POLITECNICO DI TORINO
Repository ISTITUZIONALE

Online and offline security policy assessment

Original
Online and offline security policy assessment / Valenza, F., Vallini, M., Lioy, A.. - STAMPA. - (2016), pp. 101-104. (MIST'16: 8th ACM CCS international workshop on Managing Insider Security Threats Vienna (Austria) October 28, 2016) [10.1145/2995959.2995970].

Availability:
This version is available at: 11583/2650400 since: 2021-01-28T18:45:06Z

Publisher:
ACM

Published
DOI:10.1145/2995959.2995970

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

14 October 2026

Online and offline security policy assessment

Fulvio Valenza
Politecnico di Torino
Dip. Automatica e Informatica
Torino, Italy
fulvio.valenza@polito.it

Marco Vallini
Politecnico di Torino
Dip. Automatica e Informatica
Torino, Italy
marco.vallini@polito.it

Antonio Lioy
Politecnico di Torino
Dip. Automatica e Informatica
Torino, Italy
antonio.lioy@polito.it

ABSTRACT

Network architectures and applications are becoming increasingly complex. Several approaches to automatically enforce configurations on devices, applications and services have been proposed, such as Policy-Based Network Management (PBNM). However, the management of enforced configurations in production environments (e.g. data center) is a crucial and complex task. For example, updates on firewall configuration to change a set of rules. Although this task is fundamental for complex systems, few effective solutions have been proposed for monitoring and managing enforced configurations. This work proposes a novel approach to monitor and manage enforced configurations in production environments. The main contributions of this paper are a formal model to identify/generate traffic flows and to verify the enforced configurations; and a slim and transparent framework to perform the policy assessment. We have implemented and validated our approach in a virtual environment in order to evaluate different scenarios. The results demonstrate that the prototype is effective and has good performance, therefore our model can be effectively used to analyse several types of IT infrastructures. A further interesting result is that our approach is complementary to PBNM.

Keywords

policy verification, policy assessment, configuration analysis

1. INTRODUCTION

In recent years network architectures and applications have become increasingly complex. Virtualization and Network Function Virtualization (NFV) techniques reduce hardware cost but increase management complexity.

Several approaches to automatically configure devices, application and services have been proposed in the field of Policy-Based Network Management (PBNM). These typically perform automatic configuration of devices and services from scratch, by defining high-level policies and hiding refinement process details.

However, the management of enforced configurations in production environments (e.g. data center) is a crucial and complex task. Think, for example, of updates to a firewall configuration. In particular, the problem is twofold: it requires high accuracy (e.g. to identify and perform precise modification on configuration settings) and it must limit service downtime. The management task also requires high skills and is typically expensive. For these reasons this operation should be automated. Although this task is fundamental for complex systems, few effective solutions have been proposed.

This work introduces a novel approach to handle configurations in a production environment based on (1) a formal model to identify/generate the traffic flow and to verify the enforced configurations, and (2) a slim and transparent framework to perform the policy assessment. An additional benefit is that this approach is complementary to PBNM.

2. MOTIVATION

The automatic deployment of an application instance is a common feature of recent virtualization platforms (e.g. OpenMano), on the other side, the verification and management of its configuration are currently not well addressed. A typical provisioning system does not periodically check the configuration settings of a instance. Often, this operation is performed manually, by an administrator, in case of failure or misbehaviour.

This approach has at least two drawbacks. First of all, it is error-prone and expensive task because it is performed by humans. Second, it does not address misconfigurations that do not affect service operation. For example, an attacker could add a rule on a firewall to mirror traffic to another system. In this case, the service operates correctly but its configuration is altered and the attack succeeded. Therefore, to avoid these situations, automatic and continuous monitoring of service configurations is mandatory.

The most simple approach is to periodically verify the integrity of enforced configurations, and in case of changes, redeploy the correct configuration. However, this is not effective solution because it does not detect which part of the configuration is changed (e.g. rules or part of them). On the contrary, our solution performs an accurate identification of altered settings and supports the redeployment of them.

This work proposes online and offline analysis in order to detect when high-level policies are not correctly enforced into deployed configuration. This approach introduces into the PBNM the management of deployed configurations. The provided solution also permit the detection of configuration errors and attacks (e.g. rule modifications due to human error or malicious manipulation).

ACM ISBN XXX.
DOI: XXX

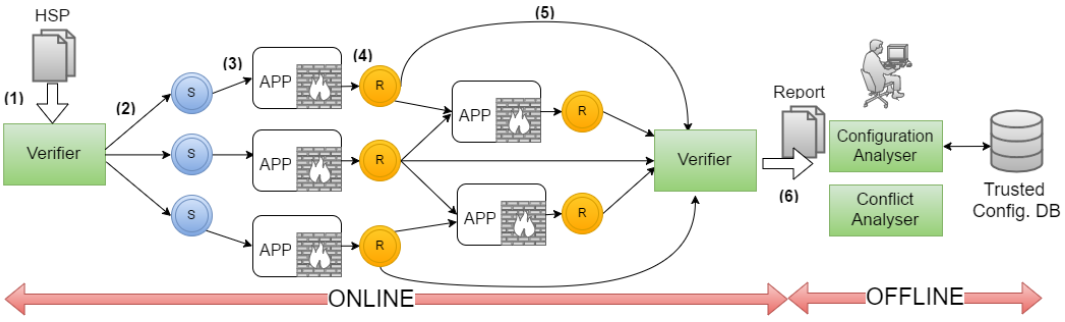

Figure 1: Verification workflow.

3. APPROACH

The goal of our work is to verify the implementation of policies in the network and, in case of failure, performing a detection of the causes, identifying mistakes and anomalies in the network configuration. To achieve this goal, first of all, we have defined a High Level Security Policy Language (HSPL), useful to define polices of different security controls (such as packet filter, application layer firewall, content inspection) by using an abstract and high-level approach. This clearly simplifies the policy authoring task for network administrators.

Starting from a set of HSPL statements, defined by an administrator, we designed a workflow to verify the correct enforcement of configurations for the security controls in the network. The wrokflow consists of several phases and involves different components. As depicted in Figure 1 some phases are performed online (i.e. dynamic generation of network packets) and others offline (i.e. static analysis of policies and configurations). Briefly, the main steps of the verification process are:

1. the Verifier receives the security policies (expressed in the high-level language) and computes the set of network packets that has to be generated in order to verify the policies correctness;
2. the Verifier configures the Sender nodes by delegating to them a sub-set of the packets; each packet will be generated to check the correctness of a security control;
3. a Sender node generates the set of packets for a specific security control;
4. a Receiver node collects the outgoing packets of a security control to send a summary to the Verifier and forwards these packets to the next hop of the chain;
5. the Verifier collects the summaries from all the Receivers and analyses those to check the policy correctness;
6. the Verifier generates a report and sends it to the administrators and to the components in charge of detecting the causes of the misconfiguration (namely Configuration Analyser and Conflict Analyser).

The next paragraphs describe the components involved in the process.

High Level Security Policy Language.

Although the full specification of the HSPL is out of the scope of this paper, here we provide a brief introduction to this language.

As discussed before, the administrator specifies the security requirements with the HSPL. Starting from previous works [1], we designed HSPL as an authorization language that follows the *subject-action-object-attribute* paradigm (also referred to as *target-effect-condition*) [2]. More precisely, a HSPL policy is a statement in the following form:

```
[sbj] action obj {(field_type,value),...}
```

A security requirement is expressed as a set of sentences close to natural language, e.g. "do not access gambling sites", "allow Internet traffic from 8:30 to 20:00 for employees". The elements of a sentence (subject, object, etc.) are chosen by the administrator from a predefined set and implemented in a editor as different lists, i.e. a list for each element (e.g. action, subject). This approach is transparent for administrators (avoiding them to learn a new language) and makes it possible to map each element of a sentence to the related HSPL component. It is clear that, an administrator can customize some elements of a sentence, for example to define timing constraints or specify a particular URL. The simplicity and completeness of this language help unskilled users (such as a business manager) to define the set of security policies.

Verifier.

This is the core component in the workflow and it performs two tasks: (1) computes the set of network packets in order to verify the policy correctness, and (2) analyses the summary generated by Receiver nodes in order to identify when a policy is not correctly enforced for a security control.

The definition of the required network packets is based on a formal model that adopts the same approach of refinement techniques. The input of this process are the HSPLs and the output are the type of traffic that each Sender nodes must generate.

More precisely, for each receiver node the Verifier produces two tables: the former contains (a) the set of packets that must pass through the receiver nodes and (b) the relative HSPLs, while the latter contains (c) the set of packets that must be blocked before reaching the received node and (d) the relative HSPLs. When the Verifier receives the summary, by the sender nodes, it analyses whether a HSPLs is correctly enforced.

Sender nodes.

Periodically, the sender nodes receive a set of packet parameters from the Verifier and generate the network traffic for a specific security control. Depending on the type of security control, the sender can generate different types of traffic (e.g. HTTP, SMTP, FTP). A sender node does not require particular hardware resources (CPU, memory), it only must support a traffic generator tool (such as Scapy, netsniff-ng, packet sender).

Receiver nodes.

The receiver nodes are transparent to the traffic flows. These nodes only observe the outgoing packets of a security control and produce a summary for the Verifier. For this reason it possible to place a receiver node directly into a security control or external to it. Similarly to the senders, a receiver node does not require particular hardware resources. These nodes only summarize the received traffic by using specific tools (such as wireshark, packetyzer, ostinato).

Report.

The Verifier summarizes the traffic information collected from the receiver nodes and produces two reports: the former, it is machine readable and the latter is human readable and it is intended for administrators.

Both reports contain for each packet (generated by a Sender node) three sets: the traversed receiver nodes, the correctly enforced HSPLs and the violated HSPLs.

The machine readable report is useful for correlating the analysis of Verifier with the analysis executed by the Configuration Analyser and Conflict Analyser. In addition can be used by a process that reads the report and, whether a policy has not been correctly enforced, sends an alert (e.g. by email) to administrator.

Conflict Analyser.

The Verifier also performs conflict analysis, a technique (applied to different types of policies) that identifies anomalies within a single policy (intra-policy) or between multiple policies (inter-policy). Policy anomalies are not always errors but can lead to misbehaviour of the policy enforcement. Policy anomaly can be compared with compilation warnings of computer programs, where a program is correctly compiled and it can run normally in most cases, but for some execution stages, the program could fail. This analysis is performed offline, i.e. directly evaluating configuration settings, without sending packets.

Policy anomalies are studied for filtering and IPsec policies. In particular filtering anomalies are well studied in literature. The definition of Conflict Analyser is out of the scope of this paper, therefore we refer to our existing works [3, 4, 5].

Configuration Analyser.

Once the Verifier detects which security control is not correctly configured, the goal of Configuration Analyser is to identify which rule(s) are added, deleted or modified. Similarly to Conflict Analyser this analysis is performed offline. In practice, our approach compares the trusted rules (which express the expected behaviour of the system) with the enforced rules deployed into security controls. Trusted rules are stored into Trusted configuration DB (depicted in Figure 1) and could be generated automatically, by a refinement process, or manually, by an administrator. Clearly, before updating the database with trusted rules, these must be tested on the system by using the proposed approach. As discussed before, security policies are described by using a high-level format, however security controls typically have low-level specific format. Therefore, in order to compare trusted with enforced configurations the Configuration analyser transforms security rules into an intermediate format. The main steps of the Configuration Analyser are: 1. collect the configuration rules of the security control to analyse 2. transform these rules into a intermediate model built upon a matrix based format 3. compare trusted rules with the ones collected from security control (enforced rules) and identify which of them are added, deleted or modified 4. notify the results to the user and suggest to him a possible remediation (e.g. re-deploy the trusted configuration into the security control)

4. IMPLEMENTATION

In order to obtain performance and scalability tests we implement our approach by using a virtual environment. As widely discussed in several works the adoption of a virtual environment has many advantages. In particular planning, management and deployment tasks are more simple than in a physical environment. Another important benefit is that each operation (e.g. deployment of a new virtual machine) can be performed on demand with very limited effort (e.g. starting from a virtual machine template).

For the definition and management of the virtual environment we use KVM[1] and Libvirt[2]. **KVM** (Kernel-based Virtual Machine) is a open source visualization solution native for Linux kernel. KVM framework allows to virtualise the physical hardware to run multiple virtual machines, each of them with specific configuration. **Libvirt** is a toolkit to interact with the virtualization capabilities of recent versions of Linux (and other OSes). Libvirt includes several commands to create, run or pause a virtual machine in different virtualization solution (KVM, Xen, VMware, ESX, QEMU).

We implement the sender and receiver node by using Scapy[3], a tool to generate network packets. More precisely, the verifier instructs each Sender node by passing the Scapy commands to generate the required packets. Then, each Receiver node sends to the verifier the summary of the received packets (these are captured by using Scapy). Finally, the models in the verifier module are developed by using Java 1.7 and two Java-based open-source frameworks: Drools[4] and MOEA[5].

Drools is a Rule Engine that uses a rule-based approach to implement a expert system. A rule is composed by a condition and a consequential action (that is triggered when the condition is matched). At system runtime, the conditions are evaluated following the processed data (i.e. facts) and the consequential actions are executed. In particular, we adopt the Drools engine to infer the set of packets that matches (i.e. verifies) each HSPL policy.

The MOEA framework is an open source Java library for developing multi-objective evolutionary algorithms. MOEA has been used to develop the optimization model.

Finally, our prototype uses the Apache Commons Mathematics Library[6] to implement the other parts of the verification model.

5. RELATED WORK

This section gives a short description of the current state of the art for firewall analysis. This description may help the reader for better understanding the proposed solution. In literature this topic is divided into three major groups: firewall testing and firewall verification.

5.1 Firewall Testing

The firewall testing approach consists on generating a set of packets to evaluate firewall decisions that are known a priori. If the firewall decision for each packet is the same as expected decision the firewall configuration is correct, otherwise there are some errors. In literature many works are based on a firewall testing approach with different testing methods.

[1] http://www.linux-kvm.org/
[2] http://libvirt.org/
[3] http://www.secdev.org/projects/scapy/
[4] http://www.drools.org/
[5] http://moeaframework.org/
[6] http://commons.apache.org/proper/commons-math/

Hoffman [6] presented a framework to generate packet streams by using covering arrays, production grammars, and replay of captured TCP traffic. This framework, namely "Blowtorch" also supports packet timing, traffic capture and replay and it is useful to check handshaking.

Jürjens [7] proposed a method for specification-based testing, enabling to formally model a firewall, surrounding network and to mechanically derive test cases to check for vulnerabilities.

El-Atawy [8] introduces a firewall testing technique by using policy-based segmentation of the traffic address space. This can adapt the traffic generation test considering potential erroneous regions in the firewall input space.

Senn [9] presents an approach to test the conformance of firewalls to a given security policy. The main contributions are: a language for the formal specification of network security policies, the combination of different methods for generating abstract test cases, and an algorithm for generating concrete test cases from the policy.

5.2 Firewall Verification

Firewall verification method evaluates whether a set of given properties are satisfied in a firewall.

Mayer et al. present "Fang" a firewall analysis engine useful to discover and test the global firewall policy (either a deployed policy or a planned one), allowing user queries for the purpose of analysis and management [10].

Lui et al. in their work face two problems: how to describe a firewall query and how to process a firewall query in order to make firewall queries practically useful. In [11] they introduce a simple SQL-like query language for describing firewall queries and present a query processing algorithm that uses firewall decision diagrams as data structure.

Finally, recent literature has extended the firewall analysis towards the verification of middleboxes. Middleboxes are stateful network functions that process traffic based on their configurations and internal states, like a learning firewall. Under this umbrella, Spinoso *et al.* [12] model middleboxes and networks as a set of logic formulas and use a solver to verify a set of policies in the network model.

6. CONCLUSIONS

This paper presents a novel approach to monitor and manage enforced configurations in a production environment. The methodology is extremely useful in many ways: for example, it can be used to detect errors both in the firewall implementation and firewall management. Our contributions are: (1) a formal model to identify/generate the traffic flow and to verify the enforced configurations; (2) a slim and transparent framework to perform the policy assessment. We have implemented and validated our approach in a virtual environment to evaluate different scenarios. The experimental results demonstrate that the prototype is effective and has good performance, therefore our model can be effectively used to analyses several types of IT infrastructures. Currently, the methodology has been implemented only for a limited set of HSPL policies, mainly related to filtering requirements (stateful packet filters and L7 filters). However, the proposed approach can be easily extended, by adding new security feature and/or VNFs. Therefore, as future work, we will extend it by adding other types of security functions (e.g. VPN, proxy, IPS/IDS). Finally, this work is clearly useful do detect changes in configurations settings of security controls. Therefore, this approach cannot be directly adopted for detecting the attacks that modify software code, e.g. to change the behaviour of an application. However, this type of attack can be identified by using trusted computing techniques, for example by performing remote attestation of the running software on a platform equipped with the Trusted Platform Module (TPM) or a virtual version of it (virtual TPM). Therefore, trusted computing techniques will be investigated in future work to manage attacks related to software modification.

Acknowledgment

The research described in this paper is part of the SHIELD project, co-funded by the European Commission (H2020 grant agreement no. 700199).

7. REFERENCES

[1] C. Basile, A. Lioy, C. Pitscheider, F. Valenza, and M. Vallini, "A novel approach for integrating security policy enforcement with dynamic network virtualization," in *Netsoft-2015: 1st IEEE Conf. on Network Softwarization*, April 2015, pp. 1–5.

[2] S. Godik, A. Anderson, B. Parducci, E. Damiani, P. Samarati, P. Humenn, and S. Vajjhala, "eXtensible Access Control Markup Language (XACML) Version 3.0," OASIS, Tech. Rep., January 2013.

[3] C. Basile, D. Canavese, A. Lioy, C. Pitscheider, and F. Valenza, "Inter-function anomaly analysis for correct sdn/nfv deployment," *International Journal of Network Management*, vol. 26, no. 1, pp. 25–43, 2016.

[4] C. Basile, D. Canavese, A. Lioy, and F. Valenza, "Inter-technology conflict analysis for communication protection policies," in *CRiSIS-2014: 9th International Conference on Risks and Security of Internet and Systems*, August 2014, pp. 148–163.

[5] F. Valenza, S. Spinoso, C. Basile, R. Sisto, and A. Lioy, "A formal model of network policy analysis," in *RTSI-2015: 1st IEEE International Forum on Research and Technologies for Society and Industry*, September 2015, pp. 516–522.

[6] D. Hoffman and K. Yoo, "Blowtorch: A framework for firewall test automation," in *ASE'05: 20th IEEE/ACM International Conference on Automated Software Engineering*, November 2005, pp. 96–103.

[7] J. Jürjens and G. Wimmel, "Specification-based testing of firewalls," in *4th International A.Ershov Memorial Conference*, July 2001, pp. 308–316.

[8] A. El-Atawy, K. Ibrahim, H. Hamed, and E. Al-Shaer, "Policy segmentation for intelligent firewall testing," in *NPSEC'05: 1st International Conference on Secure Network Protocols*, November 2005, pp. 67–72.

[9] D. Senn, D. Basin, and G. Caronni, "Firewall conformance testing," in *TestCom 2005: 17th IFIP TC6/WG 6.1 International Conferencen on Testing of Communicating Systems*, June 2005, pp. 226–241.

[10] A. Mayer, A. Wool, and E. Ziskind, "Fang: A firewall analysis engine," in *S&P 2000: IEEE Symposium on Security and Privacy*, May 2000, pp. 177–187.

[11] A. X. Liu and M. G. Gouda, "Firewall policy queries," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 6, pp. 766–777, June 2009.

[12] S. Spinoso, M. Virgilio, W. John, A. Manzalini, G. Marchetto, and R. Sisto, "Formal verification of Virtual Network Function graphs in an SP-DevOps context," in *ESOCC: European Conference on Service-Oriented and Cloud Computing*, September 2015, pp. 253–262.