An Architecture for Self-adaptive Distributed Firewall

Edmilson P. da Costa Júnior[1], Silas T. Medeiros[2], Carlos Eduardo da Silva[1], Marcos Madruga[2]

[1]Digital Metropolis Institute
Federal University of Rio Grande do Norte (UFRN)
Natal – RN – Brazil

[2]Departament of Informatics and Applied Mathematics
Federal University of Rio Grande do Norte (UFRN)
Natal – RN – Brazil

edmilsonjnior@info.ufrn.br,silastiagoo@gmail.com

kaduardo@imd.ufrn.br,marcos@dimap.ufrn.br

Abstract. *The notion of secure perimeter given by border firewalls ignores the possibility of attacks originating from inside the network. Although distributed firewalls allow the protection of individual hosts, the provided services might still be susceptible to attacks, as firewalls usually do not analyze application protocols. In this way, software vulnerabilities may be exploited until the problem has been fixed. From vulnerability discovery to the application of patches there is an exposure window that should be reduced. In this context, this paper presents an architecture for a distributed firewall system, in which a Vulnerability Assessment System is integrated for providing a self-adaptive mechanism capable of detecting vulnerabilities and executing actions to reduce exposure, contributing to mitigate the risk of vulnerability exploitation.*

1. Introduction

Several institutions nowadays, such as universities all over the world, deal with complex network infrastructure, involving an increasingly number of equipments (e.g., switches, routers) and servers, usually providing different services. Considering the diversity of activities and different research topics conducted throughout an university, it is common to find situations where several services, and servers, need to be provided for different groups of people, and more often than not, maintained by these different groups. This leads to an inconsistency in management and security procedures, where servers poorly configured and/or with outdated services, become potential targets for known vulnerabilities.

In this context, the traditional approach for network security, in which firewalls are deployed on the border of the network is no longer effective. Also known as centralized firewalls, these components are placed between the internal network of an institution and the Internet, with the objective of filtering the network traffic that goes in and out of the institution, limiting the services that would be exposed to the Internet, and defining the notion of a security perimeter. Clearly, this centralized model is not able to deal with attacks originated from inside the security perimeter [Ioannidis et al. 2000].

Today's technology movements, such as Bring Your Own Device (BYOD) and the availability of 3G/4G connections, mean that a malicious user has already penetrated the

border defenses. This is exacerbated when we consider university environments, which is usually open to the public in general, and contains a number of servers maintained by researchers, with outdated and potentially vulnerable services. Once one of those vulnerable servers is compromised, the attacker is free to search the network for other vulnerable servers, even though those servers are not exposed to the Internet. The bottom line is that, once an attacker compromises one of those vulnerable servers, he/she is inside the network, and the use of border firewall will contribute nothing in deterring his/her actions.

A solution for such scenario is the use of distributed firewall [Bellovin 1999]. A distributed firewall solution increments security by including firewalls in different points of the network and servers, besides the traditional border firewall. In this way, it is possible to control what services running on those servers are exposed on the network, and only for specific client hosts. However, the application of distributed firewall also brings some challenges, such as the management of these firewalls and their rules, and the response time in case of an incident. Traditional solutions for intrusion detection usually notify an administrator, which then assess and decides how to respond for rectifying and/or containing the situation [Meng et al. 2015]. However, this approach is usually not fast enough for avoiding service unavailability, information theft, or the infection of new systems/servers, mainly because attackers usually conduct their activities during strategic times, such as the middle of the night or weekends.

In this context, the contribution of this paper is an architecture for network security based on self-adaptive concepts. The motivation for using self-adaptation is the proven effectiveness and efficiency of self-adaptation in dealing with uncertainty in a wide range of applications, including those related to security [Bailey et al. 2014, Pasquale et al. 2012, Yuan et al. 2014]. The Self-Adaptive Distributed Firewall (SADF) architecture is based on the cooperation of different components usually found in a network infrastructure, such Vulnerability Assessment Systems and Configuration Management Systems. More specifically, we demonstrate how this architecture can be applied for managing a distributed firewall, in which possible threats can be detected (i.e., servers with vulnerabilities) and appropriate decisions be made for mitigating their impacts.

The remain of this paper is organized as follows: Section 2 contextualizes our work defining its scope and presenting some background on self-protection. Section 3 presents a conceptual view of the SADF architecture. Section 4 describes a prototype that has been implemented to demonstrate our approach feasibility. Section 5 discuss some related work. Section 6 concludes the paper.

2. Contextualization

As previously mentioned, due to the limitations of a centralized model, the border firewall does not protect equipment against internal attacks. This motivated the definition of a distributed firewall model [Bellovin 1999]. In a distributed firewall, security policies are defined in a centralized fashion using an specific language, and then distributed, by secure means, to be applied into different enforcement points. These enforcers can either be located on different segregation points inside the network, such as routers and switches, or on each host of the network [Ioannidis et al. 2000]. Figure 1 presents a general view of a network infrastructure where we can identify a Rules Management Server,

which is responsible for dealing with and distributing the firewall rules into the different enforcement points, in this case, firewalls in different servers of the network.

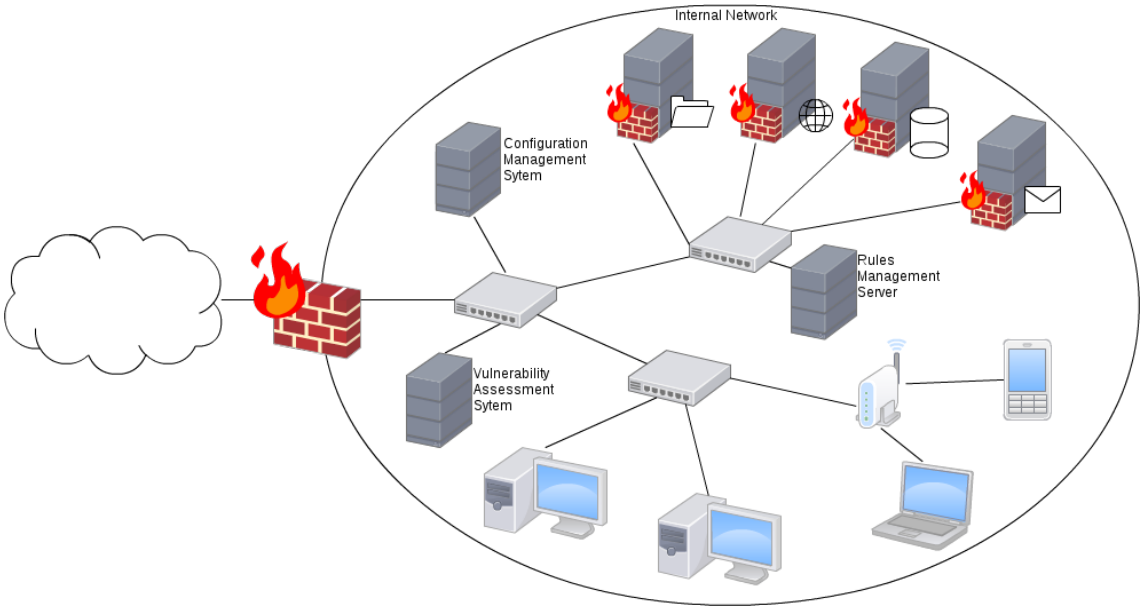

Figure 1: General view of the distributed firewall scope considered in this paper.

Operating and maintaining this infrastructure required a continuous effort as, even though there are different tools for facilitating management and maintenance actions, it is common to find out that most of these operations are still manually conducted. For example, as shown in Figure 1, the majority of institutions use, apart from firewalls, some sort of tool for configuration management, resource and service monitoring, and vulnerability assessment systems, which scans the network pointing out vulnerable services. These are important tools for maintaining the network infrastructure, but there is a lack of integration among them, requiring human intervention for conducting some tasks, and wasting valuable time between the moment an incident is detected and an administrator performs some corrective action to mitigate its impact.

A self-adaptive software system is able to modify its own structure and/or behaviour during run-time in order to deal with changes in its requirements, the environment in which it is deployed, or the system itself [Cheng et al. 2009]. Among the different properties of a self-adaptive system, self-protection has been identified as a key concept for building autonomous self-managed systems. While systems' architectures are becoming more dynamic and adaptive, the majority of the protection mechanisms have kept simple, with security policies usually manually defined, in a slow and costly way.

One way for achieving self-adaptation is through the Monitor-Analyse-Plan-Execute-Knowledge (MAPE-K) feedback control loop over a target system [Kephart and Chess 2003]. In this way, a self-protection mechanism allows the protected system to monitor and analyze its resources in order to detect possible problems, being able to react accordingly to deal with the detected problem. This reaction depends on the type of incident and the type of system being protected, and can range from emergency system shutdown, deactivation of damaged module and replacement for a new instance, user and/or connection blocking, etc [Yuan et al. 2014].

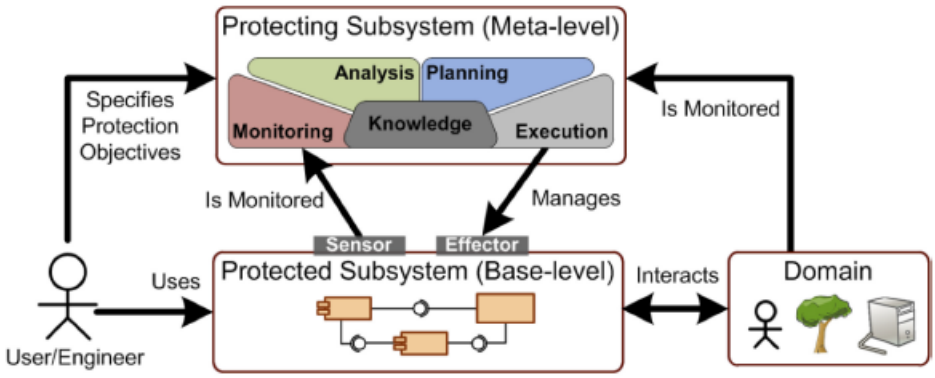

Figure 2: Self-Protection Reference Architecture [Yuan et al. 2014].

Figure 2 presents a reference architecture for a system that implements self-protection. At a meta-level we have a *protecting sub-system*, responsible for implementing the MAPE-K feedback control loop that protects the *protected sub-system* at the base level. The *protected sub-system* contains the system functionality associated with the main application logic, and may incorporate different security mechanisms, such as access control and cryptography. The meta-level subsystem is responsible for detecting security related incidents and for the decision making associated with the use of the security mechanisms by the base-level [Yuan et al. 2014]. The base-level sub-system runs over, and interacts with, a domain, which can also be monitored for helping in the decision making of the MAPE-K at the meta-level.

In this way, a Self-Adaptive Distribute Firewall (SADF) solution can be employed as a preventive mechanism for dealing with new vulnerabilities. For example, whenever a particular server contains a vulnerability with a score greater than a pre-defined value, the firewall could be configured to only allow access to server from clients in the same network.

3. Architecture for Self-Adaptive Distributed Firewall

Our solution for a Self-Adaptive Distributed Firewall (SADF) is built on top of the MAPE-K reference model as the means for logically structuring the different tasks involved in the management of the security aspects for a network infrastructure, and for integrating the different tools usually involved in those tasks, allowing for their automation. Figure 3 presents our architecture.

Each phase of the MAPE-K feedback control loop is implemented by an engine, which encapsulates the concrete components that allow for each engine functionality. To perform self-adaptation, the Monitor, Analyze, Plan and Execute engine components use different models that provide an abstraction of relevant aspects of the managed system, its environment, and the self-adaptation goals [Iglesia and Weyns 2015]. These models are maintained by a knowledge base (not represented in the Figure).

The *Monitoring engine* is responsible for collecting information about the different servers of the network infrastructure. This collection happens through *Sensor* interfaces in each server. This data is represented by a *Server description* model, which is a format that can be manipulated and reasoned upon by the components of SADF. A service

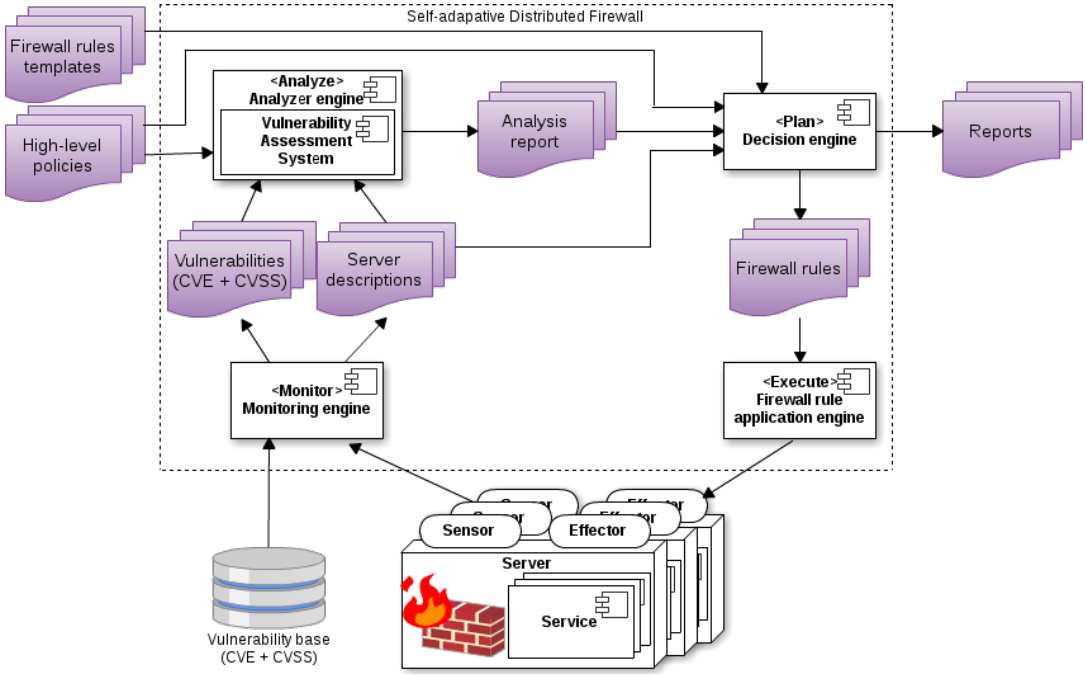

Figure 3: Conceptual architecture of proposed solution.

description model contains, among other information, details about operating system, IP Address, services' names, versions and network port. Furthermore, it captures the firewall rules currently in effect on the server. The *Monitoring engine* is also responsible for obtaining *Vulnerabilities* from an external *Vulnerability base*. Vulnerabilities are represented through CVE[1], which defines a dictionary and standard representation format for vulnerabilities descriptions. These descriptions are published through the CVE List and maintained by different vulnerabilities databases (e.g., the NVD[2]). Vulnerabilities have an associated severity score calculated based on the CVSS[3], which defines metrics and formulas for deriving a vulnerability score, and a standard format representation.

The *Analyzer engine* relies on a Vulnerability Assessment System (VAS) to search for known vulnerabilities on the services currently running on the network. A VAS works by scanning the network and conducting different tests in order to find vulnerabilities in systems and servers, producing a vulnerability report for each server. Based on the server descriptions, the VAS can be employed with higher priority to scan known services running on each server, usually when there are changes in the server descriptions or new vulnerabilities have been published. In the meantime, full vulnerability analysis of servers can still be performed. A *High-level policy* captures the requirements of the administrator, and together with the VAS report and the server description, is used for detection of policy violations. For example, servers with a vulnerability score greater than a particular threshold should only be accessible from machines in the same network, but the current firewall rules allow access from anywhere. All these data is used by the *Analyzer engine* component for producing *Analysis report*, which indicates for example, servers with known vulnerabilities.

[1]Common Vulnerabilities and Exposures - `https://cve.mitre.org`
[2]The National Vulnerability Database maintained by NIST - `https://nvd.nist.gov/`
[3]Common Vulnerability Scoring System - `https://www.first.org/cvss`

The *Decision engine* component is responsible for the plan phase of the MAPE-K loop. This component is responsible for making decisions on how to respond to the encountered situation based on the analysis report, the server descriptions, the high-level policies and a set of *Firewall rules templates*. These templates provide a sort of parameterized firewall rules for different services, which can then be employed by the Decision engine for defining specific firewall rules to be applied. The creation of firewall rules must employ mechanisms for avoiding conflicts between rules. Besides creating firewall rules to be applied onto the servers of the network, the Decision engine also produces a report intended for a human administrator.

At the execute phase we have a *Firewall rule application engine* component, which is responsible for effecting the new firewall rules on the servers. This component must take in consideration mechanisms for guaranteeing secure communication with each server, and configuration management techniques.

4. Instantiating the SADF

The proposed SADF architecture has been instantiated into a prototype implementation using a combination of existing open source and in-house developed components. This instantiation has been used to build a case study in order to demonstrate the feasibility of our approach. In order to demonstrate the prototype, in this paper we consider the server to be protected a Web server running the Apache HTTPD software and the ssh daemon.

In this section we present details about the different representation models employed in our instantiation, followed by a description of the developed prototype. We conclude this section with a brief discussion on our approach.

4.1. Representation Models

One aspect that must be considered for a self-protection solution is the representation of the protected environment, such as servers, services, and firewall rules.

For representing servers and their deployed services we chose the representation language defined by the Puppet[4] configuration management tool. The Puppet language allows the description of servers, services, and configurations using a parameterized approach and well defined semantics. Puppet configures systems in two main stages compiling and applying a catalog. A catalog is a document that describes the desired system state. It lists all of the resources that need to be managed, as well as any dependencies between those resources. The core of the Puppet language is declaring resources. Groups of resources can be organized into classes, which are larger units of configuration. While a resource may describe a single file or package, a class may describe everything needed to configure an entire service or application. The Figure 4a show a simple example of representation of a server and some services. One class was created with name *foo*. This class has an attribute *host* to describe a hostname and IP address. Furthermore, httpd and sshd were describe using the attribute *service*. The *ensure* attribute is used to indicate that those services should be in a running state.

In a similar way, it is necessary to represent firewall rules in a format that can be reasoned upon. For this purpose, we decided to employ the FLIP language

[4]`https://puppet.com/`


```
class foo {
  host { 'foo':
    host_aliases => foo.domain,
    ip           => 200.177.2.46,
  }
  service {'httpd':
    ensure     => running,
  }
  service {'sshd':
    ensure     => running,
  }
}
```

(a) Puppet class

```
domain serverHTTP = [192.168.100.35],
       serverVAS = [10.5.54.10],
service http = tcp.[port=80],
        ssh = tcp.[port=22, port=2222],
policy_group serverHTTP_policy{
  incoming:
    ssh {allow *}
    http {allow *}
}
policy_group serverHTTP_with_vulnerability{
  incoming:
    ssh {allow *}
    http {deny * except serverVAS}
}
apply serverHTTP_policy on serverHTTP;
```

(b) FLIP rule

Figure 4: Example of descriptions: (a) Class description using the Puppet language. (b) Firewall rules using the FLIP language.

[Zhang et al. 2007][Al-Shaer 2014]. In FLIP, firewall rules are defined using a high-level language that can be automatically translated into device specific format. FLIP provides as well defined language with formal semantics, together with proven sound and complete algorithms for conflict resolution and translation into device specific firewall rules. Its formalism was one of the main reasons for choosing FLIP. The Figure 4b presents an example of a rule in FLIP. The first block defines the domains that can be networks or hosts. We define a HTTP server and a server that would be our VAS. The second block of FLIP defines services, a service may have one or more ports. In this example http and ssh was specified. Then defines the group policy which specifies the behavior that will be taken in relation to services in a given scenario. Two groups were created, one that allows access to services and another that blocks the http. Finally, is necessary to make a connection between the group and the protected domain.

4.2. Prototype Implementation

In the sequence we describe our prototype implementation, whose concrete architecture is presented in Figure 5.

As previously mentioned, we employ the Puppet configuration management language for describing servers' configurations. Puppet provides tools for applying configurations, and for obtaining the current status of a host. A Puppet agent component runs on each host, and reports to (and receive commands from) the Puppet master component, which stores servers description into the Puppet catalog. In this way, Puppet agents fulfill the roles of sensor and effector of servers, while the Puppet master is responsible for the monitor and execute phases of the MAPE-K. It is important to mention that the definition of puppet description files for servers is out of the scope of this paper. We assume that this is achieved by a third party (e.g., a member of the network administration team), while our focus is on defining and applying firewall rules for the established configuration, or for dealing (i.e., mitigating the impact) with known vulnerabilities that the server may be susceptible to. In this way, the puppet agent and master components are used to obtain the current service(s) configuration on each server.

Once we know the services running on each server of the network, we can then

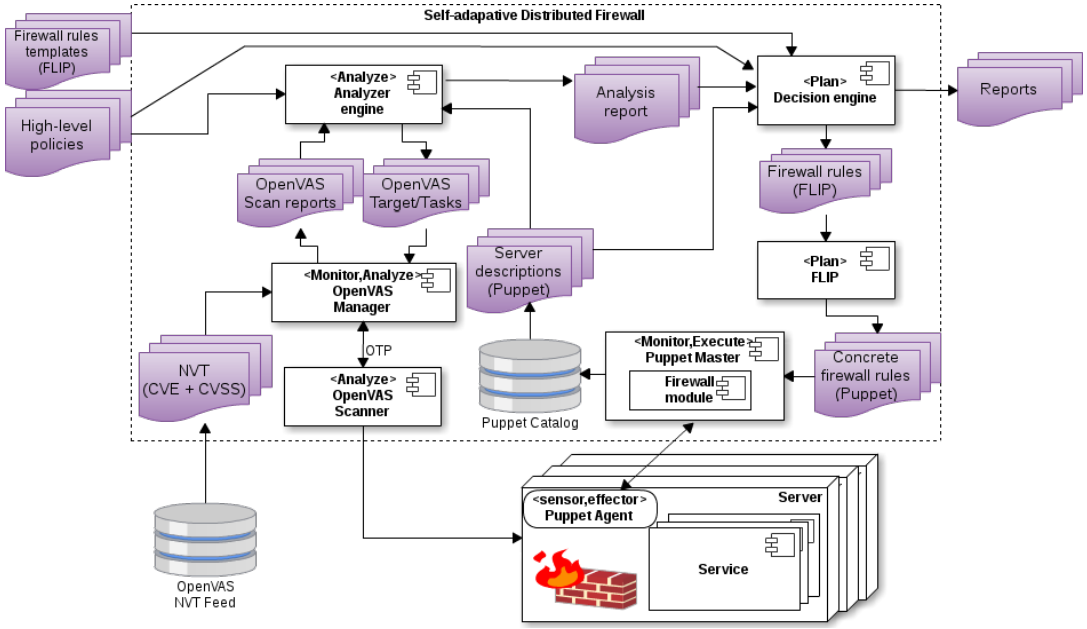

Figure 5: Architectural view of prototype implementation.

employ vulnerability assessment as triggers for adaptation. In order to achieve this, we employ the OpenVAS[5] Vulnerability Assessment System (VAS). OpenVAS provides an open-source solution for the identification, quantification and prioritization of vulnerabilities, and we explore this for our *Analyzer engine* component. The *OpenVAS Scanner* is responsible for conducting Network Vulnerability Tests (NVTs) on the hosts of the network. An NVT can be defined as a test script, together with CVE description, and CVSS score. NVTs can be obtained from the OpenVAS NVT Feed by the *OpenVAS Manager* (acting in both monitor and analyze roles), and are developed based on the CVE and CVSS score obtained from the NIST Network Vulnerability Database (NVD). The *OpenVAS Manager* is responsible for managing the OpenVAS Scanner, providing its input and processing its output. The interaction between the Manager and the Scanner happens via the OpenVAS Transfer Protocol (OTP), which provides the means for controlling scan execution. The Manager provides the OpenVAS Management Protocol (OMP), a XML-based stateless API that can be used to interact and control the OpenVAS Manager. The Analyzer engine employs the Servers' description for driving the OpenVAS scans. The interaction with OpenVAS is achieved by the OMP protocol, which receives as input an XML description of the target host and the scanning tasks (i.e., the vulnerability tests) to conduct on the host. The target can be defined as one, or a group of hosts, with an optional list of ports. A task corresponds to scanner configurations, and defines how the environment will be checked.

We detail the interaction between the *Analyzer engine* component and the OpenVAS in Figure 6, using an UML sequence diagram. Once the servers' descriptions have been obtained, the *Analyzer engine* create the inputs for OpenVAS Manager (calls 2 and 3 in the sequence diagram) and starts a scan, receiving the scan ID as response. The scan ID is then used to obtain the scan results and, in case a vulnerability has been found, the

[5]`http://www.openvas.org/`

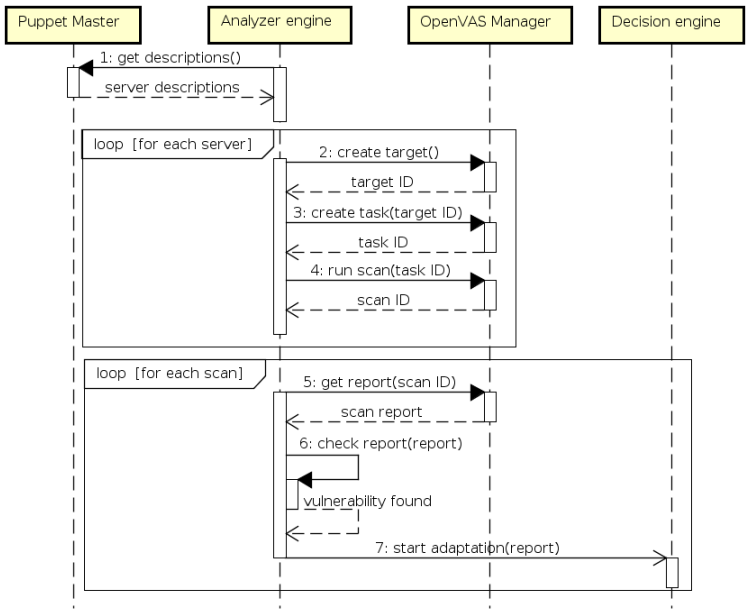

Figure 6: UML sequence diagram showing the behaviour of analysis.

Decision engine component is activated to deal with it.

Following our case study, the OpenVAS found a vulnerability with CVSS score 7.8 in HTTP service that allows remote attackers to cause a denial of service (memory and CPU consumption). An extract of the XML report is presented in Figure 7.

```
...
<port>80/tcp
  <host>10.3.128.20</host>
  <severity>7.8</severity>
  <threat>High</threat>
</port>
<nvt oid="1.3.6.1.4.1.25623.1.0.901203">
  <name>
    Apache httpd Web Server Range Header Denial of Service Vulnerability
  </name>
  <family>Denial of Service</family>
  <cvss_base>7.8</cvss_base>
  <cve>CVE-2011-3192</cve>
  <bid>49303</bid>
  ...
```

Figure 7: Extract of OpenVAS report to a vulnerable HTTP server.

Once the *Decision engine* has been activated, it must decide on how to respond to the found vulnerability. The behavior of this component is presented in Figure 8. Each report is considered together with the actual server description, including the firewall rules currently in place, against the high-level policy for deciding what to do regarding the firewall of the affected host. For example, a CVSS score above 7 might require that the service is no longer accessible. If this is the case, the *Decision engine* define the adequate firewall rules using the FLIP language, and sends it to the FLIP tool, which is responsible for checking the rule is conflict free, translating it into the puppet format, and applying them in the puppet master. Regardless of the decision, the *Decision engine* sends a notification report (represented by the *notify administrator* call). This notification can then be

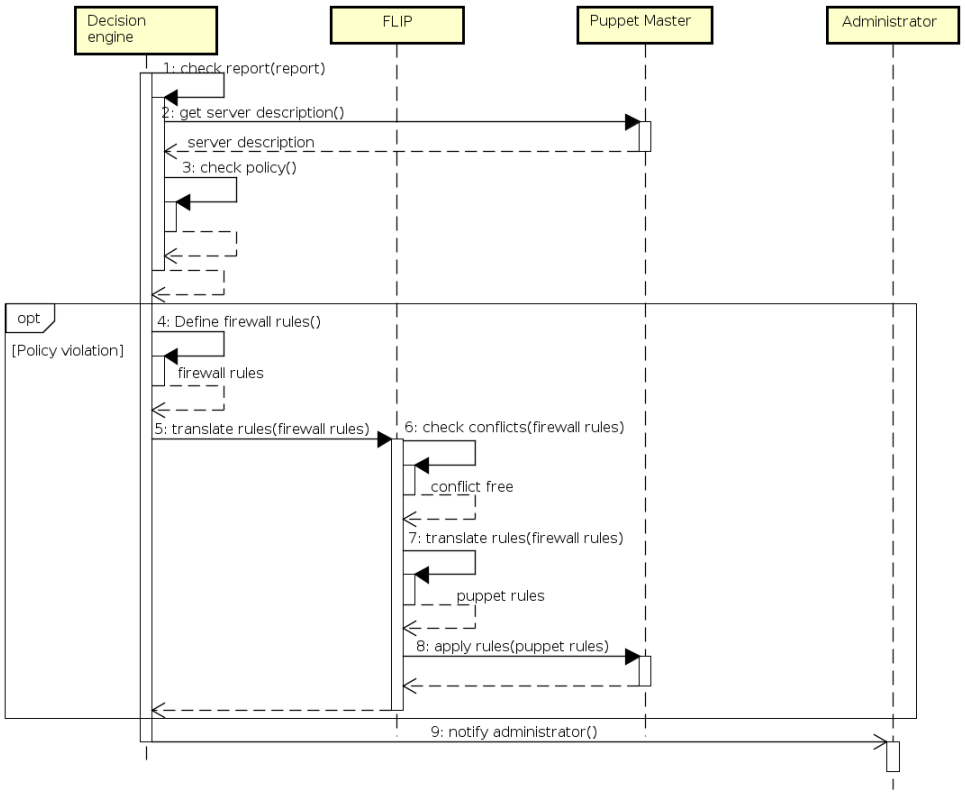

Figure 8: Sequence diagram showing behaviour of decision engine.

used by a visualization tool, for example, a Web page listing all active vulnerabilities and the respective firewall rules applied to those servers.

The Puppet master, together with its firewall module[6], manage and configure firewall rules from within the Puppet DSL. This module offers support for iptables and ip6tables, which is the most common firewall in GNU/Linux distributions, being capable of acting on a running firewall. All rules employ a numbering system in the resource's title that is used for ordering. The Figure 9a presents an example of a puppet rule for allowing http and ssh services, while Figure 9b presents the rule created by our prototype for closing access to the http service.

Once the firewall rules are in place, new alerts of the same vulnerability will be notified to administrator, but will not trigger changes in the firewall of the affected host. On the other hand, once the vulnerability has been fixed, our tool will detect it, and adjust the firewall rules accordingly.

4.3. Discussion

The main objective of this prototype was to demonstrate the feasibility of our approach in integrating a VAS and a distributed firewall in an autonomic way. For this reason, the decision making performed by our prototype has been implemented through simple if-then-else statements using the fields of the different descriptions as parameters. We have also employed very rudimentary high-level policies, simply defining range thresholds for each response (e.g., access granted to all, access granted to clients in the same network,

[6]`https://forge.puppet.com/puppetlabs/firewall`


```
class my_fw::pre {
  Firewall {
    require => undef,
  }
  firewall { '000 Allow inbound HTTP':
    dport    => [80,443],
    proto    => tcp,
    action   => accept,
  }
  firewall { '001 Allow inbound SSH':
    dport    => 22,
    proto    => tcp,
    action   => accept,
  }
}
```

(a) Allowing http and ssh

```
class my_fw::pre {
  Firewall {
    require => undef,
  }
  firewall { '000 Deny inbound HTTP':
    dport    => [80,443],
    proto    => tcp,
    action   => deny,
  }
  firewall { '001 Allow inbound SSH':
    dport    => 22,
    proto    => tcp,
    action   => accept,
  }
}
```

(b) Denying http and allowing ssh

Figure 9: Examples of firewall rules in the Puppet language.

access denied to all). The experiments conducted, although initial, have showed that we are indeed able to activate firewalls running on each server for preventing access to vulnerable services.

We believe the administrator involvement is crucial to those types of solutions, and the administrator notification aims to be a starting point for a more intelligent human interaction. For example, in our prototype we have applied the new firewall rules into the affected server without any kind of confirmation, but we also envision a scenario where the rules as presented to an administrator that then authorizes their application, or the existence of policies separating scenarios that should be automatically acted upon from those where some sort of human confirmation is needed. Our solution can also be used to identify new servers on the network that are not managed by the configuration management tool in place (e.g., Puppet). Those would be part of the notification reports sent to administrators.

The Puppet configuration management tool presents a pull based model, in which agents query the master at pre-determined intervals in search of new commands (usually, 30 minutes). This might be considered an issue when responding on-the-fly to detected situations, which is not the case in this paper. We are focused on preventing the exploit of known vulnerabilities before they happen, opposed to traditional IDPS systems, which focus on responding in real-time to incidents. In this way, we consider that 30 minutes is a reasonable time for applying firewall rules blocking vulnerable services, although this time could be reduced, and there are alternatives for employing a push-model to puppet. Regarding the secure communication between the SADF and protected servers, we employ the certificate based security provided by the Puppet tool, which takes care of authentication and secure transit between the Puppet master and its agents.

5. Related Work

The discussion on centralized and distributed firewalls is well established in the literature [Bellovin 1999] [Stallings 2010]. One of its first implementation proposal has been presented by Ioannidis [Ioannidis et al. 2000], in which kernel extensions have been developed for the OpenBSD distribution, together with a policy definition language (denominated KeyNote) and use of IPsec for secure traffic amongst the hosts of the network.

More recently [Lai et al. 2009] introduces a distributed firewall system for Linux platform that works upon Iptables/Netfilter for IPv6 networks with IPsec support. In order to improve performance for handle the additional costs of encryption of packages with IPsec, a distributed firewall architecture was proposed.

Autonomic computing and self-protection has been gaining traction as the means for dealing with new security challenges and systems, in which static and rigid security practices are not enough to deal with security threats that need to be detected and mitigated at runtime [Yuan et al. 2014]. In this context, Yuan et al. [Yuan et al. 2014] have done an extensive systematic survey of the state of the art on self-protecting software, identifying trends, patterns, and gaps. Some works focus on adapting authorization policies, such as the Self-Adaptive Authorisation Framework (SAAF) [Bailey et al. 2014] that focus on adapting access control policies on the PERMIS system [Chadwick et al. 2008], and SecuriTAS [Pasquale et al. 2012], a tool that enables dynamic decisions in awarding physical access, based on a perceived state of the system and its environment.

Several researchers focus on Intrusion Detection Systems. For example, Uribe and Cheung [Uribe and Cheung 2004] have looked into the integration between IDSs and firewalls, proposing an approach for optimizing IDS configuration by only analysing traffic that is not considered by the firewalls' rules. Zhang and Shen [Zhang and Shen 2009] employ a statistical learning based approach in order to reduce false-positives on IDSs. These works are concentrated on improving the IDS.

Few works consider the analysis of vulnerabilities for making a decision at the network level (i.e., firewall rules). Debar et al. [Debar et al. 2007] present formalisms for the definition of security policies that can be dynamically modified in response to detected threats. The formalism presented in their paper is at an abstract level, and may consider vulnerability analysis in the threat detection process. Compared to our work, their approach can be considered as complementary, providing a series of formalism that could improve the robustness of our approach regarding the definition of high-level policies.

One aspect that must be considered for a self-protection solution is the representation of the protected environment, such as servers, services, and firewall rules. We present here some of the options found in the literature during our searches. The Network Markup Language (NML) [van der Ham et al. 2013] is a generic model defined by the OpenGrid Forum (OGF)[7] as a standard for modelling networks, such as switches and links, which is out of the scope considered in this paper. Regarding the representation of firewall rules, apart from the FLIP language (already presented), there is also the AFPL2 [Pozo et al. 2009] (Abstract Firewall Policy Language 2), a domain-specific language that provides an XML Schema for the definition of firewall rules independent of firewall product. Although its support of NAT rules, AFPL2 has not evolved as FLIP, and does not provide conflict resolution of firewall rules. The Distributed Management Task Force (DMTF) proposed the Common Information Model (CIM), an specification aimed at allowing the interoperation of management information. The CIM also provides an extensible XML model and, although it has been employed by different vendors, its extension for Network Policy Management [DMTF 2016] is still considered work-in-progress, and may be subject to changes.

[7] https://www.ogf.org/

6. Conclusions & Future Work

This paper presented an approach for Self-Adaptive Distributed Firewall (SADF). We presented an architecture built on top of the MAPE-K reference model as the means for logically structuring the different tasks involved in the management of the security for the network. The *Analyze* and *Decision* engine are at the heart of the operation of our architecture for mitigating the risks of known vulnerabilities. Together, they detect vulnerabilities and protect services before they are exploited. We made a prototype implementation to demonstrate their feasibility using a combination of existing open source and in-house developed components.

Some simple experiments were conducted with encouraging preliminary results, where we are able to dynamically modify the firewalls on protected servers. The VAS in Analyze engine detect vulnerabilities caused by software bugs or misconfiguration. However, since we are at a prototype phase, our approach presents some limitations. For instance, currently we only deal with each individual host in an independent way, and more interesting possibilities arise due to the nature of distributed firewalls, in which firewall could be employed on other points of the network, such as routers and switches.

The decision making and the definition of high-level policies have been simplified, and could be improved by using other specialized components, such as rule-based systems. Some improvements on Decision engine may add capabilities to deal with a lot of information and possibilities in order to enhance the security of network. For example, a host firewall might redirect all incoming connection to an application level proxy when a particular vulnerability has been detected, adding an extra layer of authentication while the vulnerability has not been fixed. Another future work involves the integration of our solution with traditional IDPSs, allowing it to react to attack exploiting zero-day vulnerabilities.

References

Al-Shaer, E. (2014). *Automated Firewall Analytics: Design, Configuration and Optimization*, chapter Specification and Refinement of a Conflict-Free Distributed Firewall Configuration Language, pages 49 – 74. Springer International Publishing.

Bailey, C., Chadwick, D. W., and de Lemos, R. (2014). Self-adaptive federated authorization infrastructures. *Journal of Computer and System Sciences*, 80(5):935 – 952.

Bellovin, S. M. (1999). Distributed firewalls. *;login:*, pages 39–47.

Chadwick, D. W. et al. (2008). PERMIS: A Modular Authorization Infrastructure. *Concurr. Comput. : Pract. Exper.*, 20(11):1341–1357.

Cheng, B. H. et al. (2009). Software Engineering for Self-Adaptive Systems: A Research Roadmap. In Cheng, B. H., de Lemos, R., Giese, H., Inverardi, P., and Magee, J., editors, *Software Engineering for Self-Adaptive Systems*, pages 1–26. Springer-Verlag, Berlin, Heidelberg.

Debar, H., Thomas, Y., Cuppens, F., and Cuppens-Boulahia, N. (2007). Enabling automated threat response through the use of a dynamic security policy. *Journal in Computer Virology*, 3(3):195 – 210.

DMTF (2016). Network policy management profile. `https://www.dmtf.org/sites/default/files/standards/documents/DSP1048_1.0.0c_0.pdf`. [Online; accessed 19-June-2016].

Iglesia, D. G. D. L. and Weyns, D. (2015). Mape-k formal templates to rigorously design behaviors for self-adaptive systems. *ACM Trans. Auton. Adapt. Syst.*, 10(3):15:1–15:31.

Ioannidis, S., Keromytis, A. D., Bellovin, S. M., and Smith, J. M. (2000). Implementing a distributed firewall. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, CCS '00, pages 190–199, New York, NY, USA. ACM.

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

Lai, Y., Jiang, G., Li, J., and Yang, Z. (2009). Design and implementation of distributed firewall system for ipv6. In *Communication Software and Networks, 2009. ICCSN '09. International Conference on*, pages 428–432.

Meng, G., Liu, Y., Zhang, J., Pokluda, A., and Boutaba, R. (2015). Collaborative security: A survey and taxonomy. *ACM Comput. Surv.*, 48(1):1:1–1:42.

Pasquale, L. et al. (2012). SecuriTAS: A Tool for Engineering Adaptive Security. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 19:1–19:4, New York, NY, USA. ACM.

Pozo, S., Varela-Vaca, A. J., and Gasca, R. M. (2009). Afpl2, an abstract language for firewall acls with nat support. In *Second International Conference on Dependability (DEPEND 2009)*, pages 52–59.

Stallings, W. (2010). *Network Security Essentials: Applications and Standards*. Prentice Hall, 4th edition.

Uribe, T. E. and Cheung, S. (2004). Automatic analysis of firewall and network intrusion detection system configurations. In *Proceedings of the 2004 ACM Workshop on Formal Methods in Security Engineering (FMSE 2004)*, pages 66 – 74.

van der Ham, J., Dijkstra, F., Łapacz, R., and Zurawski, J. (2013). Network markup language base schema version 1. Grid Final Draft (GFD), Proposed Recommendation (R-P) GFD-R-P.206, Open Grid Forum.

Yuan, E., Esfahani, N., and Malek, S. (2014). A systematic survey of self-protecting software systems. *ACM Trans. Auton. Adapt. Syst.*, 8(4):17:1–17:41.

Zhang, B., Al-Shaer, E., Jagadeesan, R., Riely, J., and Pitcher, C. (2007). Specifications of a high-level conflict-free firewall policy language for multi-domain networks. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies*, SACMAT '07, pages 185–194, New York, NY, USA. ACM.

Zhang, Z. and Shen, H. (2009). M-aid: An adaptive middleware built upon anomaly detectors for intrusion detection and rational response. *ACM Trans. Auton. Adapt. Syst.*, 4(4):24:1 – 24:35.

