Improving Present Security through the Detection of Past Hidden Vulnerable States

Martín Barrère, Rémi Badonnel, Olivier Festor

Improving Present Security through the Detection of Past Hidden Vulnerable States

Martín Barrère, Rémi Badonnel and Olivier Festor

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

***Abstract*—Vulnerability assessment activities usually analyze new security advisories over current running systems. However, a system compromised in the past by a vulnerability unknown at that moment may still constitute a potential security threat in the present. Accordingly, past unknown system exposures are required to be taken into account. We present in this paper a novel approach for increasing the overall security of computing systems by identifying past hidden vulnerable states. In that context, we propose a modeling for detecting unknown past system exposures as well as an OVAL-based distributed framework for autonomously gathering network devices information and automatically analyzing their past security exposure. We also describe an implementation prototype and evaluate its performance through an extensive set of experiments.**

I. Introduction

By the time a piece of software is being constructed, several errors may be unintentionally introduced providing room for security vulnerabilities. These vulnerabilities can survive within active systems for a long period of time without being detected. During this period, attackers may perform well-planned and clean attacks (e.g., stealing information) without being noticed by security entities (e.g., system administrators, intrusion detection systems, self-protection modules). In deed, unaware entities do not even think about such a potential breach due to the very nature of being under-informed, constituting blind and easy targets for attackers. As a matter of fact, such attacks might never be detected. Changes in the system or even its normal activity can alter or erase the remaining evidence. This issue makes it clear why it is so important to increase the awareness of our systems as soon as security information becomes available. Under that context, this work is focused on taking advantage of current security information for analyzing system security in the past. If unknown security exposures are detected, response actions can be performed in the present for bringing system states to secure levels.

The ability to identify past unknown system exposures due to hidden vulnerabilities allows forensic activities to be performed in order to detect malicious activity [10], [12]. For instance, a bank that has detected a potential intrusion compromising data about credit cards would be able to take actions before consequences become out of control. It would be easier for the bank to block compromised credit cards and make new ones than waiting for notifications of anomalous activity from its clients. Other scenarios apply as well in general computing systems. Usually, intruders leave entry points (backdoors) for coming back to compromised hosts. If a past exposure that may allow an attacker to install backdoors has been detected, forensic analysis could be performed in order to reveal such security issue. The consequences of this investigation not only allow to know if the system has been actually compromised but also to correct a security breach in the present that could be used for future attacks. The acknowledge of current vulnerabilities is a critical factor for reducing the exposure of computing systems. Under this perspective, there is a race for getting security information early. Both security entities and attackers can benefit from their speed, being for self-defence or for breaking security barriers. Open and mature standards such as the OVAL[1] language are cornerstones at this point as they provide a strong support for openly exchanging security information within the community.

In this paper we propose a novel approach for increasing the overall security of computing systems by identifying past hidden vulnerable states. This information can be used for detecting potential unknown attacks in the past, identifying compromised assets and bringing systems up to secure states. Taking advantage of the OVAL language for representing system states and analyzing vulnerabilities, our strategy consists in autonomously generating images of network devices that represent their current state, building a history of their evolution, and capitalizing new security advisories for automatically assessing past system states in order to detect potential security breaches. Our main contributions are: (1) a mathematical model for describing and detecting unknown past security exposures, (2) a deployable infrastructure able to autonomously build historical system images and to automatically assess them when new security information is available, and (3) optimized algorithms and their evaluation for analyzing large sets of system properties over IOS Cisco devices.

The remainder of this paper is organized as follows. Section II describes existing work and their limits. Section III presents our approach for modeling and detecting unknown past security exposures. Section IV details the proposed framework describing its architecture and the strategies for performing assessment activities. Section V depicts the internals of our implementation prototype. Section VI provides an evaluation of our solution through a comprehensive set of experiments. Section VII presents conclusions and future work.

[1]Open Vulnerability and Assessment Language [9]

II. Related work

Managing large-scale networks is a complex task and by nature, humans make errors when configuring them. In addition, changes performed by autonomic entities may increase their own security exposure. Because of this, vulnerable configurations are likely within such environments and they may potentially lead to a wide spectrum of negative and unwanted issues such as instability, unavailability, confidentiality problems, and many more. Under this perspective, vulnerability management constitutes a crucial activity usually defined as the practice of (I) identifying, (II) classifying, (III) remediating and mitigating vulnerabilities [20]. In order to establish a secure process for dealing with vulnerabilities, it is necessary to specify a policy defining the desired system state and a well-known start point to identify vulnerabilities and policy compliance [27].

The CVE[2] language [3] has been introduced by the MITRE Corporation [5] as 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 [9], 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, patch states. Real analysis is performed by OVAL interpreters such as Ovaldi [7] and XOvaldi [17]. Several related technologies have evolved around the OVAL language. NIST [6] is responsible for the development of emerging technologies including the SCAP[3] protocol [14] and the XCCDF[4] language [28]. The SCAP protocol is a suite of specifications that includes OVAL and XCCDF, and it can be used for several purposes, including automating vulnerability checking, technical control compliance activities, and security measurement. The use of SCAP, particularly OVAL and XCCDF, not only allows to specify vulnerabilities, but also to bring a system into compliance through the remediation of identified vulnerabilities or misconfigurations.

Several previous contributions have taken advantage of public vulnerability databases such as [13] and the use of the OVAL language for performing vulnerability assessment activities in large scale networks [16], [15], [22]. Declaratives approaches for analyzing the security of a network are considered within other works as well [23]. When a system is found to be vulnerable, corrective changes must also be analyzed. A large variety of techniques has been proposed to evaluate the impact of changes in networks and systems [19], [18]. Under an autonomic perspective, automated techniques for assessing change associated risks as proposed in [24] and [26] are important because they provide a key support for the change management process, particularly for taking decisions about effective change implementations.

[2]Common Vulnerabilities and Exposures

[3]Security Content Automation Protocol

[4]eXtensible Configuration Checklist Description Format

Historical vulnerability information as well as security metrics and trends are highly useful as proposed in [25], [11], however these contributions do not take advantage of new security information that could have been useful in the past for detecting security exposures. As explained in the next section, the exploit for a vulnerability can be released a long time before the vulnerability is publicly known. Hence, affected systems can be exposed during this period without actually knowing it. To the best of our knowledge, no previous contributions have taken advantage of current security advisories for assessing past hidden vulnerable states. This would enable current systems to increase their own exposure awareness and to take actions in consequence if unknown past exposures are detected.

III. Modeling unknown past security exposures

Since the construction of a software program, errors are unintentionally introduced producing security vulnerabilities. At a certain time, system administrators, security modules or self-protection components, system security entities from now, may be unaware of these issues permitting attackers to take advantage of them and to breach security measures without being noticed. However, the awareness of such potential attacks later in time provides the ability to inspect possible security breaches and to take actions to ensure the security of the system. In this section we present a mathematical model that defines and supports the process for detecting past unknown security exposures.

A. Understanding past unknown security exposures

Security exposures can inadvertently occur during long periods of time. Unaware of this fact, systems become victims of unnoticed security incidents that may compromise their information and functionalities in the long term. Once a vulnerability has been introduced in a software program, a sequence of events constitutes what is called the vulnerability lifecycle [21] described in Fig. 1. Event 1 indicates the vulnerability creation time denoted by t_{creat}. Event 2 records the time where the vulnerability is discovered, specified by t_{disco}. Event 3 denoted by t_{explo} indicates the first time an exploit becomes available. Its disclosure time specified by t_{discl} occurs in event 4 where the vulnerability information becomes freely available to the public. Since the vulnerability discovery time until this point, the information about it is considered as private knowledge denoted by $\triangle_{private} = t_{discl} - t_{disco}$. Beyond this point, system security entities may acknowledge its existence. Event 5 indicates the time where a vulnerability countermeasure becomes available, denoted by t_{count}. Vulnerable states may be partially mitigated by performing certain actions that do not correct the problem but avoid it to be exploited. Since an exploit exists until this point, systems are vulnerable to security attacks. This period is denoted by $\triangle_{vulnerable} = t_{count} - t_{explo}$. Event 6 specified by t_{patch} indicates the time where a patch becomes available to the public. System security entities can install this patch in order to eradicate the vulnerability.

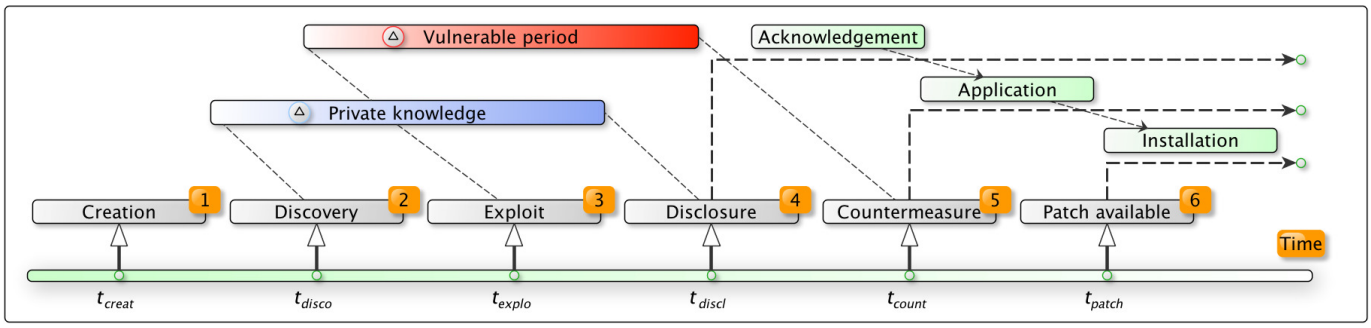

Fig. 1: Vulnerability lifecycle events

Based on the previous definitions we define an unknown past security exposure as an exploitable vulnerable state that exposes a system to security threats during a certain period of time ($\triangle_{vulnerable}$) in which neither the system nor its security entities were aware of such security weakness. In order to unveil such security exposures, an infrastructure capable of managing snapshots of the system across time would be able to analyze past system states by taking advantage of current security information. In this manner, exposure time gaps can be detected in order to perform further analysis such as forensic activities over valuable assets. In the next section we present our model for supporting the proposed infrastructure.

B. Modeling and detecting unknown past security exposures

In order to define a mathematical specification of unknown past security exposures, we first introduce a set of core definitions that constitute the main building blocks of the model. We present here three definition groups: (1) domains, (2) predicates and (3) functions, that are used for defining how a system is evaluated in order to detect past exposures. The universe of discourse is constituted by the following domains:

- $P = \{p_1, p_2, \dots\}$ denotes the set of device properties in the form of unary predicates $p_i(h)$ where h is the device under analysis.
- $S = \{s_1, s_2, \dots\}$ denotes the set of device states where a state s_i is used for describing in a compact manner the set of properties required to be observed over the device as well as for describing existing specific device states.
- $R = \{r_1, r_2, \dots\}$ denotes the sequence of system revisions (snapshots) ordered through time.
- $V = \{v_1, v_2, \dots\}$ denotes the set of known vulnerability definitions and it is also called the knowledge source.

The predicates applied over individuals of our discourse universe are defined as follows:

- All the defined domains act as membership predicates, e.g., $R(r)$ is $true$ if and only if r is a system revision.
- $isVulnerable : S \times V \rightarrow Boolean$ denotes a predicate that takes a system state $s \in S$ and a vulnerability definition $v \in V$ as input and returns $true$ if and only if the vulnerability v is present in the system state s.
- $isNew : V \rightarrow Boolean$ denotes a predicate that takes a vulnerability definition $v \in V$ as input and returns $true$ if and only if the vulnerability v is new within the current knowledge source.

The functions used in the approach are the following:

- $revision : N \rightarrow R$ denotes a function that takes a revision number $n \in N$ as input and returns the associated system revision $r \in R$.
- $number : R \rightarrow N$ denotes a function that takes a revision $r \in R$ as input and returns the associated number $n \in N$.
- $state : R \rightarrow S$ denotes a function that takes a revision $r \in R$ as input and returns its associated state $s \in S$.
- $time_R : R \rightarrow N$ denotes a function that takes a system revision $r \in R$ as input and returns the time elapsed since the revision was created.
- $time_V : V \rightarrow N$ denotes a function that takes a vulnerability definition $v \in V$ as input and returns t_{explo} if known, otherwise t_{discl} is returned.

Based on the previous core definitions, we define $E(R, V)$ shown in Equation 1 as a predicate that based on a revision history R and a vulnerability knowledge source V, indicates if the system under analysis has been unknowingly exposed in the past.

$$E(R,V) = \exists(r)\ \exists(v) \quad (R(r) \wedge V(v) \wedge isNew(v) \wedge time_V(v) \leq time_R(r) \wedge isVulnerable(state(r), v)) \tag{1}$$

Equation 1 mathematically states the main concept of our approach. If a new vulnerability is available, and exists at least one system revision made after the exploit was created or the vulnerability was disclosed, and such revision is found to be vulnerable, then the system has been unknowingly exposed in the past, even if the vulnerability is not observable in the present configuration. In the next section we present a framework capable of capitalizing security advisories and analyzing historical revisions in order to detect and warn about unknown past exposures.

IV. An OVAL-based framework for detecting past hidden vulnerable states

Detecting past unknown security exposures relies on the ability to see beyond the current status of a given system. In order to achieve this goal, we propose a distributed framework capable of organizing historical information about computing systems and analyzing them when new security information is available. In this section we present the overall architecture and explain our strategy for detecting past security exposures by taking advantage of new advisories over past system states.

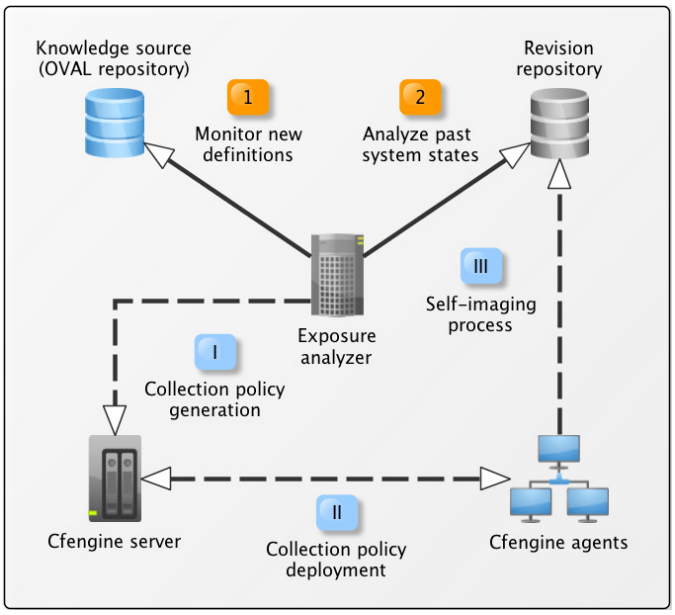

Fig. 2: High-level imaging and exposure detection process

A. Architecture overview

In order to build a framework capable of identifying security exposures in the past, we consider two independent cyclical processes. One process for imaging systems in an autonomous manner and the second one for actually detecting past security exposures. Fig. 2 illustrates the proposed architecture identifying the main components as well as the communication processes between them. The sequence denoted by Steps I, II and III constitutes the image generation process. At Step I, the exposure analyzer provides directives for data collection that will be used for building system images. These directives are specified by means of OVAL documents that are automatically translated to Cfengine policy rules. The ability to express OVAL objects without actually expecting any particular state allows us to use OVAL documents as the inventory of required objects to be collected. At Step II the generated Cfengine policy rules are transmitted to the autonomic agents distributed in the network. These agents are in charge of controlling network devices and they will perform data collection activities in order to build their system images. Finally, these images are automatically stored in the revision repository at Step III. The image generation process constitutes an autonomic activity and it is performed independently from the past exposure detection process. The latter is composed by two steps. First at Step 1, the exposure analyzer monitors the knowledge source on a regular basis checking for new vulnerability definitions. When new definitions become available, it analyzes system images stored in the revision repository at Step 2 in order to detect past unknown security exposures.

The framework proposed in this work can be easily coupled to autonomic frameworks that perform assessment activities such as the one presented in [15]. In this manner, a combined solution of past and present vulnerability assessment could highly increase the security of autonomic environments. Moreover, forensic activities could be partially automated by collecting forensic evidence using machine-readable procedures that may warn administrators about past exposures and current threats [17]. In addition, the proposed architecture allows to outsource assessment activities. By analyzing system images, target devices may provide the required data while the exposure analyzer may perform a security evaluation of it. As we mentioned before there is a period of time, noted $\triangle_{private}$, where the information about a vulnerability is not publicly known. The capability of outsourcing security assessment activities allow organizations to perform analysis with their own information without actually violating their disclosure restrictions. At the same time, clients can be warned about security exposures and be advised about actions to take without knowing internal mechanisms for detecting such threats. In the next section we illustrate our strategy for assessing and detecting unknown security exposures in the past.

B. Assessment strategy

Given a new vulnerability definition, the objective of our strategy is to identify affected system states across time after its exploit was publicly available. This process, depicted in Algorithm 1, provides a period of time where the system was potentially exposed to the security threat represented by the specified vulnerability. First, the exposure time is set, depending on the available information, between the exploit or vulnerability disclosure time and the current time (lines 1-2). Then, a sequence of available system revisions during this period is gathered, ordered by time starting with the newest revision first (line 3). For each revision (line 4), the system state is analyzed cheking if the specified vulnerability is present or not (line 5). If the system state is found to be vulnerable, the algorithm takes the longest period of potential exposure. If the vulnerable system state is not the newest one (lines 6-7), the exposure end time is set to the next revision time found not vulnerable (line 8). If the vulnerable system state is the newest one, the exposure end time is set to the current time. Finally, the exposure time is returned (line 10). This strategy has been integrated within our implementation prototype which is the heart of the next section.

Input: Vulnerability v
Output: ExposureTime e

1 $e.startTime \leftarrow time_V(v)$;
2 $e.endTime \leftarrow now()$;
3 $revs \leftarrow getRevsFromTo(e.endTime, e.startTime)$;
4 **foreach** *Revision* $r \in revs$ **do**
5 **if** $isVulnerable(state(r), v)$ **then**
6 $nextRev \leftarrow revision(number(r) + 1)$;
7 **if** $nextRev \in revs$ **then**
8 $e.endTime \leftarrow time_R(nextRev)$;
9 **end**
10 $return\ e$;
11 **end**
12 **end**

Algorithm 1: Exposure assessment algorithm

V. Implementation prototype

The actual implantation of the framework previously described requires several challenges to be addressed. First, a mechanism for describing and automatically generating and deploying system images or snapshots is required. Second, an efficient representation and storage approach able to scale with the size of the system needs to be incorporated. Third, tools and techniques for assessing system images must be provided. In this section we present our implementation prototype as well as the main artifacts that constitutes the proposed solution.

In our previous work [15] we have proposed an OVAL-based approach for increasing the security awareness of autonomic environments that relies on the Cfengine system [2], an autonomic policy-based network management system. In that work, OVAL advisories are automatically integrated into the autonomic management plane by means of automatic policy generation that represents such security information. The policy generation is performed by Ovalyzer, an OVAL to Cfengine translation system. We consider here a similar approach for generating data collection policy rules that will be used for automatically building system images. Under this perspective, autonomic agents can decide given high-level objectives, when to perform new system revisions based on different factors such as system changes and programmed tasks. While data collection policies are specified as OVAL definitions that indicate what to collect, an OVAL system characteristics document [9] includes all the information required for outsourcing vulnerability assessment activities.

Computing systems are usually constituted by large sets of configuration files and data making it hard to build historical repositories of system images. Considering the XML-based representation used in the OVAL language, we take advantage of the SVN[5] versioning system for efficiently representing past system states. Each system image is constituted by system properties specified as OVAL tests that indicate which OVAL objects must be collected. Fig. 3 shows how system properties are efficiently stored by means of an SVN repository. The main idea is that after a baseline representing the system has been made (time $T1$), only properties modified by system changes are stored within new revisions of the SVN repository. Within the example, a new system image is generated at time $T2$ when properties 2 and N are modified, now represented as 2.1 and $N.1$. At time $T3$, properties 1, 3 and N are changed and a new revision is created. Within this scenario, the latest system image can be built by taking the latest modifications of each property following the solid line (1.1, 2.1, 3.1, $N.2$). The same idea can be applied over any revision to analyze system images in the past. Our prototype uses the SVNKit [8] technology for performing activities over the SVN repository.

When new vulnerability definitions become available (transition between repository $v1$ and $v2$), the exposure analyzer (see Fig. 2) will assess those devices under control traversing the history of system images as explained in Algorithm 1. The assessment over the baseline detects one vulnerability

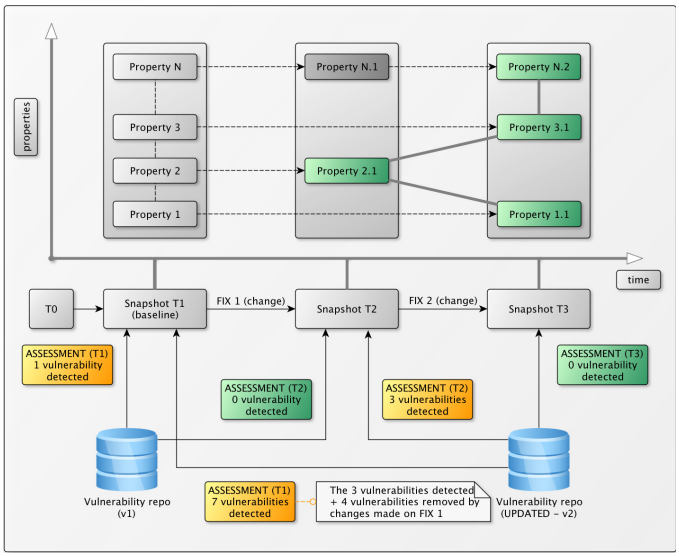

Fig. 3: SVN-based assessment

and corrective changes are performed generating $T2$. When the repository is updated ($v2$), the exposure analyzer uses this new information for assessing past system states. In the example, seven vulnerabilities are detected at $T1$ while three are identified at $T2$. It can be inferred that four vulnerabilities have been removed by changes made between $T1$ and $T2$. Corrective modifications in snapshot $T2$ produces a new snapshot at time $T3$ where no more known vulnerabilities are detected by the exposure analyzer with the repository $v2$.

In order to analyze the exposure of past system states, we have extended XOvaldi [17] for assessing system images represented by means of OVAL system characteristics files. XOvaldi is a live forensic, multi-platform and extensible OVAL-based system analyzer. The proposed extension implemented in this work allows XOvaldi to outsource vulnerability assessment activities. By consuming OVAL system characteristics files, XOvaldi is not required to be executed on each device under control. Instead, system images are generated independently, in this work by means of collection policies generated by Ovalyzer [15], and preserved in an optimized storage system using the SVN technology. By using XOvaldi services, the exposure analyzer is able to evaluate and detect past system exposures due to unknown vulnerabilities in an independent manner. In the next section we present a case study where a comprehensive set of experiments has been made for determining the feasibility and limits of our solution.

VI. A case study

Past system security exposures can provide unnoticed pathways for performing attacks on current system states. In this section we present a case study based on the IOS[6] platform for Cisco devices. We illustrate the application of the proposed approach over an emulated environment using the GNS3 emulator [4] over a regular laptop (2 Ghz Intel Core i7 with 8GB RAM) and present the obtained results.

[5]Apache Subversion [1]

[6]Internetwork Operating System

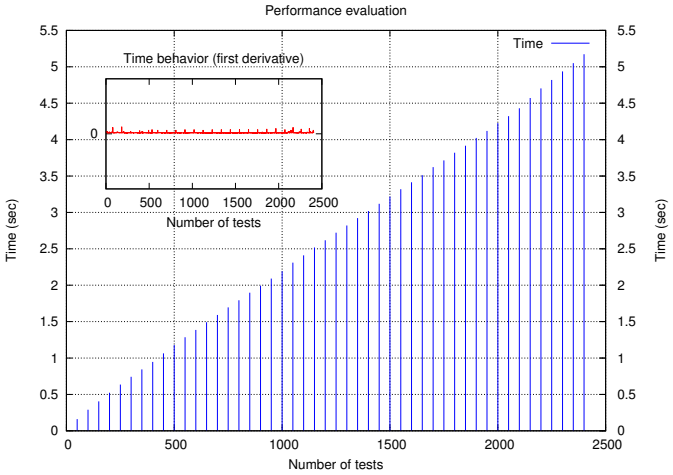

Fig. 4: Tests assessment time

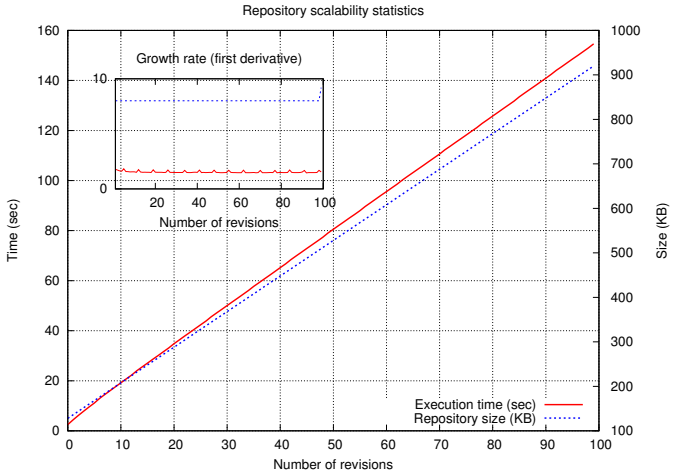

Fig. 5: Repository scalability statistics

The complexity involved within each vulnerability description usually depends on its very own nature, meaning that some vulnerability definitions may require a small set of tests to be evaluated while others may need a higher amount of systems checks. In order to avoid such nature-based size discrepancy among different vulnerability descriptions, we have increased the granularity of our experiments by independently analyzing the involved OVAL tests. Assessing the whole set of OVAL definitions for the IOS platform requires the evaluation of approximately 2400 OVAL tests. Fig. 4 illustrates the accumulated time required for assessing each system property involved in the IOS vulnerability descriptions. Within the performed experiments, it takes approximately 5 seconds for evaluating the whole set of involved OVAL tests. We also observe a linear time growth rate when the number of OVAL tests is increased as depicted in the inner graph, meaning that the proposed approach scales properly regarding the size and nature of the IOS vulnerability descriptions.

As we mentioned before, a storage mechanism able to scale with the size of system images is imperatively required. In order to measure the efficiency of our SVN-based implementation, we have performed experiments to analyze both the size of the repository and the assessment time required for evaluating the history of past system states when new revisions are generated. Fig. 5 shows the behavior of our prototype when the number of system images is increased from 1 to 100. We have cyclically analyzed the required assessment time (red solid line) when a new vulnerability definition becomes available over the proposed image range. As expected, it can be clearly identified a linear time growth along the number of revisions augments. In addition, the repository size (blue dashed line) also presents a stable growth rate in terms of storage requirements as shown in the inner graph. The frequency with which a network device changes its configuration can vary among platforms and usage. Nonetheless, our experiments over the IOS platform show that our prototype is capable of preserving a history of about 1 year with system images performed every 4 days in less than 1 MB of storage space. In addition, the assessment time of the whole year history can be performed in less than 3 minutes. We have scheduled several improvements for optimizing our solution though current experiments already show the feasibility and scalability of the proposed approach.

VII. Conclusions and Future Work

Vulnerability assessment tasks constitute a critical activity that is usually performed only over running systems. However, even though a known vulnerability may not be present on a current system, it could have been unknowingly active in the past providing an entry point for attacks that may still constitute a potential security threat in the present. In this paper we have proposed an approach for increasing the overall security of computing systems by identifying past hidden vulnerable states. In that context, we have proposed a mathematical model for describing and detecting unknown past security exposures as well as an OVAL-based framework able to autonomously build and monitor the evolution of network devices and also to outsource the assessment of their exposure in an automatic manner. We also have developed an implementation prototype that efficiently performs assessment activities over an SVN repository of IOS system images. In addition, our experiments confirm the feasibility and scalability of our solution.

The integration of vulnerability management mechanisms into autonomic environments poses hard challenges. For future work, we plan to analyze decentralized mechanisms and optimized strategies for enabling agents to independently assess their own past exposure by providing downloadable exposure analyzers. Thus, autonomous agents can perform actions on their own in order to move up to secure states. We understand that real autonomy can be accomplished if devices are capable of performing corrective tasks as well. In that context, we also plan to analyze strategies for performing appropriate corrective activities in the present based on automated forensic investigations over past system states.

Acknowledgements

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

REFERENCES

[1] Apache Subversion. http://subversion.tigris.org/. Last visited on August, 2012.

[2] Cfengine. M. Burgess. http://www.cfengine.org/. Last visited on May, 2012.

[3] CVE, Common Vulnerabilities and Exposures. http://cve.mitre.org/. Last visited on March, 2012.

[4] GNS3. http://www.gns3.net/. Last visited on August, 2012.

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

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

[7] Ovaldi, the OVAL Interpreter reference implementation. http://oval.mitre.org/language/interpreter.html. Last visited on March, 2012.

[8] SVNKit. http://svnkit.com/. Last visited on August, 2012.

[9] The OVAL Language. http://oval.mitre.org/. Last visited on May, 2012.

[10] A Road Map for Digital Forensic Research. In *Report From the First Digital Forensic Research Workshop (DFRWS)*, August 2001.

[11] M. Abedin, S. Nessa, E. Al-Shaer, and L. Khan. Vulnerability Analysis for Evaluating Quality of Protection of Security Policies. *Proceedings of the 2nd ACM Workshop on Quality of Protection (QoP'06)*, 2006.

[12] H. Achi, A. Hellany, and M. Nagrial. Network Security Approach for Digital Forensics Analysis. *Proceedings of the International Conference on Computer Engineering and Systems (CCES 2008)*, pages 263–267, November 2008.

[13] 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)*, pages 722–725, April 2008.

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

[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)*, October 2011.

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

[17] 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.

[18] 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.

[19] 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.

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

[21] S. Frei, D. Schatzmann, B. Plattner, and B. Trammel. Modelling the Security Ecosystem - The Dynamics of (In)Security. In *Proceedings of the Workshop on the Economics of Information Security (WEIS'09)*, June 2009.

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

[23] M. A. Rahman and E. Al-Shaer. A Declarative Approach for Global Network Security Configuration Verification and Evaluation. In *Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management (IM'11)*, pages 531–538. IEEE, May 2011.

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

[25] K. Scarfone and T. Grance. A Framework for Measuring the Vulnerability of Hosts. *Proceedings of the 1st International Conference on Information Technology (ICIT'08)*, pages 1–4, May 2008.

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

[27] A. Williams and M. Nicolett. Improve IT Security with Vulnerability Management. http://www.gartner.com/id=480703, 2005. Last visited on March, 2012.

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