL language as illustrated in Listing 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<doval_document>
  <doval_definitions>
    <doval_definition id="doval:fr.inria.doval:def:1" class="
          distributed_vulnerability">
      <metadata>
        <title>SIP DoS attack using DNS flooding</title>
        <description>...</description>
        <doval_repository>...</doval_repository>
      </metadata>
      <criteria>
        <criterion comment="..." test_ref="doval:fr.inria.doval:tst:4141"/>
      </criteria>
    </doval_definition>
  </doval_definitions>

  <tests>
    <doval_test id="doval:fr.inria.doval:tst:4141" comment="DOVAL test combining
          two specific tests, reachability and configuration." check_existence="
          at_least_one_exists" check="at least one">
      <object device_ref="doval:fr.inria.doval:dev:222"/>
      <object device_ref="doval:fr.inria.doval:dev:256"/>
      <state state_ref="doval:fr.inria.doval:ste:4444"/>
      <state state_ref="doval:fr.inria.doval:ste:7777"/>
    </doval_test>
  </tests>

  <objects> <!-- devices -->
    <device_object id="doval:fr.inria.doval:dev:222"/>
      <prop ovaldef_ref="oval:org.mitre.oval:def:1000"/> <!--SIP server running-->
      <prop ovaldef_ref="oval:org.mitre.oval:def:1001"/> <!--Port 5060 open-->
      <prop ovaldef_ref="oval:org.mitre.oval:def:1002"/> <!--Security module not
          installed-->
    </device_object>

    <device_object id="doval:fr.inria.doval:dev:256"/>
      <prop ovaldef_ref="oval:org.mitre.oval:def:2000"/> <!--DNS server running-->
      <prop ovaldef_ref="oval:org.mitre.oval:def:2001"/> <!--Port 53 open-->
      <prop ovaldef_ref="oval:org.mitre.oval:def:2001"/> <!--Unknown domains
          solved by external DNS servers-->
    </device_object>
  </objects>

  <states> <!-- relationships -->
      <link_state id="doval:fr.inria.doval:ste:4444"/> <!--Traffic capability-->
        <protocol operation="equals"> udp </protocol>
        <src_port device_ref="doval:fr.inria.doval:dev:222"
         operation="pattern match"> .* </src_port>
        <dst_port device_ref="doval:fr.inria.doval:dev:256"
         operation="equals"> 53 </dst_port>
      </link_state>

      <service_state id="doval:fr.inria.doval:ste:7777"/> <!--Service config-->
        <name operation="equals"> dns </name>
        <consumer device_ref="doval:fr.inria.doval:dev:222"/>
        <provider device_ref="doval:fr.inria.doval:dev:256"/>
      </service_state>
  </states>
</doval_definitions>
```

Listing 1: DOVAL document

A DOVAL definition with id *"doval:fr.inria.doval:def:1"* identifies which DOVAL tests must be performed in order to detect the distributed vulnerability that such definition is intended to describe. The DOVAL test with id *"doval:fr.inria.doval:tst:4141"* identifies the required devices as DOVAL objects and the relationships between them by referencing DOVAL states. The required devices, namely a SIP server with no flooding protection (s_1) and a local DNS server with external unknown domain resolution (s_2), are specified using two DOVAL objects, *"doval:fr.inria.doval:dev:222"* and *"doval:fr.inria.doval:dev:256"* respectively. Each DOVAL object enforces the required properties over the device it describes by considering a set of OVAL definitions, one for each needed condition. Each DOVAL state specifies the characteristics of the relationship expected to be observed between both devices.

Within the current example, two relationships are required: (1) DNS traffic is allowed between the SIP server and the DNS server (r_1), specified by a *DOVAL link_state* with id *"doval:fr.inria.doval:ste:4444"*, and (2) the SIP server has configured the DNS server as its DNS service provider (r_2), specified by a *DOVAL service_state* with id *"doval:fr.inria.doval:ste:7777"*. In order to assess such a specification, a deployable distributed infrastructure capable of enforcing its evaluation and notification is required.

IV. A FRAMEWORK FOR ASSESSING DISTRIBUTED VULNERABILITIES

The main objective of DOVAL is targeted on describing conditions involving several network devices that if observed, the underlying network presents a vulnerable state that can be exploited by an attacker. In order to detect such scenarios, DOVAL descriptions must be interpreted and evaluated on the target network. We propose a framework based on Cfengine, a widely deployed configuration and administration system, capable of enforcing security policies for discovering distributed vulnerabilities across the network.

A. Architecture overview

Due to the size and dynamics of current networks, the assessment and detection of vulnerable distributed states is not a trivial task. We partition the problem into several steps, namely, (1) generation of a minimized loop-free topology of the underlying network, (2) collection of hosts and network information, and (3) assessment of DOVAL specifications over the gathered data. Figure 4 illustrates the steps and the architecture of the proposed approach.

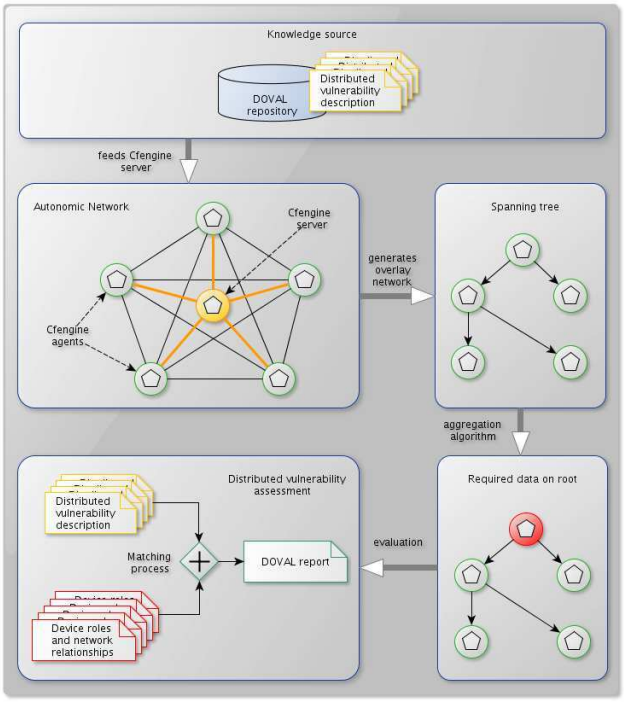

Fig. 4: Overall architecture

Within this architecture, distributed vulnerabilities are specified using the DOVAL language and stored in a database. A Cfengine server is fed with such knowledge and translated as Cfengine policy rules. Our approach considers a deployment of Cfengine agents across the network, where each agent is in charge of controlling one network device. In order to evaluate the existence of a distributed vulnerability, a spanning tree is built on top of the target network in order to minimize paths and avoid loops. The DOVAL specification is then transmitted across the tree and the required information is gathered by performing an aggregation algorithm over the nodes. Each Cfengine agent assesses the device it controls in order to discover which roles such device can play within the distributed vulnerability specification. This information is returned back until all the information is stored at the root node of the spanning tree. Finally, the specification of the distributed vulnerability is projected over the information gathered from the network, and a DOVAL report is generated informing about distributed vulnerable states across the network.

B. Assessment strategies

Several strategies for assessing the required properties on each network device can be used. Considering the number of potential combinations, an optimized algorithm for evaluating the network is required. Within our approach, we consider the aggregation algorithm proposed in [14] for building a tree-based overlay network with the information of each device in the network under analysis. We now proceed to explain our strategy in a constructive manner considering two situations. The first situation presents a simplified scenario where exists full connectivity between each pair of nodes in the network and no further relationships between nodes are required. The second situation puts forward a more realistic scenario that extends the first one by considering network constraints such as reachability restrictions or service provisioning requirements.

When starting a DOVAL definition assessment, the set of required devices for the distributed vulnerability to be present can be seen as an empty tuple $t = (\_\_1, \_\_2, ..., \_\_k)$. Each blank field represents the placeholder for a required role characterized by $P^H = \{s_1, ..., s_k\}$ according to the definitions given in Section III-A. During the assessment across the spanning tree, t will be collaboratively fulfilled as each Cfengine agent will indicate which of these fields the device it controls can play. At the end, several combinations can occur thus the final computation will be performed over a set $T = \{t_1, ..., t_w\}$. Let DV be a distributed vulnerability where $t = \{\underline{s1}, \underline{s2}, \underline{s3}\}$ specifies the set of roles required to be observed in the network. Figure 5 depicts the steps taken during the algorithm execution for discovering the roles each network device is able to play. At the initial step, a spanning tree covering every node in the network is considered. The tree is explored using a post-order traversal. At Step 1, the node $h4$ is assessed reporting that it matches roles $s1$ and $s3$. Then, the node $h2$ fulfills its temporal role list and continues with the node $h5$ as shown in Step 2. At Step 3, $h2$ is assessed in order to detect which roles it can play and the role list for the sub-tree with root $h2$ is returned to the caller $h1$. The node $h3$ is assessed at Step 4 identifying its ability to perform the role $s2$. At Step 5, the temporal role list describes the roles that the nodes $h2$, $h3$, $h4$ and $h5$ are able to play. The assessment of the node $h1$ will complete the role list indicating that role $s1$ can be performed by the nodes $h4$ and $h5$, whereas the role $s2$ can be performed by $h1$, $h3$ and $h5$, and $s3$ by $h1$ and $h4$.

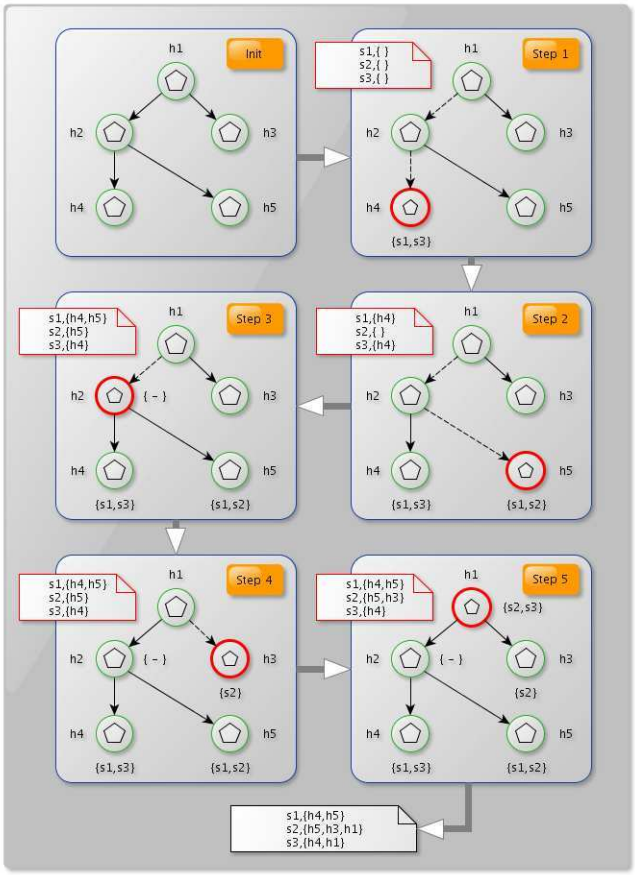

Fig. 5: Aggregation algorithm execution for role discovery

Usually, networks present complex topologies imposing reachability restrictions. We consider such constraints in the second situation, where distributed vulnerabilities require the existence of relationships among the involved network devices such as reachability or service provisioning. Under a logical perspective, a distributed vulnerability involving n roles may require at most the evaluation of n-ary predicates $r_i(h_1, ..., h_n)$ where each device h_i covers one specific role though it could cover more than one at the same time. Such scenarios require not only to discover the roles that each network device is able to play but also to assess the relationships between them. In order to deal with these more realistic situations, we extend the previous algorithm by performing a neighborhood discovery at each node and assessing the relationships each node supports with its neighbors. The idea behind this approach is to *reduce* the participation of nodes that can endorse an expected role but they do not satisfy the required relationships with other nodes. Considering the example given in Section III-B, the fact of finding a SIP server h_1 and a DNS server h_2 under

the required conditions does not mean that the distributed vulnerability is necessarily present. For instance, if DNS traffic is not allowed between them, the predicate $r_1(h_1, h_2)$ does not hold; then such pair of network devices does not constitute a candidate combination for the distributed vulnerability. In order to analyze the surrounding environment at each node, we take advantage of Cfengine's functionalities for performing a neighborhood discovery [14]. The process described by Algorithm 1 depicts the steps performed by each Cfengine agent while assessing the roles the device can play as well as the required relationships it supports with its neighbors.

```
   Input: DOVAL document
   Output: List of tuples specifying devices and relationships
           between them
1  foreach role s ∈ DOVAL objects do
2      if currentNode n is compliant with role s then
3          N ← SelectPartitionNeighbors;
4          foreach predicate r ∈ DOVAL states do
5              if r requires role s then
6                  w ← getPredicateArity(r);
7                  foreach sequence {h_2, ...h_w} ⊆ N do
8                      result ← eval state r(n, h_2, ..., h_w);
9                      if result is true then
10                         add tuple to output list
                           [r, (n, s), (h_2, s_2), ..., (h_w, s_w)];
11                     end
12                 end
13             end
14         end
15     end
16 end
```

Algorithm 1: Node roles and relationships assessment

Within our approach, a DOVAL object describes the required properties (s) that must be observed over a network device (h) that is part of a distributed vulnerability. Each device compliant with such description is able to perform the object's intended role. The proposed algorithm is executed on each agent and analyzes every role (line 1) the device it controls can perform. For each supported role (line 2), its neighbors are stored in the set N (line 3) and the required relationships are assessed against them (line 4). Such relationships are described by means of DOVAL states and they can be seen as predicates involving w network devices. Only those relationships involving the current role s are assessed (line 5). Depending on the arity of the predicate (line 6), subsets of $w-1$ network devices are built (line 7) and the relationship among them is assessed (line 8). If such relationship holds, a tuple is added in the output list (line 10) indicating that the node n is able to perform the role s, and that under that role, the relationship r (described by a DOVAL state) holds when considering devices $(h_2, ..., h_w)$ performing potential roles $(s_2, ..., s_w)$ respectively. The actual verification of devices $(h_2, ..., h_w)$ performing roles $(s_2, ..., s_w)$ is done at the end, when each node in the spanning tree has been evaluated and the root node has all the required information for the assessment of the distributed vulnerability in the network.

V. Performance evaluation

In this section we present an analytical evaluation and a technical discussion of the proposed approach. We put forward the specification of two metrics in order to analyze the performance during the evaluation of a generic distributed vulnerability, namely, (1) the number of messages sent across the network, and (2) the total time required for the assessment. We also show the scalability of the proposed approach by modeling different scenarios based on the instantiation of various parameters within the specified metrics.

Let N be the network under analysis with a set of devices $H = \{h_1, ..., h_n\}$ and relationships $R = \{(h_i, ..., h_j)*\}$. Let DV be a distributed vulnerability that requires a set of k roles defined by $P^H = \{s_1, ..., s_k\}$ and v relationships defined by the set $P^R = \{r_1, ..., r_v\}$. We consider the following assumptions during the assessment of DV over N:

- a binary spanning tree is built on top of N,
- each device h has in average q neighbors,
- each predicate r_i has in average arity b (we define the variable $a = b - 1$ in order to simplify the equations),
- the probability for a device to play a role s_j (event A) is given by $P(A) = \alpha$,
- the probability for a role s_j to occur on a predicate r_i (event B) is given by $P(B) = \beta$,
- the evaluation of any role s_j over any device h takes in average γ units of time,
- the evaluation of any predicate r_i takes in average δ units of time.

Number of messages. In order to analyze the traffic generated across the network, we consider the number of messages sent during the evaluation of a generic distributed vulnerability DV. We define the metric M that estimates the amount of messages transmitted as follows:

$$M = n * (1 + k * P(A) * P(B|A) * v * M_{pred})$$

$$\text{where} \quad M_{pred} = P_R(q+1, a) * a = (q+1)^a * a$$

When traversing the spanning tree (n nodes), each node receives one message in order to start its own evaluation. At each node, k roles must be evaluated. Because not every device will play every role, we model this uncertainty as an event A that occur with probability [6] $P(A) = \alpha$. Given the event A for a role s_j, we model the probability of such role to be involved on each predicate r_i by considering the conditional probability $P(B|A) = \beta$ and the multiplier v. The final factor M_{pred} represents an upper bound of the number of messages sent when evaluating one single predicate among the node under analysis and its neighbors. $P_R(q+1, a)$ denotes the number of a-permutations with repetition of a set of q neighbors plus the node itself. For each possible permutation that may fulfill the arguments (roles) required by

[6] This is a simplified measure of the likelihood for the event A to occur since depending on the network nature, the probability of choosing, for instance, a SIP server within a standard company network will be presumably lower than picking up a standard workstation running any version of Windows 7.

the predicate under analysis, a messages must be sent. M_{pred} represents an upper bound because we consider that multiple roles can be covered by one single device, which means that for those combinations assigning for instance the same device to each role, only one instead of a messages will be sent.

Assessment time. We analyze here two distributed approaches that have a direct impact on the total assessment time for a distributed vulnerability DV. The first one consists on sequentially assessing the network, analyzing one node at a time. In this case, we define the total assessment time as:

$$T_S = n * k * (\gamma + P(A) * P(B|A) * v * T_{pred})$$

$$\text{where} \quad T_{pred} = \delta * P_R(q+1, a) = \delta * (q+1)^a$$

For each node in the network, k roles are evaluated where each one takes γ units of time. The probability for the node to perform a role is given by $P(A)$ whereas $P(B|A)$ defines the probability for that role to be involved on each one of the v predicates. T_{pred} involves the same permutations among neighbors as M_{pred} does, but for each possible sequence T_{pred} considers the parameter δ that models the average time for evaluating a predicate r_i.

An alternative to the sequential modality consists on using a parallel computing approach. Such approach enables the evaluation of every node simultaneously thus reducing the sequential assessment time. Under this perspective, the total assessment time becomes:

$$T_P = max_{node}(k * (\gamma + P(A) * P(B|A) * v * T_{pred}))$$

$$\text{and the worst case is:} \quad T_{Pw} = k * (\gamma + v * T_{pred})$$

T_{Pw} describes the extreme case where a node is able to perform every required role that in turn is involved in each predicate. Such case maximizes the amount of time among the times required by the nodes in the spanning tree.

The proposed metrics have several parameters that affect the final result such as the number of nodes (n) or the probability for an arbitrary node to play a given role ($P(A)$). Typical scenarios usually involve predicates conceived as peer to peer properties (binary predicates, $a = 1$) as well as a restricted number of roles (small k). Each required predicate in turn usually involves all roles, thus simplifying conditional probabilities ($P(B|A) = 1$). Under this perspective, we have performed several experiments using the example illustrated at Section III-B as a case study in order to analyze the behavior of our approach. In this scenario we have $k = 2$, $v = 2$, $a = 1$ and $q = n - 1$ depicting a full mesh network. We have also simplified the units of time considering $\gamma = \delta = 1$. Networks nature can vary depending on the context and the purpose they have been built for. Therefore, we use the parameters n and $P(A)$ for analyzing such situations. The experiment depicted in Figure 6 considers a uniform distribution for role assignment ($P(A) = \frac{1}{n}$), meaning that only one of n nodes may play a given role s_j. These curves depict how much grows the number of messages (solid line) as well as the assessment time under a sequential computing approach (dotted line) and a parallel computing approach (dashed line) when the number

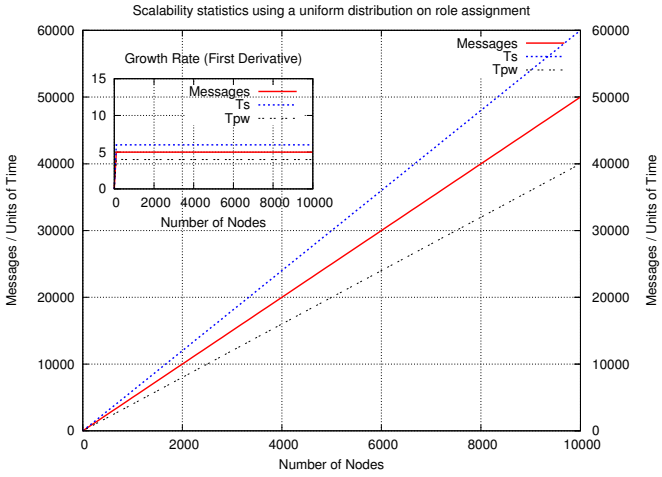

Fig. 6: Statistics with uniform distribution on role assignment

of nodes in the network becomes bigger. We can observe a proportional growth on every metric (M, T_S, T_{Pw}) under a constant rate. This is easily verifiable by looking at the inner graph of Figure 6 that shows the first derivatives of each curve.

Considering that only one device can perform a required role may apply just on special cases so we have analyzed the behavior of our approach when increasing the number of potential devices able to perform the required roles. This is achieved by modifying the probability for a device to play a role ($P(A)$) as shown in Figure 7. We observe that even though the assessment time using a sequential approach is not linear, its growth rate illustrated in the inner graph of Figure 7 remains linear. We get a maximum assessment time when $P(A) = 1$ (dotted line) because every node is able to play every role. When $P(A) = \frac{1}{2}$ (solid line), a half of the network can perform each role and the assessment time is lower though the curve demonstrates similar behavior as with $P(A) = 1$. When a parallel approach is used (dashed line), a constant behavior is observed in the worst case ($P(A) = 1$) making it clear that such approach is better than the sequential one.

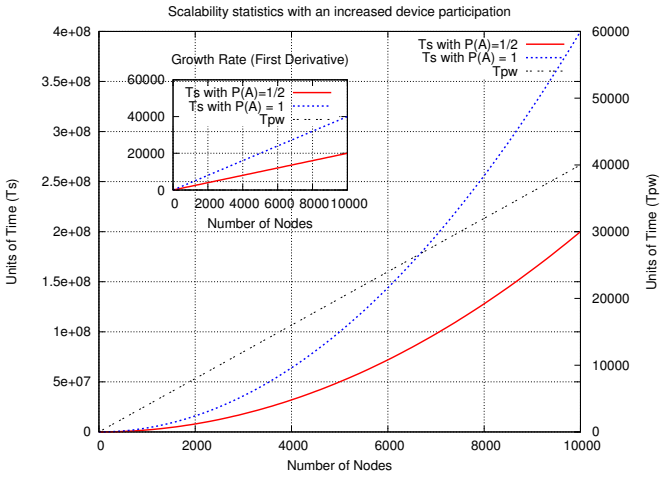

Fig. 7: Statistics with an increased device participation

From a more technical point of view, the proposed approach requires mechanisms for interpreting and assessing descriptions of distributed vulnerabilities. We have developed Ovalyzer [12], a tool capable of translating OVAL advisories to Cfengine policy rules. Within our approach, the Cfengine system is the component to be embedded within target autonomic networks and systems. It is in charge of enforcing security policies including the assessment of distributed vulnerabilities. To achieve this, policy rules directly interpretable by Cfengine are needed. We are currently developing a tool called *Dovalyzer* capable of translating DOVAL specifications into Cfengine policy rules that represent them. Dovalyzer uses the functionalities provided by Ovalyzer for translating OVAL definitions and complements the generated code in order to cover distributed assessment tasks specified by DOVAL definitions. The translator takes as input the content of DOVAL documents and produces Cfengine code that is structured as Cfengine policy files that can be later consumed by a Cfengine running instance. The Cfengine rules currently generated are compliant with *Cfengine 2.2.10* [1]. Dovalyzer is a Java-based application built on top of the *Spring framework 3.0*. It uses *JAXB* [3] for managing XML related issues and provides a plugin-based architecture enabling seamless functional extensions.

VI. Conclusions and future work

Our work addresses the integration of distributed vulnerability descriptions into the management plane of autonomic networks and systems as well as mechanisms for assessing such security advisories. We have mathematically defined the concept of a distributed vulnerability and we have developed DOVAL, an OVAL-based language capable of expressing these formal constructions. A case study has been presented showing DOVAL's main constructs. As in OVAL, DOVAL descriptions can constitute useful security repositories that in turn can be exploited by self-managed environments in order to ensure safe configurations. We have proposed a framework based on the Cfengine system for assessing distributed vulnerabilities in autonomic networks as well as optimized algorithms and collaborative strategies for performing such evaluations. We have analyzed the proposed algorithms by mathematically defining computation costs that show the feasibility of the model through a comprehensive set of experiments. We also have presented a technical discussion about the proposed framework implementation where Cfengine policies are fed by Dovalyzer, a DOVAL to Cfengine translator prototype capable of producing Cfengine policy rules that represent DOVAL security advisories. For future work we plan to investigate further the actual impact of our solution on the normal management operations performed by autonomic networks and systems. Optimized algorithms and strategies are essential due to network dynamics thus we aim at analyzing other techniques in order to tackle these problems and take advantage of previous experience for reducing computation costs. We also plan to extend the proposed approach to the execution of remediation actions. XCCDF's philosophy can be exploited over distributed scenarios in order to correct distributed vulnerable states too.

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 August 20, 2011.
[2] ITSM - IT Service Management. http://www.itsm.info/ITSM.htm. Last visited on August 20, 2011.
[3] Java Architecture for XML Binding. http://java.sun.com/developer/technicalArticles/WebServices/jaxb/. Last visited on August 20, 2011.
[4] MITRE Corporation. http://www.mitre.org/. Last visited on August 20, 2011.
[5] Nessus. http://www.tenable.com/products/nessus. Last visited on August 20, 2011.
[6] Nmap. http://nmap.org/. Last visited on August 20, 2011.
[7] OpenVAS. http://www.openvas.org/. Last visited on August 20, 2011.
[8] OVAL Language. http://oval.mitre.org/. Last visited on August 20, 2011.
[9] Ovaldi, the OVAL Interpreter Reference Implementation. http://oval.mitre.org/language/interpreter.html. Last visited on August 20, 2011.
[10] P. Ammann, D. Wijesekera, and S. Kaushik. Scalable, Graph-Based Network Vulnerability Analysis. *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS'02)*, page 217, 2002.
[11] J. Banghart and C. Johnson. The Technical Specification for the Security Content Automation Protocol (SCAP). *NIST Special Publication*, 2009.
[12] M. Barrère, R. Badonnel, and O. Festor. Supporting Vulnerability Awareness in Autonomic Networks and Systems with OVAL. *Proceeding of the 7th IEEE International Conference on Network and Service Management (CNSM'11)*, October 2011.
[13] 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)*, pages 32 –35, July 2011.
[14] 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.
[15] 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.
[16] Autonomic Computing. An Architectural Blueprint For Autonomic Computing. *IBM White Paper*, 2006.
[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)*, (1):274–283, May 2007.
[18] P. Foreman. *Vulnerability Management*. Taylor & Francis Group, 2010.
[19] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *Computer*, 36(1):41–50, January 2003.
[20] X. Ou, W. F. Boyer, and M. A. McQueen. A Scalable Approach to Attack Graph Generation. In *13th ACM Conference on Computer and Communications Security (CCS'06)*, pages 336–345. ACM Press, 2006.
[21] X. Ou, S. Govindavajhala, and A. W. Appel. MulVAL: A Logic-Based Network Security Analyzer. *on USENIX Security*, 2005.
[22] N. K. Pandey, S. K. Gupta, S. Leekha, and J. Zhou. ACML: Capability Based Attack Modeling Language. *Proceedings of The Fourth International Conference on Information Assurance and Security*, pages 147–154, September 2008.
[23] S. J. Templeton and K. Levitt. A Requires/Provides Model for Computer Attacks. *Proceedings of the Workshop on New Security Paradigms (NSPW'00)*, pages 31–38, 2000.
[24] 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.
[25] 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.
[26] N. Ziring and D. Waltermire. Specification for the Extensible Configuration Checklist Description Format (XCCDF). http://scap.nist.gov/specifications/xccdf/. Last visited on August 20, 2011.