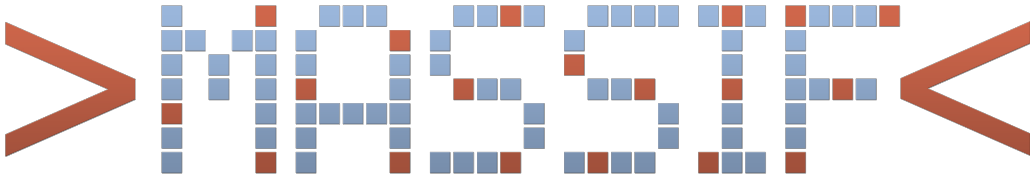


MAnagement of Security information and events in Service InFrastructures

MASSIF

FP7-257475

D5.1.1 - Preliminary Resilient Framework Architecture

Activity	A5	Workpackage	WP5.1
Due Date	M12	Submission Date	2011-09-21
Main Author(s)	Nuno Neves (editor) (FFCUL) Antonio Costa (FFCUL) Alysson Bessani (FFCUL) Paulo Verissimo (FFCUL)		
Contributor(s)	Bruno Vavala (FFCUL)		
Version	v1.0	Status	Final
Dissemination Level	PU	Nature	R
Keywords	Accidental failures and attacks; Resilient SIEM architecture; Secure Communication; Node protection mechanisms; Intrusion tolerance		
Reviewers	Luigi Coppolino (Epsilon) Claudio Soriente (UPM)		

Part of the Seventh
Framework Programme
Funded by the EC - DG INFSO

Version history

Rev	Date	Author	Comments
V0.1	2011-5-5	Nuno Neves	First version of the table of contents
V0.5	2011-7-11	Nuno Neves	Released initial version of the document
V0.6	2011-7-15	Nuno Neves	Released document for review
V1.0	2011-9-21	Nuno Neves	Final document, including comments from internal review

Glossary of Acronyms

AH	Authentication Header
BFT	Byzantine Fault Tolerance
DoW	Description of Work
EC	European Commission
ESP	Encapsulation Security Payload
EU	European Union
FP7	Seventh Framework Programme
HTTP	Hypertext Transfer Protocol
IDS	Intrusion Detection System
IP	Internet Protocol
IPsec	Internet Protocol Security
IT	Intrusion Tolerant
MAC	Message Authentication Code
MASSIF	MAnagement of Security information and events in Service InFrastructures
MIA	MASSIF Information Agent
MIS	MASSIF Information Switch
PU	Public Usage
R&D	Research & Development
REB	Resilient Event Bus
SIEM	Security Information and Event Management
SMR	State Machine Replication
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TPM	Trusted Platform Module
UDP	User Datagram Protocol
VM	Virtual Machine
VMM	Virtual Machine Manager

Executive Summary

The document presents a preliminary architecture for a Security Information and Event Management (SIEM) system, focusing on aspects related to the resilient operation of components and communications. The document includes a description of the models, building blocks and services. It analyzes the tradeoffs of fault and synchrony models that are used to guide the overall architecture, since these aspects characterize the nature of the problems (accidental or malicious) that must be addressed by the resilience enhancing solutions, and also impact on the kind of approaches that can be followed. The architecture covers the fundamental components that compose the system, such as the Edge and Core MASSIF Information Switches (MIS), and the way they are supposed to interact. Topological aspects that influence the placement of these components in the network are also discussed, as the services that must be offered in order to improve security. The document also provides a short description of the middleware services that could be employed to build the components, and explains several techniques that can be used to improve the resilience. These techniques include reliable and secure communication with strong semantics, replication (e.g., state machine replication and quorums), and approaches for proactive-reactive recovery with diversity management.

Contents

List of Figures

1 Introduction

This document gives a preliminary view of the MASSIF resilient framework architecture. The architecture is intended to provide seamless integration of resilience into distributed SIEM systems, with the objective of ensuring several levels of security and dependability in an open, modular and versatile way. Some of the features that characterize the infrastructures on which these SIEM systems may be used are the following:

- The infrastructures can be highly distributed and large-scale, both in a geographical sense and with respect to the number of entities involved;
- The infrastructures are heterogeneous, composed by end systems from possibly many vendors, with very diverse software and operating systems;
- The networks interconnecting the end systems can be of different kinds, from more confined and controlled ones to essentially open, generic and non-structured networks like the Internet.

These features are expected to become even more prevalent with the increasing inconspicuousness of computing systems and networks, and as security information starts to be collected not only from common networking devices (e.g., firewalls, routers, application servers) but also from various types of sensors that for instance observe physical processes (e.g., a dam). This trend contributes to make the monitored environments increasingly exposed to threats, and more prone to different sorts of failures. Since the SIEM subsystems that perform event collection, delivery and processing operate in essentially the same environments as the monitored systems, they can also become targets of attacks and the highly distributed nature of event sources allows for more common accidental problems. Therefore, it is important to improve the trustworthiness of SIEM systems by developing appropriate solutions to achieve resilience.

1.1 Rational for the Architecture

This section discusses the motivation and requirements that make up the rationale for the model and architecture addressing the resilience aspects of the SIEM systems. The diagnosis of the shortcomings of current SIEM systems, which led in part to the proposal of the MASSIF project, can be described succinctly by the following:

- inability of encompassing ICT infrastructures with global deployment, since they normally consider events from single organizations;

- incapability of providing a high degree of trustworthiness or resilience in event collection, dissemination and processing, thus becoming susceptible to attacks on the SIEM systems themselves;
- centralized rule processing, making scalability difficult by creating bottlenecks and single points of failure.

We establish the rationale for the MASSIF resilient architecture through a list of propositions that state a set of required macroscopic properties of the system. In consequence, the reader and/or potential developer or user can get a clear view of what is behind the architectural options proposed for MASSIF. Furthermore, since the architecture will be developed having in mind the requirements imposed by the above-mentioned propositions, one can gain confidence that the architecture and respective algorithms and middleware are bound to satisfy the imposed requirements.

- PROPOSITION 1: *Complement classical security techniques with resilience mechanisms*
  Classical security techniques are largely based on prevention, human intervention and ultimately disconnection. There is thus a need for achieving tolerance, automation and availability, both under attack and in the presence of major accidents [65].
- PROPOSITION 2: *Promote automatic control of macroscopic information flows*
  There is an growing need for SIEM systems to encompass multiple ICT infrastructures, achieving a global span. In such complex, large-scale, multi-tenant and distributed infrastructures, any security solution, to be effective, has to involve automatic mechanisms to secure the macroscopic command and information flows between the major modules, such as: between layers of different trustworthiness, from unprotected edge layers up to the more protected core realm; amongst peer layers implementing resilience-improving mechanisms.
- PROPOSITION 3: *Reconcile resilience with legacy preservation*
  One should modify and/or interfere with the observed system (payload) the least possible. As such, the SIEM system should preferably be deployed as a sort of overlay infrastructure, a system functioning in parallel with the payload system, with hooks to the latter at appropriate points. Likewise, resilience solutions should, in turn and as much as possible, be transparent to the functionality of the SIEM system and, in consequence, to the payload system. On the other hand, those solutions should be open and configurable, facilitating the porting to a diversity of SIEM systems.
- PROPOSITION 4: *Avoid single points-of-failure*
  This objective gains paramount importance with the increasing dependence on the availability of SIEM systems to secure the operation of on-line, often 24x7, large-scale infrastructures. As SIEM systems become more sophisticated and effective, there is an obvious trend for them to become targets of attack (neutralizing the sentinel) before direct attacks are staged on the payload systems. Avoiding this problem is one major reason for the objective, in MASSIF, of making the observing infrastructure itself resilient to direct attacks. Redundancy and diversity, both purposely introduced and derived from the sheer infrastructure richness and complexity, will be used to devise fault and intrusion tolerance mechanisms, keeping the system working despite the failure of individual components.
- PROPOSITION 5: *Secure timeliness in the presence of faults and attacks*
  Reconciling security with timeliness is a hard problem. Synchronous (or real-time) systems offer an additional attack plane to adversaries, where they can attempt to compromise the 'values' in

the system, but also the 'time' properties. This is why security solutions in distributed systems tend to be asynchronous. In systems providing a real-time view, and requiring real-time capability of reaction, achieving security at the cost of timeliness would be counterproductive. As such, one fundamental algorithmic and architectural challenge will consist in simultaneously preserving security and timeliness properties of the information flows coming from the collection points (the edge) to the processing engine (in the core) and vice-versa.

1.2 Organization of the document

The document is organized as follows. In Chapter 2, we will discuss two fundamental aspects of the system model, namely the fault and synchrony models. As mentioned above in the propositions, both these features are important to ensure that on one side the right level of security and dependability is enforced in the infrastructure, and on the other side, that some level of timeliness can be provided. Chapter 3 presents the architecture, addressing areas such as the main components and their topological organization in the network. After the functional description of the model and architecture, we enter the discussion of the resilience enabling aspects. In Chapters 4 and 5 we describe the functionality of the middleware support and explain the solutions that we propose to achieve resilience of MASSIF nodes. Appendix A presents the result of an experimental evaluation that was performed with the current version of the OSSIM SIEM tool, when installed in the default configuration. OSSIM was selected because of its large deployment base, making it one of the leading open source SIEM and a good representative of this sort of tools. The objective of the evaluation was to identify concrete potential security problems that can exist in streamline SIEM tools, which might be used to enrich and finesse the fault and attack model assumptions made within the context of the MASSIF project.

2 System Model

SIEM subsystems operate in heterogeneous and large-scale environments, with varying levels of exposure to attacks, and for which it is necessary to develop the right computational and resilience models that represent these characteristics. This is in contrast with settings in which the operational environment is more homogeneous, allowing a better (and simpler) understanding. The resilient SIEM architecture will necessarily encompass various nodes and devices, possibly connected through public networks, some of them operating at the edge of the system and performing data collection. We must consider that these edge nodes are typically less protected and that the communication environment might be untrusted. Other nodes, considered core nodes of the SIEM where data is processed, may be more protected. Nevertheless, they deserve a special care to ensure continuous operation (even if in a degraded mode). Therefore, it is necessary to be aware that risk factors may vary and may not be easy to perceive accurately, requiring that uncertainty is reconciled with security and timeliness requirements. For example, the different grades of real-time needs, from edge to core, should be considered in the design of the mechanisms for ensuring continuity and integrity of information flows. Additionally, other mechanisms should be in place for detecting timing failures when timeliness enforcement is impossible.

Given the simultaneous need for real-time, security and fault tolerance, this makes the problem of resilient SIEM operation hard vis-a-vis existing paradigms. As explained in the previous chapter, prevention techniques are certainly an important approach to deal with many threats. However, most defenses are dedicated to generic attacks and will likely be unable to resist to new, previously unknown, targeted attacks. Therefore, we believe that there is the need for achieving tolerance in addition to prevention. The design of solutions based on this paradigm can however only be accomplished with a good understanding of the fault and synchrony models that are more appropriate to each part of the architecture.

2.1 Fault model

The definition of the fault model is an important aspect upon which the system architecture is conceived, and component interactions are defined. The fault model conditions the correctness analysis, both in the value and time domains, and dictates crucial aspects of system configuration, such as the level of redundancy, the characteristics of the algorithms, and the placement and choice of components. Failure assumptions of a fault model can typically be organized in two classes: controlled and arbitrary failure assumptions.

Controlled failure assumptions specify qualitative and quantitative bounds on component failures. For example, the failure assumptions may specify that components only have omission failures, and that no more than f components fail during an interval of reference. Alternatively, they can admit value failures, but not allow components to spontaneously generate or forge messages, nor impersonate, collude with, or send conflicting information to other components. This approach is extremely realistic,

since it represents how common systems work under the presence of accidental faults, where they typically fail in a benign manner (e.g., by crashing), but occasionally could produce some erroneous value. This idea can be extrapolated to malicious faults (often called Byzantine faults [40]) caused by some adversary, by assuming that they are qualitatively and quantitatively limited. However, it is traditionally difficult to model the behavior of a hacker or a malicious person that is willing to disrupt the system, so there is a problem of potential incorrectness of assumptions (lack of assumption coverage) that does not recommend this approach unless it can be enforced with very high probability.

Arbitrary failure assumptions specify no qualitative or quantitative bounds on component failures. Obviously, this should be understood in the context of a universe of "possible" failures of the concerned operation mode of the component. For example, the possible failure modes of interaction, between components of a distributed system are limited to combinations of timeliness, form, meaning, and target of those interactions (let us call them messages). In this context, an arbitrary failure means the capability of generating a message at any time, with whatever syntax and semantics (form and meaning), and sending it to anywhere in the system. Practical systems based on arbitrary failure assumptions very often specify quantitative bounds on component failures, or at least equate trade-offs between resilience of their solutions and the number of failures eventually produced. For instance, by employing cryptographic algorithms to protect the messages, it is possible to prevent attacks on the network that attempt to modify or generate new messages (because these messages will be recognized as faulty at the receiver, and therefore, will be discarded). Additionally, since it takes some effort and time to compromise a component, it is acceptable to assume that over a certain interval at most f components will be intruded by the adversary,

Hybrid failure assumptions combine both kinds of failure assumptions. Generally, they can consist of allocating different assumptions to different subsets or components of the system, and have been used in a number of systems and protocols [18, 4, 58]. Hybrid models allow stronger assumptions to be made about parts of the system that can justifiably be assumed to exhibit fail-controlled behavior, whilst other parts of the system are still allowed an arbitrary behavior. For example, commodity computers are starting to be deployed with a Trusted Platform Module (TPM), which can perform a limited set of operations in a secure way, even if the rest of the machine is compromised and controlled by an adversary. Alternatively, consider a computer with virtual machines, where the hacker can intrude the guest operating systems using normal exploit techniques, but the hypervisor can be kept correct because the attack surface is much smaller (and therefore the entry points can be carefully programmed and monitored). This kind of organization is advantageous in modular and distributed system architectures, but it can only be employed if the the model is well-founded, that is, the behavior of every single subset of the system can be modeled and/or enforced with high coverage (which takes us back, at least for parts of the system, to the problem controlled failure assumptions).

Given the highly distributed nature of SIEM systems, the fault model must consider the networking environments and the nodes, and must take into account the differentiated level of threats in distinct parts of the architecture. Therefore, in what follows we analyze the potential faults affecting the flow of information from sensors to the core SIEM systems.

2.1.1 Edge layer

At the edge layer there will be sensing node devices that produce events (e.g., SYSLOG events), and then transmit them to event collectors, also at the edge layer (see a more complete discussion about the network topology in Section 3.2). Devices are exposed to several kinds of attacks, and in the extreme case,

they can be intruded by a hacker. The attacks can cause various forms of disruption, such as the deletion of specific events or complete removal of the logs, modification of event data (e.g., change some value) or creation of spurious events. However, although these problems can be severe, it is typically impossible for an adversary to have enough resources to compromise all devices at the same time. Therefore, the system will not fail as a whole, but only gradually – from a global perspective there will be partial failures leading to a increasingly degraded service, but mechanisms may be sought to reconfigure and recover the system from this problem. At least, if the right monitoring capabilities are in place, it should be possible to detect such kind of faults through correlation at the core layer.

The network that connects sensors and event collectors might also fail. This can occur either accidentally (omissions and/or crash failures), or due to attacks that tamper with the standard protocols conveying the information. In particular, the event flows can be interrupted or delayed (e.g., by controlling a router), and individual events can be for instance re-ordered, replayed, or forged. Once again, it is reasonable to assume that the adversary has limited power, and therefore, that she is only able to disrupt the networking environment in a partial way. Mechanisms can be deployed to detect such faults, which can be based on relevant sets of collected information allowing correlation and fault diagnosis (e.g., time stamps and their validity) or on structural protocol invariants that may be checked for correctness (e.g., a periodic event transmission did not arrive).

Collector nodes, on which SIEM services and protocols are executed, might also be the target of intrusion and their operation might be disrupted. However, since these nodes are managed by the SIEM solution, it is possible to deploy specific measures to protect their operation. In particular, depending on the value of the data being collected, distinct mechanisms can be implemented in order to achieve different levels of resilience (and typically also cost).

2.1.2 Edge to core communications

The networks through which events are transmitted to the processing nodes are prone to several kinds of failures. Depending on the configuration of the observed system, the information might be sent through a local LAN, and therefore, it is easier to enforce a more controlled behavior. For organizations with offices spread across a region, in most cases the communication has to be provided by some third party telecom operator, which has its own policies regarding for instance security. In both cases, the communications can fail accidentally due to the crash of some node or messages can be lost because of network congestion. Attackers can also tamper with the SIEM protocols for conveying events between the edge and core nodes, causing for example the delay, re-order, or replay of messages.

2.1.3 Core layer

The core layer, which includes the processing engine that does correlation on the events, is typically protected with some sort of devices (e.g., a firewall) aimed at increasing the resilience from external attacks. However, these devices can also become a target of attack – in fact, over the past years, several vulnerabilities have been described for the most commonly used firewalls [36, 15, 46]. This means that in order to ensure the safe operation of the core layer, specific mechanisms will need to be developed to offer higher levels of resilience to attacks and also to control the in and out flows of information. Given that attackers are assumed to have limited power, avoiding single points-of-failures by using enough redundancy will be one way to deal with these problems.

Certain services running in the core layer may require specific mechanisms to ensure correct operation even under improbable attack scenarios. For example, the historian is responsible for storing collected events and information in such a way that it can be presented and used in a court of law. In this case, it may make sense to consider a scenario where an inside employee could try to disrupt the operation. Addressing these problem requires the design of specific solutions, based for instance in redundancy and self-healing capabilities, and strict authentication and access control policies.

2.2 Synchrony model

The synchrony model refers to assumptions related to the notions of time and timeliness. Since the complexity of the solutions and the correctness of the system depend on these assumptions, they should reflect as accurately as possible the real characteristics of the execution environments. Traditionally, distributed systems have been developed by considering one of the two extreme models of synchrony. The asynchronous model, also called time-free model, does not make any time-related or timeliness assumption. On the other extreme, the synchronous model assumes that all system activities are executed within known temporal bounds, which includes local activities (process execution) and distributed ones (message transmission). However, many real systems are not fully asynchronous nor fully synchronous. Therefore, there exist models of partial synchrony to cover various intermediate cases, for instance assuming that there are reliable local clocks or that only some components are temporally predictable.

The environments considered in MASSIF are heterogeneous in several aspects, also with respect to timeliness. Therefore, it will be wise to consider different synchrony models or different assumptions depending on the characteristics of the specific environments or networks. Hence we review the main synchrony models and discuss their appropriateness for MASSIF.

2.2.1 Asynchronous model

In one of the extremes of the synchrony spectrum we can find the asynchronous model. In asynchronous systems there is absolutely no notion of time, which means that there are no assumptions about the relative speeds of processes, about the delay time to transmit and deliver messages or about the existence of time references [31]. Given that it is impossible to specify timed services in this model, some authors prefer to call it the time-free asynchronous system model [20], to make a distinction with timed asynchronous models.

An extremely attractive aspect of the asynchronous model is its simplicity. Since it makes no assumptions about timeliness, any solution designed for the asynchronous model is easily ported to any other model making stronger synchrony assumptions. Moreover, the coverage achieved with the implementation of asynchronous solutions is the best possible, since any behavior with respect to timeliness is permitted by the model. In other works, since no assumptions are made about the temporal behavior of the system, any activity can take as long as necessary without compromising correctness.

This model is thus appropriate for the parts of the infrastructure that are exposed to malicious attacks, which could disturb, delay or deny the execution of operations. In fact, protocols developed under the asynchronous model are immune to these attacks because they do not depend on any timeliness assumption and are always correct independently of real delays.

On the other hand, the absence of time notions makes it impossible to satisfy temporal requirements or enforce some required levels of Quality of Service. The system performs in a best-effort way since per-

formance cannot be controlled. To some extent, with asynchronous models there is a trade-off between safety and the ability to deal with Quality of Service (QoS) requirements.

Finally, it is important to note that in the asynchronous model it is impossible to deterministically solve agreement problems, such as consensus or total order broadcast, in the presence of even a single process crash [31]. This extremely important result (usually referred to as the FLP impossibility result) implies that any problem requiring processes to reach agreement cannot be solved without making some synchrony assumptions or assuming failure-free environments.

2.2.2 Synchronous model

The synchronous model lies on the other side of the synchrony spectrum, in opposition to the asynchronous model. In synchronous systems both communication delays and processing delays are known and bounded, and the rate of drift of local clocks is also known and bounded.

A direct consequence of these assumptions is that clocks can be synchronized, which can be useful to perform synchronized actions and to order distributed events. Perhaps even more important is that in synchronous systems it is possible to address the timeliness requirements of applications by means of appropriate algorithms and protocols.

The synchronous model would therefore be the elected model to address timeliness, performance or QoS requirements that exist in MASSIF. However, this model suffers from a major drawback, which is related to the lack of coverage of the synchrony assumptions. The problem is that in some of the considered execution environments it is difficult, inadequate or even impossible to determine worst-case load scenarios and upper bounds for the communication and processing latencies. In consequence, assumed bounds may not always hold, compromising the correctness of the SIEM system. The problem is even amplified if one considers the possibility of time-based attacks. These attacks may introduce artificial delays, leading to the violation of assumed temporal bounds, or may change values (e.g. time stamps), which can lead to temporal disruptions and ultimately to the violation of safety properties. In summary, considering the synchronous model when the infrastructure is unpredictable, unreliable or prone to attacks, is an impediment for achieving trustworthiness.

An adequate approach to the problem of handling timeliness requirements in uncertain environments requires taking a step back and relaxing the synchrony assumptions. This leads us to consider models of partially synchrony.

2.2.3 Partial synchrony models

One approach to escape the problems encountered by developing solutions under the synchronous or asynchronous models is to consider partial synchrony. Partially synchronous models essentially make additional assumptions that allow achieving important properties or circumventing impossibility results like the FLP one, without falling into the problems caused by the lack of assumption coverage.

The partially-synchronous model [27, 29] was one of the first to adopt the concept of partial synchrony. In this model it is assumed that there exist fixed upper bounds for the relative speeds among processes and for the message delivery delays, but that these bounds are not known a priori or they will only hold after an unknown time instant, called Global Stabilization Time (GST). With any of these assumptions it is possible to design various solutions for the consensus problem, with different resilience to failures.

Another approach to increase the strength of the asynchronous model, also allowing circumventing the FLP impossibility result, is to assume the existence of an external failure detection mechanism that provides information about crash failures. The asynchronous model augmented with unreliable failure detectors was first introduced by Chandra and Toueg [13]. In this model the system is fully asynchronous, but each process has access to a local failure detector module that is allowed to make mistakes. Failure detectors provide a very elegant way to structure consensus-related algorithmic, requiring the system to be or become synchronous enough, and for long enough periods, in order to solve consensus [14].

The timed asynchronous model [21] can be described as being fundamentally an asynchronous model with the additional assumption that processes have access to a physical clock with a bounded rate of drift. The authors of [21] have observed that most computing systems have high-precisions quartz clocks, which renders the assumption reasonable enough. Practical systems can then be built in infrastructures that alternate between synchronous and asynchronous behavior, with the system making progress when there is enough synchrony, being possible to detect timing failures otherwise.

Finally, another example of a partial synchrony model is the quasi-synchronous model that was proposed by [61]. Here it is assumed that bounds exist for the system properties (process speed, message transmission delay, clock drift rate), but some of the chosen values have a non-null probability of being incorrect.

2.2.4 Hybrid models

The previously mentioned models of partial synchrony build on the idea that the infrastructure will eventually behave in a synchronous way. They implicitly believe that synchrony is not a homogeneous property in the time domain, that is, that systems become faster and slower during the execution and thus that synchrony comes and goes. But it is also possible to consider that synchrony is not a homogeneous property in the space domain. Some parts of the infrastructure may be more predictable and synchronous while other parts may not. This perspective was introduced in [62], and it later led to the Wormholes hybrid distributed system model.

Both perspectives are important and relevant for defining protocols in uncertain and attack-prone environments, such as those considered in MASSIF. However, there is an important difference between relying on heterogeneous synchrony in the time and in the space domains. In the former case, one just *expects* the system to eventually become synchronous, whereas by exploring the space dimension i.e., acting on the system structure, one *makes* the necessary synchronism happen. From a trustworthiness perspective this difference can be crucial, since the time-domain behavior for at least one part of the system is well-known and can be relied upon.

A hybrid synchrony model, in which different synchrony assumptions can be stated for different parts of the system, presents several advantages over homogeneous models, as explained in detail in [60]. In the context of MASSIF, a hybrid synchrony model can be explored for instance by assuming that some nodes (e.g., edge nodes) have trusted components able to deliver trustworthy time stamps. In a more general sense, and given that hybridization has to be enforced by construction, the overall distributed system model can be described as a hybrid distributed system model, composed by trusted components that are added to the baseline legacy, unreliable and intrusion-prone components.

One general assumption that may be made in MASSIF is that nodes at the edge and core layers have access to local clocks providing a global notion of time. This assumption can be enforced either through the use external synchronization with an absolute reference (e.g., GPS time synchronization) or through internal synchronization using clock synchronization protocols such as the Network Time

Protocol (NTP). GPS-based clock synchronization is considered more robust to faults and attacks, given that it relies on radio signals directly received from satellites, which are typically trustworthy. The disadvantage is that it requires specialized hardware and may not be easily implementable in practice, since the receiver antennas need to be in direct view of satellites. On the other hand, software-based internal synchronization relies on the network for the exchange of synchronization messages, possibly suffering from malicious or just accidental faults. Fortunately, clocks can usually be kept in synch, provided that a minimum level of synchrony is eventually achieved from time to time, which is very likely to be true. Furthermore, NTP can be considered an off-the-shelf solution, readily available in main stream operating systems.

One important implication of assuming the existence of a global notion of time is that time stamps can be used, in general, to infer about the timeliness of events and about their ordering. This is very important in the context of MASSIF, for the operation of the SIEM.

Some nodes at the edge layer, namely sensor nodes, may not have access to local clocks (e.g., physical sensors). This means that events produced by these nodes cannot be time stamped locally, but only at collector nodes. The accuracy of these time stamps with respect to the real time instant at which events were produced will then be dependent on the behavior of the network. In any case, since this time stamp can be made trustworthy by construction, this should make it easier to reason about the validity of the collected events at higher abstraction levels.

In the case where sensor nodes have access to local clocks, it can happen that the produced time stamps may not be trustworthy. In practice, the situation will be similar to the one mentioned above, although here the time stamps produced at the sensor node might be used for correlation purposes and for inferring about the trustworthiness of collected sensor information.

Regarding the assumed synchrony properties for the networking infrastructure, one should assume that there is eventual synchrony, that is, that message transmission latency is bounded, although it is difficult to state the exact bound. Given the real-time requirements, specific bounds may have to be assumed, which means that the network will alternate between synchronous and asynchronous behavior. We must note that the underlying infrastructure can be the target of attacks (is not trusted by default) and therefore this must be reflected on the assumed synchrony. This will also dictate the kind of architectural solutions and protocols that will be developed in MASSIF.

3 Architecture Description

This chapter presents the architecture. It begins by introducing the key options of the architecture, in the context, when appropriate, of the propositions enumerated in Chapter 1 and of the requirements laid down in the Scenario Requirements deliverable (D2.1.1) [16]. Next, the structural model is explained, proposing a topology relating the payload system (observed system) with the SIEM system (observing system), discussing the network structure and the placement of the main components. Then, these main system components are discussed in more detail. Finally, the chapter gives an overview of the services to be provided by the SIEM architecture.

3.1 Key architectural options

When reflecting about the key aspects of the MASSIF SIEM architecture, one has to take into account the need for a resilient architecture considering:

- Different interaction realms, such as: multiple and (mainly) unprotected edge facilities; hostile large-scale communication environment; more protected, centralised or decentralised core facilities.
- Distinct levels of risk accepted for different instantiations of the architecture in various scenarios, leading to different levels of resilience as a tradeoff for cost and complexity.
- The difficult combination of characteristics such as: security, timeliness, multi-tenancy.

The main desirable characteristics of the SIEM architecture must be laid down so as to fulfil a set of more specific requirements, both functional and non-functional, which we outline below. Functional requirements are mainly dictated by the findings of [16] cited above, whereas non-functional requirements are mainly implied by the rationale propositions.

Functional requirements:

- scalable data acquisition of huge amounts of events, from diverse and geographically spread nodes;
- distributed and real-time collection, aggregation and processing of events; alert generation; and incident notification;
- integrated and distributed correlation engine implementation alternatives;
- clear decoupling between the target and the SIEM system, for minimal impact on the observed infrastructure.

Non-functional requirements:

- high resilience of whole SIEM system under attack, concurrent component failures, and/or unpredictable network operation conditions;
- event flow protection, from the collection points through their distribution, processing and archival;
- authenticated and unforgeable component status reporting;
- modular functions and protocols, to be re-used by different instantiations of the architecture;
- flexible and incremental solutions for node resilience, providing for seamless deployment of necessary functions and protocols.

We propose to address these requirements by an architecture structure as described below, having the following main characteristics:

- A topology laid down as a logical overlay over the target system, so as to preserve legacy but allow seamless integration of the observing and observed systems.
- Modular structure achieved by concentrating all functions in configurable conceptual devices which act as the nodes of the overlay: MASSIF Information Switches (MIS). The MIS are usually hardware implemented, however there can be software based implementations, called MASSIF Information Agents (MIA).
- Information flow in the overlay implemented as a secure and real-time event bus, modeled essentially as a producer-consumer SCADA-like [1] system upstream, with low-bandwidth commands downstream.
- Resilience procurement based on: securing the information flow; making the dissemination infrastructure itself (event bus) resilient; protecting crucial processing units (MIS, MIA) with incremental resilience strategies relying on hardware and software based alternatives; and differentiating between edge-side and core-side configurations.

3.2 Structural model

The structure of a MASSIF SIEM system is shown in Figure 3.1. We model both the payload and the SIEM system interconnection as a WAN-of-LANS [63], a useful construct to represent loosely-coupled wide-area infrastructures such as some of those envisaged as of the target scenarios for the MASSIF technology. They are typically made-up of several facilities sometimes widely separated geographically, whose local intranets are interconnected through public networks like the Internet, possibly under the protection of secure channels or tunnels. It is easy to decouple the threat scenarios faced by the WAN part from the LAN parts and, moreover, it is quite simple to consider distinct levels of trustworthiness for different selected facilities and their LANS. The 'LAN' concept is used in a generic way to mean "short-range", whose implementation may in fact involve switching or routing topologies at layers 3-1.

[1] Supervisory Control and Data Acquisition are distributed systems used in physical infrastructures, often large-scale, e.g., electrical grids, which, as the name implies, acquire data from all the infrastructure, to feed a real-time dynamic image of its state, and sometimes produce control decisions, which are materialized by commands back down.

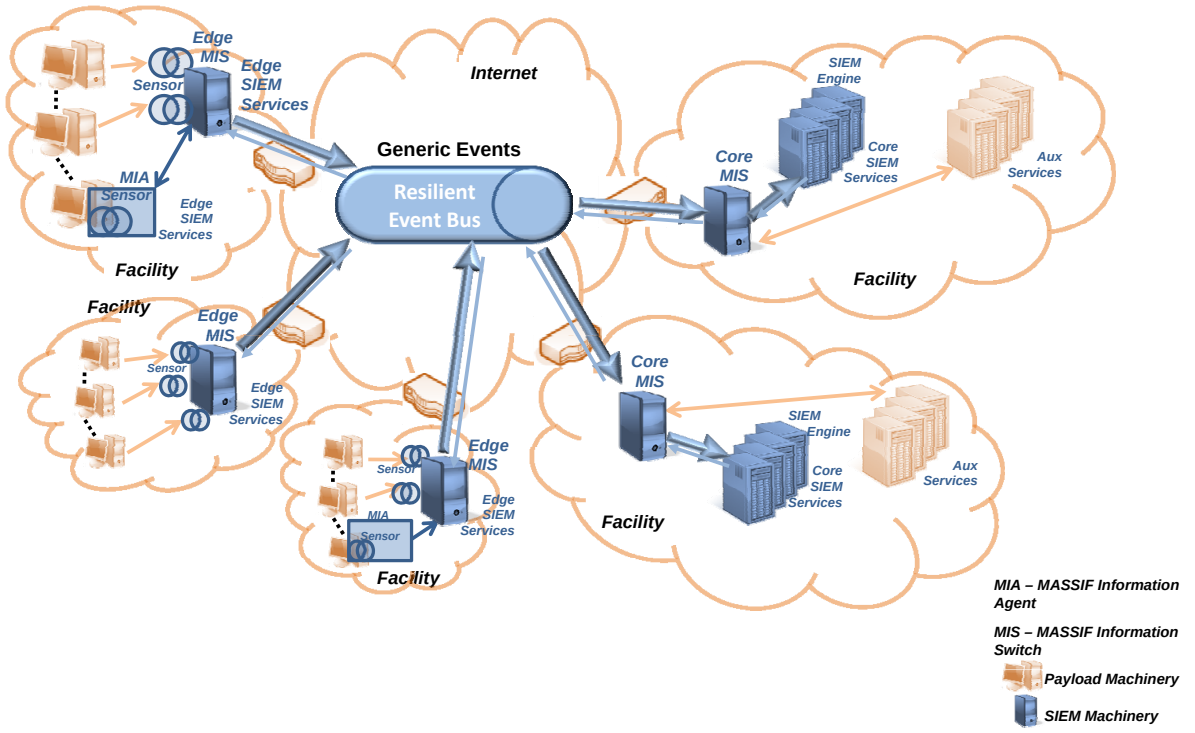


Figure 3.1: MASSIF overall architecture structure - payload vs. SIEM.

We can notice that the payload system can retain its essential characteristics when the SIEM infrastructure is superimposed on it, since both work essentially in parallel. The hooks or contact points between both are materialised by the devices mentioned above: the MASSIF Information Switches (MIS) and the MASSIF Information Agents (MIA). The Resilient Event Bus (REB) is an abstraction created through a suite of protocols that will be discussed ahead, and which can use essentially the same kind of substrate of communication as the payload system. More secluded architectures for highly critical applications can nevertheless be foreseen, with dedicated secure circuits or virtual private networks.

The essential difference between the edge-MIS and the MIA depicted in Figure 3.1, is that the first is implemented by a 'box' which resides on the network and can be addressed by any device of the payload, through standard protocols like TCP/IP. This does not involve any modification of the information producing devices, which send their log, event or alarm files to the nearest edge-MIS. However, certain devices, such as firewalls or IDS (Intrusion Detection Systems) are so rich and sophisticated in the information they provide, that it makes sense to incur the cost of porting (some of) the MIS services to a software module compliant with the architecture of the former. The gains are the capability of preprocessing and filtering the information, and even tuning the firewall or IDS devices, and the reliability of MIA to edge-MIS communication, which can use the reliable protocols of MASSIF.

As shown in Figure 3.1, the REB conveys this information collected by the edge-MIS to the core part of the architecture, where the SIEM core processing engines reside. Once more, the contact point is a MIS, this time a core-MIS (the difference lying essentially on the services it runs, and on the robustness of its construction, as we will see ahead), which communicates reliably with the core SIEM engines, at the same time it protects them from external attacks, acting pretty much as a sophisticated firewall.

3.3 Main system components

The main building blocks of the architecture are: the edge and core MIS, which are typically implemented in a separate machine/device; the MIA, which will be software implemented; the REB, implemented as a set of protocols being run among the edge and core-MIS; and, the SIEM engine (which includes, besides the event processing engine, other services like the historian, where the security and event information is stored; and graphical interfaces). Incremental levels of resilience may obtained both at micro (local node architecture) and macroscopic levels (inter-node algorithms), by the definition of tradeoffs between resilience and cost, complexity or performance of the solutions.

Figure 3.2 depicts in a block diagram of the architecture. The bulk of MASSIF distributed services reside between what we call *Edge-side Infrastructure Interface* and the *Core-side Infrastructure Interface*, and are implemented by the MIS or MIA.

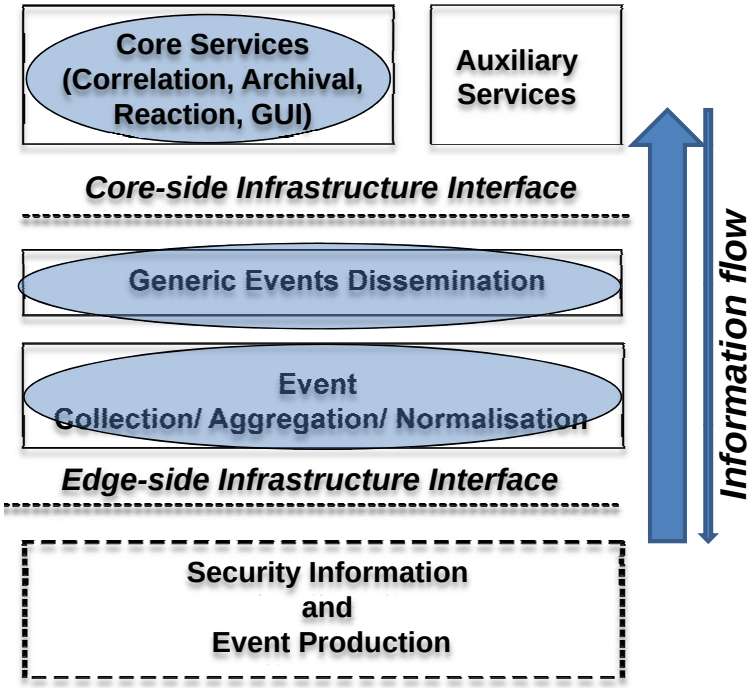


Figure 3.2: MASSIF Architecture Block Diagram

The lower layer is materialised by the payload devices, supplying raw security information and event data. These devices can be of different forms depending on the application scenario, but they can consist of specialized servers (e.g., a mobile payment application), network management and protection systems (e.g., a firewall or an IDS), and physical sensors (e.g., water level sensor). This data is offered at the Edge-side Infrastructure Interface and is collected by the edge-MIS and/or MIA. Individual MIS or MIA implement services related to data acquisition, namely they do at least event collection. However, they might also implement other services, namely the normalization of the event formats, aggregation of several events, and some local processing and correlation.

The set of MIS devices run the secure, reliable and real-time communication protocols needed to implement the REB. The REB performs generic event dissemination towards the processing engine in

the core-side of the infrastructure. The bulk of the traffic will be in this direction, from the edges to the core, but in some cases it might be necessary to relay back commands. For example, when there is a suspicion that an attack might be in progress, the processing engine might instruct the edge MIS and/or sensors to start collecting more detailed information about the status of the network and specific nodes. Additionally, the reaction components at the core may send commands to selected network management systems to stop the progression of attacks.

The SIEM engine runs core application services like correlation, and archival of event information. In MASSIF, for example, it will also include reaction mechanisms and tools for the visualization of security events and policies.

3.3.1 MASSIF Information Switches (MIS)

The MIS can be built with incremental levels of resilience, depending on its criticality. The edge-MIS is the simpler instantiation, since it is placed at more locations on the data collector side and costs may be a concern. It receives information with a limited degree of trustworthiness, since it is produced by untrusted machines and mainly conveyed by standard protocols.

The edge-MIS is located on the sensors side, it is normally single-homed, but in some cases may be dual homed for protection of specific bulk and/or critical source traffic (i.e., IDS). The core-MIS is positioned on the core processing elements side, and it is normally dual-homed, to actively protect the core SIEM servers.

The trustworthy MIS-to-MIS interconnection, through the communication services implemented by the REB, secures information flows. The flow from edge to core, as the figure shows, is expected to have much greater bandwidth than the flow in the opposite direction, used to carry commands in reaction to the analysis performed by the correlation engines.

3.3.2 MASSIF Information Agents (MIA)

The MIA is a software appliance residing with end target hosts, implementing a smart sensor, that is, a MASSIF compliant sensor where part of the usual MIS functions may be performed. It may in consequence be co-located with important event sources (e.g., a relevant application machine). MIAs require minimal host modifications to achieve MASSIF functionality. The trustworthy MIA-MIS interconnection through MASSIF communication services secures the information flow, in a way not possible to achieve by the standard device-to-MIS communication.

3.3.3 Resilient Event Bus (REB)

The collection of MIS devices run the secure, reliable and real-time communication protocols needed to implement the REB. The event bus should encompass both events created by the periphery and events generated from within the SIEM machinery.

Figure 3.3 offers a view of the architecture stack, detailing the block diagram view shown previously in Figure 3.2. As the figure shows, the L-shaped structure of the REB aims at providing a *generic events* abstraction where events from different origins coalesce on the event bus layer. Generic events are events

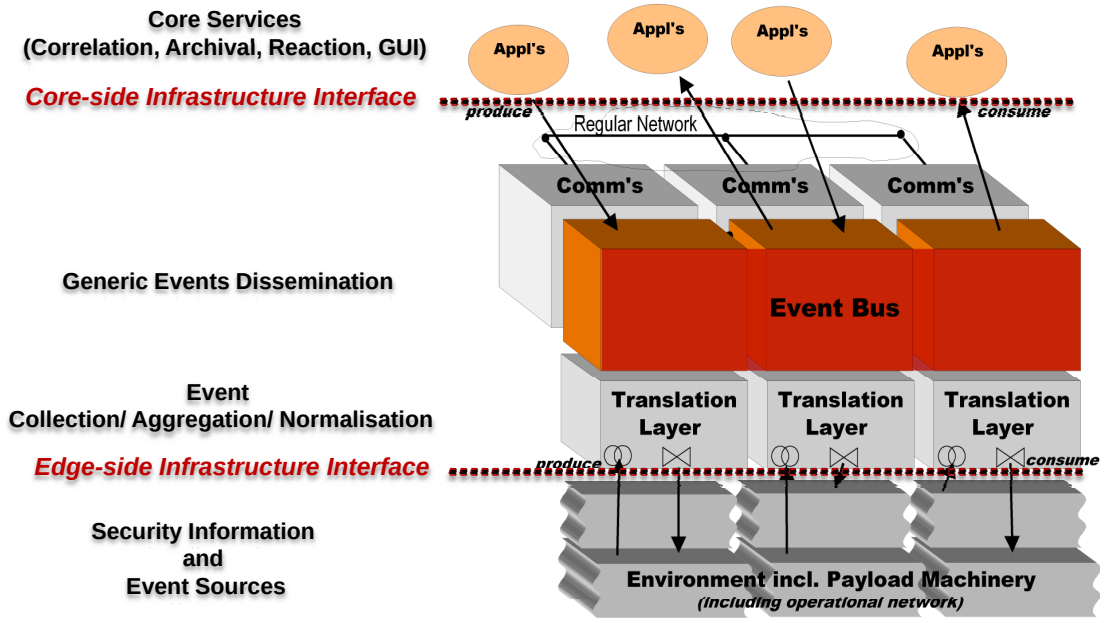


Figure 3.3: MASSIF Resilient Event Bus.

following a common syntax and semantics, whatever their origin: (i) events produced or consumed by the environment; or (ii) events produced or consumed by the SIEM machinery.

The *Environment* is anything from which sensory information should be collected. It includes the payload machinery containing the security information and event sources, and the operational networks, i.e. any networks used to convey information from sensors, dedicated or not.

The *Translation* layer captures and processes sensory information of diverse origins, transforming it into generic events. To achieve this purpose it implements services such as collection, aggregation and normalisation. The translation layer may be implemented in a MIS (general case), in a MIA (smart sensor case), or split between both.

The *Event Bus* layer provides a "generic events bus" abstraction, that is, it publishes events following a common syntax and semantics, whatever their origin, and following pre-defined delivery reliability and causal and temporal ordering properties. It is bidirectional but asymmetric: upstream, it conveys high-throughput data, sourced by edge-MIS and sinked by core-MIS; downstream, it conveys low-throughput commands, sourced by core-MIS and sinked by core and/or edge-MIS, and possibly passed-on to environment machinery, e.g. MIA, by the latter.

The *Communication* layer is concerned with the MASSIF protocols responsible for the resilient propagation of events via the regular network system (the WAN-of-LANS mesh), creating the "bus" abstraction in all MIS units. Actual media may involve a combination of the Internet, and/or corporate LANs and WAN. Native media redundancy may be used, e.g., dual MPLS links.

3.3.4 SIEM Engine

The SIEM engine performs complex event analysis, normally using the stream data processing model [17]. The SIEM engine's core application services process the information and events arriving from the edge, trying to find correlations in the data and detect anomalies (failures, intrusions). Besides correlation, data is also archived in resilient storage, in order to allow ulterior forensic analysis. Reaction modules may generate commands to modify the sensing and collecting conditions, or even modify protection or filtering apparatus like firewalls or IDS, namely those mediated by MIA.

SIEM engines can be integrated or distributed, and its functions can be centralized or decentralised. In SIEM engine implementations compliant with the MASSIF architecture, these variants can be transparently supported by the modularity provided by the MIS, and core-MIS resident services may manage issues like query fragmentation or dispatching, parallelism and replication.

3.4 Overview services

Below is given an overview of the main services that are provided by the components of the resilient architecture. We start we a description of two more generic services, communication and protection, and briefly explain more specific edge and core services.

3.4.1 Overview of generic services

There are two generic services offered by the collection of MIS: communication and protection.

Communication The communication service is implemented by protocols running amongst the MIS. This service should be resilient both to accidental and malicious (or Byzantine) faults, given the fault-/attack model outlined for MASSIF (see Chapter 2). It should take advantage of the overlayed MASSIF networking infrastructure, for example to achieve routing resilience. It can benefit from the foreseen asymmetry between edge-MIS and core-MIS in terms of sinked and sourced throughput, for more efficient solutions. The combination of security and real-time requirements however makes the implementation of this service a challenging objective.

Protection The protection service is implemented by services residing in a dual-homed MIS, which acts as a bastion providing perimeter defense to specific critical subsystems, such as the core SIEM engines. As depicted in Figure 3.1, the SIEM engines lie behind a core-MIS, which filters all access, both from the network and from the facility intranet. In fact, note that the Auxiliary services, which belong to the payload, can only interact with the SIEM engine via the core-MIS.

3.4.2 Overview of edge services

As depicted in Figure 3.2, the edge services feed on the raw security information and event data provided by the payload devices, at the Edge-side Infrastructure Interface. This information is collected by the edge-MIS or MIA, which may implement the following services:

1. *Collection* : the events are collected from the periphery machinery by sensors which acquire the data through the appropriate standard protocols (e.g., SYSLOG).
2. *Aggregation* : related events are aggregated, to avoid repeated notifications of the same real life cause event, and prevent event showers from cluttering the processing elements.
3. *Anonymisation* : for privacy or confidentiality reasons, it may be necessary to anonymize the events being sourced by specific components.
4. *Normalisation* : all events and information are translated into a generic MASSIF event format, to hide heterogeneity from the upper layer services.
5. *Correlation* : typically a core service to be performed by the SIEM engine. In the MASSIF architecture, we may want to transfer some intelligence to the periphery, in the interest of balancing the load of the core engines and/or reduce communication traffic. As such, it may be foreseen that the edge-MIS perform some form of correlation, specifically about the semantics that can be learned from components in a same facility intranet, for example, common mode attack or failure syndromes.

3.4.3 Overview of core services

Likewise, the following services are implemented at the core, by the SIEM engine and the Historian and GUI servers.

1. *SIEM engine* : the SIEM engine implements the main function of the core, namely the correlation of events. Additionally, as a consequence of this processing, alarms can be raised and reactions can be triggered, some of them to be sent down in the form of commands.
2. *Historian* : the Historian manages the archival service, using the resilient storage service provided by the middleware.
3. *GUI* : the graphics user interface provides the console operation, and it is by no means less critical, since loss or corruption of view may be as serious as the intrusion of the engine itself.

4 Resilient Middleware Support

This chapter and the following discuss approaches to make the SIEM infrastructure and services resilient to faults and attacks of possibly large and/or uncertain magnitude. It is interesting to start by analysing the attack vectors discussed earlier in Chapter 2, put in context with the MASSIF architecture, as depicted in Figure 4.1. As shown in the figure, in such a distributed and large-scale architecture, there are obviously several attack vectors:

- integrity of the sensing flow, which typically uses standard protocols (illustrated as arrow 1);
- edge-MIS, targeting its availability and/or the integrity of event collection and/or communications (arrow 2);
- MIA, with the objective of attacking its availability and/or the integrity of remote event collection and/or MIA-to-MIS communications (arrow 3);
- Event bus, targeting its confidentiality, integrity and availability (arrow 4);
- core-MIS, aiming at attacking its availability and/or the integrity of the protection service and/or the communications (arrow 5);
- SIEM engine, targeting its availability and/or the integrity of the core services (arrow 6);
- integrity of the interaction with the auxiliary services (arrow 7).

These attack vectors have to be prevented from compromising the correctness of the SIEM system, by employing the appropriate mechanisms and protocols that safeguard the operation of the nodes and the communications. These mechanisms and protocols will be implemented in a middleware software that will offer primitives for the development of specific SIEM services. In this chapter, we consider two important aspects of the middleware, namely the support for communications and storage of events.

4.1 Communication

The communication among the MIS plays a fundamental role in the MASSIF resilience architecture. This feature is responsible for delivering events from the edge services to the core SIEM correlation engine despite the threats affecting the underlying communication network. To give this kind of guarantee we will employ application-level routing strategies among the MIS nodes, in such a way that they form an *overlay network* able to deliver messages in a secure and timely way.

Overlay networks have been used as mechanisms to implement routing schemes that take into account specific application requirements [3]. In the MASSIF resilience architecture we want to employ

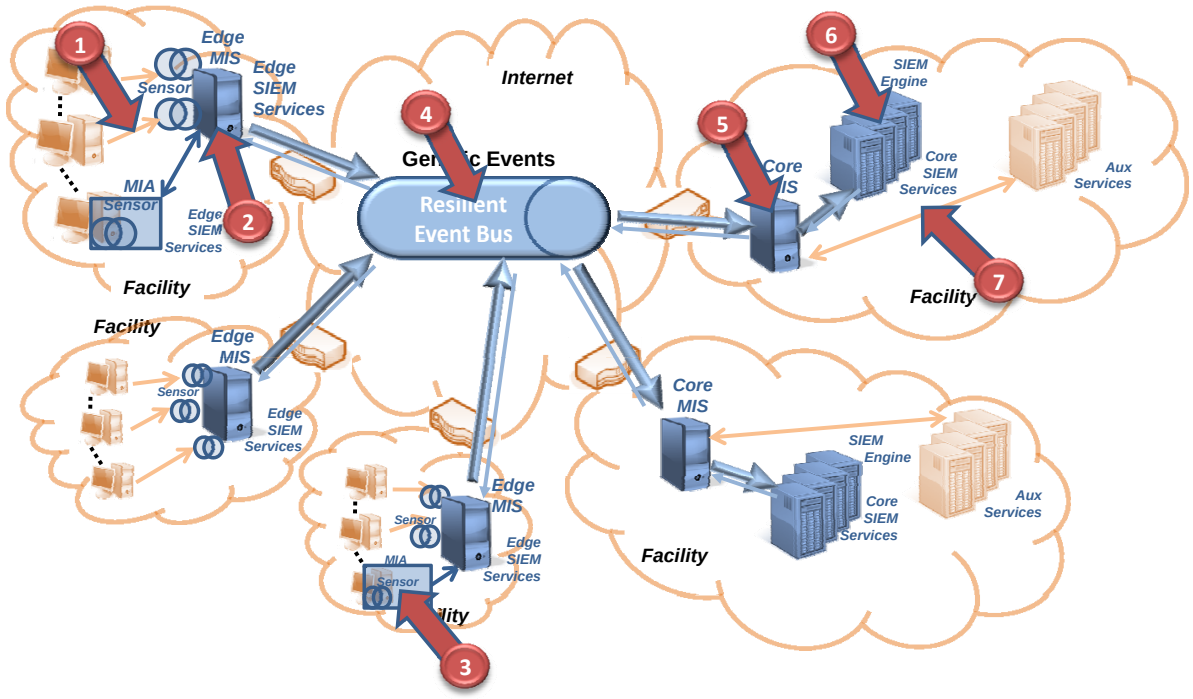


Figure 4.1: Attack vectors to the MASSIF architecture.

overlay networks to create redundant network-agnostic channels for robust event transport from the edge sensors to the core event correlation engine.

There are two main requirements on the inter-node MASSIF communication middleware. First, messages should be transmitted respecting some delivery deadline. The objective is to make the events be processed at the correlation engine while they are (temporally) valid, which requires the communication subsystem to enforce timeliness properties of the communication. Unfortunately, most overlay networks do not have this objective, and therefore, we will develop specific solutions to enforce these guarantees in the MASSIF SIEM system. In the past, some approaches had the aim of improving the end-to-end communication latency, but not of attaining application-defined maximum delays (e.g., [2, 57]). Only more recently, some of the authors proposed a timeliness-aware application-level routing solution using overlay/multihoming techniques called Calm-Paranoid (CP) [23]. Although the CP algorithm solves in part the timeliness requirement of the communication, it was designed considering a static set of nodes that only fail by crashing, therefore, it can not address the case where some nodes might be compromised by a malicious adversary (i.e., nodes that are subject to Byzantine failures).

Second, the middleware should tolerate malicious intrusions in some of the nodes, such as in data forwarding devices (e.g., routers) and eventually in a subset of the MIS. Our initial approach to solve the problem is enhance the CP algorithm with Byzantine-routing capabilities [52, 50] and network coding techniques [41]. The idea is to send each message through 1 to $2t+1$ different paths chosen based on how disjoint they are (i.e., minimizing the number of common nodes among them) and their timeliness (their expected delivery time must be smaller than the message delivery deadline), being t a bound on the number of channel faults during the message transmissions. In order to avoid the bandwidth overhead of sending the same requests more than once, one idea is to use network coding algorithms to generate message blocks to be sent using different channels. With this technique, each channel will only transmit

a fraction of the message size and a receiver will be able to recover the message as long as it receives at least a subset of the blocks and decode them.

Besides the secure and timely message delivery, another important feature of the communication system is to support trusted timestamping of events. The idea is to add a trusted timestamp to the events when they arrive at the edge MIS, in order to complement the untrusted timestamp of the event assigned on the distributed sensors. The timestamps of the MISes are trusted in the sense that MIS clocks will be synchronized and their error and drift rates bounded due to the use of internal and external robust clock synchronization algorithms.

4.2 Storage

The storage solutions to be deployed in the MASSIF architecture aims to provide one or more intrusion-tolerant historian for archival of security events ensuring properties like confidentiality and unforgeability.

The system will be deployed together with the processing engine, as described in Figure 3.1, to archive important security events used for criminal/civil prosecution of attackers after a security breach. Replication, diversity management and threshold cryptography will be employed to make sure that the stored information is unforgeable, and to guarantee that the storage system will be tolerant to faults and intrusions (see a description of these techniques in Chapter 5). Additionally, access control needs to be used to ensure that certified records of security breaches will be made available only to authorized parties, based on existing and upcoming regulations.

5 Node Resilience Solutions

This chapter presents an overview of the concepts and protocols used in the implementation of the MASSIF resilient architecture. Our intent is to describe the main techniques that are being explored to improve the resilience of specific nodes of the architecture, such as the edge and core-MIS. It is worth to notice that this is a *preliminary description*, as we expect that the solutions presented here in high level will evolve to a more mature platform in future deliverables.

In our approach to increase resilience, we will employ prevention techniques whenever possible to deal with various types of threats, such as eavesdropping and/or tampering of messages. For instance, traditional cryptographic solutions based on in symmetric encryption and Message Authentication Codes (MAC) are highly effective at averting this sort of attacks, and nowadays they provide efficiency levels that can address information flows with huge amounts of events. However, some more severe attacks are hard to solve with prevention solutions alone (e.g., an intrusion in the core-MIS machine), and therefore, it advisable to employ mechanisms to achieve tolerance [64, 65]. In short, instead of trying to prevent every single intrusion or fault, they are allowed, but tolerated: systems remain to some extent faulty and/or vulnerable, attacks on components can happen and some will be successful, but the system has the means to trigger automatic mechanisms that prevent faults or intrusions from generating a system failure. Additionally, while disconnection can be an effective solution to avoid the propagation of attacks, it may imply significant performance degradation and may have very negative and costly implications to service provision. It is thus important to seek for solutions that allow availability under attack.

The chapter is organized in the following way. Section 5.1 presents the architecture of a node, including a description of the main modules and layers of the middleware software. Sections 5.3 and 5.4 explain the basic communication support required by the nodes and some distributed protocols. Section 5.5 describes the replication management options that can be used for supporting core and edge MIS high availability. The runtime support subsystem responsible for fault recovery and periodic rejuvenation is presented in Section 5.6.

5.1 Local node architecture

The organization of the nodes follows the structuring principles for intrusion-tolerant systems [64, 65]:

- *trusted - versus untrusted - hardware* : most of the hardware of a node is considered untrusted, with small parts of it being considered trusted.
- *trusted - versus untrusted - support software* : some small parts of the software can be trusted to execute a few critical functions correctly, while the rest being subjected to malicious faults.

- *run-time environment* : offers trusted and untrusted software and operating system services in a homogeneous way.
- *trusted distributed components* : software services implemented by collections of components, interacting with a middleware that tolerates intrusions in a subset of the components.

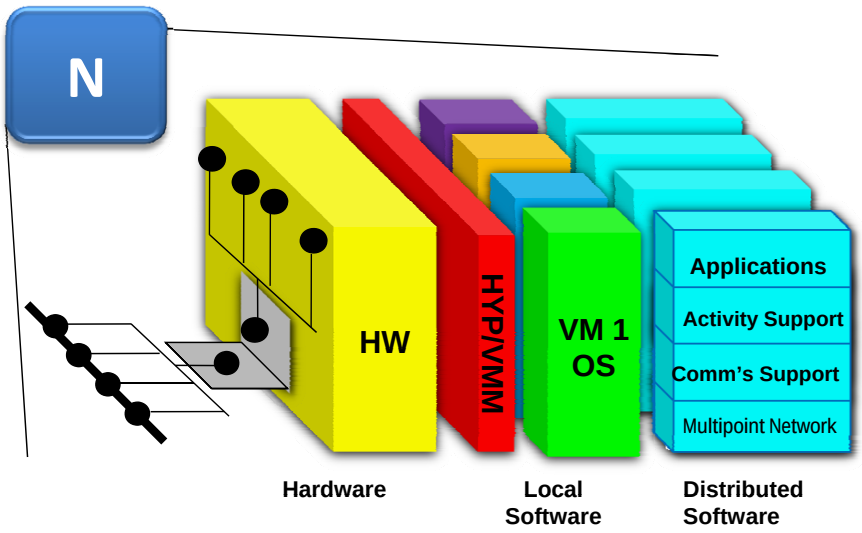


Figure 5.1: Local architecture of a MASSIF node

A snapshot of the node is depicted in three dimensions in Figure 5.1, where the above-mentioned node structuring principles can be perceived. Firstly, there is the hardware dimension, which includes the node and networking devices. We assume that most of a node's operations run on untrusted hardware, e.g., the usual machinery of a computer connected through a normal networking infrastructure. However, some nodes may have pieces of hardware that are trusted, for example, that by construction prevent intruders from having direct access to the inside of those components. For example, TPM offers capabilities for the secure generation of cryptographic keys and pseudo-random numbers, and includes for instance services to support remote attestation and sealed storage. Alternatively, an appliance board with processor, which may or not have an adapter to a control channel (an alternative trusted network), could be plugged to the node's hardware for more generic trusted services.

Secondly, services based on the trusted hardware are accessed through the local software. As depicted in the figure, the local software can take the form of a hypervisor (also called virtual machine manager (VMM)) that employs virtualization techniques to run multiple operating systems (called guests) concurrently on the same host hardware. The hypervisor presents to the guest operating systems a virtual platform and manages their execution. Typically, hypervisors are shielded from different forms of remote attacks, and enforce isolation between the guest operating systems and itself, and also among the guests. Therefore, even if a guest operating system is compromised, the remaining ones can continue to operate correctly.

Thirdly, there is the distributed software provided in a middleware with several layers, on top of which distributed applications can run, even in the presence of malicious faults (far right in Figure 5.1).

In the next section, we will discuss alternative approaches to setup a node with incremental resilience capabilities, and describe in more detail the distributed software to be used in the project.

5.2 Incremental resilience strategies

Given that, as we have assumed earlier, different Facilities/Networks may have different levels of trustworthiness, and that distinct application and systems will require different levels of trust, the architecture must allow for an incremental range of resilience solutions, in the interest of the best tradeoff with performance, cost, or complexity.

A key issue is the resilience of the MASSIF nodes (MIS) against direct attacks. We give a few examples illustrating the possible MIS construction methods, to achieve the desired incremental range of resilience:

1. *Ruggedised simplex* : single ruggedised machine, where various intrusion prevention techniques are applied to increase security (e.g., a better identification and authentication scheme; careful configuration of network services and removal of unnecessary applications);
2. *Loosely coupled duplex or N-plex* : the service is replicated in two or more machines loosely coupled in the network;
3. *Closely coupled N-plex* : the service is replicated on N machines connected with a private broadcast network;
4. *Tightly coupled N-plex* : the service is replicated in a virtualized node running N diverse guest operating systems (to prevent common vulnerabilities);
5. *Twin quad* : two replicas of virtualized nodes, each one running four virtual guest operating systems (to guarantee Byzantine fault tolerance and availability in case of a node crash).

5.3 Multipoint Network

The multipoint network is the lowest level of our distributed software, and is used to enable more advanced communication support for the SIEM under a common interface. In this layer, we assume only the existence of the TCP/IP and UDP/IP network protocols, potentially with security extensions. In particular, we consider the *Internet Protocol Security* (IPSec) [51] and *Secure Socket Layer* (SSL/TLS) [26] standard security technologies, as well as custom implementations of secure point-to-point communication solutions[1].

IP Security IPSec is an extension of the IP Protocol that provides some level of security [51]. In its basic form, IP messages can be modified and its content read by anyone with access to the network, e.g., a hacker controlling a router. IPSec prevents this problem by ensuring host-to-host end-to-end integrity, authenticity and confidentiality. It offers several services, such as access control, connectionless integrity,

[1]For example, with a public key infrastructure, one can use Secure Diffie-Hellman to establish shared keys between pairs of processes, and then employ message authentication codes and (optionally) symmetric encryption for secure communication [45].

data origin authentication, protection against replays, confidentiality (through symmetric encryption) and limited traffic flow confidentiality. Since these services are offered at the IP layer, they can be used by any higher layer protocol, such as TCP, UDP, HTTP.

IPSec is divided in two (sub)protocols, which may be applied alone or in combination with each other to provide the desired set of security properties at IP-level:

- *Authentication Header (AH)* : provides connectionless integrity, data origin authentication, and an anti-replay service.
- *Encapsulation Security Payload (ESP)* : provides payload confidentiality (using encryption) and limited traffic flow confidentiality. Optionally, it may also provide connectionless integrity, data origin authentication, and an anti-replay service.

Both AH and ESP are vehicles for access control, based on the distribution of cryptographic keys. These protocols support two different modes of operation: with Transport mode, IPSec essentially protects upper layer protocols (e.g., TCP); with Tunnel mode, the protocols are applied to tunneled IP packets, i.e., the IP datagrams themselves are sent through a secure tunnel.

IPSec allows the user or the system administrator to control the granularity at which a security service is offered, allowing, for example, the creation of a single encrypted tunnel to carry all the traffic between two security gateways or a separate encrypted tunnel for each TCP connection between a pair of hosts communicating across these gateways. Since IPSec works at the operating system level, most implementations do not have an API that can be used by applications to transmit secure data, other than the socket API used for IP, UDP or TCP. However, a system administrator can define the policy for IPSec on a host basis, determining the ways by which a host can connect securely to another.

Secure Socket Layer The SSL, and the later standardization called Transport Layer Security (TLS), is a security extension to TCP. It basically provides authentication of the hosts involved in the communication, and confidentiality and integrity of the communication. SSL/TLS is a modification of TCP: the initial handshaking is followed by a negotiation to select cryptographic algorithms and create a session key. Authentication is normaly based on public-key cryptography and digital certificates, and can be mutual (both peers authenticate themselves), one-way or simply not done. Integrity and (optionally) confidentiality of data are guaranteed using the session key, respectively by adding a MAC and encrypting the data. The security guarantees provided by SSL/TLS are similar to those provided by TCP over IPSec, except for the more powerful authentication scheme and the usual availability of a user-level API, something that is not common with IPSec. Another difference is that while IPSec provides security from OS-to-OS, SSL/TLS provides a complete security protection from process-to-process.

SSL/TLS is provided by software packets like OpenSSL, and directly in languages like Java. The basic APIs tend to be quite similar to the TCP sockets API. However, there are usually a set of calls to define the location of the certificates and other configuration options, namely to decide if confidentiality is turned on or off, and to choose which cryptographic algorithms should be used.

5.4 Communication Support

Communication support offers a set of primitives for distributed communication that can be used directly by the processes implementing a service or by the Byzantine fault-tolerance replication mechanisms. Currently, we are considering two forms communication support: reliable and atomic broadcast.

Reliable Broadcast A reliable broadcast protocol is used to ensure that if a message is sent to a group of processes, either all correct processes deliver this message or none will do that. Formally, a reliable broadcast protocol can be defined in terms of the following properties [35, 10, 19]:

- *Validity* : if a correct process broadcasts a message m, then some correct process eventually delivers m.
- *Agreement* : if a correct process delivers a message m, then all correct processes eventually deliver m.
- *Integrity* : for any identifier ID, every correct process p delivers at most one message m with identifier ID, and if $sender(M)$ is correct then M was previously broadcast by $sender(M)$.

We consider that the sender also delivers the messages it broadcasts, and the predicate $sender(M)$ gives the field of the message header that identifies the sender.

An example protocol for Byzantine fault tolerant (BFT) reliable broadcast was proposed by Bracha [8]. The protocol ensures correct operation if there are n process where at most f of them can be controlled by a malicious adversary (with $n \geq 3f+1$). The main steps of the protocol are: 1) the sender transmits an initial message $\langle \text{INITIAL}, ID, m \rangle$ to all processes; 2) when this message is received by a correct process, it sends $\langle \text{ECHO}, ID, m \rangle$ to all processes; 3) when a correct process receives $\lceil \frac{n+f}{2} \rceil$ of these messages, it broadcasts a $\langle \text{READY}, ID, m \rangle$ to all processes; finally, 4) when a correct process receives $\lceil \frac{n+f}{2} \rceil$ of these messages, it reliably delivers m. The messages broadcast in steps 2 and 3 can also be triggered in a process if it receives $f+1$ messages $\langle \text{READY}, ID, m \rangle$ before executing these steps. An interesting feature of this protocol is that it does not require any timing assumptions (i.e., asynchronous model), and therefore, it is immune to attacks in the time domain.

Total order broadcast A total order broadcast protocol is similar to a reliable broadcast protocol, but it ensures an additional property [19]:

- *Total Order* : if two correct processes deliver two messages m_1 and m_2 then both processes deliver them in the same order.

This additional property makes the implementation of a total order broadcast much harder than implementing a reliable broadcast. More precisely, total order broadcast is equivalent to the well known consensus problem, and thus cannot be solved deterministically in asynchronous systems with crash failures (and therefore, cannot also be solved in systems with Byzantine failures) [32].

Castro and Liskov presented an efficient protocol for implementing total order broadcast of client requests sent to a set of $3f+1$ replica servers, in the context of state machine replication (discussed in the next section) [11]. The protocol of these authors can seen as a BFT version of the Paxos solution proposed by Lamport [39]. The protocol begins with a sender transmitting a signed message m to all servers. One of the servers, called the leader, is responsible for ordering the arriving messages. When the leader receives m, it sends a PRE-PREPARE message to all servers assigning a sequence number i to m. A server accepts a PRE-PREPARE message if the proposal of the leader is *good* – the signature of m verifies and no other PRE-PREPARE message was accepted for sequence number i. When a server accepts a PRE-PREPARE message, it sends a PREPARE message with m and i to all servers. When a server receives $2f$ PREPARE messages from other servers with the same m and i, it marks m as prepared and sends a COMMIT message with m and i to all servers. When a server receives $2f$ COMMIT

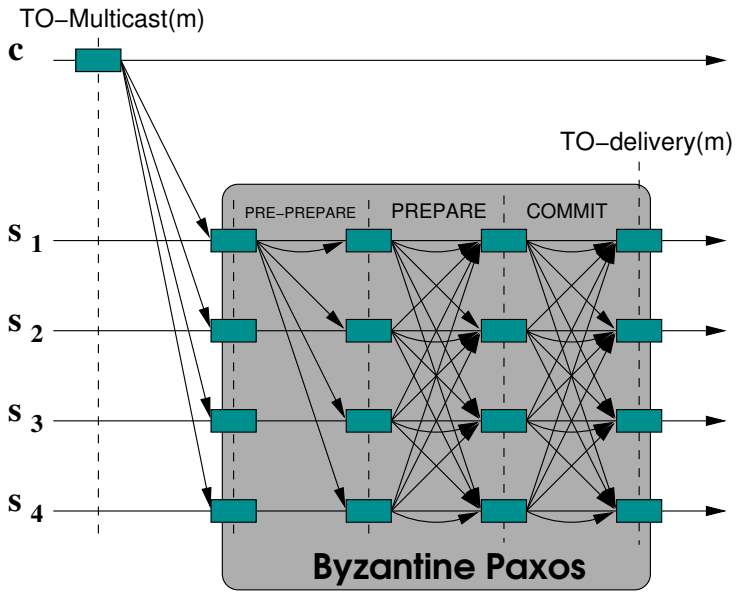


Figure 5.2: Byzantine fault tolerant total order broadcast (normal case).

messages from other servers with the same m and i, it commits m, i.e., accepts that message m is the i-th message to be delivered. Figure 5.2 illustrates the protocol execution.

While the PREPARE phase of the protocol ensures that there cannot be two prepared messages with the same sequence number i (which is sufficient to order messages when the leader is correct), the COMMIT phase ensures that a message committed with sequence number i will have this sequence number even if the leader is faulty.

When the leader is detected to be faulty, a leader election protocol is used to freeze the current round of the protocol, elect a new leader and start a new round. When a new leader is elected, it collects the protocol state from $2f+1$ servers. The protocol state comprises information about accepted, prepared and committed messages. This information is signed and allows the new leader to verify if some message was already committed with some sequence number. Then, the new leader continues to order messages.

It is worth to notice that this protocol implicitly implements the reliable broadcast described in previous section: the leader reliably broadcast the message plus sequence number in a reliable broadcast (PRE-PREPARE corresponds to INITIAL, PREPARE corresponds to ECHO and COMMIT corresponds to READY). Moreover, this protocol requires a partially synchronous system model [30] to work: the aforementioned reliable broadcast by the leader must be completed before a timer (started in each process when sender' message is received) expires on a quorum of $f+1$ correct processes. For every timer expiration, the value of the timer is doubled to ensure that eventually it will be larger than the required communication and processing delay of the protocol.

5.5 Activity Support

5.5.1 Replication management

One of the key mechanisms for implementing fault tolerant services is replication. In a replicated system there is an unbounded set of clients $C = \{c_0, c_1, ...\}$ interacting with a set of n servers $S = \{s_0, ..., s_{n-1}\}$ acting as replicas of a service. The clients and servers communicate among themselves using a protocol that implements a replication scheme. In this section we describe three basic replication schemes that potentially will be used in MASSIF, to enhance the resilience of the MIS and event archival.

Notice that the replication model usually employed for (crash only) fault tolerance, passive or primary-backup replication [9], is not adequate for our purposes. In this model there is a primary replica that executes all operations issued by the clients, and forwards the results of these operations to a set of backup replicas that can take over the primary role in case of faults. The problem in using this model with Byzantine failures is that a malicious primary may execute client' operations in a wrong way to fool both clients and backup replicas. Since the reply sent by the primary is not verified or compared with the result of the execution of the operation in other replicas (in fact, the backup replicas do not execute the request, only get the update result), there is no way to verify the correctness of the primary' computation.

Given this limitation, in this section we discuss other replication models in which clients effectively compare request' results from different replicas to extract meaningful responses.

State Machine Replication A natural way to make a service fault-tolerant is to model it as a deterministic state machine, and replicate the service implementation in a set of servers, while ensuring that all of them start with the same state and execute the same sequence of operations. This is the core idea of the State Machine Replication (SMR) model [54], also called active replication.

In this model, the servers, hereafter called replicas, receive an operation request issued by a client, process it (possibly) modifying its state and send a reply. Formally, a state machine replication is characterized by three properties:

1. *Initial state* : All correct replicas start in the same state;
2. *Determinism* : Two correct replicas on identical states that execute the same request go to the same next state and generate equal results;
3. *Coordination* : All correct replicas receive and execute the same sequence of requests.

Although property 1 is trivial to implement, property 2 severely constraints the kind of services that one can replicate using this technique. The problem is that by requiring determinism, the state machine replication model rules out the possibility of replicas independently generate timestamps, random numbers or even run multiple threads, due to the fact of the inherent non-determinism of these actions that can make correct replicas have divergent states. Some work has been devoted to replicate non-deterministic state machines (e.g., [12]), but it is still hard to support it in practice.

Property 3 requires the implementation of a total order broadcast. The idea is to make clients issue their requests through this communication primitive to ensure all replicas receive all requests in the same (total) order. One of the early protocols for state machine replication with Byzantine failures was described in the previous section [11]. The core of this protocol is used for ordering requests from clients. After a message is ordered, replicas execute it and send a reply to the client that issued the request. A

client waits for $2f+1$ equal replies from different replicas in order to extract a response. This larger quorum is needed (instead of the $f+1$ equal replies from [54]) to ensure linearizability of operations, even with read-only requests optimistically being processed without total order.

Forwarder Replication An inherent limitation of the SMR is its programming model: there is a client that invokes an operation on a replicated service and waits for replies. Unfortunately, several components of the MASSIF resilience architecture do not follow this client server model. For example, the core-MIS is fundamentally an extensible message broker that can be used to firewall, translate and aggregate events coming from several sources, and then forward them to one or more destinations. More generically, this component receives, process and forwards messages. Load balancers, routers, stream processing engines, publish/subscribe systems are other examples of critical systems that follow this pattern of communication. In order to make this type of systems resilient, we are developing a new model called *Forwarder Replication*, that will be employed in the project to implement a MIS.

A basic implementation of a replicated forwarder can be done straightforwardly with small modifications to the SMR model. The idea is to replicate a deterministic forwarder and use a total order broadcast to make all replicas process the same sequence of requests/messages. The difference from the SMR model is that the result of the message processing is delivered to the destination of the message, and not the sender.

The SMR-based implementation of the forwarder can be seen as the starting point for more efficient and scalable solutions that take into account the specificities of the forwarder being developed. As an example, one would like to avoid total order broadcasts if the service is stateless (i.e., if the messages being forwarded do not affect the state of the forwarder). Moreover, research is required to deal with the forwarder scalability and its integration with existing services. More specifically, in MASSIF we want to provide infinite scalability for the event correlation engine, in the sense that by adding more machines we increase the event processing throughput. However, to make an end-to-end scalable solution, the core-MIS must also be able to increase its forwarding throughput if we add more replicas to it, without endangering its resilience properties.

The main idea for implementing a scalable forwarder is to dynamically assign clusters of at least $2f+1$ replicas to each message in such a way that one of the replicas is responsible to forward the message and the others witness this operation. The key problem to be solved in this solution is how to efficiently make the witness verify if the forwarder is correctly forwarding its messages. We are working on this problem in the context of the core MIS, and we intend to provide a solution in future deliverables. This method has two potential benefits: First, the total order broadcast can be avoided if the witnesses are able to verify the forwarder execution without receiving all its requests (causing a significant improvement to performance); Second, adding more $2f+1$ replicas to the system increases the throughput of the forwarder by a factor of $\frac{n+2f+1}{n}$, and scalability can be achieved as long as more replicas are added.

Quorum systems *Quorum systems* is a technique for implementing dependable shared memory objects in message passing distributed systems [34]. Given a universe of data servers, a quorum system is a set of server sets, called *quorums*, that have a non-empty intersection. The intuition is that if, for instance, a shared variable is stored in all servers, any read or write operation has to be done only in a quorum of servers, not in all servers. The existence of intersections between the quorums allows the development of read and write protocols that maintain the integrity of the shared variable even if these operations are performed in different quorums.

Byzantine quorum systems are an extension of this technique for environments in which clients and servers can have Byzantine failures [42]. Formally, a Byzantine quorum system is a set of server quorums $\mathscr{Q} \subseteq 2^U$ in which each pair of quorums intersect in sufficiently many servers (*consistency*) and there is always a quorum in which all servers are correct (*availability*). The servers can be used to implement one or more shared memory objects. Among the many types of quorum systems, two of them are fundamental for implementing Byzantine replication.

In the first type, the servers form a *f-masking quorum system* that tolerates at most f faulty servers, i.e., it masks the failure of at most that number of servers [42]. This type of Byzantine quorum systems needs that the majority of the servers in the intersection between any two quorums are correct, thus $\forall Q_1, Q_2 \in \mathscr{Q}, |Q_1 \cap Q_2| \geq 2f+1$. Given this requirement, each quorum of the system must have $q = \lceil \frac{n+2f+1}{2} \rceil$ servers and the quorum system can be defined as: $\mathscr{Q} = \{Q \subseteq U : |Q| = q\}$. This implies that $|U| = n \geq 4f+1$ servers.

The second type is called *f-dissemination quorum system*, in which a value is disseminated among n servers despite the existence of f faulty servers. This type of system requires data to be *self-verifiable*, i.e., any faulty server that corrupts its replica of the shared object will be detected. This type quorum systems needs at least one correct server is in the intersection between any two quorums, thus $\forall Q_1, Q_2 \in \mathscr{Q}, |Q_1 \cap Q_2| \geq f+1$. Given this requirement, each quorum of the system must have $q = \lceil \frac{n+f+1}{2} \rceil$ servers and the quorum system can be defined as: $\mathscr{Q} = \{Q \subseteq U : |Q| = q\}$, which implies $|U| = n \geq 3f+1$ servers.

With these constraints, a f-masking quorum system (resp. f-dissemination quorum system) with $n = 4f+1$ (resp. $n = 3f+1$) will have quorums of $3f+1$ servers (resp. $2f+1$ servers).

There are several shared storage implementations using quorum-based replication tolerating Byzantine failures [42, 43, 24], and in all of them each server saves at least the values of data being stored and a timestamp. Given that, a write protocol for implementing a storage service using an f-dissemination quorum system works in the following way [43]: first a writer client reads the current timestamp from a quorum; then, it increments the higher value found by one unit; and, finaly, it writes the signed value with this new timestamp. The corresponding read protocol consists of: the client reads the value-timestamp pair from a quorum; next, it chooses the value with the higher timestamp as the read value. Optionally, the read protocol can have a write-back phase, in which the client sends the value with highest timestamp to the servers that reported older values to ensure atomicity [38]. This very simple protocol works with non-malicious writers in a completely asynchronous distributed system model.

5.5.2 Confidentiality of replicated data

In some services, it might be important to ensure the confidentiality of data stored at replicated servers even if a subset of the replicas is compromised by an adversary. In this case, a useful cryptographic technique to ensure confidentiality of replicated data is *secret sharing* [56]. This technique comprises the use of a special party called dealer that distributes a secret to n players, where each player gets only a share of the secret. The main properties of the scheme are: 1) at least $k \leq n$ different shares of the secret are needed to recover it; and 2) no information about the secret is disclosed with $k-1$ or less shares. This kind of scheme can be integrated in f-fault-tolerant replication protocols making $k \geq f+1$ and distributing a share of the data being written to each of the n replicas. This ensures that f or less individual replicas will not have access to the data being stored, but that clients that have authorization to access the data will be granted access to the shares of (at least) $n-f \geq f+1$ different replicas, and will be able to rebuild the original data.

An important benefit of using secret sharing schemes is to integrate confidentiality guarantees to the stored data, without using a key distribution mechanism to create shared secret keys between the writers and readers of stored objects.

A drawback of using secret sharing is that each generated share has at least the same size of the original data, which increases the total amount of storage by a factor of n. In order to avoid this limitation, it is possible to compose a secret sharing scheme with an *information-optimal erasure code*, reducing the size of each share by a factor of $\frac{n}{f+1}$ of the original data [53]. This composition was originally described in [37]: first, the data is encrypted using a random secret key; then, the encrypted data is encoded using the erasure code; next, the key is divided using the secret share scheme; and finally, each of the n replicas receive a block of the encrypted data and a share of the key.

In the last few years, we have applied confidentiality solutions on stored data in different scenarios [6, 7]. In MASSIF, we intend to extend these ideas and to apply them to design an intrusion-tolerant storage for the SIEM engine, while taking into considerations its particularities.

5.6 Runtime Support

The runtime support of the nodes is mainly related with the rejuvenation and recovery from intrusions, and in parallel ensure that replicas do not share common vulnerabilities by employing diversity techniques.

5.6.1 Proactive recovery

Byzantine fault-tolerant protocols for state machine or quorum replication may be used to deal with the arbitrary failures of a finite set of f out-of n replica servers [42, 11, 66, 47]. Such protocols have limited utility in long-lived systems where malicious adversaries are constantly deploying attacks and causing intrusions. In fact, fault/intrusion tolerant protocols are useful to delay the corruption of the overall replicated system by a certain interval of time that depends on the actual value of f and on how different/diverse the individual replicas are [50]. Unfortunately, while a higher f allows more compromised replicas to be tolerated, it also means that more individual servers are need to be different from each other (to avoid common vulnerabilities, which could be easily exploited over a short period o time). For instance, some of the previously mentioned protocols require at least $n = 3f + 1$ replicas to tolerate f faults. This means that when f is increased from 1 to 2, the minimum total number of replicas is increased from 4 to 7.

We argue that fault- and intrusion-tolerant protocols are useful as long as they are complemented with *recovery mechanisms* able to reduce both the value of f and the time window for an adversary to compromise more than f replicas. The way to implement this idea is to rejuvenate each replica periodically even if they are not compromised, a technique called *proactive recovery*. The rationale is that compromised servers may not exhibit any detectable behavior until the adversary controls more than f servers, but then it is too late to avoid the corruption of the overall system. Some existing works implement proactive recovery mechanisms, thus making the system tolerate an unbounded number of malicious faults during its lifetime [11, 67].

Proactive recovery mechanisms are however a natural candidate for being the first targets of an attack – a malicious adversary will start by stopping, delaying, or even by corrupting the recovery process, before deploying the actual attack that compromises the replicated system. Sousa et al. [59] demonstrates

how proactive recovery approaches may collapse precisely because these mechanisms are not adequately protected. That work analyzes systems that are designed under the asynchronous model, but end-up making a few synchrony assumptions, namely related to recovery triggering and execution time. Such assumptions may have acceptable coverage in the presence of accidental (non-intentional) faults, but represent a vulnerability in an environment where malicious attacks can occur.

We argue that recovery mechanisms must be architected in a way that their timeliness and effectiveness cannot be affected, namely by a malicious adversary. When a replica is recovering, there exists normally a non-negligible interval of time during which messages (from clients or other replicas) are lost, and thus a recovering replica behaves as being crashed during that time. The unavailability of individual replicas may provoke the unavailability of the overall replicated system. Consider for instance a BFT replicated system with 4 replicas that, therefore, is able to tolerate one faulty replica. If this system is enhanced with recovery mechanisms, then its availability will be affected if 1) more than one replica recovers at the same time, or 2) one replica is recovering while another one is compromised. In both cases, the sum of arbitrary faulty replicas and purposely crashed (recovering) replicas is greater than f, and thus the replicated system will become unavailable during those periods.

This means that recovery mechanisms cannot be added in a straightforward manner to existing BFT replicated systems. In order to avoid unavailability, recoveries have to be coordinated in an agreed schedule, and the system needs a higher total number of replicas to tolerate both Byzantine (arbitrary) faults and recovery-induced (crash) faults.

5.6.2 Reactive recovery

An inherent limitation of proactive recovery is that a malicious replica can execute any action to disturb the system's normal operation (e.g., flood the network with arbitrary packets) until its recovery time, and there is little or nothing that a correct replica (that detects this abnormal behavior) can do to stop/recover the faulty one. Our observation is that a more complete solution should allow correct replicas to force the recovery of *detected or suspected* faulty replicas. This solution is called *proactive-reactive recovery* [58], and it can improve the overall performance of a system under attack by reducing significantly the amount of time a malicious replica has to disturb the normal operation.

A proactive-reactive recovery mechanism is implemented in a hybrid system model, where on a synchronous and trusted part is implemented a service that manages both the periodic rejuvenations and the forced ones. This service needs that the payload replicas indicate when some other replica is executing incorrectly. This information is provided through two interface functions: $PR\_suspect(j)$ and $PR\_detect(j)$.

A replica i calls $PR\_suspect(j)$ to notify the service that replica j is suspected of being faulty. This means that replica i suspects replica j but it does not know for sure if it is really failed. Otherwise, if replica i knows without doubt that replica j has problems, then $PR\_detect(j)$ is called instead. Notice that the service is generic enough to deal with any kind of replica failures, e.g., crash and Byzantine. For instance, replicas may: use an unreliable crash failure detector [13, 14] (or a muteness detector [28]) and call $PR\_suspect(j)$ when a replica j is suspected of being crashed; or detect that a replica j is sending unexpected messages or messages with incorrect content [5], calling $PR\_detect(j)$ in this case.

If $f+1$ different replicas suspect and/or detect that replica j is failed, then this replica is recovered. This recovery can be done immediately, without endangering availability, in the presence of at least $f+1$ detections, given that in this case at least one correct replica detected that replica j is really faulty. Otherwise, if there are only $f+1$ suspicions, the replica may be correct and the recovery must be coordinated

with the periodic proactive recoveries in order to guarantee that a minimum number of correct replicas is always alive to ensure the system availability. The quorum of $f + 1$ in terms of suspicions or detections is needed to prevent recoveries triggered by malicious replicas – at least one correct replica must detect/suspect a replica for some recovery action to be taken.

Notice that the proactive-reactive recovery service is completely orthogonal to the failure/intrusion detection strategy used by a system. The proposed service only exports operations to be called when a replica is detected/suspected to be faulty. In this sense, any approach for fault detection [14, 28, 5], system monitoring [22] and/or intrusion detection [25, 48] can be integrated in the system.

5.6.3 Diversity management

If a recovery procedure restores a replica to a previous known-to-be-good state (e.g., by rebooting the operating system of a virtual machine from a clean media), this replica is still vulnerable to whatever flaw that caused its compromise in the first place. Therefore, it is not enough to simply restore replicas to known states, since this would allow an attacker to exploit the same vulnerabilities as before. To address this issue, the recovery procedure should itself introduce some degree of diversity to restored replicas, so that attackers will have to find other vulnerabilities in order to compromise a replica.

Recall that an intrusion-tolerant replicated system maintains a correct behavior even if there is an undetermined number of malicious users and/or if an attacker controls up to f out-of n replicas. The aim of the diversity rejuvenation service is to ensure that this last invariant continues to be valid throughout the lifetime of the system. It basically employs two mechanisms. First, replicas run diverse software to guarantee that vulnerabilities are not shared. If this is true, then the adversary would need to spend a considerable time to compromise each replica, since previously found exploits cannot be re-used to create intrusions in further servers. Second, periodically each replica is rejuvenated with a new diverse software, removing the effects of some prior intrusion, and therefore making the adversary start over.

The implementation of diverse proactive-reactive recovery works by dividing each replica in two logical components: the server software is run in a separate virtual machine and the *diversity rejuvenation module* (DRM) is executed in the hypervisor. This setup provides an acceptable level of protection for the DRM because the hypervisor is isolated from the virtual machines. Therefore, if an adversary manages to exploit a vulnerability in the operating system supporting the server execution, he or she will not be able to propagate the intrusion to the hypervisor and affect the correctness of the DRM. Additionally, replica rejuvenation can also be performed in an efficient manner by carrying out the following steps:

- DRM starts a new virtual machine with a diverse *OS configuration* stored in a local cache. This virtual machine runs in parallel with the current server replica;
- A new server is initiated in the virtual machine by running the necessary setup operations, which might include contacting the other server replicas to obtain an updated state of the service;
- The virtual machine of the current server is shutdown and discarded, and the new server takes the place of the old one;
- DRM runs a selection algorithm to find out which OS configuration should be run in the next rejuvenation;
- DRM fetches from a *configuration repository* the chosen OS configuration and stores it in the cache; This occurs in the background, while the server is processing the user requests.

An OS configuration basically contains the operating system, plus other auxiliary programs, and a replica of the service being provided by the intrusion-tolerant system. They are stored in a virtual machine disk (i.e., a file) that can be run by the virtualization solution. System administrators typically create these configurations, which should only contain fully patched software without any known vulnerabilities, and save them in a secure repository. The access to this repository is protected by employing a separate LAN or by using cryptographic mechanisms to safeguard the communications.

Intuitively, the selection algorithm should pick from the available alternatives the *best* OS configuration, in the sense that it should not have common vulnerabilities with the already running replicas. This would considerably delay the adversary to compromise more than f replicas[2]. This solution however suffers from one difficulty – given two fully patched configurations, one does not know if they share some vulnerability (which might be discovered in the future). Therefore, when designing the selection algorithm, we should attempt to fulfill the following objectives:

P1 : The new selected OS configuration does not share vulnerabilities with the configurations already executing in the other replicas.

P2 : Given the group of configurations currently running, the adversary can not predict the configurations that will be selected in the future.

P3 : All diverse OS configurations available in the configuration repository[3] for selection are picked by the algorithm with a reasonable probability.

P4 : The algorithm is run individually by each DRM of the replicas.

As explained, P1 cannot be ensured with absolute certainty. However, a recent study about OS diversity [33] found strong empirical evidence for: 1) it is possible to find OS pairs that have had no (or only a few) vulnerabilities in common in the past; and 2) if OS pairs share few vulnerabilities in the past, then with high probability no (or very few) common vulnerabilities are found in the future. This study was based on vulnerability data from the NVD database [49] over a period of 15 years, and it allowed the collection information about vulnerabilities that are present in more than one OS version. For each OS version pair one can obtain the list of shared vulnerabilities and the CVSS score of each vulnerability[4]. There are studies that cross-validate this idea, also based on data from NVD (see [1, 55] for more details). Therefore, by combining this data it is possible to calculate a rough criteria for deciding if two OS configurations share vulnerabilities: $score(OS_A, OS_B) = \sum_{v \in \mathcal{V}_{A,B}} CVSSscore_v$, where $v \in \mathcal{V}_{A,B}$ is the set of past common vulnerabilities of OS_A and OS_B, and $CVSSscore_v$ is the score of a vulnerability v.

P2 is necessary to address the following attack – to increase the available time to find vulnerabilities, the adversary predicts a system configuration that will be used some time from now (e.g., in a month); then, he or she starts to attack the corresponding OS versions, so that when this configuration is eventually installed, more than f replicas can be corrupted in a limited amount of time. Since we only have a limited number of OS configurations, our aim should be to make the prediction as hard as possible. This means

[2]Notice that we are working under the assumption that finding and exploiting new vulnerabilities in mature software takes some time.

[3]Which is a subset of all available configurations containing the operating systems that match some performance or dependability criteria.

[4]The Common Vulnerability Scoring System (CVSS) score provides an indication of the impact of a vulnerability in a system, and it takes into consideration aspects like ease of exploitation and the impact on the integrity/confidentiality/availability [44].

that selecting an OS configuration from the available ones should entail some level of randomness, even if this implies choosing a system configuration that has a somewhat higher score among some of the executing replicas.

Some OS pairs share much less vulnerabilities than others, and therefore, there is the risk that some of the available OS configurations are never selected. To address this problem, the algorithm should enforce P3. The last objective is useful because it simplifies the implementation, since this allows the DRM to determine which OS configurations are (and will be) used in replicas without having to exchange information through the network. This requires that the algorithm executes in a deterministic way (after some potential initial random setup step).

The algorithm for choosing specific replica configurations is under development. We expect to use it as a key component together with the proactive-reactive recovery service to enhance the dependability of the MIS.

6 Conclusions

This deliverable presents a preliminary architecture for a resilient SIEM framework, covering the aspects related to its operation when faced with accidental faults and malicious attacks. The document address the following main areas:

- *System Model* : provides a detailed discussion on two fundamental models - the fault and synchrony. These models have a strong impact on the organization of the architecture and on the design of the protocols.
- *Architecture* : describes the building blocks and services of the MASSIF resilient architecture. It includes the main components that will enforce the security, namely the Edge and Core MIS and MIA, and the communication services.
- *Middleware and Node Resilient Solutions* : explains mechanisms and techniques that can be implemented in a middleware, and that can be applied to construct increasingly more resilient components.

In the next deliverable of WP5.1, D5.1.2, the specification of the resilient architecture will be further refined with a detailed description of services and protocols that will be used to implement the various components.

Bibliography

[1] O. Alhazmi and Y. Malayia. Application of vulnerability discovery models to major operating systems. *IEEE Transactions on Reliability*, 57(1), March 2008.

[2] Y. Amir, C. Danilov, S. Goose, D. Hedqvist, and A. Terzis. An overlay architecture for high quality VoIP streams. *IEEE Transactions on Multimedia*, 8(6):1250–1262, December 2006.

[3] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01)*, pages 131–145, October 2001.

[4] M. Backes and C. Cachin. Reliable broadcast in a computational hybrid model with Byzantine faults, crashes, and recoveries. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 37–46, June 2003.

[5] R. Baldoni, J.-M. Hélary, M. Raynal, and L. Tangui. Consensus in Byzantine asynchronous systems. *Journal of Discrete Algorithms*, 1(2):185–210, April 2003.

[6] A. Bessani, E. Alchieri, M. Correia, and J. Fraga. DepSpace: a Byzantine fault-tolerant coordination service. In *Proc. of the ACM/EuroSys Conference on Computer Systems*, pages 163–176, April 2008.

[7] A. Bessani, B. Quaresma, M. Correia, F. André, and P. Sousa. DepSky: Dependable and secure storage in a cloud-of-clouds. In *Proc. of the 6th ACM/EuroSys Conference on Computer Systems*, pages 31–45, April 2011.

[8] G. Bracha. An asynchronous $\lfloor (n-1)/3 \rfloor$-resilient consensus protocol. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 154–162, August 1984.

[9] N. Budhiraja, K. Marzullo, F. B. Schneider, and S. Toueg. *Distributed Systems*, chapter The Primary-Backup Approach, pages 199–216. ACM Press, 1993.

[10] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols. In *Advances in Cryptology: CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.

[11] M. Castro and B. Liskov. Practical Byzantine fault-tolerance and proactive recovery. *ACM Transactions Computer Systems*, 20(4):398–461, November 2002.

[12] M. Castro, R. Rodrigues, and B. Liskov. BASE: Using abstraction to improve fault tolerance. *ACM Transactions Computer Systems*, 21(3):236–269, August 2003.

[13] T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 325–340, August 1991.

[14] T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.

[15] Cisco. Cisco security advisories and notices. http://www.cisco.com/en/US/products/products_security_advisories_listing.html.

[16] MASSIF Consortium. Scenario requirements. Deliverable D2.1.1, Project MASSIF EC FP7-257475, April 2011.

[17] STREAM Consortium. State of the art of data streaming. Deliverable D2.1, Project STREAM EC FP7-216181, May 2008.

[18] M. Correia, L. C. Lung, N. F. Neves, and P. Veríssimo. Efficient Byzantine-resilient reliable multicast on a hybrid failure model. In *Proceedings of the 21st Symposium on Reliable Distributed Systems*, October 2002.

[19] M. Correia, N. Neves, and P. Veríssimo. From consensus to atomic broadcast: Time-free Byzantine-resistant protocols without signatures. *The Computer Journal*, 49(1):82–96, January 2006.

[20] F. Cristian. Synchronous and Asynchronous Group Communication. *Communications of the ACM*, 39(4):88–97, April 1996.

[21] F. Cristian and C. Fetzer. The timed asynchronous distributed system model. *IEEE Transactions on Parallel and Distributed Systems*, 10:642–657, June 1999.

[22] A. Daidone, F. Di Giandomenico, A. Bondavalli, and S. Chiaradonna. Hidden Markov models as a support for diagnosis: Formalization of the problem and synthesis of the solution. In *Proc. of the 25th IEEE Symp. on Reliable Distributed Systems*, pages 245–256, October 2006.

[23] W. Dantas, A. Bessani, and M. Correia. Not quickly, just in time: Improving the timeliness and reliability of control traffic in utility networks. In *Proc. of the Workshop on Hot Topics in System Dependability*, June 2009.

[24] W. Dantas, A. Bessani, J. Fraga, and M. Correia. Evaluating Byzantine quorum systems. In *Proceedings of the IEEE Symposium on Reliable Distributed Systems*, October 2007.

[25] D. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, 13(2):222–232, 1987.

[26] T. Dierks and C. Allen. The TLS Protocol Version 1.0 (RFC 2246). IETF Request For Comments, January 1999.

[27] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34:77–97, January 1987.

[28] A. Doudou, B. Garbinato, R. Guerraoui, and A. Schiper. Muteness failure detectors: Specification and implementation. In *Proceedings of the 3rd European Dependable Computing Conference*, September 1999.

[29] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, April 1988.

[30] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–322, 1988.

[31] M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[32] M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[33] M. Garcia, A. Bessani, and N. Neves. OS diversity for intrusion tolerance: Myth or reality? In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks*, Jun 2011.

[34] D. Gifford. Weighted voting for replicated data. In *Proc. of the ACM Symposium on Operating Systems Principles*, pages 150–162, December 1979.

[35] V. Hadzilacos and S. Toueg. A modular approach to the specification and implementation of fault-tolerant broadcasts. Technical Report TR 94-1425, Department of Computer Science, Cornell University, New York - USA, May 1994.

[36] S. Kamara, S. Fahmy, E. Schultz, F. Kerschbaum, and M. Frantzen. Analysis of vulnerabilities in internet firewalls. *Computers and Security*, 22(3):214 – 232, 2003.

[37] HugoH. Krawczyk. Secret sharing made short. In *Proc. of the International Cryptology Conference*, pages 136–146, August 1993.

[38] L. Lamport. On interprocess communication (part II). *Distributed Computing*, 1(1):203–213, January 1986.

[39] L. Lamport. The part-time parliament. *ACM Transactions Computer Systems*, 16(2):133–169, May 1998.

[40] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programing Languages and Systems*, 4(3):382–401, July 1982.

[41] D. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[42] D. Malkhi and M. Reiter. Byzantine quorum systems. *Distributed Computing*, 11(4):203–213, 1998.

[43] D. Malkhi and M. Reiter. Secure and scalable replication in Phalanx. In *Proc. of the IEEE Symposium on Reliable Distributed Systems*, pages 51–60, October 1998.

[44] P. Mell, K. Scarfone, and S. Romanosky. Common vulnerability scoring system. *IEEE Security & Privacy*, 4(6), Nov–Dec 2006.

[45] A. Menezes, P. Van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[46] C. Middela, A. Dommeti, and K. Deekonda. Vulnerability analysis and management of an internet firewall. Technical report, George Mason University, Fairfax, USA, 2007.

[47] H. Moniz, N. Neves, M. Correia, and P. Veríssimo. Randomized intrusion-tolerant asynchronous services. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks*, Jun 2006.

[48] B. Mukherjee, L. Heberlein, and K. Levitt. Network intrusion detection. *IEEE Network*, 8(3):26–41, 1994.

[49] National Vulnerability Database. http://nvd.nist.gov/.

[50] R. Obelheiro and J. Fraga. A lightweight intrusion-tolerant overlay network. In *Proceedings of the 9th IEEE International Symposium on Object and Component-oriented Real-time Distributed Computing*, 2006.

[51] R. Oppliger. Security at the Internet layer. *IEEE Computer*, 31(9):43–47, September 1998.

[52] R. J. Perlman. *Network Layer Protocols with Byzantine Robustness*. Phd thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1988.

[53] M. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36(2):335–348, February 1989.

[54] F. Schneider. Implementing fault-tolerant service using the state machine aproach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, December 1990.

[55] GuidoG. Schryen. Security of open source and closed source software: An empirical comparison of published vulnerabilities. In *Proceedings of the Americas Conference on Information System*, August 2009.

[56] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979.

[57] A. Snoeren, K. Conley, and D. Gifford. Mesh-based content routing using XML. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 160–173, October 2001.

[58] 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*, 21(4), 2010.

[59] P. Sousa, N. F. Neves, and P. Verissimo. Hidden problems of asynchronous proactive recovery. In *Proceedings of the Workshop on Hot Topics in System Dependability*, Jun 2007.

[60] P. Verissimo. Travelling through wormholes: a new look at distributed systems models. *SIGACT News*, 37:66–81, March 2006.

[61] P. Verissimo and C. Almeida. Quasi-synchronism: a step away from the traditional fault-tolerant real-time system models. *Bulletin of the TCOS*, 7(4):35–39, 1995.

[62] P. Verissimo and A. Casimiro. The Timely Computing Base model and architecture. *Transaction on Computers - Special Issue on Asynchronous Real-Time Systems*, 51(8):916–930, August 2002.

[63] P. Verissimo, N. Neves, and M. Correia. The middleware architecture of MAFTIA: A blueprint. In *Proceedings of the IEEE Third Survivability Workshop*, pages 157–161, October 2000.

[64] P. Verissimo, N. Neves, and M. Correia. Intrusion tolerant architectures: Concepts and design. In R. de Lemos, C. Gacek, and A. Romanovsky, editors, *Architecting Dependable Systems*, pages 3–36. Springer-Verlag LNCS 2677, 2003.

[65] P. Verissimo, N. Neves, M. Correia, and P. Sousa. Intrusion-resilient middleware design and validation. In H. Raghav Rao and S. Upadhyaya, editors, *Information Assurance, Security and Privacy Services (Handbooks in Information Systems : volume 4)*, pages 615–678. Emerald Group Publishing, 2009.

[66] G. Veronese, M. Correia, A. N. Bessani, and L. Lung. Spin one's wheels? Byzantine fault tolerance with a spinning primary. In *Proceedings of the IEEE Symposium on Reliable Distributed Systems*, September 2009.

[67] L. Zhou, F. Schneider, and R. Van Rennesse. COCA: A secure distributed online certification authority. *ACM Transactions Computer Systems*, 20(4):329–368, November 2002.

A Security Evaluation of OSSIM

SIEM systems offer various capabilities for the collection and analysis of security information in networked infrastructures. Currently, they are being employed by organizations around the world as a way to facilitate operations related to maintenance, monitoring and analysis of networks, by integrating a large range of security and network tools, which allow for instance the correlation of thousands of events and the reporting of attacks and intrusions in near real-time.

This appendix presents the results of an experimental evaluation that was made with a current SIEM system, when faced with adversaries with different capabilities, ranging from access to the local network to physical control of the event processing engine. The objective of the evaluation was to identify concrete potential security problems that can exist in streamline SIEM tools, which might be used to enrich the fault and attack model assumptions made within the context of the MASSIF project.

The study was performed by creating a testbed where the OSSIM SIEM system was installed with the default configuration. Then, the available documentation of OSSIM was analyzed to derive potential attack vectors that could compromise the security of the system. Finally, specific attacks were implemented and tried in the testbed.

In the following pages, we describe several attack vectors that were successfully executed, and therefore that were able to expose a potential vulnerability in OSSIM. Each *MASSIF vulnerability report* is named with an individual indentifier with the following format: $< Letter1 > . < Letter2 >< letter3 > . < number1 > . < number2 >$; $< Letter1 >$ takes the value *S* to indicate that it is about a SIEM itself; $< Letter2 >< letter3 >$ indicate the type, which is *Vu* for vulnerability; $< number1 >$ is a unique number of the vulnerability within a type; and $< number2 >$ is a unique number of the same vulnerability version.

Vulnerability: S.Vu.1.0

Code Name: login-sniffing
Access Vector: local machine or/and network
Classification: high
Access Complexity: low
Type: impersonation attack
Authentication: none
Description: password can be stolen by means of a http packet interception

Target

Products: AlienVault OSSIM v2.3
Components: web service
Configuration: default
Comments: authentication procedures through the provided php-forms are susceptible to tapping

Detailed Description

The OSSIM system, or part of it, can be configured, managed and monitored by an operator through a web service. The operator is authenticated by means of a web login php-form. Though the typed password is hidden on the screen, no encryption is performed before sending it to the server via the http protocol. The password is therefore sent in plaintext, up to a simple *base64 encoding*. The interception of the http packet carrying it (e.g. through a network monitoring tool like Wireshark) enables the attacker to recover it, so to impersonate the operator.

Impact

Type: unauthorized unrestricted access
Integrity: almost complete
Confidentiality: almost complete
Availability: complete
Comments: the attacker has complete control over the service provided through the web interface, which usually runs on the server host. Therefore, if the stolen credentials have high privileges (e.g., the administrator account is compromised) she can: shut down some components, have access to the database and modify events, rules, directives, policies and actions.

Solutions

1) confine the web access to a trusted domain (network, machines)
2) utilize cryptographic protocols (e.g., SSL) to build logical private channels for web authentication

Remarks

The attack is feasible as long as it is possible to tap the network or the local machine where the login is performed.

References

URL: http://www.ossim.net/wiki/doku.php?id=user_manual:introduction
Other: http://*ossim_webserver*/ossim/session/login.php

Vulnerability: S.Vu.2.0

Code Name: db-info-injection
Classification: high
Type: information injection through impersonation
Access Vector: adjacent network
Access Complexity: low
Authentication: none
Description: unauthorized commands can be executed and unauthentic events can be stored in the database by means of an impersonation attack through TCP packet spoofing

Target

Products: AlienVault OSSIM v2.3
Components: MySQL database
Configuration: default
Comments: arbitrary data can be (legally) injected into the database by leveraging the unencrypted communication

Detailed Description

In the OSSIM system, each server that correlates events is usually provided with a MySQL database to store the information. Such database is accessed directly by the OSSIM server and by tools like Snort. At each connection established between the components, the client is authenticated with a unique username and encrypted password. Once the authentication succeeds, the communication is performed in plaintext. This is immediately visible by performing a *tcpdump* of the transmitted packets: the payload associated with each one contains the command/response data just as it was issued. Therefore, this suggests the following attack: assume that the adversary has access to the same domain in which these communications frequently take place; she can eavesdrop on the channel and attempt to impersonate the client by spoofing packets, thus being able to execute all the commands allowed by the client's privileges. The impersonation attack is not difficult since the previous packet eavesdropping provides the adversary with all the required information to forge and inject legal packets: Ethernet, IP and TCP headers indeed carry the MAC and IP address and port number that represent the client; also, the sequence and the ACK number are available to proceed correctly with the TCP session. On the other side, the server cannot distinguish between the actual client and the impostor just by looking at the contents of the packet. Also, if the packet carries the correct sequence and ACK numbers, it cannot reject it (because the packet could have been sent by the client itself). Therefore, the packet and its content are accepted and acknowledged, and the request (whatever it is) is executed.

Impact

Type: unauthorized command execution
Integrity: complete
Confidentiality: complete
Availability: complete
Comments: the attacker can have either persistent or repeated access to the database with the client's privileges. Therefore, she can execute all the commands that are legal for the client's authentication credentials. This usually translates in the ability to store/modify/delete security events (possibly just the ones that may raise an alert) in order to hide ongoing more dangerous attacks. In the worst-case, if the client is running under *root* privileges, the attacker would be able to create new user accounts, or to drop entire databases.

Solutions

1) confine the database access to a trusted domain to avoid communication eavesdropping
2) secure the communication between the client and server to enforce message integrity, authenticity and confidentiality

Remarks

References

URL: http://www.alienvault.com/wiki/doku.php?id=documentation:serverd

Vulnerability: S.Vu.3.0

Code Name: my-db-is-there
Classification: high
Type: password disclosure
Access Vector: adjacent network
Access Complexity: medium
Authentication: none
Description: the password to access the database is disclosed

Target

Products: AlienVault OSSIM v2.3
Components: database password storage
Configuration: default
Comments: the communication channel between the *non-local* database and the other components suffers from confidentiality issues

Detailed Description

In the default configuration of the mysql server, after the client authenticates, the communication is performed in the clear. While the database is local (on the same machine), it is difficult to eavesdrop on the channel since the loopback interface is usually used to improve communication speed, thereby avoiding forcing it through the network interface. However, if the loopback interface is not used, particularly when the database is not local, the communication actually takes place on the wire and information can be captured. The vulnerability thus arises from the content of the database table *ossim.conf*. Such table indeed contains the database access passwords of tools like *snort* and *nagios* (hopefully, but not surely, different from the root account) stored in plaintext. Since the retrieval of the table turns out to be somewhat frequent, sensitive information is therefore rapidly disclosed.

Impact

Type: jeopardy of correlation process
Integrity: almost complete
Confidentiality: almost complete
Availability: almost complete
Comments: the threat level depends on the privileges of the compromised account. Since many tools have at least the privilege to insert new events into the database, then spurious information can be injected with the intent to cause: event hiding, alarm flooding, resource exhaustion. It must be noted that if all the authentication credentials of the tools that reside on a specific component/machine get compromised (i.e. all of the information inserted in the database can no longer be considered trustworthy) then an impersonation attack can take place. In this case, the original component/machine would be disconnected and replaced with another one thoroughly under control of the attacker. Such component would be undistinguishable by/from the other trusted ones.

Solutions

1) enforce the usage of a local database and implement authentication mechanisms and confidential channels among the components
2) dedicate the database to a specific server; encrypt all of the sensitive information with its key; enforce database information requests to pass through the server (which in turn retrieves, decrypt, and relay the responses); implement authenticated and confidential channels between the server and the other components

Remarks

though it is usual that each server has a local database, this may not be the case for a sensor component. Indeed, it turns out that even the sensor component frequently and remotely requests the table and receives it in plaintext

References

URL: http://www.alienvault.com/wiki/doku.php?id=documentation:serverd
URL: http://www.alienvault.com/wiki/doku.php?id=documentation:agent
URL: http://forge.mysql.com/wiki/MySQL_Internals_ClientServer_Protocol\#Client_Authentication_Packet

Vulnerability: S.Vu.4.0

Code Name: events-from-some(no)where
Classification: high
Type: data forgery
Access Vector: adjacent network
Access Complexity: low
Authentication: none
Description: the database can be populated with spurious events arbitrarily forged

Target

Products: AlienVault OSSIM v2.3
Components: server collector
Configuration: default
Comments: the collector does not perform any check on the event source

Detailed Description

The server collects normalized events from other components, but no checks are performed. The normalized event is sent in the clear, using the TCP protocol, with no integrity protection. It is just a text string. Also, no authentication is carried out, and the event's information is not checked against the data contained inside the packet headers. In particular, no information from these headers is stored in the database. Therefore, as long as the collector is reachable, a TCP connection can be established and the collector can be fed with any arbitrarily forged normalized event.

Impact

Type: security information mismanagement
Integrity: none
Confidentiality: none
Availability: complete
Comments: first, the database in which the events are stored can be flooded with spurious information, with the intent of causing resource exhaustion. Second, if no attack is in progress and some machines periodically send heartbeat events, just to inform that the monitoring system is alive, then these may be expressly disconnected and their events spoofed. In this way, part of the network would unconsciously go unmonitored. Third, since correct events (possibly related to an ongoing attack) are still being delivered and may generate specific alarms, the attacker's strategy could be to raise deliberately other false alarms, by sending events which fully match different directives, thus sidetracking the threat detection process.

Solutions

1) isolate the infrastructure from external interference, possibly using component authentication mechanisms

Remarks

Once this problem is solved, ideally all events are generated in cascade starting from the monitored machines. However, since most of the events are generated from their *syslog*, which send messages using the UDP protocol and without encrypting them, the vulnerability could remain somehow open. Authentication and integrity of these messages, which usually are the ones that transit from the monitored to the monitoring system, is thus necessary.

References

URL: http://www.alienvault.com/wiki/doku.php?id=documentation:serverd
URL: http://www.alienvault.com/wiki/doku.php?id=documentation:agent

Vulnerability: S.Vu.5.0

Code Name: I-forward-it
Classification: medium
Type: man-in-the-middle
Access Vector: adjacent network
Access Complexity: medium
Authentication: none
Description: a man-in-the-middle attack can be carried out when forwarding security information events

Target

Products: AlienVault OSSIM v2.3
Components: ARP protocol
Configuration: default
Comments: ARP messages are not authenticated and properly monitored

Detailed Description

An ARP poisoning attack can be successfully accomplished to detour the normal path of security event routing. The attack is performed by repeatedly spoofing ARP replies, as ARP information are frequently refreshed. The victims are the ones responsible for forwarding security information, namely: the monitored machines that create syslog events sent through the UDP protocol, and sensors and servers that create normalized events sent through the TCP protocol. The attacker's machine(s) present inside the same network can receive and tamper with that information, and then (if necessary) at forward it to the right destination.

Impact

Type: stealthy attack and unreliable event correlation
Integrity: complete
Confidentiality: almost complete
Availability: complete
Comments: syslog and normalized event communication (respectively using the UDP and the TCP protocol) are by default performed in clear, with no authentication and any other form of protection. Hence, security information events are susceptible to additions, deletions and modifications, disrupting the event correlation and so the threat detection process. An attacker would therefore be able to attack the monitored network (or part of it) and to hide this threat.

Solutions

1) implement authenticated channels among the components to prevent outsider's packet injection
2) utilize message integrity protection mechanisms to prevent man-in-the-middle and packet spoofing attacks
3) use counters inside events, to prevent message deletion and to check for security information gaps
4) if confidentiality is a must, then use encryption techniques
5) replicate network monitoring tools to increase the cost of the attack

Remarks

The attack is feasible with whatever UDP/TCP-based communication. Though OSSIM is provided with tools to monitor ARP messages, these may not be always effective. First, by default they generate events that are not considered reliable enough to raise alarms. Since the security event flow in the system is considerable, such events are likely to go unnoticed. Second, if some part of the network is monitored by just one sensor (which has these tools enabled) and the attack successfully changes the sensor's ARP table before it is detected by the ARP monitoring tool, the event itself generated by the tool is compromised. The attack detection and reaction mechanism is therefore favorable to the attacker.

References

URL: http://www.alienvault.com/wiki/doku.php?id=documentation:agent

Vulnerability: S.Vu.6.0

Code Name: where-is-the-event
Access Vector: local
Classification: low
Access Complexity: high
Type: timing attack
Authentication: single
Description: alarms and events can be concealed from the operator's sight by changing the system time

Target

Products: AlienVault OSSIM v2.3
Components: Collector System
Configuration: default, NTP disabled
Comments: the date field of the events/alarms is tied to the time of the system in which it was generated

Detailed Description

In OSSIM, the way an operator can look at the most recent events/alarms is to order them by date through the web interface. The date of each event depends strictly on the time of the system in which it was generated: an event generated by a sensor carries its system-time; if after the event is correlated (suppose at another machine) it triggers the creation of a directive event, this last one will carry the system-time of the machine that generated it. Since there is a considerable number of events flowing through the monitoring infrastructure that are stored in databases, the modification of the time of the system that generates (perhaps critical) events may prevent the operator from understanding what are the most recent ones, by dispersing them into the myriad of events already collected.

Impact

Type: delay in countermeasure actuation
Integrity: none
Confidentiality: none
Availability: none
Comments: since a huge number of events may to flow in the infrastructure, it can be difficult for an operator to find out the last received events in order to understand what monitored component generated them. Additionally, one should notice that OSSIM focus fundamentally on security monitoring, while the actuation procedure –to reestablish the normal trustworthy system execution– is left to be reactively performed by an operator. Therefore, the recovery procedures may be delayed if the display of the events is not carefully managed.

Solutions

Since the infrastructure is hierarchical, a tamper-proof timestamp (possibly supplied by a TPM) could be associated to each event by the generating machine or in the forwarding nodes. In this way, the association of the event with the physical time can be performed with a more trustworthy approach, which can provide several advantages: First, it can represent better the causal order of the raised events. Second, it enables the event collector to detect (possibly malicious) gaps in the serial flow of the events. Third, if the event collection is performed through multiple paths, the presence of gaps may trigger mechanisms to examine the network conditions through event arrivals.

Remarks

Since the events are not dropped, the attack is only related to how these are ordered. Despite the fact that the operator might not be immediately able to locale them, he can still observe that the alarm/event counter keeps increasing, thus being sure that events are nevertheless collected.

References

URL: http://www.alienvault.com/wiki/doku.php?id=documentation:serverd
Other: http://www.alienvault.com/wiki/doku.php?id=documentation:agent

Vulnerability: S.Vu.7.0

Code Name: alarm-prevention
Classification: medium
Type: timing attack
Access Vector: local
Access Complexity: high
Authentication: single
Description: alarms can be prevented from being raised by changing the system time

Target

Products: AlienVault OSSIM v2.3
Components: correlation mechanism
Configuration: default, NTP disabled
Comments: correlation is tied to the time of the system in which correlation is performed

Detailed Description

In OSSIM, directive correlation is used to give a meaning to a set of events that get aggregated. Directives may be composed of several rules and many levels. The first level of a directive is particular in that it waits for a single occurrence of a specific event and does not contain any timeout. However, starting from the second level is usual to find one. The timeout is used because it is assumed that *"a brute force attack will generate a lot of events in a short period of time and not in the next two years"*. Hence, when it expires, *"the directive process defined in that rule is discarded"*. This means that subsequent events will not be aggregated to that specific directive, but (if possible) will have to restart a new directive correlation process. Therefore, by modifying the correlation engine system time, an attacker would able to disrupt the event-rule matching process.

Impact

Type: alarm prevention
Integrity: none
Confidentiality: none
Availability: none
Comments: directives aim at recognizing attacks through successive events correlation. Each time an event matches a rule, it increases the reliability value, which measures both the truthfulness and the dangerousness of the attack. The reliability value is a parameter used to compute the risk following a successful event correlation. If the risk is greater than 1, an alarm is raised. Since it is a good policy not to use high reliability values at low correlation levels, in order not to have lots of false alarms raised, the risk is not greater than 1 until enough events are collected. The modification of the system time thus can prevent future events to make the correlation process progress in the directive. The situation may be even worse if the attacker were aware of the timeouts used inside the directives and the number of events that his attacks generate. The ultimate result of the attack is that: the correlation process is disrupted and no alarm is generated from directive events.

Solutions

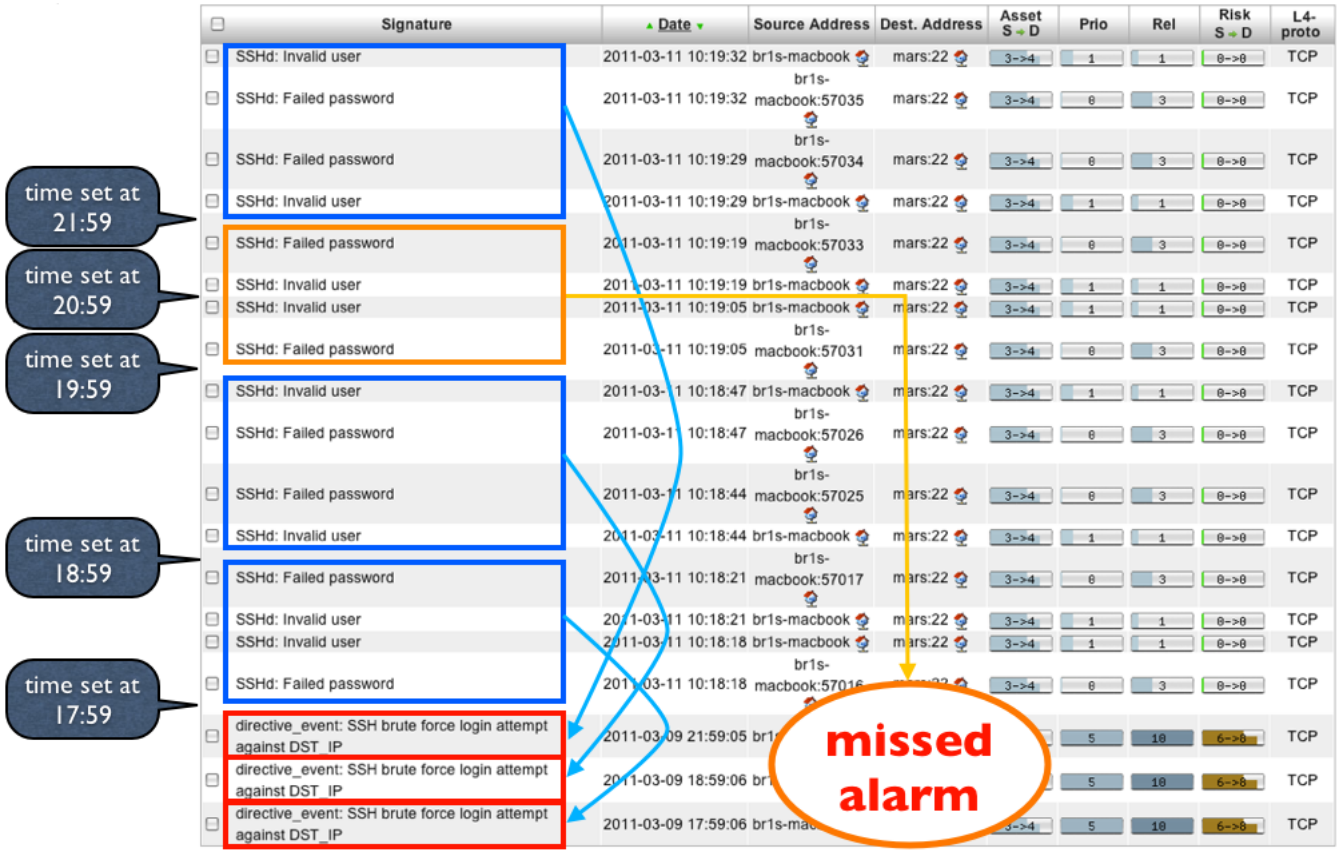
A solution that does not use timeouts in the rules would require the rethinking of the correlation system. Therefore, a more straightforward solution is to secure the system time update. It is true that changing the time already requires high privileges, but this change may also be induced by tampering with the NTP protocol (when enabled). Consequently, since the OSSIM machines can be replicated for security reasons, a fault-tolerant time synchronization protocol could be executed locally (among the local replicas) to ensure time consistency.

Remarks

Even though the events are still available in the database, if no alarm is raised then it is unlikely that the operator will notice the incongruence just by looking at them.

References

URL: http://www.alienvault.com/wiki/doku.php?id=user_manual:intelligence:correlation_directives:directives\#intelligence_-_correlation_directives_-_directives



Signature	Date	Source Address	Dest. Address	Asset S→D	Prio	Rel	Risk S→D	L4-proto
SSHd: Invalid user	2011-03-11 10:19:32	br1s-macbook	mars:22	3->4	1	1	0->0	TCP
SSHd: Failed password	2011-03-11 10:19:32	br1s-macbook:57035	mars:22	3->4	0	3	0->0	TCP
SSHd: Failed password	2011-03-11 10:19:29	br1s-macbook:57034	mars:22	3->4	0	3	0->0	TCP
SSHd: Invalid user	2011-03-11 10:19:29	br1s-macbook	mars:22	3->4	1	1	0->0	TCP
SSHd: Failed password	2011-03-11 10:19:19	br1s-macbook:57033	mars:22	3->4	0	3	0->0	TCP
SSHd: Invalid user	2011-03-11 10:19:19	br1s-macbook	mars:22	3->4	1	1	0->0	TCP
SSHd: Invalid user	2011-03-11 10:19:05	br1s-macbook	mars:22	3->4	1	1	0->0	TCP
SSHd: Failed password	2011-03-11 10:19:05	br1s-macbook:57031	mars:22	3->4	0	3	0->0	TCP
SSHd: Invalid user	2011-03-11 10:18:47	br1s-macbook	mars:22	3->4	1	1	0->0	TCP
SSHd: Failed password	2011-03-11 10:18:47	br1s-macbook:57026	mars:22	3->4	0	3	0->0	TCP
SSHd: Failed password	2011-03-11 10:18:44	br1s-macbook:57025	mars:22	3->4	0	3	0->0	TCP
SSHd: Invalid user	2011-03-11 10:18:44	br1s-macbook	mars:22	3->4	1	1	0->0	TCP
SSHd: Failed password	2011-03-11 10:18:21	br1s-macbook:57017	mars:22	3->4	0	3	0->0	TCP
SSHd: Invalid user	2011-03-11 10:18:21	br1s-macbook	mars:22	3->4	1	1	0->0	TCP
SSHd: Invalid user	2011-03-11 10:18:18	br1s-macbook	mars:22	3->4	1	1	0->0	TCP
SSHd: Failed password	2011-03-11 10:18:18	br1s-macbook:57016	mars:22	3->4	0	3	0->0	TCP
directive_event: SSH brute force login attempt against DST_IP	2011-03-09 21:59:05	br1s-mac		3->4	5	10	6->6	TCP
directive_event: SSH brute force login attempt against DST_IP	2011-03-09 18:59:06	br1s-mac		3->4	5	10	6->6	TCP
directive_event: SSH brute force login attempt against DST_IP	2011-03-09 17:59:06	br1s-mac		3->4	5	10	6->6	TCP

Vulnerability: S.Vu.8.0

Code Name: ntp-poisoning
Classification: high
Type: NTP information corruption
Access Vector: adjacent network
Access Complexity: low
Authentication: none
Description: arbitrary system time change can be induced by means of corruption of the external messages received by the time-synchronization server

Target

Products: AlienVault OSSIM v2.3
Components: Network Time Protocol
Configuration: default
Comments: the system's NTP server/client can be fed with arbitrarily forged legal data due to lack of authentication

Detailed Description

In the OSSIM system, each component that normalizes and correlates events uses the (local) system time/date whenever it has to create a new event. Event timestamping can be subjected to unexpected malicious alterations, by tampering with the NTP protocol. In order to synchronize the system time, a local NTP server (that periodically gets synchronized with some external reference time source) can run in the background and act as a synchronizer for the machines in the same domain. Equivalently, it is possible to run the NTP client from time to time, specifying the a synchronization server. In these cases, the synchronization is usually performed in more than one communication round, using the UDP protocol and no authentication. Therefore, the access to the collision domain of the target machine enables the attacker to eavesdrop the synchronization requests and to reply properly by spoofing NTP packets, thus succeeding in the alteration of the system time.

Impact

Type: unsecure information event management
Integrity: none
Confidentiality: none
Availability: none
Comments: the attack compromises the reliable management of security information. First, since events are usually displayed by date, the change of the system time may not make new events be listed on the top, perhaps preventing a prompt security threat detection and consequent actuation. Second, the attack can prevent alarms from being raised. Typically, alarms are raised from directive events, which are created following correlation directives. Such directives are composed by rules and conditions that are progressively matched with incoming events, so to generate directive events to provide a more and more accurate security threat level. This matching process however is not asynchronous, as it makes extensive use of timeouts: the correlation engine waits for a bounded time to match other events. When either the directive is fully matched, or no further related events are delivered for some time, the directive expires and it is discarded. Since these timeouts are tied with the system time, the attack can indeed prevent an accurate risk assessment, and thus alarms.

Solutions

1) create a centralized NTP server and isolate the domain to make it trusted.
2) disable local system time management, or at least consider every time change as a critical event that must be reported. Enable and configure the available NTP authentication options.

Remarks

References

URL: http://alienvault.com/docs/Installation_Guide.pdf
URL: http://www.alienvault.com/wiki/doku.php?id=user_manual:correlation\#correlation_directives

Vulnerability: S.Vu.9.0

Code Name: direct-disk-read
Classification: high
Type: password disclosure
Access Vector: local
Access Complexity: high
Authentication: none
Description: the password to access the database is disclosed

Target

Products: AlienVault OSSIM v2.3
Components: ossim_setup.conf
Configuration: default
Comments: the OSSIM system configuration file contains the database password stored in plaintext

Detailed Description

If the attacker has physical access to the machine and downtime happens, the disk can be externally connected and read, and therefore the database password is disclosed. An equivalent problem can occur with passwords stored inside the database (such in the *ossim.config* table).

Impact

Type: jeopardy of security information
Integrity: almost complete
Confidentiality: almost complete
Availability: almost complete
Comments: the threat level depends on the privileges of the compromised account. In any case, even if the access is limited to part of the database, then: read operations impact on confidentiality and availability (through DOS); write operations impact on integrity and availability (by means of resource exhaustion). The correlation engine that uses the database in question is thus compromised.

Solutions

1) physically protect the machine where the files are stored
2) encrypt the filesystem or implement a password storage to keep confidentiality

Remarks

The password must not be stored on disk in plaintext, but it must be safely used at runtime. According to the authentication protocol of mysql, only the hash of the password need to be know to log in. Storing the hash directly on disk does not solve the vulnerability.

References

URL: http://www.alienvault.com/wiki/doku.php?id=installation
URL: http://forge.mysql.com/wiki/MySQL_Internals_ClientServer_Protocol\#Client_Authentication_Packet

Vulnerability: S.Vu.10.0

Code Name: hidden-loading
Classification: high
Type: execution integrity attack
Access Vector: local
Access Complexity: high
Authentication: none
Description: OSSIM execution on a specific machine can be arbitrarily deviated by means of either code injection or impersonation attack

Target

Products: AlienVault OSSIM v2.3
Components: system storage
Configuration: default
Comments: the OSSIM system does not use an encrypted filesystem by default and lacks in code execution integrity check and plugin attestation mechanisms

Detailed Description

The OSSIM system is based on the Debian Linux distribution. On one side, the operating system uses by default neither an encrypted file system, nor driver signing enforcement, nor code execution integrity checks. On the other side, OSSIM does not perform any check both on the plugins that are loaded and on their content. The system on the whole is thus seriously exposed to offline attacks and others by which the attacker can gain the highest privileges (e.g., by exploiting some buffer-overflow). In the first case, the content on disk can be arbitrarily modified during a possibly malicious downtime. In the second case, the whole system is accessible and its execution can be arbitrarily deviated.

Impact

Type: security information management hijack
Integrity: complete
Confidentiality: complete
Availability: complete
Comments: since the filesystem is not encrypted, all of the information stored in plaintext is compromised. Furthermore, ad-hoc kernel modules and OSSIM plugins can be loaded to tamper with the reliable security information management. As a result, an attacker can fully control the component, disrupting the process of event collection and undermining the information that has to be forwarded (i.e. normalized and correlated events). If the component is the only gateway between (some part of) the monitored network and the monitoring infrastructure, attacks can be hidden through impersonation.

Solutions

1) use an encrypted filesystem to mitigate confidentiality issues
2) use kernel module (resp. plugin) digital signing techniques, and enforce verification-based module (resp. plugin) loading through cross-certificates, up to the trusted certification authority
3) use code execution integrity check or software-based attestation techniques to prevent malicious execution detouring
3) enforce safe boot through a TPM to prevent offline attacks to module signatures and operating system's keys

Remarks

References

URL: http://www.alienvault.com/wiki/doku.php?id=installation
URL: http://www.alienvault.com/wiki/doku.php?id=architecture:plugins