An Intrusion-Tolerant Firewall Design for Protecting SIEM Systems

Miguel Garcia, Nuno Neves and Alysson Bessani
University of Lisbon, Faculty of Sciences, LASIGE
mhenriques@lasige.di.fc.ul.pt, {nuno,bessani}@di.fc.ul.pt

***Abstract*—Nowadays, organizations are resorting to Security Information and Event Management (SIEM) systems to monitor and manage their network infrastructures. SIEMs employ a data collection capability based on many sensors placed in critical points of the network, which forwards events to a core facility for processing and support different forms of analysis (e.g., report attacks in near real time, inventory management, risk assessment). In this paper, we will focus on the defense of the core facility components by presenting a new firewall design that is resilient to very harsh failure scenarios. In particular, it tolerates not only external attacks but also the intrusion of some of its components. The firewall employs a two level filtering scheme to increase performance and to allow for some flexibility on the selection of fault-tolerance mechanisms. The first filtering stage efficiently eliminates the most common forms of attacks, while the second stage supports application rules for a more sophisticated analysis of the traffic. The fault tolerance mechanisms are based on a detection and recovery approach for the first stage, while the second stage uses state machine replication and voting.**

Keywords*—*Intrusion Tolerance; Intrusion Prevention Systems; Firewall;

I. Introduction

Security Information and Event Management (SIEM) systems offer various capabilities for the collection and analysis of security events and information in networked infrastructures [1]. They are being employed by organizations as a way to facilitate operations related to maintenance, monitoring and analysis of networks, by integrating a large range of security and network capabilities, which allow the correlation of thousands of events and the reporting of attacks and intrusions in near real-time [2], [3], [4].

A SIEM operates by collecting data from the monitored network and applications through a group of sensors, which then forwards the events towards a core facility for processing at a correlation engine (see Figure 1). The engine performs an analysis on the stream of events and generates alarms and other information for post-processing by other SIEM components. Examples of such components are an archival subsystem for the storage of data needed to support forensic investigations, or a communication subsystem to send alarms to the system administrators.

As SIEMs play an increasingly relevant role in the network and security management tasks of the organizations, it becomes imperative to ensure their correct operation under a wide range of fault scenarios. In particular, since security solutions often have been the target of malicious actions (e.g., anti-virus software [5], intrusion detection systems [6] or firewalls [7], [8], [9], [10]) as part of a wider scale attack, SIEM systems should be built/deployed under the assumption that this sort of actions will eventually occur. In this paper, we will focus on the protection of the core facility components from outside attacks, improving the security and dependability of such essential processing.

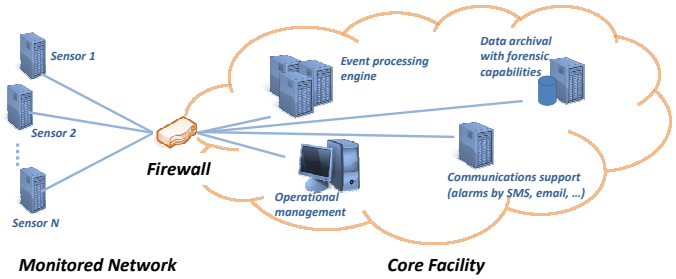

Fig. 1. Overview of a SIEM architecture, showing some of the core facility subsystems.

The traditional solution to defend a critical network from malicious outsiders is a firewall. Firewalls are intended to separate security perimeters, such as a LAN from a WAN, and their main goal is to control the traffic that flows in and out of a facility. Typically, a firewall decides on letting a packet go through (or drop it) based on the analysis of its header and/or contents. Over the years, this analysis has been performed at different levels of the OSI stack, but the most sophisticated rules are based on the inspection of application data included in the packets. State-of-the-art solutions for application-level firewalls include network appliances provided by several vendors, such as Juniper[1], Palo Alto[2], and Dell SonicWall[3].

Firewalls are in general a single point of failure, and as such they have been the target of many attacks[4]. When successful, these attacks can impact on the system's security in different ways — for instance, they can allow complete access to the internal network resources or they can compromise availability by preventing traffic from going through the firewall. To address some of these issues, the paper presents a new resilient firewall design that is able to remain operational under very harsh failure conditions, including both accidental crashes or

[1] http://www.juniper.net/us/en/products-services/security/

[2] http://www.paloaltonetworks.com/products/platforms/PA-5000_Series.html

[3] http://www.sonicwall.com/us/en/products/Next-Generation_Firewall.html

[4] An inspection of the Open Source Vulnerability Database (OSVD) shows that there have been many security issues in commonly used firewalls. During the 4 year period between 2009 and 2012, there were for example the following number of security reports: 36 for the Cisco Adaptive Security Appliance; 64 in Juniper Networks solutions; and 29 related to netfilter.

the compromise of some of its components.

A key design choice on the firewall architecture is to divide filtering operations in two stages. The rationale for this decision was related with optimizing performance under attack and to give flexibility on the selection of the fault tolerance mechanisms. Performance is increased if common forms of attacks are handled earlier and with highly efficient tests. Therefore, the first stage, which is called *pre-filtering*, carries on lightweight checks on the messages and aims at discarding all attacks from external adversaries. In particular, it only lets messages go through that come from a pre-defined set of senders (typically, the sensors of the SIEM) and that are correctly authenticated. Denial-of-service (DoS) traffic from external sources is immediately dropped, preventing these messages from overloading the next stage. The second stage is named *filtering*, and it is responsible for enforcing more refined application level policies, which can require the inspection of specific message fields and/or the observation of certain ordering rules (e.g., a sensor can only send data after some initial setup is performed with the engine).

Different fault tolerance mechanisms are employed at the two stages. Pre-filtering is implemented by a dynamic group components, whose size is adapted to the current demand. Hence, as the traffic load grows or extra senders have to be supported, the firewall creates more pre-filters to amplify the aggregated processing capabilities (within the constraints of the hardware). Since pre-filters face the external network, they can experience various kinds of attacks and eventually be intruded. Therefore, we take the conservative approach of assuming that pre-filters can fail in an arbitrary (or Byzantine) way, meaning that they may for instance crash or start to act maliciously (when compromised by an adversary). The faults are tolerated by forcing the recovery of the erroneous pre-filters after their identification. The filtering stage is performed by a static group of filter components that are organized as a replicated state machine. Since we also want to consider filters that fail with an arbitrary behavior, the replication protocol needs to tolerate Byzantine faults and the final destination needs to vote on the results that are produced (to eliminate malicious data).

The rest of the paper is organized in the following way: Section II discusses some of the design choices that guided our design. The model of the system is presented in Section III, and then the firewall operation is described in detail. Finally we present an overview of possible deployment alternatives (Section V), and the conclusions in Section VI.

II. Design Choices

Traditional firewall designs are based on a single logical component, and consequently, they are unable to tolerate most failures. A simple crash prevents the firewall from fulfilling its function, and more elaborated failure modes may allow an adversary to penetrate into the protected network. Some organizations deal with crashes (or DoS attacks) by replicating the firewall to support multiple entry points. This solution is helpful to address some (accidental) failures, but is incapable of dealing with an intrusion in the firewall. In this case, the adversary gains complete access to the internal network, allowing an escalation of the attack, which at that stage can only be stopped if other protection mechanisms are in place.

Over the past years, there has been a important amount of research in the development of systems that are intrusion-tolerant. To our knowledge, however, only very little work was devoted to design intrusion-tolerant protection devices, such as a firewalls (e.g., [11], [12]). Performance reasons might explain this, as Byzantine fault-tolerant (BFT) replication protocols are usually associated with reasonable overheads and limited scalability. For example, a straightforward implementation of a BFT firewall would require replicas to process every arriving message and then agree on their delivery. BFT solutions based on a leader can also become prey of an adversary, as they have a natural bottleneck replica that can be selected for the attack (instead of having to disperse the attack power over all replicas [13]).

The main motivation for this work is to address these limitations and propose a design for an intrusion-tolerant firewall. The fundamental design options that guided our solution were:

1) *Application-level filtering:* Allow for sophisticated filtering rules that take advantage of application knowledge. To support these rules, the firewall has to maintain state about the current communications. The state will have to be consistently replicated using a BFT protocol.
2) *Performance: a)* Address the most probable attack scenarios with highly efficient tests, and as earlier as possible in the filtering stages. *b)* Reduce communication costs with external senders, as these messages may have to travel over high latency links (e.g., do not require message multicasts).
3) *Resilience: a)* Tolerate a broad range of failure scenarios, including: malicious external attackers; compromised authenticated senders; and intrusions in a subset of the firewall components. *b)* Prevent malicious external traffic from reaching the internal network by requiring explicit message authentication.

III. System Model

Since we want to address faults both of an accidental nature and also caused by a malicious adversary, we assume that erroneous components can behave arbitrarily (or in a Byzantine way). To be conservative, we assume that all failed components are controlled by a single entity that will make them act together to defeat the normal operation of the system. Therefore, they can for instance stop sending messages, skip some steps of the protocols, provide erroneous information to correct components, or try to delay the system.

We address three failure scenarios on the components, which provide increasingly more power to the adversary. The most common scenario occurs when an external adversary attempts to attack the internal network systems. He can deploy a large number of (unauthenticated) senders, whose aim is either to delay the communications or to bypass the firewall protection and reach the internal network. In particular, he can try to masquerade the messages as coming from normal senders, or perform a DoS by transmitting many erroneous packets to the firewall. However, as the firewall cannot stop DoS attacks that completely overload its incoming channels, which cause most of the normal messages to be dropped by the routers, it is assumed that the network includes other defense mechanisms to address this sort of attack (e.g., see [14]).

As the authenticated senders (e.g., the sensors of the SIEM) are potentially spread over several outside networks, it is advisable to consider a second scenario where the adversary is also capable of taking control of some of these nodes. When this happens, it is assumed that the adversary gains access to all keys stored locally. Therefore, he will be able to generate traffic that is accepted by the firewall as correctly authenticated, but the packets will still be checked by the application level filtering. In any case, if the messages follow the rules, the firewall has to let them go through, as they are indistinguishable from other valid messages.

A third scenario occurs when the adversary is able to cause an intrusion in the firewall, and compromises a few of the pre-filters and/or filters. We assume that at most f_{pf} pre-filters fail of a total of $N_{pf} = f_{pf} + k$ (with $k > 1$), and that out of the $N_f = 3f_f + 1$ filters at most f_f fail. To enforce this assumption it is necessary to ensure that firewall components fail independently, which typically can be achieved with good coverage if one employs diversity [15]. A failed pre-filter can for instance modify the received traffic or generate invalid messages that are given to the filtering stage. A malicious filter can perform similar attacks, and in addition send erroneous data to the final destination.

We assume that the communications with the firewall can also suffer from accidental faults and/or attacks. Thus, messages may be lost, delayed, reordered or corrupted. Most of the faults will have to be tolerated by the applications, for example, by retransmitting the messages.

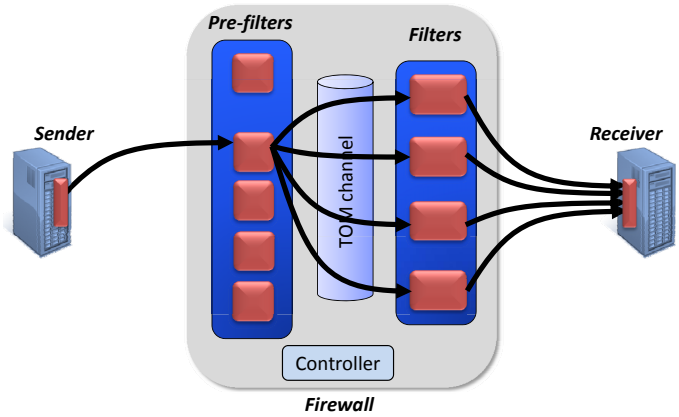

Fig. 2. Overview of the architecture and communication pattern with the firewall.

IV. Firewall Operation

This section describes the firewall execution when messages arrive from the senders and are forwarded to the final destination. Traffic identified as malicious by one of the filtering stages is discarded. Figure 2 displays the architecture of the firewall and shows the communication pattern among the components.

A. Components of the Architecture

The main components involved in the operation of the firewall are:

a) Sender-module: There is a communication module deployed on the sender's side that is responsible for the interactions with the firewall. It performs the authentication with the firewall, and encapsulates the data received from the application to ensure its correct delivery. Each sender-module is associated with one preferred pre-filter that is selected during setup. Messages are transmitted towards the preferred pre-filter.

b) Pre-filters: They appear as the external interface of the firewall, and perform basic filtering actions. Although tests are kept simple to improve efficiency, they are nevertheless effective at deterring most attempts of DoS. Pre-filters forward to the filters the accepted messages using a *Byzantine total ordered multicast (TOM)* protocol [16]. This protocol ensures that all filters receive identical messages in the same order.

c) Filters: They implement a state machine replication service that filters messages based on application knowledge. Each of them acts as a replica, applying exactly the same rules to every message that was deemed valid by the pre-filters. Therefore, correct filters should reach to the same conclusion regarding the validity of the messages. Accepted messages are transmitted to the final destination.

d) Receiver-module: It is a communication module deployed on the receiver's side, whose responsibility is to deliver the messages to the application. The main role of this component is to vote Filters' messages in order to accept only correct messages (recall that compromised filters may send invalid data). A message is considered correct is $f_f + 1$ filters vouch for it.

e) Controller: Is a trusted component of the firewall that runs with a higher privilege level (depending on the actual firewall implementation, it can be developed in different ways; see Section V for a discussion). The controller takes input from the filters to decide on the creation of more pre-filters, or to delete one of them.

The deployment of the firewall requires a *key distribution scheme* to create shared keys between sender-modules and the pre-filters, and between filters and receiver-modules. These shared keys can be distributed based on some long term secrets (e.g., private-public key pairs), using for instance protocols similar to IPsec (the Internet Key Exchange [17]).

B. Transmitting a Message

The sender-module receives from the application a buffer with *DATA* to be transmitted to a certain destination behind the firewall. The buffer needs to be encapsulated in a message with some extra information required to protect the communication. It is necessary to add the sender-module identifier c_i and a sequence number value sn. The sequence number is incremented in every message, and is used to prevent certain replay attacks either from the network or a compromised pre-filter.

Next, information is added to the message to protect its integrity and allow for its authentication. A signature *sign* is performed over the messages contents, and a MAC M_{pf} is obtained for the pre-filter associated with this sender (covering all fields, including the *sign*). The MAC is computed using a shared key established with the pre-filter, and serves as an optimization to speedup checks (as testing a MAC is faster than a signature). The signature is calculated with the private key of

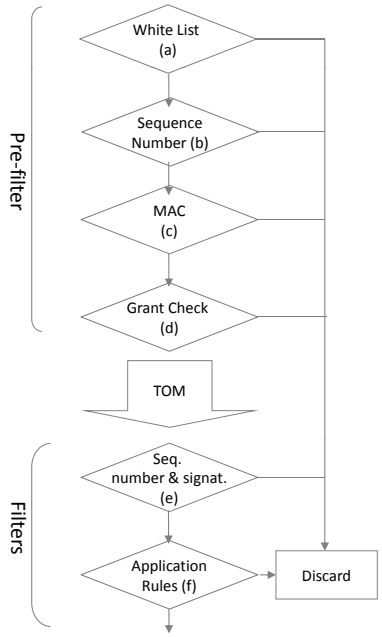

Fig. 3. Filtering stages at the firewall.

the sender-module (the corresponding public key is provided to the firewall components for signature verification, during the setup of the system). The message then is sent via UDP to the pre-filter pf:

$$M = \langle c_i, sn, DATA, \text{sign} \rangle_{M_{pf}}$$

f) Pre-filter: Upon receiving the message, the pre-filter applies a few checks to determine if the message should be forwarded or discarded (see Figure 3):

(a) White list: each pre-filter maintains a list with the nodes that are allowed to transmit messages (i.e., which were authorized by the system administrator). Messages coming from other nodes are simply dropped. This check is based on the address of the sender of the message.
(b) Sequence number: determines if a message with sequence number *sn* from sender c_i was already seen. In the affirmative case, the message is discarded to prevent replay attacks. Furthermore, messages are also dropped if their *sn* is much higher than the largest sequence number ever seen from that sender (a moving window of acceptable sequence numbers is used).
(c) MAC test: MAC M_{pf} is verified to authenticate the message contents. If the check is invalid, the message is dropped and the sequence number information is updated to indicate that this message was not seen.
(d) Grant check: each pre-filter controls the amount of messages that sender-module is transmitting. Messages that fall outside the allocated amount are dropped, to ensure that all senders get a fair share of the available bandwidth (and to avoid DoS attacks by intruded senders).

Then, the pre-filter invokes the total order multicast channel to forward the message to the filters. This channel ensures that all filters receive the message in the same order.

g) Filter: Each filter applies identical checks to the message:

(e) Sequence number and signature test: as the pre-filter might have been intruded, it is necessary to perform an extra check on the integrity/authenticity of the message fields. The test on the sequence number is similar to the pre-filter, but then the signature is used to ensure that all filters reach the same decision regarding the validity of the contents.
(f) Application-level rules: apply the application-level filtering rules to determine if the message is acceptable. These rules may look into the contents of *DATA* and relate it with context information about messages that were previously processed, i.e., we support iptable rules.

Although uncommon in several deployment scenarios, it can happen that the network re-orders the delivery of messages. The impact of this problem is that application-level rules may drop some of the out of order messages, which later on will have to be re-transmitted by the application. For example, if messages A and B should appear in this sequence but are re-ordered, then the rules may consider B invalid and then accept message A. At some point, the application would determine that B was lost, and would re-send it.

To address this issue, each filter enqueues for a while messages with a sequence number greater than the expected (but that do not exceed a threshold above the last processed sequence number). Next, it will continue to process other messages, until either: 1) the missing message(s) arrive, and then they are all tested in order, or 2) it gives up on waiting, and checks the message. This last decision is made after processing a pre-determined number of other messages.

Messages that are considered valid are encapsulated in a new format, and are then sent to the final destination. Basically, the filter removes the signature and substitutes the MAC with a new one M_f. This MAC is created using a shared key between the filter and the receiver-module.

$$M1 = \langle c_i, sn, DATA \rangle_{M_f}$$

The receiver-module accumulates the messages that arrive from the various filters, until a quorum is collected to ensure correctness. Furthermore, it validates the MACs to authenticate the content. A message can be delivered to the application when there is a quorum of $f_f + 1$ equal copies, as this indicates that at least one correct filter accepted the message.

C. Handling Component Failures

In the Byzantine model, every failed component can behave in an arbitrary way, intentionally or accidentally. Therefore, when designing the firewall, it is necessary to incorporate mechanisms that are resilient to very harsh failure scenarios. Given the architecture of Figure 2, one has to address faults in the pre-filters and filters, and needs defenses against erroneous (authenticated) senders and external attackers (regarding the receiver-module, there is not much that can be done about its failures).

Although pre-filters carry out the same function, i.e., check the messages arriving from a few sender-modules, they are not replicated. Furthermore, since pre-filters face the external firewall interface, there is a higher risk of being compromised. As such, in order to keep the firewall operational, it is required that failed pre-filters are identified and recovered. We leverage from the filter setup to perform the fault detection (of either crashes or misbehavior), and then use the controller to re-start erroneous pre-filters.

Filters implement state machine replication, and as long as they process the same messages in identical order, identical results should be produced. Consequently, filter faults can be tolerated by employing a voting technique that selects results supported by a sufficiently large quorum (as explained above, an output with at least $f_f + 1$ votes).

1) Pre-Filter failures: Since pre-filters can fail arbitrarily, they can exhibit very different invalid behaviors. Moreover, sometimes they may look as acting in a flawed way, but in fact they are correct. For example, when a pre-filter is under a DoS attack, messages can start to be lost on the network due to buffer overflows, but this is indistinguishable from a failure that causes omissions. This sort of ambiguity precludes exact failure detection, and therefore, our aim should be to provide a number of mechanisms that allow the firewall to recover from failures and continue to deliver a correct service (i.e., let clients send messages to a given destination). One however should accept right from the start that occasionally there might be mistakes on the fault detection — a good pre-filter may be erroneously considered failed (e.g., it is just overloaded), while a flawed one might go undetected (e.g, if messages are only dropped rarely).

An initial step on the detection process is for the pre-filter to evaluate its own conduct. In particular, it observes the amount of traffic that is arriving to determine if there is a risk of becoming overloaded. An easy way to carry out the analysis is to measure the waiting intervals for message arrivals over a certain period. If those intervals are very small on average, then the per-filter is working at its full capacity or there is already some overload. When this happens, the pre-filter sends over the TOM channel a *WARNREQ* request to the filters, so that they may take some action to solve the problem (see below).

To detect failures in general, it is necessary the cooperation of the filters and the senders. Since a sender generates the data that is transmitted to a pre-filter, it knows how many messages should have arrived to the filters. Consequently, by providing a mechanism where a sender can tell the filters how many messages were supposed to be delivered, it is possible to determine if many omissions are occurring on a pre-filter (or in the network connecting to it). The procedure is the following:

- Periodically, the sender-module transmits to the filters a special *ACKREQ* request, where it indicates the sequence number of the last message that was transmitted (plus a signature and a MAC to ensure authenticity). This request is first sent to the preferred pre-filter, but if no answer is received within some time, it is forwarded over the other pre-filters. The waiting period is adjusted in each retransmission by doubling its value.
- When the filters get the request, they use the included sequence number together with some local information, to find out how many messages have been missing. The local information is basically the set of sequence numbers of the messages that were correctly delivered since the last *ACKREQ*.
- Based on the number of missing messages, the filters transmit through the same pre-filter a response *ACKRES* to the sender-module, where they state the observed failure rate and other control information (plus a signature). Filters may also specify some recovery action if the failure rate is too high (see below).

Finally, filters can also learn about failures based on the validations performed on the messages. For example, an invalid signature check is highly suspicious because all messages corruptions should be captured by the pre-filters with the MAC test. This would indicate that either the pre-filter or/and the sender is failed (since it could have generated a wrong signature but a good MAC). Filters may also become suspicious if there is a sudden increase on the level of out-of-order message arrivals. This could indicate an attack on the network or a malicious pre-filter. The filters should attempt to fix these behaviors when they are observed.

Three kinds of recovery actions are used to solve the above mentioned problems. These actions are taken depending on the extent of the perceived failures, but they should be safe from the point of view of the firewall operation:

- Redistribute load: For example, if a pre-filter provides a warning about its load, or high failure rates start to be observed for a specific pre-filter, the first course of action is to move some of the message flows to other pre-filters. This is achieved by specifying in the *ACKRES* response of a sender-module the identifier of a new pre-filter that should be used. At that point, the sender-module is expected to contact the chosen pre-filter to initiate communications through it.
- Increase pre-filtering capacity: If the existing pre-filters are unable to process the current load, a second course of action is to create more pre-filters (until a certain maximum is reached, depending on the hardware resources). To accomplish this, the filters contact the controller informing that a extra pre-filter should be started. When the controller receives $f_f + 1$ of such messages, it performs the necessary steps to fulfill the request (which are dependent on the actual deployment of the firewall). The new pre-filter begins by doing a few startup operations, which include the creation of a communication endpoint, and then it uses the TOM channel to inform the filters that it is ready to accept messages from the sender-modules.
- Kill pre-filters: When there is a reasonable level of suspicion on a pre-filter, the safest course of action is to have the filters ask the controller to destroy it. Moreover, if the load on the firewall is perceived as having decreased substantially, the filters select the oldest pre-filters for elimination, allowing eventual aging problems to be addressed. The controller carries

out the needed actions when it gets $f_f + 1$ of such requests (once again, these depend on how the firewall was deployed). The sender-modules that may have had their flows affected, will be informed about the replacement pre-filter through the *ACKREQ* mechanism (they will use another pre-filter to send the request, and get the information about a new pre-filter to be used in the response).

2) Filter failures: Since filters receive the same messages and execute in a deterministic way, they are expected to produce the same results. Therefore, it is possible to detect erroneous behaviors by comparing the outputs of the filters. Namely, when a receiver-module sees divergent messages being forwarded, for a specific sequence number of a given sender, this provides evidence that filters may have failed. Additionally, the controller should receive similar requests to update the pre-filters configuration, and missing or disagreeing messages also give an indication of a problem. Since we anticipate that filter failures will be rare, we decided to exclude automatic filter recovery — the component that discovers a misbehaving filter sends a warning to the system administrator, so that recovery can be initiated manualy.

3) Sender-module failures: By looking at the arriving messages, the filters may also detect some sender-module failures. In particular, we are concerned with behaviors that may influence the normal execution of the firewall. For example, if a correctly signed message arrives with a much larger than expected sequence number, this gives a clue that something may be wrong with the sender. More serious is a sender-module that is constantly complaining about the pre-filters (e.g., by apparently showing high failure rates), or that is transmitting at a speed above the allowed by the grant check (test (d) of Figure 3). Here, some defense action needs to be carried out, as these cause the firewall to spend extra resources (e.g., constantly moving the sender through the various pre-filters, or wasting network bandwidth).

To be conservative, we decided to follow a simple procedure to protect the firewall from a specific sender-module:

Filters maintain a counter per sender that is increased whenever new evidence of failure can be attributed to it. When the counter reaches a pre-defined value, the sender is disallowed from communicating with the firewall by temporarily removing it from the white list (test (a) of Figure 3) and a warning is sent to the system administrator. To let excluded senders regain access to the service, the counter is periodically decreased. When the counter falls below a certain threshold, the sender is moved back into the white list.

V. Overview of Deployment Alternatives

The firewall requires several separate components (filters and pre-filters) for effective deployment. However, costs are a major concern of any administrator. Therefore, we present a few deployment alternatives that can be made based on our solution. In any case, one must keep in mind that resilience usually has associated costs.

Figure 4 presents the rational for making deployment decisions. The considered solutions try to tradeoff costs with the use of virtualization [18], [19], [20]. In all options, different

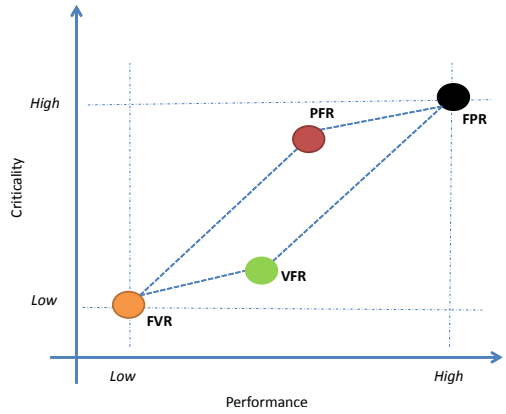

Fig. 4. Tradeoffs in some deployment alternatives.

components run in separate virtual machines and/or physical machines (or hardware boards). A solution with more physical machines is desirable for more critical systems, due to the higher fault isolation and also because of the potentially better performance. With virtualization, each physical machine supports several virtual machines, which means that there might be less machinery costs but performance can be reduced as resources are shared. Although virtual machines provide some level of isolation among the different components, preventing in most cases intrusions from propagating from one replica to the others, hardware faults may have an impact in all replicas.

- Full Physical Replication (FPR): every pre-filter and filter runs in a different physical machine.
- Full Virtual Replication (FVR): every pre-filter and filter runs in different virtual machines in the same physical hardware.
- Virtual Filter Replication (VFR): pre-filters and filters run in virtual machines, but the pre-filters are in a physical machine and the filters are in another physical machine.
- Physical Filter Replication (PFR): pre-filters run in virtual machines of the same physical machine, and each filter runs in a separate physical machine.

The controller should be run in a separate component that is protected from failures in the rest of the system. In the case of a virtualized environment, it can be implemented in the hypervisor, allowing complete access to the other components. With physical replication, there are a few alternatives, such as implementing it in a separate hardware module that can force the rebooting of a machine, or hybrid solutions that use a separate control network and a privileged software module [11]

VI. Conclusions

This paper presents a design for a resilient firewall that we are developing to protect the core services of a SIEM system. The firewall employs a two-level filtering strategy to improve performance and allows for some flexibility on the selection of fault tolerance mechanisms. The pre-filtering stage eliminates the most common forms of attacks in a very efficient way, while the filtering stage supports application rules for a more sophisticated analysis of the traffic. The

fault tolerance mechanisms consider a harsh failure scenario, where the components of the firewall may be compromised and behave arbitrarily. Two approaches are followed to address failures: the pre-filters use a detection and recovery solution, while the filters use BFT state machine replication and voting.

ACKNOWLEDGMENT

This work was partially supported by the EC through project FP7-257475 (MASSIF) and by the FCT through the Multiannual (LaSIGE) Program.

REFERENCES

[1] D. Miller, Z. Payton, A. Harper, C. Blask, and S. VanDyke, *Security Information and Event Management (SIEM) Implementation*. McGraw-Hill Education, 2010.

[2] The Security Division of EMC, "Security information and event management: Expectations for mid-sized organizations," RSA, Tech. Rep., 2010.

[3] M. Nicolett and K. Kavanagh, "Magic quadrant for security information and event management," Gartner RAS Core Research, Tech. Rep., 2011.

[4] ——, "Magic quadrant for security information and event management," May 2012. [Online]. Available: http://www.gartner.com/technology/reprints.do?id=1-1AOG9W9&ct=120529&st=sb&elq=1aacb714c9db45019551292e9050da2f

[5] J. Chauhan and R. Roy, "Is Firewall and Antivirus Hackers Best Friend?" iViZ Techno Solutions Pvt Ltd, Tech. Rep., Jan 2011.

[6] R. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*, 1st ed. John Wiley & Sons, Inc., 2001.

[7] S. Kamara, S. Fahmy, E. E. Schultz, F. Kerschbaum, and M. Frantzen, "Analysis of vulnerabilities in Internet firewalls," *Computers & Security*, vol. 22, no. 3, pp. 214–232, 2003.

[8] S. Surisetty and S. Kumar, "Is McAfee securitycenter/firewall software providing complete security for your computer?" in *Proceedings of the International Conference on Digital Society*, Feb 2010, pp. 178–181.

[9] Cisco, "Multiple vulnerabilities in firewall services module," Feb. 2007. [Online]. Available: http://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20070214-fwsm

[10] ——, "Multiple vulnerabilities in Cisco firewall services module," Oct. 2012. [Online]. Available: http://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20121010-fwsm

[11] P. Sousa, A. Bessani, M. Correia, N. Neves, and P. Verissimo, "Highly available intrusion-tolerant services with proactive-reactive recovery," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, pp. 452–465, 2010.

[12] T. Roeder and F. Schneider, "Proactive obfuscation," *ACM Transactions on Computer Systems*, vol. 28, no. 2, pp. 4:1–4:54, 2010.

[13] Y. Amir, B. Coan, J. Kirsch, and J. Lane, "Prime: Byzantine replication under attack," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 4, pp. 564–577, 2011.

[14] A. Mishra, B. B. Gupta, and R. Joshi, "A comparative study of distributed denial of service attacks, intrusion tolerance and mitigation techniques," in *Proceedings of the Intelligence and Security Informatics Conference*, Sep 2011, pp. 286–289.

[15] M. Garcia, A. Bessani, I. Gashi, N. Neves, and R. Obelheiro, "Analysis of OS diversity for intrusion tolerance," *Software: Practice and Experience (accepted for publication)*, 2013.

[16] J. Sousa and A. N. Bessani, "From Byzantine consensus to BFT state machine replication: A latency-optimal transformation," in *Oroceedings of the European Dependable Computing Conference*, May 2012, pp. 37–48.

[17] D. Harkins and D. Carrel, "The Internet Key Exchange," 1998. [Online]. Available: http://tools.ietf.org/rfc/rfc2409.txt

[18] H. Reiser and R. Kapitza, "Fault and intrusion tolerance on the basis of virtual machines," in *Tagungsband des 1. Fachgesprch Virtualisierung*, 2008.

[19] Xen, "http://www.xen.org/," 2013.

[20] Vmware, "http://www.vmware.com/," 2013.