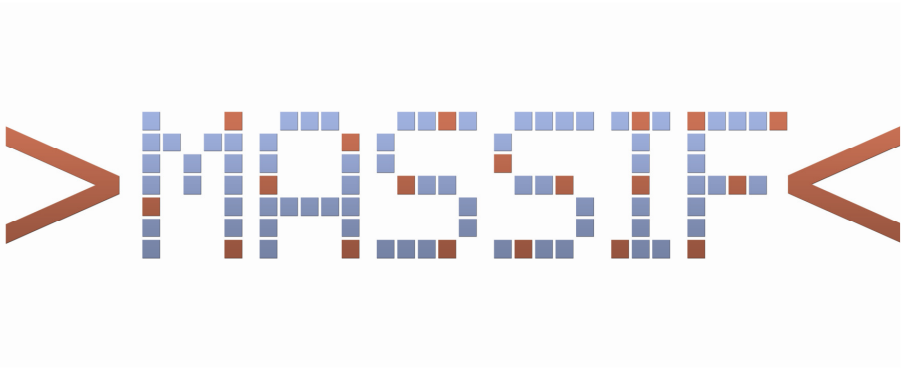


MAnagement of Security information and events

in Service InFrastructures

MASSIF

FP7-257475

Architecture Document

Activity	N/A	Work Package	N/A
Due Date	2012-04-23	Submission Date	2012-04-27
Main Author(s)	Paulo Verissimo, Nuno Neves (FFCUL); Alexander Goller, Alberto Roman Limancero (AlienVault); Susana González, Rubén Torres (ATOS); Luigi Romano, Salvatore D'Antonio (CINI); Hervé Debar (IT); Roland Rieke, Zaharina Stoynova (SIT); Igor Kotenko, Andrey Chechulin (SPIIRAS); Ricardo Jimenez-Peris, Claudio Soriente (UPM); Nizar Kheir, Jouni Viinikka (6cure).		
Contributors	Alysson Bessani, Nuno Neves, António Casimiro (FFCUL); Carlos Arce, Elsa Prieto(ATOS); Gustavo Gonzalez Granadillo, Malek Belhaouane (IT); Juergen Repp, Maria Zhdanova (SIT); Olga Polubelova, Evgenia Novikova (SPIIRAS).		
Version	V1.5	Status	Final
Dissem. Level	PU	Nature	R
Keywords	SIEM, Distributed Systems Architecture, structural view, functional view, payload system, data layer, event layer, application layer, Intrusion Detection, Security, Resilience, Generic Event Translation framework, MASSIF Information Switch, Resilient Event Bus, Event Processing, Decision Support & Reaction, Predictive Security Analyser, Attack Modelling and Security Evaluation component, visualization component,		
Reviewers	Rodrigo Díaz (ATOS), Luigi Coppolino (Epsilon), Alexander Goller (Alienvault), Jouni Viinikka (6Cure), Gunnar Bjoerkman (ABB AG, Advisory Board Member)		

Part of the Seventh Framework Programme

Funded by the EC - DG INFSO

Version history

Rev	Date	Author	Comments
V0.1	2011-12-12	P. Verissimo (FCUL)	ToC
V0.2	2011-12-29	E.Prieto, S.González (ATOS)	Rationale, roadmap, functional requirements and application services.
V0.3	2011-01-10	C.Arce, S.González, E.Prieto, R.Torres (ATOS)	Functional requirements update.
V0.5	2012-02-02	P. Verissimo (FFCUL)	First integrated draft.
V0.6	2012-02-24	P. Verissimo (FFCUL)	Second integrated draft.
V1.1	2012-03-05	P. Verissimo (FFCUL)	Final draft for comments.
V1.2	2012-03-31	P. Verissimo (FFCUL)	Final draft for comments (2nd ed.) after revision of application service interactions.
V1.3	2012-04-23	P. Verissimo (FFCUL)	First release after appointed reviews.
V1.4	2012-04-25	R. Diaz, E.Prieto (ATOS)	Integration of final comments and final review.
V1.5	2012-04-27	E.Prieto (ATOS)	Final updates, final edition and official delivery.

Glossary of Acronyms

AGG	Attack graph generator
AMSEC	Attack Modelling and Security Evaluation Component
APA	Asynchronous Product Automata
API	Application Programming Interface
CEP	Complex event processing
CP	Calm-Paranoid
CRUD	Create, Read, Update, and Delete
DB	Database
DBMS	Database Management System
DMZ	Demilitarized zone
DoS	Denial of Service
DS&R	Decision Support and Reaction
GAP	GET Access Point
GET	Generic Event Translation
GPS	Global Positioning System
GUI	Graphical User Interface
HIDS	Host-based Intrusion Detection System
ICT	Information and Communications Technology
IDMEF	Intrusion Detection Message Exchange Format
IDS	Intrusion Detection Systems
IP	Internet Protocol
KPI	Key Performance Indicator
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
MAC	Message Authentication Codes
MASSIF	Management of Security information and events in services infrastructures.
MEH	MASSIF Events Handler
MIA	MASSIF Information Agents
MIS	MASSIF Information Switch
MM	Malefactor modeller
NTP	Network Time Protocol
OrBAC	Organization-Based Access Control
OSSEC	Open Source Security

OSSIM	Open Source Security Information Management
PEP	Policy Enforcement Points
PSA	Predictive Security Analyzer
PyOrBAC	Python OrBAC
QoS	Quality of Service
RBAC	Role-Based Access Control
REB	Resilient Event Bus
RORI	Return on Response Investment
SCADA	Supervisory Control and Data Acquisition
SE	Security Evaluator
SG	Specification Generator
SIEM	Security Information and Event Management.
SOA	Service-Oriented Architecture
SP	Security Probe
TCB	Timely Computing Base
TCP	Transmission Control Protocol
TPM	Trusted Platform Module
VLAN	Virtual LAN
VPN	Virtual Private Network
WAN	Wide Area Network
XML	eXtensible Markup Language
XSD	XML Schema

Executive Summary

The main goal of the present architecture document is to depict a global view of the MASSIF system and of the solution it intends to achieve. This document is primarily for external users to understand how the MASSIF solution is structured and how its components work together. Therefore the document provides a rather high-level description of different elements (structural layers, components and functionalities). Nevertheless, for complete information, we strongly recommend the reading of official MASSIF project deliverables found in the project website (http://www.massif-project.eu/). Consortium developers may also be interested in this documentation to understand the relationships and dependencies among components within an integrated framework.

MASSIF Architecture Overview

The MASSIF architecture is represented through several views that convey different perspectives, levels of abstraction and needs. Yet a technical approach has been considered for all of them. The MASSIF architecture is intended to be as general as possible and does not address any particular scenario or use-case that could add specific constraints to the common solution. Thus it is worth pointing out that not every single component is subject to be adapted to each considered scenario, but we will have different instantiations of the general architecture depending on the given characteristics of the scenario. Furthermore, the MASSIF architecture, which is open, is intended to be powerful and generic enough to allow future system integrations, and support different use-cases from the ones foreseen in this project, by third parties.

The MASSIF architecture document provides an integrated view of the project, cross-cutting the different activities. In order to give the reader a perspective on the correspondence between components and activities, such mappings onto the concrete project activities can be found at the end of this document in the annexes section.

A diagnosis of the shortcomings of current Security Information Event Management (SIEM) systems, which led in part to the proposal of the MASSIF architecture, 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; insufficient correlation and lack of countermeasure capabilities; centralized rule processing, making scalability difficult by creating bottlenecks and single points of failure.

Addressing these problems implies a set of functional, as well as non-functional requirements, to be met by the MASSIF architecture. Some functional requirements bring about innovative functionality compared with existing SIEM. On the other hand, satisfying non-functional requirements such as resilience, understood as the capacity to maintain acceptable levels of security and dependability in harsh operating conditions, is considered by MASSIF a key asset of critical SIEM systems, given the current and expected severity of advanced persistent threats or targeted attacks. Satisfying those attributes and requirements implies meeting a set of key objectives:

- Scalable data acquisition and collection of huge amounts of events from diverse and geographically spread nodes.
- Distributed and near real-time aggregation, dissemination and processing of events; alert generation and incident notification; countermeasure propagation.
- Scalability and elasticity of correlation, across integrated and distributed engine implementation alternatives.

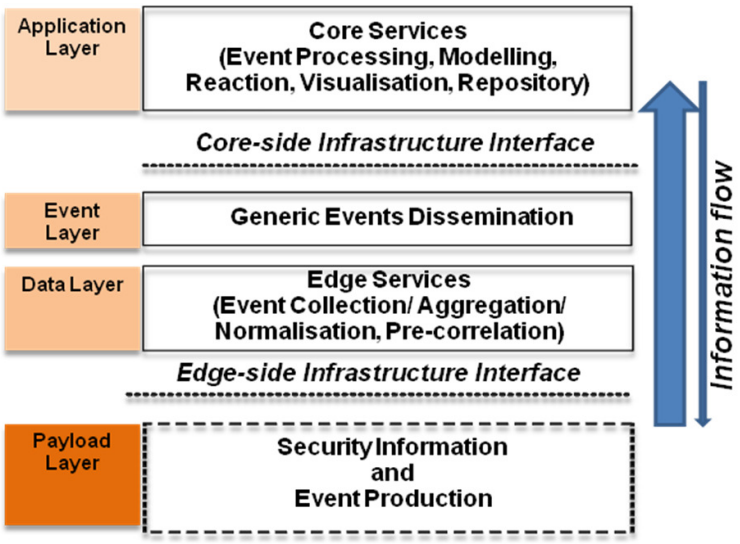
- Clear decoupling between the target (monitored) and SIEM (monitoring) system, for minimal impact on the observed infrastructure, and adaptation to varying target/SIEM system combinations.
- Resilient operation of the above against faults and attacks of incremental severity, maintaining availability, integrity and confidentiality.

When reflecting about key non-functional aspects of the MASSIF SIEM architecture, such as scalability, versatility and resilience, one has to take into account:

- 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 trade-off for cost and complexity.
- The difficult combination of characteristics such as: security, timeliness, multi-tenancy.

MASSIF Architecture Components

The MASSIF architecture intends to address the aforementioned objectives. The MASSIF SIEM system is structured as an infrastructural overlay of the monitored payload system. The overlay is implemented by devices which provide the hooks to the monitored system, MASSIF Information Switches (MIS), whilst they themselves serve as nodes of the overlay. The MASSIF SIEM architecture features several layers: Data layer, Event layer, Application layer.



The main purpose of the MASSIF Data layer services is to deliver security information flows up to the MASSIF core, as indicated by the thick arrow. In order to do so, this layer must provide the functionalities for the collection, aggregation and normalization of the events generated by the payload machinery and use the services offered by the MASSIF resilient infrastructure to provide the relevant security data to MASSIF applications.

The Event layer is provided by a generic events dissemination service, implemented by the Resilient Event Bus (REB), supported on a dedicated communication service resilient to faults and attacks. The communication service is implemented by protocols running amongst the MIS. The REB performs

generic event dissemination towards the services in the core-side of the infrastructure, namely the event processing engine.

The Application layer features several key services. Processing of events in the MASSIF SIEM is performed by a highly-scalable, elastic correlation engine. The latter is materialized as a parallel Complex Event Processing (CEP) system. Security monitoring in MASSIF SIEM is supported by the Predictive Security Analyser (PSA), which performs multi-level predictive security monitoring. The Attack Modeling and Security Evaluation Component (AMSEC) is intended to complement the direct analysis functionality of the SIEM system, by providing the architecture with the capability of attack modeling and security evaluation. The Decision Support and Reaction (DS&R) component provides an administrative tool based on the OrBAC model, which allows to consolidate the security policy through the different infrastructure's components in an organization, and to configure automatically those components, enforcing the countermeasures to be applied (as indicated by the fine arrow).

These services are helped by a generic visualisation service and a repository service. The purpose of the visualisation component is to provide a convenient and effective GUI for the interaction with MASSIF SIEM components. The common MASSIF repository provides cross-layer information integration from different components of the MASSIF SIEM.

MASSIF Architecture Resilience

Several mechanisms support the seamless integration of resilience into the distributed MASSIF SIEM system, with the aim of ensuring several levels of security and dependability in an open, modular and versatile way. The solutions proposed were essentially inspired by two main issues of the current SIEM arena:

- the monitored environments are increasingly exposed to threats, and more prone to different sorts of failures;
- dependence on the monitoring systems to ensure secure and dependable operation of the monitored systems in real-time is increasing dramatically.

For example, in MASSIF we discuss techniques to improve the resilience of specific nodes of the architecture, such as the MASSIF Information Switches (MIS). The MIS can be built with incremental levels of resilience, depending on its criticality, from baseline ruggedised simplex machines, up to physically replicated Byzantine resilient units. Recall that we leave the monitored system essentially untouched, and base our resilience solutions on the overlay, of which the MIS are key points.

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. The Resilient Event Bus is an overlay communication subsystem internal to the MASSIF SIEM and thus itself protected, much in the sense that secure VPN (virtual private networks) are. 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.

The MASSIF architecture allows for multiple strategies for protection of the core components executing application layer services. The simplest one is perimeter defence, by isolating the core components within trusted intranets, only communicating with the outside through a MIS, in two ways: with the Resilient Event Bus; and with auxiliary systems. Besides executing protection functions, the core-MIS is itself built with resilience enhancing mechanisms, to protect it from direct attacks. Besides this baseline protection, SIEM core resilience can be enhanced through more sophisticated forms of protection, through fault and intrusion tolerance. Such solutions would for example provide resilience against insider attacks.

The publish-subscribe nature of the Resilient Event Bus communication model extends the modularity of the edge subsystems to the core systems: application servers may actually reside in more than one protected intranet, offering a multitude of deployment and server placement strategies.

The storage solutions to be deployed in the MASSIF architecture have several purposes, requiring different levels of resilience. Amongst them, MASSIF foresees storage units dedicated to archival of critical security information and events, requiring properties like integrity, confidentiality and unforgeability. One of the obvious uses of such resilient storage is to archive important security information and events in a way justifiably usable for criminal/civil prosecution of attackers after a security breach.

MASSIF Architecture vs. existing systems

Security Information and Event management systems have existed for about ten years. Even though they still can be improved, they are being commercially successful today, which shows that designing an entirely new SIEM system from the ground up would be an enormous effort for little benefit. Instead, the MASSIF project has chosen to partner with vendors of two prominent open source SIEM, OSSIM and Prelude[1], to complement them with enhanced MASSIF functionality whilst reusing existing functions provided by these SIEM implementations. The open-source choice (even though we are also looking at commercial SIEM environments) has been made because it eases analysis and integration.

[1] Websites of these products: http://communities.alienvault.com/community; http://www.prelude-technologies.com/en/welcome/index.html.

Table of Contents

List of Figures

List of Tables

1. Introduction and Scope

1.1 Roadmap

The present document introduces the MASSIF architecture. This document is established in the frame of MASSIF project but is not related to a specific activity, work package or task. In fact it does not appear as contractual document of the Annex I of the project. This document is organized as follows:

- The "Introduction and Scope" section provides the roadmap, purpose and document overview. It also establishes the premises and considerations to build the MASSIF system.
- The "General Overview" section introduces the main architectural blocks and services offered per block.
- The "Structural View" section provides a macroscopic view of the topology and of the main components encapsulating the system functions.
- The "Functional View" section introduces the system decomposition into the different major components. For each component an explanation of its organisation, function, operation and interaction with other components is given.
- The "Resilience Mechanisms" section focuses on a core non-functional aspect, presenting the MASSIF resilient framework architecture, revealing the most important aspects on resilience that will be taken into account in the final solution.
- Finally, the "MASSIF Architecture vs. Existing Systems" section presents a comparison between MASSIF architecture and the elements that can be found in existing SIEM solutions, namely OSSIM (Open Source Security Information Management) and Prelude, the exemplary systems within MASSIF project.

The figure below shows the roadmap for MASSIF architecture, developments and integration tasks. This indicates what have been done up to the delivery date of the architecture (marked by a vertical brown line) and what is still left. At the date of submission, there will be some design deliverables pending that could have an impact on the current architectural design. Therefore this document cannot be considered as a final version. Additionally, the integration task may require additional changes of the document. If applicable, these changes will be performed when needed.

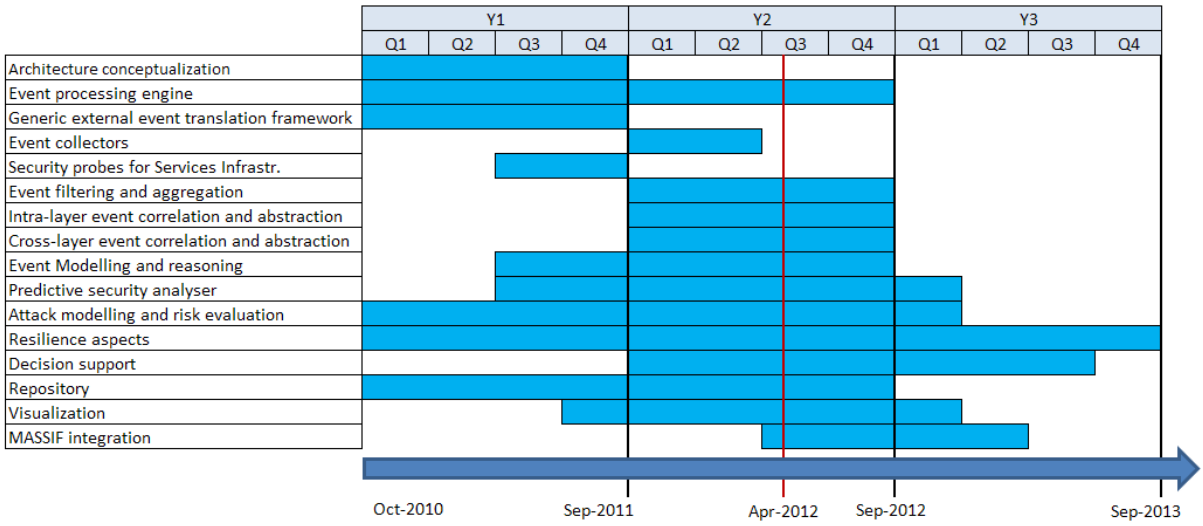


Figure 1 - Roadmap

1.2 Rationale

The main goal of the present architecture is to depict a global view of the MASSIF system and of the solution it intends to achieve. This document serves two purposes.

Firstly, consortium developers will be interested in this documentation to understand the relationships and dependencies among components within an integrated framework. Such an architecture document is important for ensuring that partners share a common vision of the production of the project, know where and with whom they should be prepared to integrate and test, and what interfaces and information they are consuming, producing and offering. Furthermore, the present document will serve as a guide for the future tasks of integration, tool adaptation and evaluation.

Secondly, it is intended for an external audience, to explain how the MASSIF solution is structured and how its components work together, providing a high-level description of different elements (structural layers, components and functionalities). Furthermore, for complete information, this document is complemented by the official MASSIF project deliverables found in the project website (http://www.massif-project.eu/). .

The MASSIF architecture is represented through several views that convey different perspectives, levels of abstraction and needs. Yet a technical approach has been considered for all of them.

The MASSIF architecture is intended to be as general as possible and does not address any particular scenario or use-case that could add specific constraints to the common solution. Thus it is worth pointing out that not every single component is subject to be adapted to each considered scenario, but we will have different instantiations of the general architecture depending on the given characteristics of the scenario. Furthermore, the MASSIF architecture, which is open, is intended to be powerful and generic enough to allow future system integrations, and support different use-cases from the ones foreseen in this project, by third parties.

The MASSIF architecture document provides an integrated view of the project, cross-cutting the different activities. In order to give the reader a perspective on the correspondence between components and activities, such mappings onto the concrete project activities can be found at the end of this document in the annexes section.

1.3 Understanding the Functional Requirements

Guidelines for the MASSIF framework (and future SIEMs) were developed in [27] , which established a set of functional, as well as non-functional requirements, to be met by the MASSIF architecture. Their analysis serves as a guide to the architectural decisions followed. In this section, we start by understanding the functional requirements:

- The MASSIF system must interface the monitored system. The MASSIF system and the monitored system will exchange information between each other. There will be an upstream of security information (events) consisting of both events pushed by the monitored system to the MASSIF Security Information Event Management (SIEM) and events requested by the MASSIF SIEM platform from the monitored system. In addition there will be a downstream consisting of commands and countermeasures from the MASSIF SIEM to the monitored system that would describe modifications of the monitored system. However the monitored system should be left as undisturbed as possible, or at least the capabilities required by the MASSIF SIEM system in terms of monitoring and countermeasures should be fixed and acceptable to the business system owners.
- The MASSIF system must be able to collect security data (events) generated by different kinds of probes at both the network and the services layer of the monitored system. These probes can be highly distributed in the monitored system. Since there are many different formats of collected

data (events), these must be translated into an internal format (the MASSIF format) independently of the source format.

- Similar events should be aggregated in a single event. Conditional filters (ruled-based) should be able to be applied to different stages of the event collection phase. Standardized event attributes should be used in conditional constructions used for filtering, action triggers, correlation and reporting.
- Any event should include a common trusted timestamp reference independently of the security probe.
- The MASSIF system should inform an appropriate user about possible attacks or abnormal behaviors. Advanced event processing should be able to filter out unwanted events and generate alerts on key issues. The alerts should be prioritized to support the decision-taking.
- The system should rely on simulation of attacks and countermeasures to evaluate their possible impact at the operational level.
- The system should be able to select automatically the most suitable remedial actions. The selected countermeasures should be transformed into commands applicable to the selected components and tools. The system should be capable of applying certain countermeasures automatically to protect assets.
- MASSIF should be able to predict security threats before the occurrence of possible security incidents, and detect close future violations of security monitoring rules.
- MASSIF system monitoring should be compliant with security policies, rules, models and metrics defined beforehand by an authorized user. The system should be flexible enough to express a wide variety of rules and deploy them over a likely distributed environment.
- Security directives, rules and models should be introduced or modified in an easy and dynamic way. Users can interact with MASSIF components through a Graphical User Interface (GUI). No information should be lost during the change and internal processes should not be affected during the change. New security policies, rules, models and metrics must be able to be added without interrupting the service.
- The system must be able to display in a management GUI (Graphical User Interface) an overview of all security events that constitute security incidents of interest in real time (the correlated events and alerts).
- The system must be able to show in a management GUI the list of the original basic events that triggered the correlation rule and generated the security incident.
- The system must be able to detect the compromise of the event data integrity, either in transit or in storage, and to indicate this in a management GUI.
- MASSIF components must assure that for internal events (failures detected, user accesses or any configuration change) in the system a log is produced and stored for internal audit.
- The security information must be stored at different stages of the information management (raw data, normalized). The system must apply the least persistence principle: it stores only the information needed to forensic analysis, historical and trend evaluations, recovery actions and statistics. The received security events should be able to be indexed, compressed and archived. The search engine should be available to easily access to stored events.
- The system must support data access isolation. The system must allow the management of different roles and identities (authentication). Only specific operators may be authorized to perform certain operations, after a proper authorization procedure. The events or the data reported or generated by the management and control system components (i.e. the measurements parameters) should not be seen by unauthorized persons. Data records containing information

related to security breaches must be available only to authorized parties, based on existing and upcoming policies.

- Users' actions in the system must be logged. The logs must be stored and kept for a certain period of time (depending on the country's local regulation and attending to the company's records and Information Management procedures) on a secured and dedicated log-service platform.

1.4 Understanding the Non-Functional Requirements

In this section, we review and understand the non-functional requirements laid down in [27] , that is, those related to, e.g., performance, scalability and mainly, resilience, understood as the capacity to maintain acceptable levels of security and dependability in harsh operating conditions:

- The system should be flexible enough to integrate a growing number of devices or probes deployed in the architecture, to operate through diverse administrative domains, maintaining its performance and timeliness capabilities. Namely:
    - at the collection point, the system must be able to handle data peaks; this is, collect data at the highest rate the probes can produce and maintain the performance capabilities;
    - collection, aggregation, normalization, prioritization and correlation of security data (events) must be performed in (near) real-time.
    - archival should be capable of storing and retrieving generated events in a scalable way.
- The system must ensure a reliable flow of information upstream and downstream, as well as storage, generically preserving integrity and confidentiality, namely:
    - event flow protection, from the collection points through their distribution, processing and archival, maintaining ordering;
    - authenticated and unforgeable component status reporting;
    - authenticated, unforgeable and non-repudiable (auditable) internal event log production and storage;
    - timely and orderly generation of alarms and countermeasures when needed.
- The system should maintain availability and integrity in face of the occurrence of accidental faults and of isolated attacks, namely through:
    - appropriate mechanisms like redundancy and cryptographic protection;
    - flexible and incremental solutions for node resilience, providing for seamless deployment of necessary functions and protocols.
- In case of severe fault/attack patterns, such as multiple component failures and/or unpredictable network operation conditions, the whole SIEM infrastructure should achieve high resilience. Namely:
    - data generated by the monitoring devices and tools (sensor data) must keep feeding the core machinery with acceptable quality vis-avis normal attack/fault situations;
    - likewise, in such harsh conditions the system must possess the necessary reconfiguration ability in order to preserve its crutial functions;
    - if necessary, both in communication and processing, activities and information flows may be resourced by order of criticality, specially the generation of alarms and countermeasures.

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

- A topology following the WAN-of-LANs model [17] , and laid down as a logical overlay over the target system, so as to preserve legacy but allow seamless integration of the monitoring and monitored systems, possibly across different and wide-scale administrative domains.
- Modular and adaptive structure, achieved by: (i) using modular functions and protocols, to be re-used by different instantiations of the architecture; (ii) 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). MIS/MIA construct the MASSIF architecture "LEGO" in symbiosis with the monitored system.
- Information flow in the overlay implemented as a secure and real-time event bus, modelled essentially as a producer-consumer SCADA-like (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) 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.

2. General Overview

2.1 Main Architectural Features

This section presents the architecture. It begins by introducing the key options of the architecture and the system model, in the context, when appropriate, of the requirements laid down in the Understanding of Requirements sections (1.3 and 1.4). Next, the architecture block diagram and several service categories are succinctly presented, to be detailed further ahead in the document.

The main desirable characteristics of the SIEM architecture are laid down so as to fulfil the set of requirements, both functional and non-functional, outlined earlier. They can be summarized into a few key objectives:

- Scalable data acquisition and collection of huge amounts of events from diverse and geographically spread nodes.
- Distributed and near real-time aggregation, dissemination and processing of events; alert generation and incident notification; countermeasure propagation.
- Cross-layer correlation and predictive security monitoring capabilities,
- Integrated and distributed correlation engine implementation alternatives.
- Clear decoupling between the target and SIEM system, for minimal impact on the observed infrastructure, and adaptation to varying target/SIEM system combinations.
- Resilient operation of the above against faults and attacks of incremental severity, maintaining availability, integrity and confidentiality.

When reflecting about key non-functional aspects of the MASSIF SIEM architecture, such as scalability, versatility and resilience, one has to take into account:

- 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 trade-off for cost and complexity.
- The difficult combination of characteristics such as: security, timeliness, multi-tenancy.

These considerations educated the organisation of the MASSIF architecture.

2.2 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. Classical intrusion prevention techniques are certainly an important approach to deal with many threats. However, most defences 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 fault and intrusion 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.

2.2.1 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. 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. However, in the presence of a hacker or a malicious person that is willing to disrupt the system, this approach is not recommended unless perhaps in parts where it can be enforced with very high probability (protected subsystems, use of trusted components, etc.).

Arbitrary failure assumptions specify no qualitative or quantitative bounds on component failures. In this context, an arbitrary failure means the capability of affecting a value or a message, at any time, with whatever syntax and semantics (form and meaning), anywhere or in parts of in the system. 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. Hybrid models allow stronger assumptions to be made about parts of the system that can justifiably be assumed to exhibit fail-controlled behaviour, whilst other parts of the system are still allowed an arbitrary behaviour. For example, commodity computers with a Trusted Platform Module (TPM), 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.

Practical systems based on arbitrary or hybrid failure assumptions very often specify quantitative bounds on component failures, or at least equate tradeoffs 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, opening the door for the so-called intrusion-tolerant protocols, or protocols which mask up to a given number of arbitrary failures. Note the power of these protocols: given the assumption that the hacker can compromise up to f components (f is a parameter, can take any value) during the

execution time, the protocol neutralises *any* intrusions, and plus does so in an automatic way. That is, we get rid of the stress of having to completely *prevent* intrusions: we allow up to *f* of them, whilst still *preventing security failure*, the ultimate goal.

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 define the assumptions on the faults affecting the flow of information from sensors to the core SIEM systems.

Edge-side

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. 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, we assume the following of the edge-side components:

- 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;
- with the right monitoring capabilities in place, it should be possible to detect such kind of faults through correlation at the core layer.

The network that connects payload sensors to 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 he 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, we assume:

- it is possible to deploy specific measures to protect the operation of collectors and make them resilient; 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).

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 monitored system, the information might be sent through a local LAN, and therefore, it is easier to enforce a more controlled behaviour. 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. However, we assume that:

- measures can be taken in the WAN part of the WAN-of-LANs infrastructure typical to the SIEM system, ensuring desirable properties of end-to-end communications, such as availability and near real-time, confidentiality and integrity; in particular, path redundancy and overlay, and cryptographic message protection.

Core-side

The core layer, which includes the processing engine that does correlation on the events and several other critical core services (see Section 4.3), is classically protected with some sort of devices (e.g., a firewall) aimed at preventing external attacks. However, under these circumstances, those devices can also become a target of attack, and most certainly will, within the scope of highly skilled targeted attacks as expected for critical IT systems and infrastructures – in fact, over the past years, several vulnerabilities have been described for the most commonly used firewalls [13] , [14] , [15] . This means that in order to ensure the safe operation of the core services, 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. We assume the following:

- measures can be taken to ensure a seamless protected end-to-end flow of information from the protected edge nodes, to protected core nodes which also establish a resilient protection perimeter to the core services;
- intrusion tolerance through redundancy and diversity may further avoid single points-of-failure in the presence of both faults and attacks, to selected critical core servers.

Certain core services 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, although we have been considering perimeter defence for the core-side services, it may make sense to consider a scenario where an inside employee could try to disrupt the historian operation. In consequence, we assume that:

- selected critical core server intrusion tolerance measures can be extended to enforce unforgeability and non-repudiation of information storage, in addition to classical strong authentication and access control policies.

2.2.2 Synchrony model

We briefly address the synchrony model, which refers to assumptions on time and timeliness.

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 neither fully asynchronous nor fully synchronous. Therefore, there exist *partially-synchronous* and *hybrid synchrony* models 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. Next we discuss the appropriateness of the several models for MASSIF.

An extremely attractive aspect of the asynchronous model is its simplicity. 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 would be 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. 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 failures.

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, the rate of drift of local clocks is also known and bounded, allowing clocks to be synchronized, performing synchronized actions, and time stamping distributed events. The synchronous model would seem to be the elected model for the MASSIF subsystems dealing with the above operations (e.g. near real-time event dissemination and processing).

However, this model suffers from a major drawback, which is related to the lack of coverage of the synchrony assumptions, be it due to uncertain performance of parts of the system, or in the presence of time-based attacks, e.g. introducing artificial delays or changing clock or timestamp values, 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 resilience and may compromise the correctness of the related SIEM subsystems.

One approach to escape the problems encountered by developing solutions under the synchronous or asynchronous models is to consider partial or hybrid synchrony. These models essentially make additional assumptions that allow achieving some timeliness properties without falling into the problems caused by the lack of assumption coverage. Essentially, they build on the idea that synchrony is not a homogeneous property in the time or in the space domains, that is, that the infrastructure either becomes faster or slower during the execution and thus synchrony comes and goes (partial), or that some parts may be more predictable and synchronous than other parts (hybrid).

One typical example of the partially-synchronous model assumes 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. Another well-know example, the *timed asynchronous* model, assumes an asynchronous model with the additional assumption that processes have access to a physical clock with a bounded rate of drift, making it possible to detect timing failures. One typical example of hybrid synchrony is the TCB model [16] , which assumes an asynchronous system enhanced with small synchronous components providing the necessary anchor to real-time.

Hybrid Synchrony in MASSIF

It looks like the adequate model for defining protocols in uncertain and attack-prone environments, with heterogeneous loci of synchrony, such as the settings we consider in MASSIF, should lie in the partial or hybrid synchrony group. Both perspectives are interesting, but it should be further observed that there is an important difference between them. 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 resilience perspective, in the presence of malicious faults, this difference is definitely crucial, since the time-domain behavior for at least one part of the system is well-known and can be relied upon *despite the attacker power*.

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. In general, we can assume:

- edge and core layers MASSIF nodes have access to local clocks providing a global and trustworthy notion of time;
- time stamps can, in turn, be used to infer about both the timeliness of events and about their ordering.

Some sensors in payload nodes at the edge layer 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 the edge MASSIF 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 payload-to-collector network. Whether or not sensors have access to local clocks, it can happen that the produced time stamps may not be trustworthy, which in practice, leads to a situation similar to the one mentioned above, made worse with the possibility of malicious time stamp manipulation, at the sensors or in transit. In any case, we may assume that:

- analysis of the time series of events affected by accidentally caused delays (e.g. variable load), but correctly time stamped at the collectors, should allow correct reasoning about their semantics, at higher abstraction levels, filtering out timing errors;
- correlation of the time series of different event flows from the same payload-to-collector network should allow correct reasoning about their semantics, at higher abstraction levels, masking out time stamp tampering.

Regarding the WAN part of the networking infrastructure, recall that we do not consider it inside the defense perimeter. As such, its synchrony properties are bound to be quite uncertain, this made potentially worse by attacks (e.g., DoS). However, one can still make interesting assumptions that will be used in the MASSIF WAN communication protocols:

- individual links between any two edge or core nodes have a form of partial synchrony, in the sense that message transmission latency is bounded, although it is difficult to state the exact bound;
- at deployment time, specific bounds will have to be assumed, which means that the specific link will alternate between synchronous and asynchronous behavior (respectively when the bound is or is not met);
- in the presence of overloads or attacks to the network in general, in an interval of time, and given a sufficient number of alternate links between any two edge or core nodes, there is at least one link which behaves synchronously.

The assumptions made in the fault and synchrony models described above will dictate the kind of architectural solutions and protocols that will be developed in MASSIF.

2.3 Architecture Block Diagram

The MASSIF SIEM architecture features several layers: Data layer, Event layer and Application layer. These layers are superimposed over a Payload layer that we describe for the sake of clarity, but which is actually formed by the monitored system, external to MASSIF. The Payload layer actually produces the security information and events to be processed by the several MASSIF layers. Indeed, that describes reality since, as will be seen later in Section 3, the MASSIF infrastructure is laid down as an overlay over the monitored system.

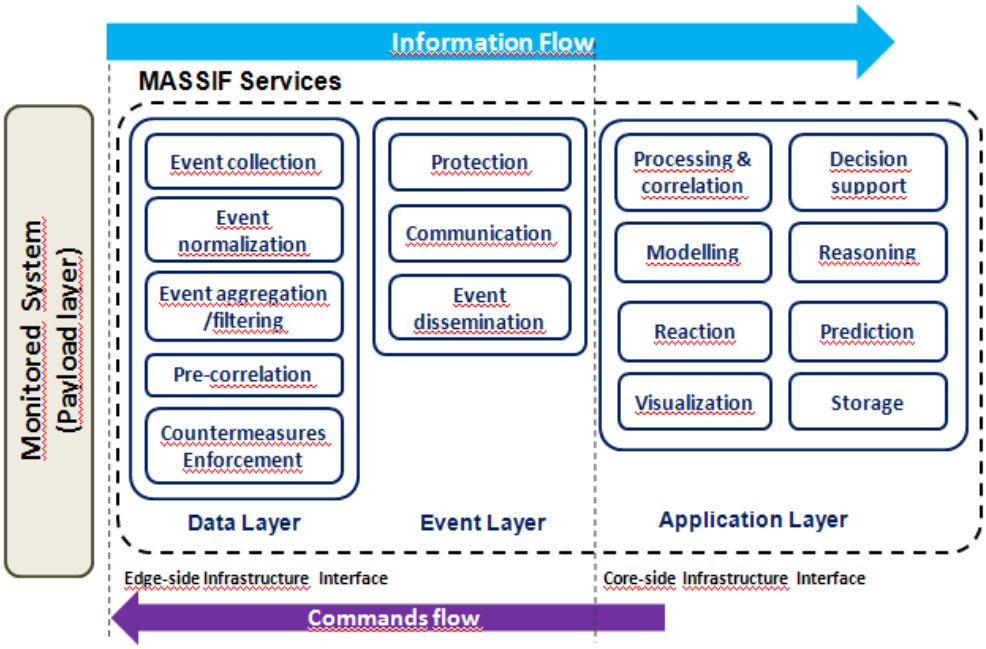


Figure 2 - Block diagram of the architecture

The main building blocks of the architecture are shown in the block diagram of Figure 2:

- Data layer – event collection, aggregation, normalisation, and pre-correlation
- Event layer – communication; event dissemination (reliable, in time stamp order); protection.
- Application layer – event processing, modelling and simulation (including reasoning and prediction), decision support and reaction, visualisation, repository.
- The information flow model features high-bandwidth producer-consumer upstream (cyan arrow): events and other security information are produced by the payload layer, pre-processed by the Data layer, disseminated by the Event layer to the potentially several elements of the Application layer, to be finally processed (consumed) by the latter. Low-bandwidth, low-latency channels provide notifications downstream (as suggested by the purple arrow), which may convey commands prefiguring reconfigurations of Data layer artefacts, as well as reaction actions (countermeasures) to these artefacts or even to payload system nodes.

Structurally, the MASSIF architecture is quite simple, as will be seen in detail in Section 3, aiming at disturbing the monitored system structure (the payload layer) in the least possible way. All MASSIF Event and Data layer functions are encapsulated in conceptual modules, placed according to the needs, in strategic places of the payload system. These are the MASSIF SIEM parts subject to a greater deal of threat. In consequence, they are generally implemented by specialised nodes, which we call MASSIF Information Switches (MIS), and which actually make information flow around in a reliable way, being also capable of providing some level of perimeter protection. MIS are themselves protected against intrusions and tolerant of accidental faults, as will be seen in Section 5.

The set of interconnected MIS form a distributed infrastructural overlay superimposed over the payload, implementing the Data and Event layers, and some ancillary services described in the next section. The edge-side interface of this infrastructure is made, as shows, with the monitored system (Payload layer). The core-side interface is made with the core servers, which acquire the disseminated information and process it. Core servers host services like the SIEM event processing engine, and other services like modelling, decision support and reaction, visualisation, repository.

For the architecture in general, 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, as will be discussed in Section 5.

2.3.1 Data Services

The aim of the services in the Data layer is to collect the relevant security data from the payload machinery (i.e. the lower layers devices supplying raw security information and event data). These devices can be of different forms depending on the application scenario: they may consist of specialized servers (e.g., a mobile payment application), network management and protection systems (e.g. an intrusion detection system (IDS) or a firewall), or physical sensors (e.g., water level sensor).

In order to manage and process such heterogeneous data within one common framework, we should create conditions for event fusion from those different sources. MASSIF events with different formats and origins and from different application domains, need to undergo a process of abstraction and coalesce into what we call *MASSIF Generic Events*, following a common syntactic and semantic format, which allows events from anywhere in the infrastructure to be fused into the Event Bus, which performs reliable and ordered event dissemination, and gives services at the Application layer a convenient way to treat this uniform event flow.

To perform such task the Data layer provides services for the collection of sensory information from the monitored environment, for the processing and filtering out of non-relevant information, and for the final normalization of the events. This process transforms the collected events into generic events that flow through the events layer for their final delivery to the MASSIF SIEM core.

Another key service provided by the data layer of the MASSIF architecture is pre-correlation at the edge: the objective of pre-correlation is to transfer part of the SIEM intelligence to the edges of the architecture in order to balance the load on the core processing engines and to reduce the communication traffic. There are several reasons for having pre-correlation at the edge side of the network: in the space domain, some specific semantics can be learned from several components in a same facility intranet of the setting under attack or going to failure; in the time domain, pre-correlation can help compact successive events related to the same syndrome or root cause. Pre-processing can, in general, reduce the volume of messages (and data in general) that travel through the infrastructure, reducing the data load on the Application level services at the core side of the MASSIF architecture, like the Processing Engines and MIS.

Last but not least, reaction and adaptation services can be provided by agents of the Decision Support and Reaction application module, to be performed on MASSIF's smart sensors, or on native sensors and event sources of the monitored payload machinery. They may serve, for example, for configuration and reconfiguration of the sensing policies.

2.3.2 Infrastructure Services

In this section we give an overview of the 'infrastructure services', as we collectively designate the services that implement the above-mentioned distributed infrastructural overlay superimposed over the payload. The infrastructure services include the Event layer services, essentially supporting the reliable flow of information and also controlling communication between MASSIF nodes. They can also implement protection, for example of the core services, as will be seen ahead. The following services are provided, to be detailed later in the document (see Section 4.2):

Communication. The communication service is implemented by protocols running amongst the MIS. These baseline protocols guarantee that this service is resilient both to accidental and malicious faults. Additionally, resilience to overload or denial of service (DoS) attacks is achieved by diverse routing.

Generic Events Dissemination. An event bus abstraction (Resilient Event Bus) is implemented over the communication service, inheriting its resilience, and implementing additional useful properties, such as time-stamp based ordering and event fusion from any source, since events are converted to a generic syntactic and semantic format. Decoupling between producers and consumers is also achieved by the publish-subscribe nature of the event bus. However, given the latency and throughput demands of the expected event flows from the edge to the core, the publish-subscribe event bus is intended to push the information in near real-time from the publisher to the subscribers (rather than being of an

asynchronous, persistent nature). The resilience aspects will be discussed with more detail in Section 5.

Protection. When needed, MIS are dual-homed, implementing, besides the SIEM functionality, a resident protection service akin to an application-level firewall, acting as a bastion providing perimeter defence. As mentioned earlier, this is especially interesting for core-side specific critical subsystems, such as the core SIEM event processing engines.

2.3.3 Application Services

The services offered by the application layer are aiming to provide the MASSIF core intelligence together with the common middleware services required to orchestrate and manage the application components. Consequently, the application layer will implement new intelligent and effective ways to derive information on the overall state using the observed events acquired and disseminated by the above commented layers. Application services can be grouped in the following types of services, to be detailed later in the document (see Section 4.3):

Event processing. A highly scalable event processing engine is the responsible of processing large amounts of streaming data in real time, as well as stored events for forensic analysis, with multi-level abstraction and correlation capabilities based on user-defined rules in a distributed, efficient, elastic and scalable way.

Modelling and simulation. This group of services will implement new process/attack analysis and simulation techniques in order to be able dynamically to relate events from different execution levels, define specific level abstractions, evaluate them with respect to security issues and during runtime interpret them in context of specific security properties. Two main modelling and analysis complementary approaches are integrated at this level. On the one hand, the Predictive Security Analyser (PSA) provides application aware security monitoring capabilities, supporting the near future application simulation and the prediction of potential security violations. And, on the other hand, the Attack Modelling and Security Evaluation Component (AMSEC) provides techniques for attack modelling and simulation, threat analysis and risk evaluation.

Decision support and reaction. The Decision Support and Reaction subsystem develops an administrative tool allowing the security policy consolidation through the different infrastructure's components in an organization, and to configure automatically those components based on selected countermeasures (by the operator/automatically). In addition to the modelling of systems and dependencies, this service will include also simulation capabilities that will allow submitting simulation models and simulation parameters to support quantitative evaluation and comparison of the attack and counter-measures impact.

Storage and Visualization. In addition to the functional services described above, some middleware application services are required to agglutinate the application layer components and to interface with the end user. These middleware services are mainly composed by: the short-term storage and long-term storage (for events, alerts, vulnerabilities, attacks, configuration, weaknesses, platforms and countermeasures) and visualization capabilities, allowing real-time security-incident notification, security status monitoring, analytics and reporting. The visualization tools will also support the specification of security parameters, queries, rules, policies, procedures and models of the overall infrastructure.

3. Structural View

In this section, the structural model of MASSIF is explained, proposing a topology relating the payload system (monitored system) with the SIEM system (monitoring system), discussing the placement of the main components and the network and event dissemination structure.

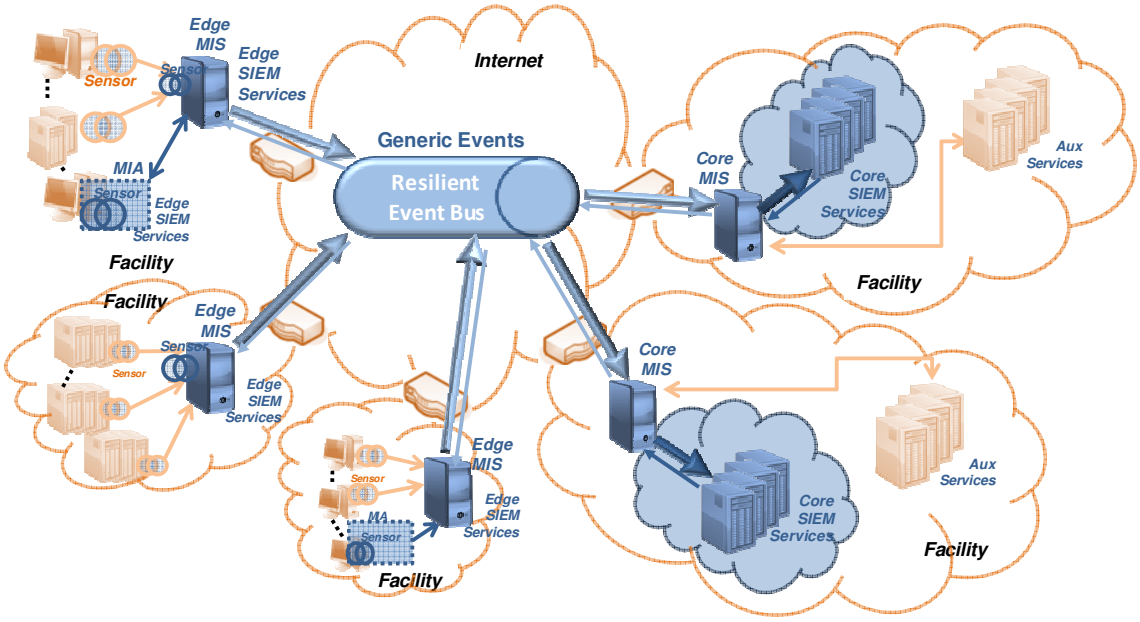


Figure 3 - MASSIF architecture structural view - payload (brown) vs. SIEM (blue)

The structure of a MASSIF SIEM system is shown in Figure 3. Let us recapitulate the notion of the MASSIF SIEM system (blue) actually laying out an infrastructural overlay of the monitored system (brown). The overlay is implemented by MASSIF Information Switches (MIS).

We model both the payload and the SIEM system interconnection as WAN-of-LANS [17] a useful construct to represent loosely-coupled wide-area infrastructures, pertaining to the same or different administrative domains, such as those envisaged as 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 forming virtual private networks 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 (physical to network layer).

We note 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 clearly materialised by the devices mentioned above: the MASSIF Information Switches (MIS). An alternative implementation, also shown in the figure, is the MASSIF Information Agent (MIA), which brings MASSIF intelligence deeper into the payload, implementing MASSIF remote *smart sensors*, as we explain in the next sections.

3.1 MASSIF Information Switch/Agent

For the sake of taking advantage of the architecture asymmetry, we separate between edge and core MIS which, though similar in nature, may have different configurations, be treated differently e.g., by producer-consumer protocols, and/or have different complexity and resilience.

Additionally, MIS being typically implemented as stand-alone machine/devices, for modularity, ease of configuration, performance and protection reasons (we may think of a MIS as an appliance box plugged onto the network), we also foresee software implemented versions of the same module, which we call MASSIF Information Agents (MIA). An MIA is a software appliance residing in edge payload nodes. The essential difference between the edge-MIS and the MIA depicted in Figure 3, 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 is the standard situation, where MASSIF SIEM relies on the payload's *own sensors*, and does not involve any modification of the information-producing devices, which send their log, event or alarm files to the nearest edge-MIS.

On the other hand, the MIA implements a remote *smart sensor*, that is, a MASSIF compliant sensor which allows part of the data layer functions to be performed in the payload machinery. This requires payload nodes to offer a local API to the basic sensing apparatus (syslogs, event services, etc.), and be open to installing external software modules, but apart from that, it should require minimal host modifications, allowing swift integration of MASSIF functionality into non-closed payload nodes.

3.2 Event Bus

MASSIF Information Switches also play an important role as generic communication servers, namely implementing the Resilient Event Bus, REB. The collection of MIS devices run the secure, reliable and real-time communication protocols needed to implement the Resilient Event Bus abstraction. These protocols 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 to implement the REB. Though this component will henceforth be designated Resilient Event Bus, the resilience aspects will be discussed later in Section 5.3, whereas here we introduce the functional aspects.

The Resilient Event Bus (REB) is mainly in charge of disseminating the events collected by the edge-MIS/MIA, after being pre-processed by the Data services implemented in the same edge-MIS/MIA, to the core-side Application layer services. The trustworthy MIS-to-MIS interconnection secures these information flows. The REB delivers the information to the core-MIS, which communicate reliably with the core engines, at the same time protecting them from external attacks, acting pretty much as a sophisticated firewall.

The REB should encompass both events created by the periphery and events generated from within the SIEM machinery. As shown in Figure 3, events are published into the event bus, mainly by the edge-MIS, to be delivered to the core-MIS subscribers. But this does not preclude edge-MIS from subscribing, or core-MIS from publishing events. In fact, that happens each time there are notifications or commands sent from the core services down to the edge of the infrastructure.

The flow from edge to core, as the figure suggests, 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 and other application services. In fact, given the latency and throughput demands of the expected event flows from the edge to the core, the event will push the information in near real-time from the publisher to the core entities having subscribed to it, the Event Processing Engine being the main subscriber.

3.3 Edge-side Services

In the MASSIF model, we consider that edge-side monitored system payload devices actually have their own basic sensory apparatus in place, be them raw event sources/emitters --- e.g. logs --- or native sensors --- purposely made metrology artefacts that measure Key Performance Indicators (KPIs) or alarm conditions of the payload systems. These are normally supplied with the monitored systems, even if they are extensions to basic configurations. We will generally call (monitored system) *sensors* to whatever is in place to acquire the raw security information and event data from the payload. The edge-MIS then act as collectors of information from the payload sensing apparatus, at the Edge-side Infrastructure Interface. Each edge-MIS is then in charge of implementing part or all of the Data services foreseen in MASSIF, which perform some sophisticated data processing. Namely, an edge-MIS should do at least event collection. However, it will normally implement other services as well, namely the normalization of the event formats and contents, aggregation of several events, and even some local pre-processing and correlation. Likewise it may also host agents capable of performing reaction and adaptation commands. These services will be detailed in Section 4.1. Generally speaking, we talk of MASSIF *smart sensors*, to address the edge-MIS data modules that acquire and process the basic information coming from payload sensors. This duality is shown in Figure 3.

An important alternative is brought into play by MASSIF Information Agents (MIA), described earlier. Whereas edge-MIS rely on the payload sensors and are confined to their limitations, some of functionality, some caused by faults or attacks (see Section 2.2), MIA introduce the notion of *remote smart sensor*, with two key advantages: part of the Data services may be performed in the MIA, in symbiosis with the local sensory apparatus, improving the quality of information and event acquisition. Actually, Data services can be split as wished between the MIA and the edge-MIS to which it connects (see Figure 3).

The additional integration effort of MIA into selected existing payload devices may well be justified for nodes offering reasons for local MASSIF intelligence: critical nodes such as core routers; nodes that are themselves very rich in information and event sources. As a matter of fact, 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. Another reason for resorting to an MIA is when a given payload device, albeit important, does not have incorporated sensors (i.e., lacks software modules capable of generating syslogs, events, etc. in a format exportable or understandable to the MIS). This will be rare in IT, but may happen in control devices such as used in critical infrastructures. A slightly higher integration effort may be well justified for critical devices lacking sensing capability.

In any case, one of the advantages is the capability of pre-processing and filtering the information, and even tuning those firewall or IDS devices in special ways, in response to commands issued by the Application layer. Another advantage of the MIA approach is guaranteeing a more trustworthy information and event feed from/to that particular payload node, not subject to the communication faults and attacks discussed in Section 2.2, since MIA-MIS interconnection is made through MASSIF reliable communication protocols.

3.4 Core-side Services

Core SIEM services are for example the event processing engine, and other services like modelling, decision support and reaction, visualisation, repository. The core application services process the information and events arriving from the edge, performing complex event analysis, normally using the stream data processing model, 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.

Auxiliary services are any non-critical services that are not part of MASSIF, but may be of interest to the operation of MASSIF as a whole (long-term archival, email, web apps, reporting, printing, etc.).The application services are bound to reside in data centres either of the monitored system's organisation (running its own MASSIF SIEM system) or of a third party organisation (in the case of an outsourced MASSIF SIEM managed service). Remember that MASSIF is supposed to operate through diverse administrative domains, and this is one of the reasons.

As Figure 3 suggests, these core-side critical subsystems (Core SIEM services), for example core SIEM event processing engines, are supposed to be housed in perimeter-protected LANs connected to the MASSIF WAN-of-LANs. As mentioned earlier, dual-homed MIS can implement this protection: as depicted in the figure, such core services lie behind a MIS, which filters all access, both from the network and from the facility intranet. In fact, with regard to the latter, note that the Auxiliary services, which belong to the payload, can only interact with the core services via a MIS.

Structurally, SIEM event processing engines deserve special attention, since they are the most data intensive of all core-side, application services. SIEM engines can be integrated or distributed, and its functions can be centralized or decentralised. The MASSIF architecture actually supports any of these variants of SIEM engine implementations, due to the modularity provided by the MIS concept. The content-based information dissemination characteristic yielded by the publish-subscribe paradigm offers an easy way for core-MIS resident services to manage issues like fragmentation and dispatching, parallelism and replication, depending on the way SIEM engines are for example distributed or replicated by different facilities or data centres.

4. Functional View

The global information flow in the MASSIF architecture is depicted in Figure 4. Information is captured at the edges, as shown earlier in the structural view of the system (Figure 3), processed in the Generic Event Translation modules which perform event collection, aggregation and normalisation, and then disseminated to the application services. The Event Processing module (Complex Event Processing Engine, CEP), performs correlation amongst related events from the raw event flow coming from the periphery and stores both raw events and correlated events in the Repository. Note that event processing is enhanced by pre-correlation that already happens in the Generic Event Translation module. The Repository allows indirect communication with the other application modules. The Model Management services perform modelling and simulation, and further generate additional syndromes: threat models and security alerts, which are fed back to the repository. Finally, the Decision Support and Reaction (DS&R) service analyses both the original event digests and the threat model digests and security alerts, and triggers reaction and adaptation measures, translated for example in modified policies, which are sent back to the edge, to the DS&R agents, and affect the sensing and periphery event processing modules.

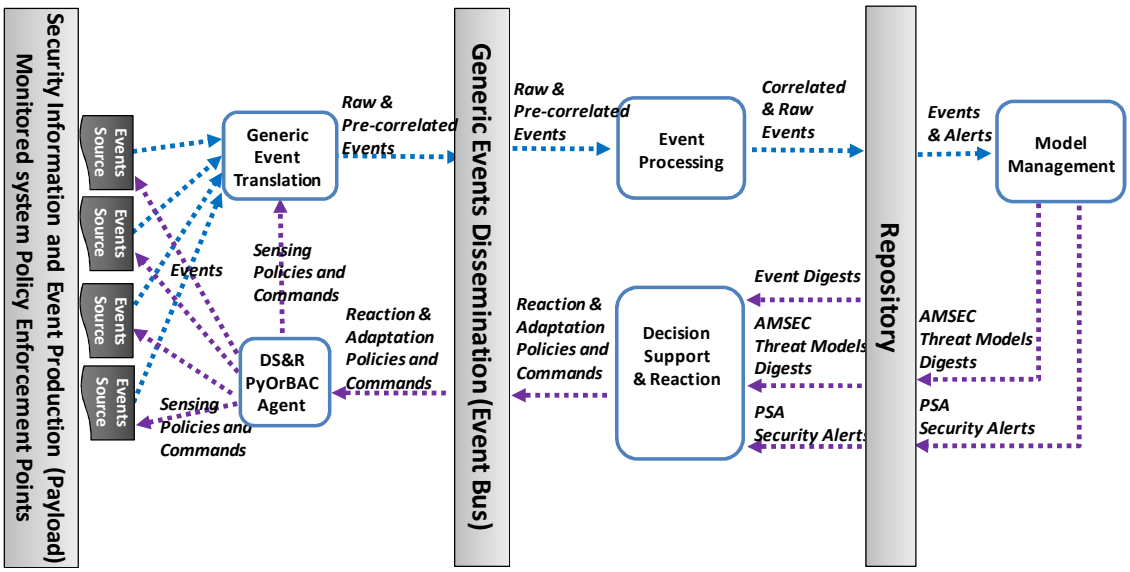


Figure 4 - Global Information Flow in the MASSIF Architecture

In the following sections, the functional aspects of each service module will be explained in detail.

4.1 Data Services

4.1.1 Event Collection, Aggregation and Normalisation

The main purpose of the MASSIF data services is to deliver security information flows to the MASSIF core. In order to do so, this layer must provide the functionalities for the collection, aggregation and normalization of the events generated by the payload machinery and use the MASSIF Infrastructure level services to provide the relevant security data to MASSIF applications by exploiting the services offered by the MASSIF resilient framework. When required (e.g., in outsourced SIEM systems), this is also the place in the architecture to perform anonymisation.

The Data layer must be able to handle a wide set of security-related data formats, generated by the security event sources deployed in the MASSIF use cases. Thus it provides the functionalities for the translation from these data formats to the unified MASSIF *generic event* format which is tractable by the MASSIF applications. It also provides the functionalities for time-stamping and disseminating translated events through the MASSIF infrastructure to the MASSIF Application layer, using the Resilient Event Bus.

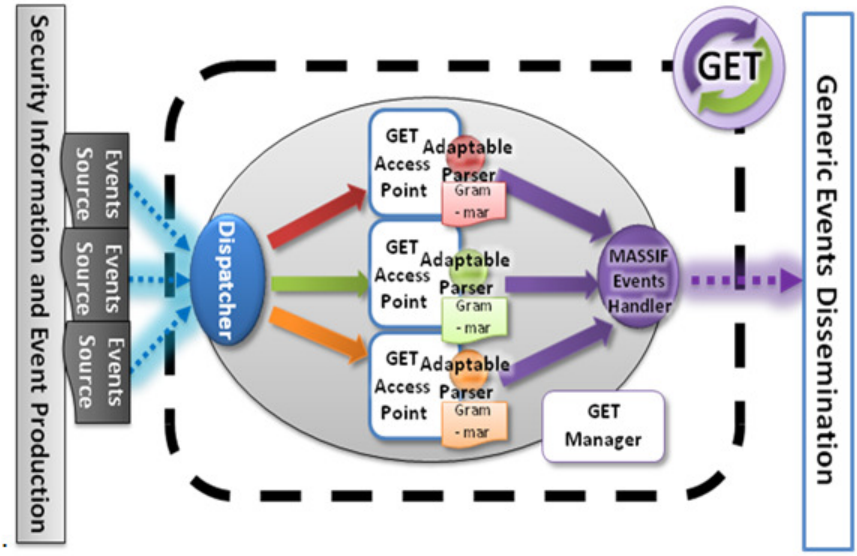


Figure 5 - GET data flow diagram

In order to perform these tasks, the data layer includes the following three functionalities:

- *Collection* of raw events from the Payload Machinery. All the raw events produced by the security related sensors in the monitored systems are delivered to the data layer, which allows the security data to enter the MASSIF SIEM platform (see the block diagram in Figure 2). Raw events are collected from the data sources, which may pertain to different layers of abstraction and follow different formats, and they are transferred to the data layer in a textual form through the appropriate protocols (e.g., syslog). This diversity is suggested by the colour coding of the different flows from the Dispatcher.
- *Aggregation* of closely related events. In some cases the same real world event can generate many redundant computer events, which carry few or no additional information. In these cases events need to be aggregated before being delivered to the upper levels, in order to avoid repeated notifications and also to prevent flood of events that may clutter the dissemination and processing elements of the MASSIF framework;
- *Normalisation* of all events and information to the MASSIF *generic event* format. The MASSIF SIEM is called to deal with highly heterogeneous types of events and information. The collected events go through a normalisation process that converts them to a common and generic representation format hiding this heterogeneity (this normalisation is suggested by the uniform colour of the flows converging on the Events Handler after processing). In fact, any event message exchanged between MASSIF modules follows this common format. This allows the SIEM platform to transparently manage all the different data, whatever its source.

In order to implement these translation functionalities, the abovementioned data layer services are organised in a framework for event translation named Generic Event Translation (GET), whose main components are represented in Figure 5. For the extraction of the relevant data fields from the different types of events, the GET framework relies on a component called Adaptable Parser [8] , [9] . GET is a dynamically reconfigurable framework that allows collection and identification of the events from

different sources. It acts as wrapper for the functionalities of the adaptable parsers in order to make their usage and management more efficient.

The translation process within the GET framework is organized as follows: the events are collected from their source, the input format is recognized, the event is parsed by the adaptable parsers, the relevant data from the event is converted to the MASSIF event format, and finally the MASSIF event is time-stamped and forwarded to application layer, through the event layer generic events dissemination services. The GET framework has a modular architecture, whose operation is coordinated by the GET Manager.

The entry point for the events generated by the payload machineries is the Event Dispatcher. The Dispatcher connects events from different sources and formats to the appropriate adaptable parser, through a GET Access Point (GAP). The GET Manager creates GAP instances on demand, to handle new sources. The Manager also supervises the dynamic activation, deactivation, and update of the Adaptable Parsers, which implement the actual grammar-based parsing functions. After parsing and extracting the relevant information from a specific event format, the MASSIF Events Handler (MEH) deals with the final conversion into the MASSIF Generic Event format, and the dispatching to the event layer, so that events can be sent to the application layer, where the SIEM core services reside.

4.1.2 Pre-correlation

The MASSIF SIEM architecture features early correlation mechanisms at the periphery, on the edge side, which we call *pre-correlation*. We believe that it makes sense to try and recognize security-relevant patterns which may be symptoms of malicious or anomalous activities perpetrated over *a zone* of the target distributed system --- both on events belonging to the same architectural layer (intra-layer correlation) and on events belonging to distinct architectural layers (cross-layer correlation), but taking advantage of the natural topological affinity of events gathered in a same zone or subsystem of the target (e.g., a VLAN, a DMZ, etc.).

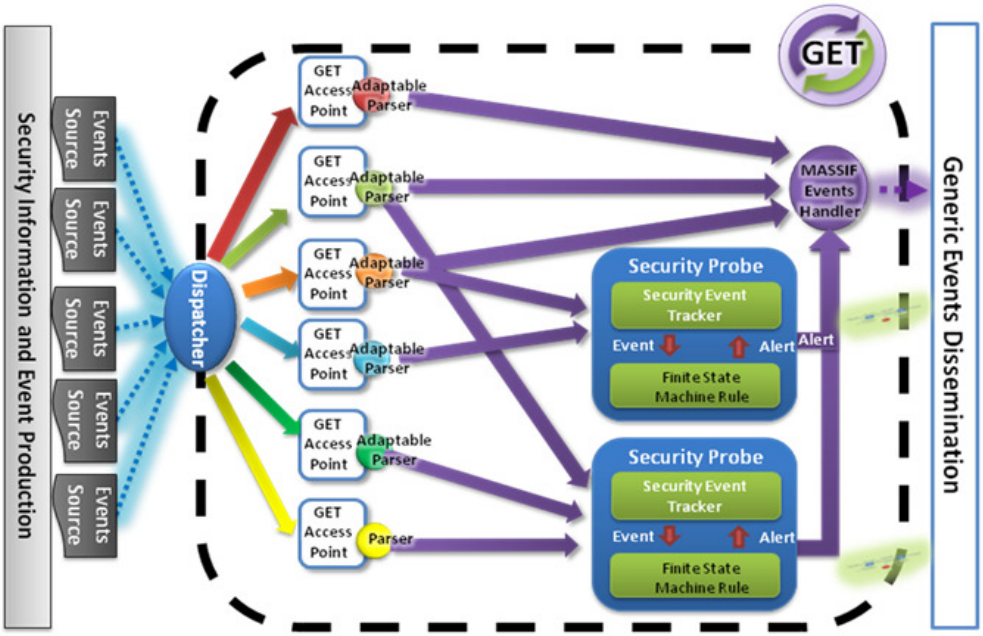


Figure 6 - GET framework enhanced with pre-correlation Security Probes

The component in charge of pre-correlation is called a Security Probe (SP). A Security Probe is capable of including new event classification rules and using them in order to improve the accuracy of the detection function. SP modules are deployed within a Generic Event Translation (GET) framework instantiation, as shown in Figure 6 and, as such, they reside in edge-side MASSIF Information Switches (MIS). There may be more than one SP within the same GET framework. Besides processing events to be directly disseminated to the upper layers, the GET framework performs the necessary upstream data processing for the SPs. A Security Probe is selectively fed by the output of parser components of the GET framework. Besides the generic adaptable parsers, there may be fixed parsers specially designed to fit the SP needs. The parser outputs may be configured to go exclusively to the SP or to the application layer correlation engines, or to both.

SPs, upon detection of positive correlations, act by generating alert events, which are created directly in the MASSIF generic event format, so that no further translation process is required. These alert events are sent to the application layer through the MASSIF Event Handler of the GET framework. These alert messages receive a trusted timestamp, so that they can be meaningfully correlated with other events, e.g., in the core application layer services. As a matter of fact, a typical consumer of the alert messages is the MASSIF Processing Engine. The Security Probe is essentially an event detector. As shown in Figure 6, the Security Probe uses the information extracted by the parsers to identify anomalous service events or patterns: the event detector's engine is based on Finite State Machine modules built from predefined service rules.

In order to achieve such objective, a dedicated software module, the Security Event Tracker, is in charge of identifying specific events occurring in the infrastructure based on the information extracted by the Data Parsers. For example in case of service pattern recognition, the normal infrastructure behaviour is modeled through the Finite State Machines (FSMs). Whenever an unauthorized transition from one service status to another one is detected by the Security Event Tracker, an alert is generated.

It's worth highlighting here the main differences between the GET and the combined GET and SP event flows. The GET framework agnostically translates the input generated by the monitoring sensors, integrating at run-time new and heterogeneous event sources, by dynamically and seamlessly reconfiguring and deploying adaptable parsers. It is also in charge of performing a normalization process of the pre-existing event sources. The Security Probes (SP) have been conceived to add some intelligence to the information flow generated by GET, by spotting and reporting anomalous events based on predefined sets of security patterns expressed by means of Finite State Machine rules. Moreover its output messages do not require any additional normalisation.

4.1.3 Reaction and Adaptation

As discussed in Section 4.3.3 ahead, the Decision Support and Reaction (DS&R) application module may issue decisions about pending threats, part of which imply sending down to the edge, the necessary reaction and adaptation commands, to be effected on MASSIF smart sensors, or on native sensors and event sources of the monitored payload machinery.

These hooks for configuration and reconfiguration of the sensing policies are called Policy Enforcement Points (PEP), and they are implemented by a set of DS&R agents, as suggested in Figure 7, which ensure the configuration of the associated components' policies. So, every time a new component needs to be managed, a specific agent will be created based on the component's need.

DS&R agents aim both at reconfiguring Policy Enforcement Points (PEP) components and at providing assistance to manage contextual information that is not natively handled by the PEP. PEP is used herein as a generic term for components within an IT infrastructure that act as gateways for requests, and can alter them; examples of such components are firewalls, intrusion detection sensors, anti-virus, web application gateway, LDAP directories or radius servers. The DS&R agent subscribes to instructions from the DS&R application module, published in the form of MASSIF generic events, as depicted in Figure 7. Then, it takes the request and configures the PEP according with the policies

specified. The DS&R agent is able to detect context changes and define the new set of policies, in order to apply the required configurations.

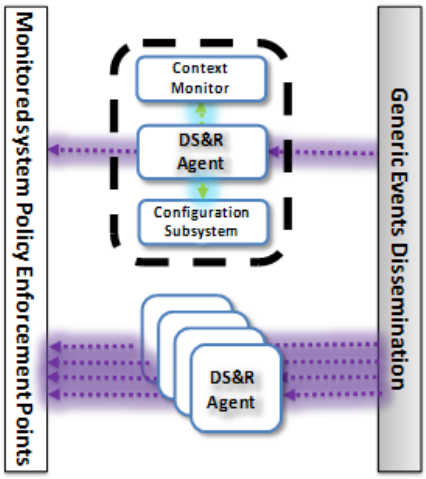


Figure 7 - Decision Support and Reaction (DS&R) Agent: Reaction and Adaptation

Decision Support and Reaction agent architecture

The DS&R agent resides either on an edge-MIS or on a MIA co-located with payload components. Its goal is to execute the policies dictated by the core DS&R module, adapting the monitored system's security policy (and thus its configuration) to the detected threats. It will implement any functions needed to adapt to new circumstances or react to threats by reconfiguring the sensing apparatus and payload system's security policy.

The edge-MIS solution applies when the Policy Enforcement Points (PEP) is a closed environment and has a channel with sufficient QoS to the edge-MIS. The MIA solution applies in the case where it makes sense to have local intelligence and/or it is made possible by those PEP component's openness.

In the edge-MIS installation, the DS&R agent: (1) effects new configurations on MIS-local smart sensors (receiving directly from remote payload native sensors or event emitters); or (2) it can remotely effect the configuration of payload components PEP (the very native sensors or event sources/emitters). In the MIA installation, the DS&R agent: (3) effects new configurations on smart sensors local to the MIA (receiving locally from local payload native sensors or event emitters); or (4) locally effects the configuration of payload components PEP (the very native sensors or event sources/emitters).

The DS&R agent is actually composed of two sub-components, the configuration agent and the context monitor. Both components are hosted together. The configuration agent receives information from three different sources: the PyOrBAC engine of the core DS&R module[2], in order to create organizations and subjects; the context monitor, in order to implement the policies; the administrator, through an API that allows the user to set the mapping information between the PyOrBAC and the PEP elements. This agent translates the policies from the PyOrBAC format to the set of instructions needed to apply and execute the rules. The context monitor is used to manage all the contexts defined in PyOrBAC. It performs this by monitoring the environment to check if the context has changed, in such a case, it informs the configuration agent of the new condition in order to apply the corresponding policy.

[2] PyOrBAC is an OrBAC-based security policy engine implemented in Python, which is the core of the DS&R module, to be discussed further in Section 4.3.3.

Interaction with other MASSIF components

The Generic Event Translation Framework consolidates the MASSIF data services components. It stands in between the security information and event sources of the payload machinery and the MASSIF infrastructure services performing generic events dissemination (i.e., the event layer), feeding the MASSIF application layer. The GET framework is normally deployed in edge-side MASSIF Information Switches (MIS).

A subset of the translation capabilities of the framework can also be made available in MASSIF Information Agents (MIA). MIAs are smart security sensors that are deployed together with important sources of security events. As these smart sensors are specifically developed for the integration of these sources in MASSIF, local translation capabilities make it easier to plug them into the MASSIF framework.

4.2 Infrastructure Services

4.2.1 Generic Events Dissemination

In this section we deal with the functional aspects of the generic events dissemination service, implemented by the Resilient Event Bus (REB). As explained earlier, the resilience aspects will be discussed later in Section 5.3, whereas here we focus on the functional aspects.

The REB performs generic event dissemination towards the services in the core-side of the infrastructure, namely the event processing engine. 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 the Model Management service forms a suspicion that an attack might be in progress, the Decision Support and Reaction service, upon processing this alert, might instruct its agents (see Sections 4.3.3 and 4.1.3) in the edge MIS and/or payload sensors to start collecting more detailed information about the status of the network and specific nodes.

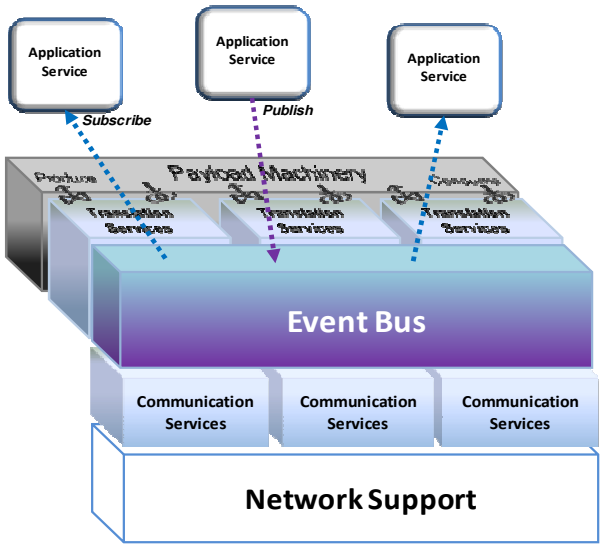


Figure 8 - Resilient Event Bus architecture

Figure 8 offers a detailed view of the Resilient Event Bus internal architecture, making its implementation clearer. REB aims at providing a 'generic events' abstraction where events published from different origins coalesce on the layer, are temporally ordered, and made available to the subscribers.

One of the key factors to achieve this goal is that events follow a common syntax and semantics, whatever their origin, the MASSIF *generic event* format mentioned earlier. Another key factor is the availability of trusted timestamping, i.e., that event timestamps must be globally meaningful, that is, clocks at the relevant end-points must be globally synchronised to an adequate precision, and trustworthy. A third and final key factor is that events are treated seamlessly, whatever their origin or destination: (i) events produced or consumed by the payload machinery (outgoing events and information; incoming commands); or (ii) events produced or consumed by the SIEM machinery (incoming events and information; outgoing commands).

The unusual L-shaped structure of the REB architecture guarantees these objectives, by making a seamless connection between the external system, the monitored payload machinery, on one leg, and the internal MASSIF SIEM system, on the other leg.

Recall that the payload machinery contains the raw security information and event sources. This alien sensory information of diverse origins is captured and undergoes a *translation*, normalizing it into generic events understood by the MASSIF SIEM system, which are fed into the Resilient Event Bus. On the other hand, Application services are also capable of producing native MASSIF generic events, which for example convey notifications or commands, also fed to the REB. Finally, pre-correlation modules at the edge are also capable of synthesizing events to be fed to the REB. These events of several natures and origins coalesce on the bus and are then published to eventual subscribers (actually, MIS nodes).

The event bus layer publishes events following a common syntax and semantics, whatever their origin and direction, 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. Some of these events may possibly undergo a reverse translation and be passed-on to environment machinery, e.g. MIA, by the subscribing edge-MIS. This is the way we foresee commands or notifications getting ultimately to payload devices in a uniform way.

Mapping the specific REB architecture onto the MASSIF layers, Translation services map onto some of the Data layer services, such as collection, aggregation and normalisation. In consequence, this part of the REB functionality will be implemented in MIS (general case), in MIA (smart sensor case), or split between both. The Communication services are concerned with the MASSIF protocols responsible for the actual propagation of events via the regular Network Support system (the WAN-of-LANS mesh), ensuring they reliably arrive at all intended destinations. This part of the REB functionality will be implemented in MIS. Ordering and synchronisation mechanisms create the event bus abstraction in all MIS units for the intended subscribers. The event bus obviously maps onto the Event layer of the general MASSIF diagram.

4.2.2 Secure Communication

We saw that the set of MIS form the distributed infrastructural overlay superimposed over the payload. This overlay is actually implemented by a dedicated resilient communication service. The communication service is implemented by protocols running amongst the MIS. These baseline protocols guarantee that this service is resilient both to accidental and malicious (or Byzantine) faults, given the fault/attack model outlined for MASSIF (see Section 2.2). Additionally, they establish concurrent routes between MIS, to overcome severe threat scenarios like overload or denial of service (DoS), by achieving routing resilience. Communication protocols can benefit from the foreseen asymmetry between upstream and downstream communication, 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. The communication service also serves additional purposes: to send down-stream commands; to effect control communications between peer MIS.

4.3 Application Services

This section describes the several Application Service modules. Figure 9 gives a high-level view of the logical interactions between these modules, which will be detailed below.

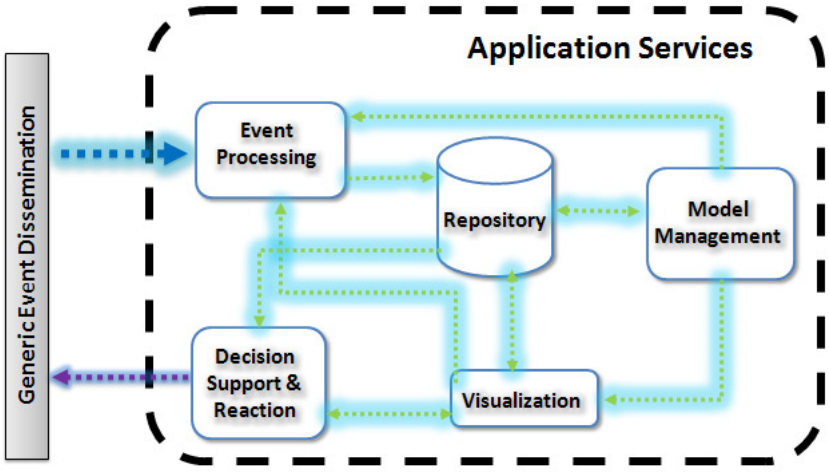


Figure 9 - Overview of the Application Service modules and interactions

The application layer services use the event dissemination infrastructure to get event information from the edge, and also to send back events (e.g. reaction and reconfiguration) to the edge. Once inside the Application Layer, application services rely on the event processing engine to process all incoming events and generate alarms. Input and output events are also persisted in the event repository, part of the general repository module depicted in Figure 9, as an alternative means for application service components to manipulate events, as well as to enable historical forensic analysis of events. Therefore, the different application components may choose between using a streaming interface and/or the data store API to get access to events. Whenever required, communication between application modules can also be made through other direct communication protocols suited for the purpose. The current architecture configuration favours indirect communication through the repository as depicted in Figure 9.

4.3.1 Event Processing

Processing of events in the MASSIF SIEM is performed by a highly-scalable, elastic correlation engine [1] . The latter is materialized as a parallel Complex Event Processing (CEP) system that is capable of (i) aggregating the computing power of a large cluster to process massive amounts of events per second and (ii) adjusting the number of allocated resources to the real input load.

The behavior of the engine can be extensively customized through CEP queries (CEP queries are transparently generated from user-defined standard SIEM directives, designed e.g., with OSSIM) that define how to abstract, transform, aggregate and correlate input events. A CEP query is composed of a number of operators, defined in [2] .

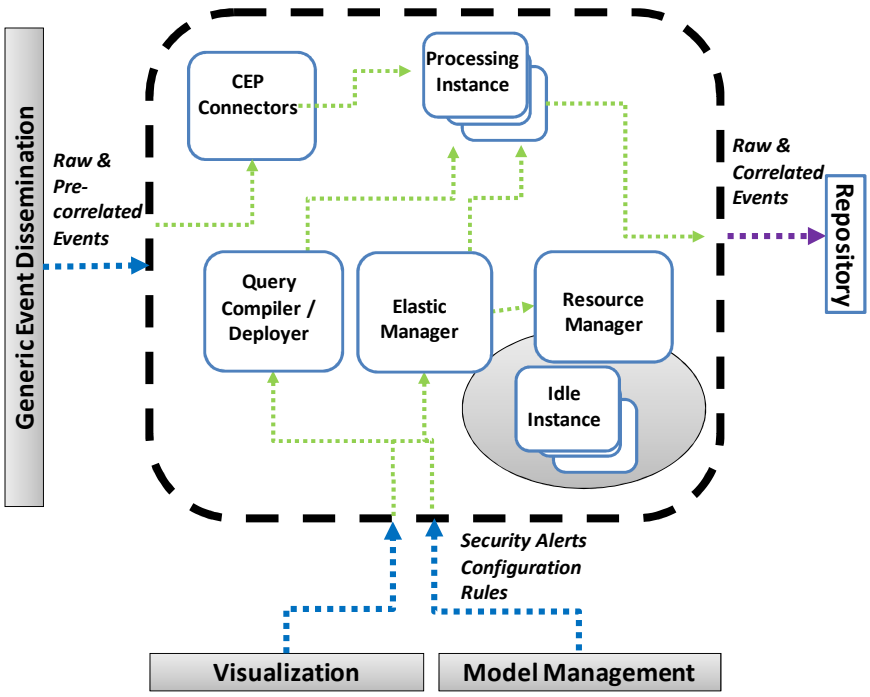


Figure 10 - Event Processing: Correlation Engine overview

The internal architecture of the engine and its interaction with the other components of the MASSIF SIEM are shown in Figure 10. The engine is characterized by a number of processing instances arranged in a sequence of subclusters. All processing instances of a subcluster run the same portion of the CEP query, called subquery (efficient techniques to partition a query into subqueries are discussed in [1]), receiving input events from the previous subcluster and feeding output events to the following subcluster. The first subcluster of the sequence receives input events directly from the Resilient Event Bus, with the use of the CEP connectors. Output events of the last subcluster are available both through a streaming API to other application modules and to the MASSIF Repository. Partial results from intermediate subclusters are also available.

The Elastic Manager monitors the status of each processing instance and adjusts the size of a subcluster (e.g., adding or removing instances) according to its current input load. Adding or decommissioning processing instances requires the Elastic Manager to interact with the Resource Manager, which keeps a pool of available instances. At any time, the Elastic Manager can also re-distribute the load across the processing instances currently allocated to the subcluster. Efficient techniques to add/remove processing instances or to re-distribute the input load are discussed in [3] .

Finally, the Query Compiler/Deployer receives a standard SIEM directive as input, either from the visualization component interface or the Model Management component, translates it to a CEP query, partitions it into subqueries and deploys each subquery to a subcluster.

Interaction with other MASSIF components

Input events come from the edge services through the resilient event bus and are injected into the correlation engine via CEP connectors. The output of the CEP queries is consumed by the modelling, reaction, and visualization services. Currently, they can be consumed via the event API. However, an alternative method will be provided to access the data by storing them in the repository based on a highly scalable cloud data store.

4.3.2 Model Management

The Model Management block is composed of two modules, described below: the Predictive Security Analyser (PSA), and the Attack Modeling and Security Evaluation Component (AMSEC).

Predictive Security Analyser

The Predictive Security Analyser (PSA) provides advanced, application aware security monitoring capabilities to the MASSIF SIEM. Specifically, it supports *close-future process behavior simulation* and *prediction of possible security violations*. Its block diagram is depicted in Figure 11. The quality of the performed analysis strongly depends on the quality and granularity of the process description as well as on the appropriate security event specifications.

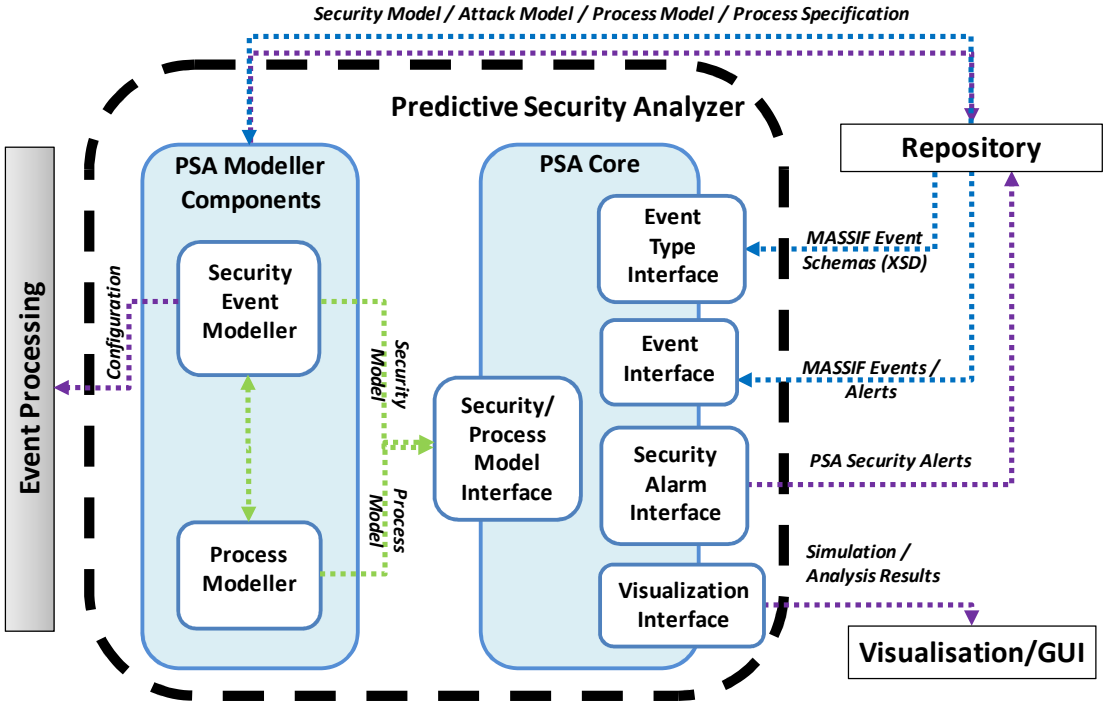


Figure 11 - Predictive Security Analyser Component (PSA)

Security event modeller and *process modeller.* Prior to the start of the engine, the process description and security goals/events will be transformed into PSA understandable models (Asynchronous Product Automata (APA)), which are going to be used for the continuous real-time analysis and close-future simulation. This will be done in the *security event modeller* and the *process modeller* components. They will communicate with the *model repository*, which contains the *attack models* of the AMSEC tool and the previous models composed by the modellers. The *security model* and *process model interfaces* will provide access to the repository for the PSA engine. The interpreted models will be imported into the PSA in the initialization phase.

The PSA modeller components support the *security requirements elicitation*, the *specification of a simulation model*, and the *development of monitoring rules (CEP queries)*. High level security goals, security requirements, monitoring rules, the developed specifications and the relations between them will be stored in the MASSIF repository. These relations will enable correlation of PSA alarms with high level security goals and security requirements. Furthermore, asset descriptions and event formats, which are stored in the Repository, are needed in order to associate asset information with events received and alarms transmitted by the PSA via the repository.

Interaction with other MASSIF components

In order to assure the seamless interaction with other components of the MASSIF SIEM, the PSA supports a number of external interfaces.

- *PSA interaction with the repository*

The MASSIF repository is the main data source and information exchange interface between the PSA and other MASSIF components.

The *event type interface* is a point of interaction between the PSA and the event type registry. This registry manages a database of XSD event schemas, which defines the format and the content of all events processed by the PSA. The event schema database is provided by the MASSIF repository. The *event interface* communicates through the MASSIF repository with the event processing engine. It will feed the PSA (low-/high-level) events and alerts. The seamless interpretation of these events within the engine relies on the content of the database of XSD event schemas in the MASSIF repository.

The *security alarm interface* delivers the identified current or close-future security violations to the MASSIF repository in the form of PSA alarms. This notifies other MASSIF components about detected critical situations in order to enable immediate/proactive actions against the upcoming security threats based on the provided information.

Other MASSIF components, such as AMSEC, deliver their results via the repository to the PSA. This will enable the PSA to incorporate these results into the simulation process and to adapt the simulation strategy.

- *PSA interaction with the event processing component*

Correlation rules identified by the PSA modeller components have to be delivered to the Event Processing component in order to enable the activation of the developed monitoring rules with events delivered by the generic event dissemination.

- *PSA interaction with the visualization interface*

Results of the multi-level predictive security monitoring performed by the PSA, such as *current or close-future security violations*, can be presented to the security officer through the MASSIF visualization component in order to facilitate decision making and choice of countermeasures.

Attack Modeling and Security Evaluation

The Attack Modeling and Security Evaluation Component (AMSEC) is intended to complement the direct analysis functionality of the SIEM system, by providing the architecture with the capability of attack modeling and security evaluation [6] .

The main inputs are: configuration of the computer network (or the system), security policy for the computer network (or the system) determining a set of permissions or policy rules, event and alerts in the computer network (or the system), external databases (DBs) of vulnerabilities, attacks, platform, etc., possible malefactor profiles (as a set of malefactor characteristics), required values of security metrics (as a set of requirements to security).

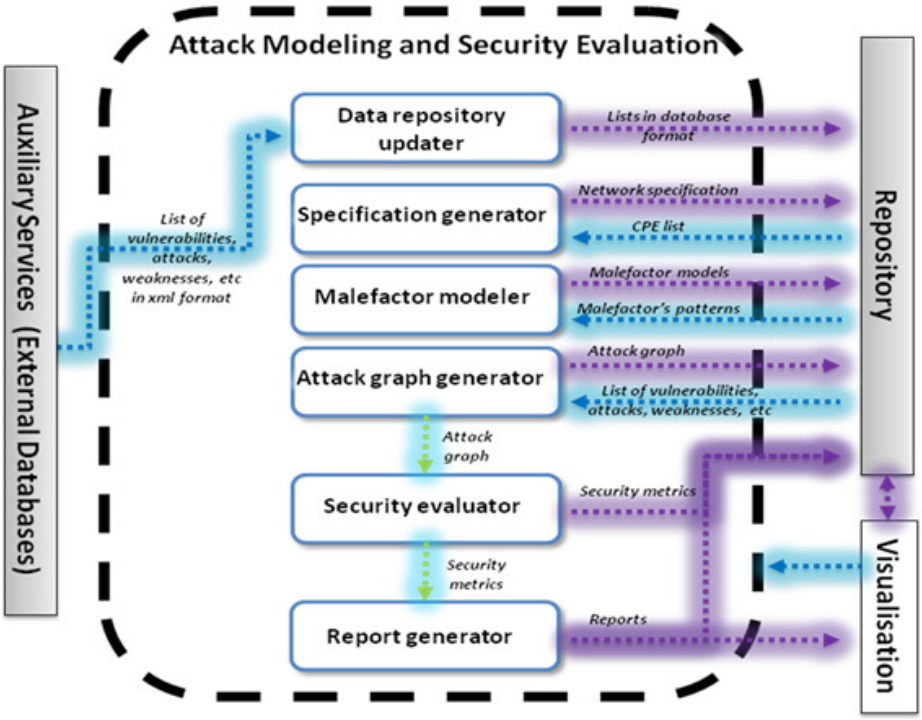


Figure 12 - Attack Modelling and Security Evaluation Component (AMSEC)

The main results are as follows: vulnerabilities detected; possible routes (graphs) of attacks and attack goals; payload internal dependencies; bottlenecks ("weak places") in network security; adjusted attack trees based on changes in the network; predictions of the intruder's next steps taking into account the current situation; security metrics, which can be used for general security level evaluation of computer network (system) and its components; attack and countermeasures impacts; guidelines for increasing the security level and solutions based on security measures/policies/tools.

The AMSEC operates in two main modes [6] : (1) Design time (or configuration) stage, where AMSEC is used for design and initial analysis of the network analyzed (or the system under protection). It is a non real-time mode; (2) Exploitation stage, where AMSEC is used for real-time or near real-time operation in the framework of the SIEM system.

The general architecture of AMSEC and its interaction with other components of MASSIF SIEM are shown in Figure 12. Connections, depicted in the figure, show the direction of interactions between different components.

- The *Data repository updater* downloads the open databases of vulnerabilities, attacks, configuration, weaknesses, platforms, and countermeasures from the external environment (sending requests to external databases for updates and communicating with data sources).
- The *Specification generator* (SG) converts the information about network events, configuration and security policy, from other MASSIF SIEM components or from users, into an internal representation.
- The *Malefactor modeler* (MM) determines malefactors' individual characteristics, skill level, their initial position (insider/outsider, available points of entry, etc.), the set of permissions, possible actions/attacks already fulfilled (which can be predicted according to events and alerts) and knowledge about the analyzed network.
- The *Attack graph generator* (AGG) builds attack graphs (or trees) by modeling sequences of malefactor's attack actions in the analyzed computer network using information about available

attack actions of different types, services dependencies, network configuration and used security policy. Attack graph generator can also build attack traces taking into account zero-day vulnerabilities – unknown vulnerabilities which are required to compromise network assets.

- The *Security evaluator* (SE) assists the selection of solutions (validated events and alerts, possible future security events, countermeasures) needed for other MASSIF SIEM components. It simulates stochastically multi-step attacks and studies the cost and effect of various countermeasures. For example, it generates combined objects and calculates their security metrics in order to evaluate the common security level and possibly make recommendations on strengthening it.
- The *Reports generator* shows vulnerabilities detected by AMSEC, represents "weak" places, generates recommendations on strengthening the security level and depicts other relevant security information.

Interaction with other MASSIF components

Results and setting of the Attack Modeling and Security Evaluation Component (AMSEC) are presented at and controlled through the Visualization Component. The major part of input and output data flows goes through Repository. Interaction with Visualization and Repository components takes place at all stages and modes of AMSEC operation.

At each stage of functioning (Design (configuration) and Exploitation) AMSEC interacts with different MASSIF components.

- *Design (configuration) stage*

AMSEC needs to have a detailed description of protected network topology and configuration for correct and efficient operation. This information is retrieved from the user (through the Visualization system), from predefined data (through Repository) and from sensors placed in the network (through the Resilient Event Bus or the Event Processing module). As a result, AMSEC produces attack graphs and calculates security metrics.

Attack graphs can be used to refine Event Processing rules, and security metrics can be transferred to the Decision Support and Reaction (DS&R) component to form the list of recommendations to increase the security level. Since at this stage real-time mode is not required, the information flow can go through the Repository.

- *Exploitation stage*

There are several tasks performed by AMSEC at this stage: (1) attack graphs adjustment; (2) attack detection improvement by searching matches between real-time events and attack graphs; (3) security metrics evaluation and prediction of potential threats and attacks.

AMSEC needs interaction with other components to fulfill these tasks.

- Task 1 (attack graphs adjustment): AMSEC needs information about changes in controlled network. The Event Processing (through the Repository) is the source of this information. Thus, information on network changes comes to SG and MM that make changes in the network and malefactor models stored in Repository. After that AGG and SE recalculate stored attack graphs and security metrics basing on updated models.
- Task 2 (attack detection improvement). It is assumed, that the Event Processing uses attack graphs to increase the precision of intrusion detection and to detect zero-day vulnerabilities. This data flow is not a direct, because not real (but near real-time) mode of AMSEC. Therefore information flow goes through the Repository.
- Task 3 (security metrics evaluation, threat and attack prediction): in this task AMSEC interacts with the Predictive Security Analyzer (PSA). It should be noted that this information flow is not direct - both AMSEC and PSA are not real-time components, so flow goes through the Repository.

We assume that security metrics, produced by AMSEC, can be used in "on-if-do-why" chain and attack graphs can be applied in the PSA state model. Moreover the determination of malefactor transition through the attack graph can initiate the transition on the state graph and vice versa. The information from the PSA can be also used by AMSEC.

4.3.3 Decision Support and Reaction

The OrBAC model defines the security policy formalism, but it does not specify the way it should be implemented. The Decision Support and Reaction subsystem designs and develops an administrative tool based on the OrBAC model, which allows to consolidate the security policy through the different infrastructure's components in an organization, and to configure automatically those components. This section describes the architecture of PyOrBAC, our implementation of the OrBAC-based Decision Support and Reaction subsystem.

Architecture overview

The proposed architecture follows a client-server model, as shown in Figure 13. The PyOrBAC Engine acts as a server that allows the centralization of the access control policy administration; and a set of agents ensures the policies configuration of the PyOrBAC associated components.

DS&R agents were discussed earlier in Section 4.1.3: these components aim both at reconfiguring Policy Enforcement Points (PEP) components and at providing assistance to manage contextual information that is not natively handled by the PEP. The DS&R agent receives instructions from the PyOrBAC Engine. These flow down the infrastructure from the core to the edge, as depicted in Figure 13.

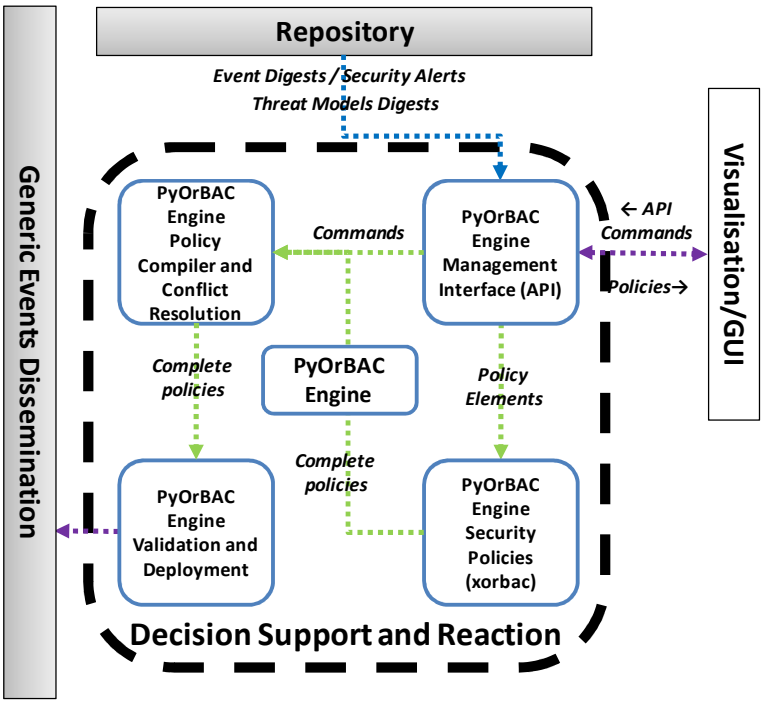


Figure 13 - Decision Support and Reaction Component (DS&R)

PyOrBAC engine architecture

PyOrBAC is an OrBAC-based security policy engine implemented in Python that aims at creating a centralized security policy infrastructure based on the requests received by the administrator. The implemented solution offers the following advantages:

- It allows the security policy configuration of external systems called associated components (e.g., Apache, MySQL, LDAP, etc) from the PyOrBAC engine, which means that administrators do not need to know the configuration rules of other components; they only need to manage the PyOrBAC platform to configure all the security policies.
- The administrator is able to easily identify the existence of conflicts among rules. For instance, it would not be possible for the administrator to detect that one security policy is affecting LDAP and Apache if he had to manually configure each infrastructure. In contrast, by configuring the security policies through PyOrBAC, the system automatically detects and informs the administrator of the existence of a conflict before apply it.
- Security policies are dynamically configured through the use of contexts, which allow the system to react more rapidly to any change (e.g., intrusion attempts, attacks). It is therefore necessary to clearly define the contexts and the monitoring system in order to properly detect the context changes thus allowing PyOrBAC to execute the respective changes in the configuration.
- All the new generated security rules can be applied simultaneously to all the components associated to the organization. For this, PyOrBAC broadcasts the new rules so that all the components change the configuration accordingly.
- PyOrBAC is able to identify pre-existing configurations and save them in its repository so that the engine knows all the security policies of a given organization in order to validate them and detect conflicts. For instance, let us suppose that LDAP and MySQL have already their own security policies, PyOrBAC should be able to detect and store them in its repository so that every time the administrator wants to configure a new policy, PyOrBAC can verify first, that the policy has been already created and second, that the new policy does not create any conflict with other existing policies.

The PyOrBAC engine consists of five modules: Management, Compiler, Security Policies, Validation and Deployment. Further details about the functions of each module will be provided in.[12]

Interaction with other MASSIF components – operations (runtime) view

In operation, the PyOrBAC engine expects to receive alerts in IDMEF format [5] , that is XML messages, over a stream interface. Four MASSIF components have been identified that provide such messages:

- The Generic Event Translation (GET) component provides alerts that are directly usable by the PyOrBAC engine, provided that the appropriate configuration action has taken place (see next subsection), as this represents immediate and current threat information that needs to be mitigated. However, GET components are likely to provide high volumes of information, which feed the Event Processing module for correlation and as such a direct interface between GET and PyOrBAC is not desirable.
- The Event Processing (EP) component provides correlated events and alerts that are stored in the Repository. These are directly usable by the PyOrBAC engine, which gets them from the Repository, provided that the appropriate configuration action has taken place (see next subsection). As the EP engine provides correlated, high-density information, it is the preferred alert stream interface to the PyOrBAC engine.
- The Predictive Security Analyzer (PSA) component provides alerts that are also usable by the PyOrBAC engine. These events are also stored by the PSA in the Repository and made available

to the PyOrBAC engine, provided that the appropriate configuration action has taken place (see next subsection), and that the evaluation of the change of security state takes into account the fact that contexts may be cancelled if the prediction of security state change is invalid. This requires a manual parameterization of the PyOrBAC engine.

- The AMSEC component provides alerts that are usable by the PyOrBAC engine. These events are also stored by the AMSEC in the Repository and made available to the PyOrBAC engine, provided that the appropriate configuration action has taken place (see next subsection), and that the evaluation of the change of attacker posture takes into account the fact that contexts may be cancelled if the prediction of attacker posture change is invalid. This requires a manual parameterization of the PyOrBAC engine.

Interaction with other MASSIF components – configuration view

In order to properly configure the PyOrBAC engine, there is a need for a mapping between certain alert parameters (typically the IDMEF Alert.Classification field) and security policy contexts. The current need is limited to a common dictionary of signature "words", so each of the GET, CEP, PSA and AMSEC modules should include the ability to list the Alert.Classification.text information that they can generate.

In addition, and at a later stage, the countermeasure evaluation component can be coupled with PSA and AMSEC. The objective of this coupling is to shift the configuration of the costs used in the computation of the Return on Response Investment (RORI) index (see [12]) from design time to run-time.

4.3.4 Visualisation

The Visualization component is responsible for displaying, managing and responding to different information (e.g: events, situations, alarms), providing a convenient and effective GUI to interact with some MASSIF components, visualizing data and fulfilling management and administrative tasks.

The main inputs for visualisation are directed from Repository or from application services through the Repository or directly from other MASSIF components: alerts from Decision Support and Reaction (DS&R) component through Repository; security metrics and simulation results, generated by Model Management component (PSA and AMSEC) and available from Repository; settings of AMSEC and PSA components; security events, generated by Event Processing and available from the Repository; access to historical data historical analysis and produce statistical reports.

Besides, it provides a convenient user interface to: to schedule tasks and edit rules for Model Management component and DS&R; to configure security policies and perform repository maintenance; to create and configure security models (event/process models, attack/attacker models, simulation models).

The following architecture of Visualization component (Figure 14) is considered. It can be viewed as a simple two component model which includes *User Interface* and *Control Middleware*.

The *User Interface* is separated from the *Control Middleware* to facilitate the development of different user interfaces (starting with simple command line finishing with rich user interface).

- In the general case, *User Interface* is a simple main window form of an application.
- The Control Middleware consists of two main modules: Plug-ins Manager and Visual Component Controller.
    - *Plug-ins Manager* interacts with other components using plug-in mechanism. It is responsible for registering, managing, including or excluding different plug-ins.

- *Visual Component Controller* is responsible to manage graphical items including start/stop visualization threads (pipelines) on demand of requests both from *Plug-in Manager* and users. *Visual Components* provide a set of graphical primitives (charts, treemaps, graphs, etc) to process input data and render it.

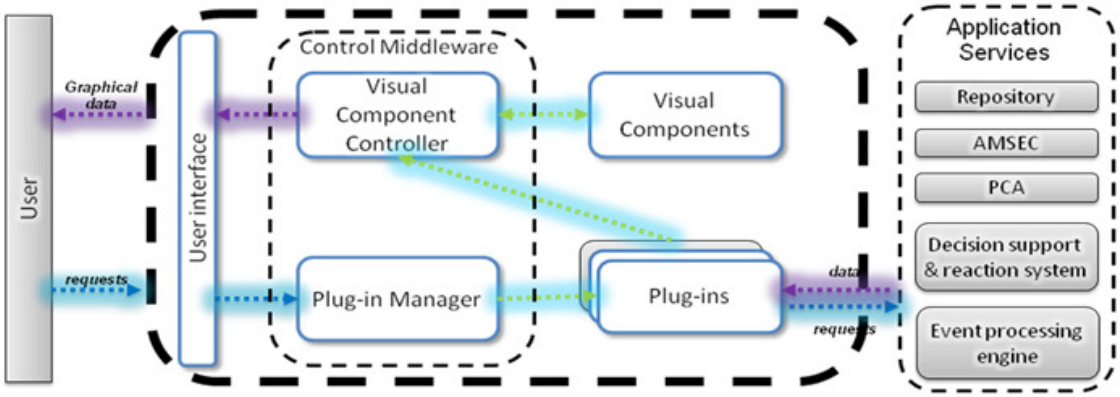


Figure 14 - Visualization Component

The following example illustrates general principle of Visualization Tool functioning. The user via *User Interface* generates the request to some MASSIF component that is managed by the corresponding *plug-in.* The plug-in forwards the request to the MASSIF component and then receives the generated response using the MASSIF component API. The plug-in interface allows this plug-in to communicate with graphical items via the *Visual Components Controller*, selecting the adequate primitive (e.g., a net graph), producing the needed graphical presentation (picture).

4.3.5 Repository

The Repository offers a general storage service, safeguarding data and allowing indirect communication between different MASSIF components. The repository is implemented according to the SOA principles [4] . The advantages of this architecture are the flexibility and loose coupling of components, which provides high scalability and extensibility of the system. Figure 15 shows the general architecture of Repository (CRUD designates basic operations Create, Read, Update, and Delete).

In accordance with the main principles of SOA, the MASSIF repository architecture can be divided into Web services API, Repository services layer and Repository storage layer, as detailed in [4] and summarized here. In addition to these layers, an *access control layer* provides the necessary authorization rights [4] . The Repository manipulates the following types of information: Events; Incidents; Alarms; Decisions; Vulnerabilities; Attack graphs; Users (for RBAC) and others.

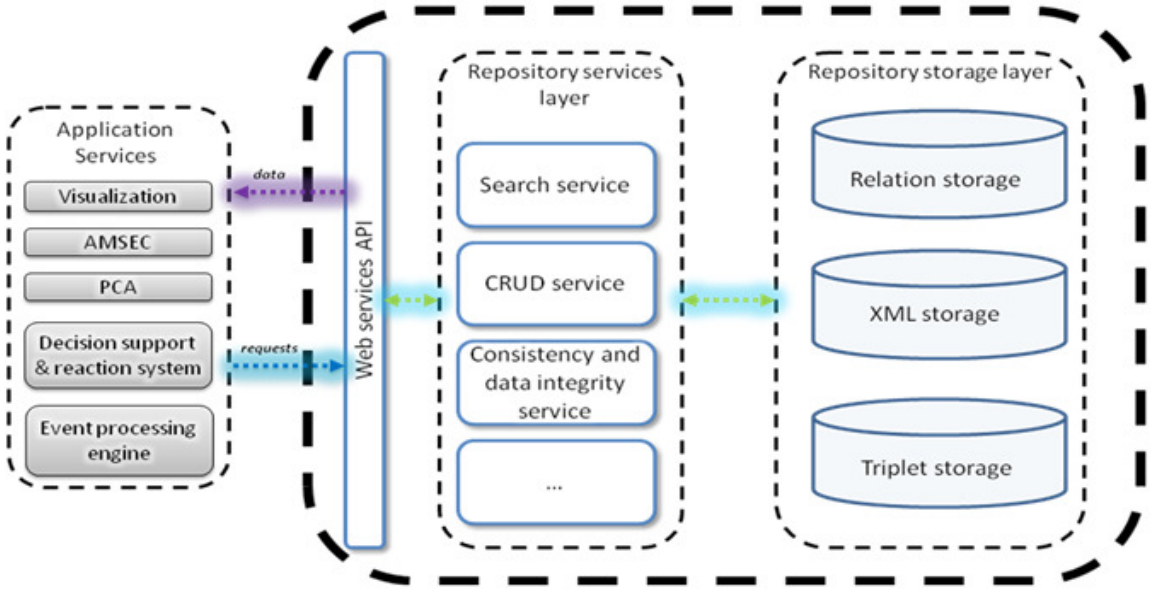


Figure 15 - Repository architecture

The *Web services API* is an interface for interaction with application layer components.

The *Repository services layer* allows abstracting the interaction between one or more objects, workflows and services through an intermediate interface API. It consists of the presentation level and data access level: presentation level covers everything that is related to interaction with the MASSIF components; data access level interprets the queries for data retrieval, received from MASSIF components in the language notation used by the underlying database management system (DBMS).

The *Repository storage layer* includes generic data and data related to different MASSIF application services: event data, AMSEC data, PSA data and DS&R data.

For full support of different information models being developed in the MASSIF, hybrid approach in the repository storage layer is used [4] . This approach combines the possibilities of relational DBMSs, XML-based repositories and triplet stores. Triplet store provides an ontological representation of the data model, and uses the logical reasoning to select the data.

One of the most important components of Repository storage layer is the *event data*. It has very strong scalability requirements due to it should be able to store not only the output events that can be a small load, but also the input events from the edge services that can be a very high load. For this purpose, the event repository will be parallel-distributed and will use a state of the art cloud data store solution providing high scalability. To store AMSEC data we intend to use all three kinds of data representation (relational, XML-based and triplets). Predictive Security Analyzer data and Decision Support and Reaction data will be stored in relational and XML-based stores.

5. Resilience Mechanisms

This section gives an overview of the resilience aspects of the MASSIF architecture. As such, we discuss the motivation, requirements and mechanisms behind one of the objectives of this architecture: to provide seamless integration of resilience into distributed SIEM systems, with the aim 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.

On the other hand, a diagnosis of the shortcomings of current SIEM systems, which led in part to the proposal of the MASSIF architecture, 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.
- lack of reaction capabilities (current SIEMs being “detect only”, it is difficult to take action on the information they provide);

This scenario points to three main issues:

- the monitored environments are increasingly exposed to threats, rendering the monitoring task more complex;
- this dramatically increases the dependence on the monitoring systems to ensure secure and dependable operation of the monitored systems in real-time;.
- the monitoring systems become a target of attack themselves, being prone to different sorts of failures.

Since the SIEM subsystems that perform event collection, delivery and processing have today a highly distributed nature and operate in essentially the same environments as the monitored systems, they can also become targets of attacks and accidental faults. These problems are expected to become even more prevalent with the increasing inconspicuousness of computing systems and networks, and as security information and events start to concern not only common IT devices (e.g., firewalls, routers, application servers) but also critical information infrastructures, which for instance observe and control physical processes (e.g., a dam or a power plant).

Therefore, it is important to improve the trustworthiness of SIEM systems by developing appropriate solutions to achieve resilience. 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 resilience. 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 [18] .
- Proposition 2: *Promote automatic control of macroscopic information flows*
    - There is a 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 monitored 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 monitoring 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 engines (in the core) and vice-versa.

The intent of this section is to describe the main techniques that are being explored in MASSIF to improve its resilience. 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 [18] [19] . 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.

Given that, as we have assumed earlier, different Facilities/Networks of the payload and the SIEM system 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 trade-off with performance, cost, or complexity.

5.1 Attack Vectors

This section analyses the susceptibility of the MASSIF architecture to faults and attacks, some of which of possibly large and/or uncertain magnitude. It is interesting to start by analysing what are the potential attack vectors, put in context with the MASSIF architecture, as depicted in Figure 16.

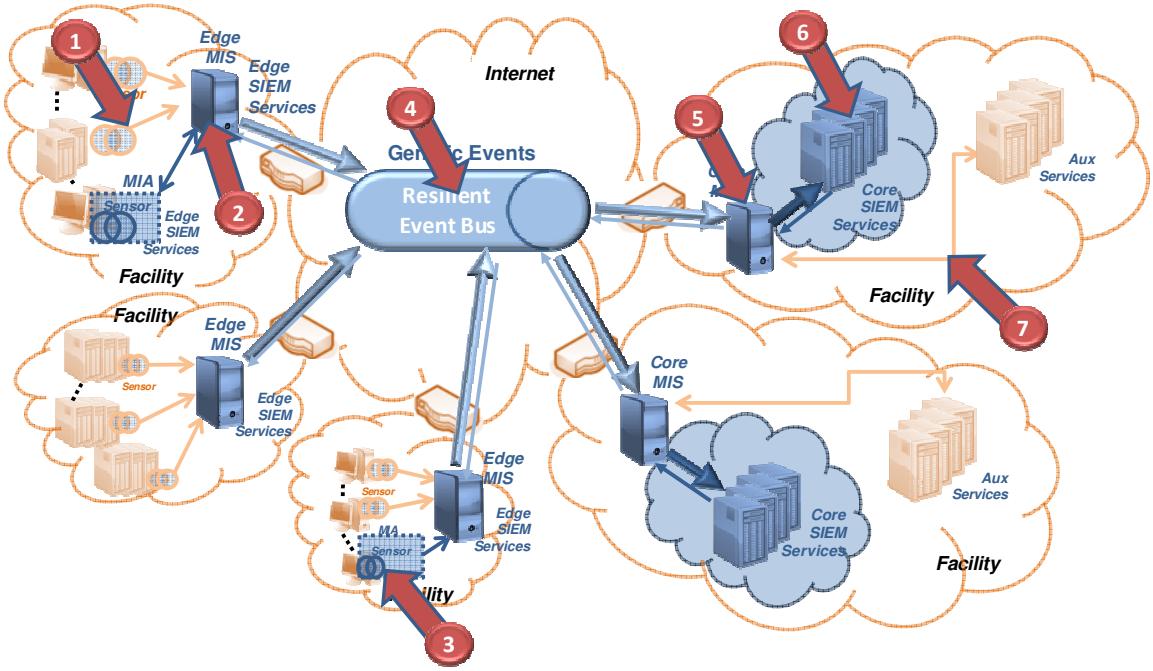


Figure 16 - Estimated attack vectors to the MASSIF SIEM architecture

As shown in the figure (illustrated by arrows), in such a distributed and large-scale architecture, there are obviously several attack vectors:

- *sensing flow integrity*, which typically uses standard protocols (arrow 1) --- e.g., tampering with the standard protocols conveying information (e.g., SYSLOG) from devices to the edge-MIS: interrupting, delaying, re-ordering, replaying, forging, etc.;

- *edge-MIS*, targeting its availability and/or the integrity of event collection and/or communications (arrow 2) --- e.g., disruption (DoS) or penetration attacks on edge-MIS: SIEM services and/or communication protocols;
- *MIA*, with the objective of attacking its availability and/or the integrity of remote event collection and/or MIA-to-MIS communications (arrow 3) --- e.g., disruption (DoS) or penetration attacks on device-resident MIA: SIEM services and/or communication protocols;
- *Event Bus*, targeting its confidentiality, integrity and availability (arrow 4) --- e.g., tampering with the MASSIF protocols conveying information between MISs: interrupting, delaying, re-ordering, replaying, forging, etc.;
- *core-MIS*, aiming at attacking its availability and/or the integrity of the protection service and/or the communications (arrow 5) --- e.g., disruption (DoS) or penetration attacks on core-MIS: SIEM services and/or communication protocols;
- *core systems*, targeting their availability and/or integrity (arrow 6) --- e.g., disruption (DoS) or penetration attacks on core services (SIEM Engine, Historian, GUI, etc.);
- *auxiliary services*, targeting integrity of interactions (arrow 7) --- e.g., disruption (DoS) or penetration attacks on auxiliary services.

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 middleware software that will offer primitives for the development of specific SIEM services. In this section, we discuss how to achieve integrity of the key component MASSIF Information Switch (MIS), and then we consider two important aspects of the middleware, namely the support for communication and event dissemination and storage of information.

5.2 Incremental MIS Resilience

In this section, we discuss techniques to improve the resilience of specific nodes of the architecture, such as the edge and core-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. It is also bound to have the most sophisticated resilience mechanisms, since it protects key core servers.

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:

- *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);
- *Loosely coupled duplex or N-plex*: the service is replicated in two or more machines loosely coupled in the network;

- *Closely coupled N-plex*: the service is replicated on N machines connected with a private broadcast network;
- *Tightly coupled N-plex*: the service is replicated in a virtualized node running N diverse guest operating systems (to prevent common vulnerabilities);
- *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).

Resilience of the time related mechanisms is also of great interest, since the MASSIF global time base is implemented by the MIS/MIA. Data-layer services implemented in the edge-MIS/MIA, as well as the Even-layer services, have access to local clocks which are collectively synchronized, providing a global and trustworthy notion of time.

Details of implementation are outside the scope of this paper, but this can be achieved either through the use of external synchronization to an absolute reference (e.g., GPS time synchronization) or through internal synchronization using clock synchronization protocols such as the Network Time Protocol (NTP), highly-fault-tolerant versions existing as well. One important implication of assuming the existence of a global and trustworthy notion of time is that time stamps can be used, in general, to infer about the timeliness of events and about their ordering, which is very important in the context of MASSIF, since it allows faithful correlation of events in the SIEM. The ability to correctly time-stamp events at the MIS/MIA is all the more important because, as assumed (see Section 2.2): some sensors at the edge layer may not have access to local clocks (e.g., physical sensors); some sensors with local clocks may exhibit poor *synchronization*, or be vulnerable to timing attacks.

5.3 Event Bus Resilience

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 [20] . In the MASSIF resilience architecture we want to employ overlay networks to create redundant network-agnostic channels for *timely* and *robust* event transport from the edge sensors to the core event correlation engine. There are two thus main requirements on the inter-node MASSIF communication middleware.

First, timeliness: 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. One should thus assume that there is eventual synchrony, that is, assume that message transmission latency is bounded. However, we must note that the underlying infrastructure can be the target of performance instability, or of attacks (is not trusted by default) which impact on the coverage of those latency assumptions. Although it may be difficult to state the exact bound, specific bounds have to be assumed at run-time, which means that the network will alternate between synchronous and asynchronous behavior, which is undesirable for our objective. Overlay networks might provide the necessary path redundancy to provide for timing fault-tolerance. 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., [21] [22]). Recently, a timeliness-aware application-level routing solution called Calm-Paranoid (CP) was proposed [23] , using overlay/multi-homing techniques. 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). We will build on these results.

Second, robustness: 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 to enhance the CP algorithm with Byzantine-routing capabilities [24] [25] and network coding techniques [26] . 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.

5.4 SIEM Core Protection

The MASSIF architecture allows for multiple strategies for protection of the core components executing application layer services.

The simplest one is perimeter defence, by isolating the core components within trusted intranets, isolated from the outside by core-MIS. Actually, this is the baseline protection offered in the architecture, as depicted earlier in Figure 3. This kind of protection is quite effective, since all the application subsystems are inside a perimeter which only communicates with the outside through a MIS, in two ways: with the Resilient Event Bus; and with auxiliary systems.

The Resilient Event Bus is an overlay communication subsystem internal to the MASSIF SIEM and thus itself protected, much in the sense that secure VPN (Virtual Private Networks) are. Auxiliary systems are any external systems accessed by the core application layer services (e.g., email, web, corporate servers) resident in networks alien to the MASSIF SIEM, either operated by the monitored system owners or by SIEM managed service subcontractors. They are considered untrusted in the SIEM fault/threat model, and as such traffic with them is carefully filtered by the core-MIS.

We also recall a second pillar of perimeter protection: besides executing protection functions, the core-MIS is itself built with resilience enhancing mechanisms, as discussed in Section 5.2, to protect it from direct attacks.

It should be noted that the publish-subscribe nature of the Resilient Event Bus communication model extends the modularity of the edge subsystems to the core systems (in fact, suggested in Figure 3): application servers may actually reside in more than one protected intranet, offering a multitude of deployment and server placement strategies.

Besides this baseline protection, SIEM core resilience can be enhanced through more sophisticated forms of protection, namely by mechanisms providing forms of defence in depth, for example, solutions featuring fault and intrusion tolerance of application servers themselves. Such solutions would for example provide resilience against insider attacks. Though they are not the focus of the instantiations foreseen for the project, several of the mechanisms advanced in Section 5.2 might be reapplied successfully to achieve fault and intrusion tolerance of core servers.

5.5 Resilient Storage

The storage solutions to be deployed in the MASSIF architecture have several purposes, requiring different levels of resilience.

In this section we are concerned with a specialised kind of storage: those units dedicated to archival of critical security information and events, requiring properties like integrity, confidentiality and unforgeability. Furthermore, integrity should be strong, that is, not only detecting, but also preventing successful attacks to integrity, so that information cannot actually be destroyed. This requires a combination of security and dependability, and calls for example for intrusion-tolerant techniques in the construction of the resilient storage units. One of the obvious uses of such resilient storage is to archive important security information and events in a way justifiably usable for criminal/civil prosecution of attackers after a security breach. Those attackers may include privileged users (insiders), with ample access to internal systems. The need for going beyond classical security techniques based on prevention is that erased log records are of no use, even if they were strongly signed and/or integrity protected before erasure.

Techniques like replication, diversity management, coding and threshold cryptography will be employed to *guarantee* unforgeability of the stored information, *and* to *guarantee* that the storage system itself is tolerant to faults and intrusions. Additionally, traditional techniques such as access control need 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.

The architecture of the Resilient Storage is illustrated in Figure 17. The figure also illustrates the interactions with other MASSIF components. The Processing Engine is the main MASSIF component in charge of feeding the resilient storage. The Processing Engine is able to process a massive amount of events per second e.g. 100s of thousands of events per second. From this huge pile of security events, only a few are of interest for forensic analysis. So only the events which are generated during a security breach are sent to the Resilient Storage by the Processing Engine to be stored for the potential forensic analysis (at a later time). We refer to this design approach as to the “Least Persistence” principle. In order to store data, the Resilient Storage exposes a very simple interface (very much like a “write” command), that hides its internal - sophisticated - mechanisms. In order to provide extreme unforgeability guarantees, data is not signed by means of standard techniques. Instead, a threshold cryptography mechanism is used. In a nutshell, threshold cryptography involves the distribution of secret key into different shares. The key is divided into shares in such a way that a certain number of shares, when combined, yield the original secret but corrupt share holders less than this certain threshold cannot calculate the secret. Also, the shares do not contain or reveal full or even partial information about the secret key that may be helpful to guess the secret. In this way the system is made Intrusion and Fault Tolerant, since if some of the shareholders are compromised or become corrupt, the system continues to provide its functionalities correctly.

The process for creating a forensic record, i.e. an unforgeable record related to a security breach is as follows. The event related to a security breach is fed to all the individual threshold cryptography units, i.e. to all participants of the threshold cryptosystem. Each unit implements a hash function which calculates the digest of the security event whose unforgeability is required. The hash function accepts a variable size message as input and produces a fixed size output. The output of the hash function goes as input to an encrypt function which encrypts the message digest with the secret share as the encryption key, i.e. produces a cipher which is called a partial signature of the security event. A component, called combiner, is responsible for assembling all partial signatures received from participants of threshold cryptosystem to generate a full signature. The full signature is attached to the original event, thus forming a signed security record, i.e. a forensic record. Forensic records are eventually stored using a persistent storage facility. They will be made available to application level services implementing forensic features. In order to read the stored messages a “read” command is in charge of authenticating the reader and of retrieving the stored data.

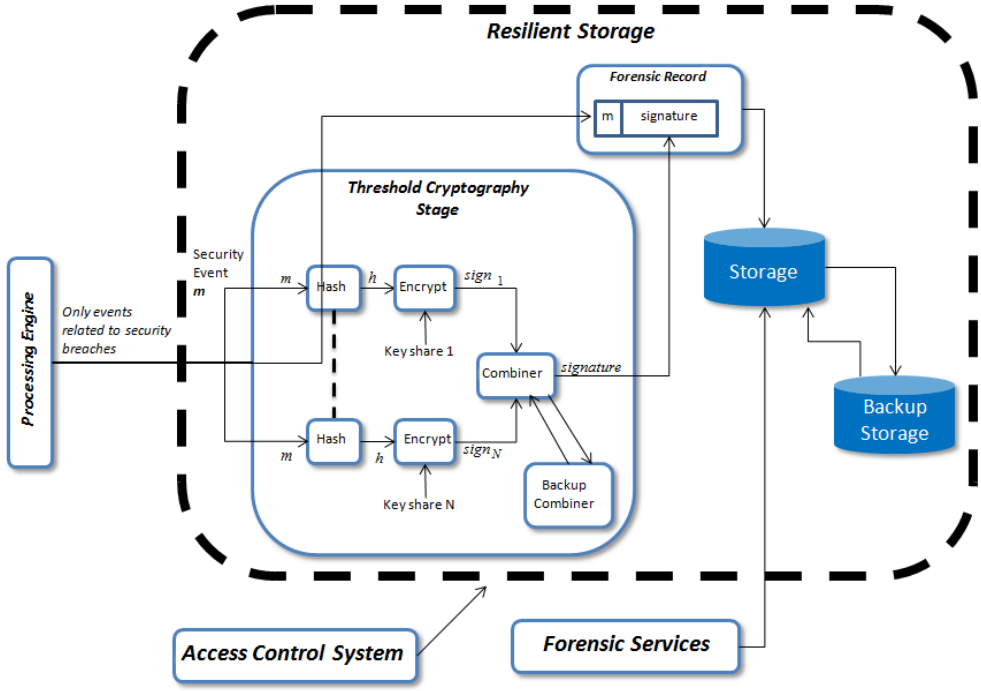


Figure 17 - Architecture of the Resilient Storage

Replication and diversity are employed in each processing stage to further improve the Resilient Storage tolerance to faults and intrusions. In particular, the threshold-based signature scheme, the combiner, and the storage are all replicated via a set of software replicas, which are deployed on a set of independent servers, implementing diversity at several architectural levels (i.e. hardware, Operating System, system software, and more).

The signed security events are stored according to the occurrence time, making it easy to access the signed events corresponding to a particular security breach. The backup Storage ensures the availability and resilience of signed security events in case of attacks of storage systems.

6. MASSIF Architecture vs. Existing Systems

Security Information and Event management systems have existed for about 10 years. Even though they still can be improved, they are being commercially successful today, which shows that designing an entirely new SIEM system from the ground up would be an enormous effort for little benefit. Instead, the MASSIF project has chosen to partner with two prominent open source SIEM, OSSIM and Prelude, to reuse existing functions provided by these SIEM implementations and to improve on the ones that make the most sense. The open-source choice (even though we are also looking at commercial SIEM environments) has been made because it eases analysis and integration. This section will thus focus on the MASSIF components that could supplement, improve or replace existing SIEM functions with advanced functionalities. This section will focus on malicious activity detection, infrastructure resilience, alert correlation, decision support and counter-measures, which are the key contributions of MASSIF to existing SIEMS.

6.1 OSSIM

Typical OSSIM installations share many concepts with the design of MASSIF. It's common sense to collect events at the edges and transport events up to the center where the SIEM collects events, correlates, analyses etc.

We will discuss some analogies and differences between OSSIM and MASSIF in this chapter, based on functionality and component.

6.1.1 Functional view

Translation Layer

The MASSIF translation layer is responsible for the conversion of events from event emitters (usually at the edge infrastructure) to a generic event format. This conversion or in the OSSIM case normalization is done directly on OSSIM sensors. The resulting format is the OSSIM message format, which is sent to an OSSIM server component for further processing.

Normalization

Normalization in the MASSIF architecture is planned to be done on the Edge MIS through the GET architecture.

The MASSIF GET architecture is very similar to the architecture of OSSIM data sources (often referred as plugins). OSSIM agent features pluggable parsers (analog to the Adaptable Parsers in the MASSIF architecture) that compute specific events sent from any attached system and translates (normalizes) those events to the OSSIM message format, which can then be forwarded to the next destination, which is typically a OSSIM server instance.

Right now OSSIM server does not feature a automatic recognition of the input data source as described in the MASSIF architecture via an event dispatcher. A manual recognition is implemented and is typically done when setting up the infrastructure. This approach has some tradeoffs, but simplifies normalization and decreases complexity in the running system.

Automatic event dispatching is not trivial and may not always produce the right dispatcher decision (e.g. duplicate process names in syslog sources that are used to make a dispatch decision).

Correlation

Correlation in MASSIF is performed by the Complex Event Processing system, correlation logic is implemented by CEP queries.

OSSIM does not implement elastic correlation in a cluster of nodes. Correlation is happening on single OSSIM servers. No precorrelation is done on sensors.

The OSSIM correlation is based on correlation directives, that are modelled in XML files. As part of the MASSIF project the translation of OSSIM correlation directives into Complex Event Processing queries is part of WP 3.4.1.

Aggregation

Aggregation in the OSSIM architecture is typically happening on a SIEM server (OSSIM server). OSSIM sensors do not preaggregate any events. This happens only on final destinations where correlation is done and typically only for events that have a calculated risk equal to or greater than 1.

Generic events dissemination

OSSIM does not provide a generic events dissemination mechanism as described in the infrastructure services section of the MASSIF architecture. The dissemination in OSSIM is done solely based on configurations.

Decision support and reaction system

A decision support and reaction system is not implemented in OSSIM. OSSIM features a action mechanism that is attached to policies. A policy can have any number of attached actions. A policy can be limited IP addresses, network assets (Hosts, Hostgroups, Network Groups, Networks). A policy can also be limited to a specified time range (e.g. Mon-Fri, 8-5) and different data sources or a set of data sources.

Actions include opening a ticket in the internal ticketing system, sending an email with the event data or executing a program on the SIEM server. This functionality can be used to provide a reaction to specific events, event groups, events on single hosts or hostgroups etc (e.g. trigger a script to lock an attacking IP address out of the network on the firewall, shut down a switch port). There is no framework and no enforcement or limitations on what the specified command can do.

6.1.2 Per component view

MASSIF information switch (MIS)

The MASSIF information functionality can be seen as a hybrid of an OSSIM agent and a OSSIM server residing on the same platform.

The entry point for security events is usually an OSSIM sensor running an agent, that collects and normalizes data from various event emitters. That data will then be normalized and delivered to one or several OSSIM servers (an OSSIM server can be installed on the same system as the sensor). After the event is received the server will take a policy-based decision how to compute the incoming event. An event can be correlated and stored locally, can be dropped from further processing, etc.

A traversal of events through a hierarchy of SIEM servers (MASSIF MISes) is not provided in the open source OSSIM server.

MASSIF information agents

OSSIM does not know the concept of a agent running on the payload machine. However we make heavy use of e.g. HIDS agents (OSSEC) that run on various payload machines and deliver security events in OSSEC format to the OSSIM agent, which normalizes into OSSIM message format.

Generic Event Translation Platform

As explained in Section 4.1.1 (Normalization) the generic event translation GET finds it's counterpart in the OSSIM agent with its plugin architecture. The adaptable parsers very much do the same as the OSSIM plugins do.

The GET manager behaves much like the OSSIM agent. It collects logs, decides on the adaptable parser to use and normalizes the event. The GET manager also features automatic recognition of log sources, which the OSSIM agent right now does not do. The OSSIM agent has similar controls as the GET manager to activate and deactivate plugins from the framework.

The MASSIF event manager functionality is also included in the OSSIM agent, if not provided by the parsers the OSSIM agent will prefill fields with reasonable data (e.g. timestamps, sensor information etc.). The agent will also take care of timely delivery and buffering in case a upstream infrastructure component is not connected. Events will be buffered until the remaining disk space is less than or equal to 5% of total capacity. After that limit events will be discarded.

Resilient Event Bus

In the OSSIM architecture the concept of a Resilient Event Bus is not existing. Communications are modelled in policies and configuration files. The event flow is not based on a publish-subscribe pattern, but on the wired policies and configuration files.

Core MASSIF services

The core MASSIF services include the Complex Event Processing and the decision support module. OSSIM does not provide a decision support functionality.

The Complex Event Processing engine finds its counterpart in the correlation engine that is built into OSSIM server. Correlation directives are written in a XML syntax and deployed on the OSSIM server. A correlation directive is triggered if the initial condition for the correlation directive is met. All subsequent events are then checked if they match the current hierarchy in the correlation process. If the correlation reaches the final rule and matches that rule a event is generated.

Repository

The repository in OSSIM is the OSSIM event storage engine, a database of all security events, alarms, configurations and all metadata. OSSIM does not have the concept of a short or long-term storage as described in Section 4.3.5. Neither does OSSIM feature an API to that information.

6.2 Prelude

We draw in this section an analogy between MASSIF architecture and components, and their Prelude counterparts. In fact, although they may add new functionalities, or they may have a better performance and scalability, most of the MASSIF components have their counterparts in Prelude. This section compares those similar components. It shows the limitations of Prelude modules and motivates the need to replace these modules with new MASSIF components.

Figure 18 shows an example of a distributed Prelude architecture and we will point out the correspondences between different components in the following subsections.

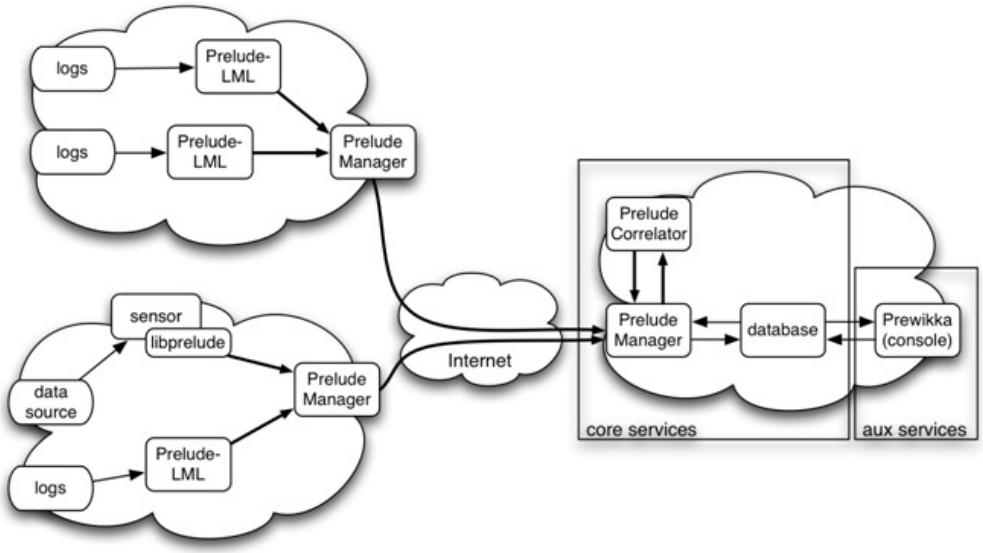


Figure 18 - Prelude Architecture with distributed Managers

6.2.1 MASSIF Information Switch - Prelude Manager

MASSIF Information Switches (MISes) correspond to Prelude Manager modules in the Prelude architecture. Just as in the MASSIF architecture, Prelude Managers can form a hierarchy that is similar to the hierarchy of edge and core MISes. Local Prelude Managers collect security events in a local domain and relay them to the core Prelude Manager. Internally the Prelude components use a binary representation of IDMEF [5] to exchange events and IDMEF corresponds to the MASSIF event format. Within a given domain, the local Prelude manager gathers events either directly from Prelude compatible sensors or indirectly using log and event parsing capabilities of the Prelude-LML component.

6.2.2 Generic Event Translation Platform - Prelude LML

Sensors connecting directly to the Prelude Manager use the libprelude library. It implements a set of functions required for these sensors to interface with the Prelude Manager.

Sensors that do not use libprelude are coupled with a Prelude-LML, which translates their output to binary IDMEF, and forwards it to a Prelude Manager. Prelude-LML can be also used to process any other log files to translate all or selected log events to IDMEF alerts. The translation is based on rules that associate log fields to IDMEF elements and attributes. A set of predefined rules for various log files exists, and user-defined rules can be added.

Prelude-LML reads logs from a file and doesn't provide means for receiving remote logs. Thus it is either co-located with the data source or the log delivery to Prelude-LML is to be handled by other means (e.g. via syslog).

Prelude-LML corresponds to the Generic Event Translation (GET) platform described in the MASSIF deliverable D3.3.1. Nonetheless, the GET platform offers more functionality than Prelude-LML, as summarized in Table 1.

We thus expect an enhanced performance and scalability for event translation when replacing Prelude-LML with the GET platform.

Prelude-LML	MASSIF Generic Event Translator
Does not support event dispatching	Dispatches events according to their format.
Does not support event grammars.	Handles event grammars and creates adaptable parsers.
Does not support type recognition.	Supports type recognition through the event dispatcher.
Parallel parsing requires several LML instances	Parallel event dispatching based on event format.
Output only to a Prelude Manager	Supports any output format whose grammar is defined.

Table 1: Prelude-LML vs. MASSIF GET

6.2.3 Resilient Event Bus - Prelude communications

The MASSIF Resilient Event Bus is a publish-subscribe type bus connecting edge MISes to core MISes. The resilient MASSIF architecture enables redundancy over core MISes, thus withstanding single node failures.

The Prelude architecture supports Manager redundancy (events can be sent to all n Managers every time or the sender can try the n Managers one at a time to find one that is available). The communication channel is point-to-point and uses a client/server model.

All agents (i.e. Prelude-LML and compatible sensors) register themselves to their Prelude Manager, and in a hierarchy of Managers the lower level Managers register themselves at the upper level Managers. The registration process allows the exchange of public key certificates, which allows mutual authentication of sensor-Manager and Manager-Manager connections.

All communications can be, and are by default, encrypted. Prelude thus guarantees the authenticity of the communication endpoints, and the confidentiality and integrity of the exchanged messages.

6.2.4 Core MASSIF Services - Prelude Correlator, database and Prewikka

Core MASSIF services mainly include the Complex Event Processing Engine (CEP) for alert correlation and the decision support module. The core MIS, which retrieves events from the Resilient Event Bus, relays these to the core MASSIF services for event processing.

Prelude functions similarly. The central Prelude Manager collects events from local Managers. It persists these events into a database and forwards them to the Prelude Correlator. The communication between the central Prelude Manager and the core Prelude services (the Correlator and the database) is bidirectional: the central Manager pushes events into these components, but also receives correlation results and retrieves events from the database for the web-based visualization component called Prewikka.

The Prelude Correlator module, illustrated in Figure 19, has an extensible architecture based on a set of correlation classes built with Python. As shown in Figure 19, the Prelude Correlator is conceived only to interface with the Prelude Manager. It registers to the Manager using the libprelude library. Events are sent to the Correlator in the IDMEF binary format that is supported by Prelude. The Correlator module builds internal data structures using those events, and then they are relayed to a plugin manager. The plugin manager applies each of the correlation plugins, which are implemented

as Python classes, to the IDMEF data structures. Prelude Correlator supports context-based correlation. A matching correlation class may thus trigger a new context that designates requirements to be satisfied by events sent through the Manager. Correlation plugins are independent, and so the Correlator design makes possible to define as many plugins as needed. Nonetheless, these plugins are checked sequentially for every incoming event. A high number of plugins thus degrades the performance of the Correlator. Compared to the CEP engine, Prelude Correlator is neither scalable nor elastic.

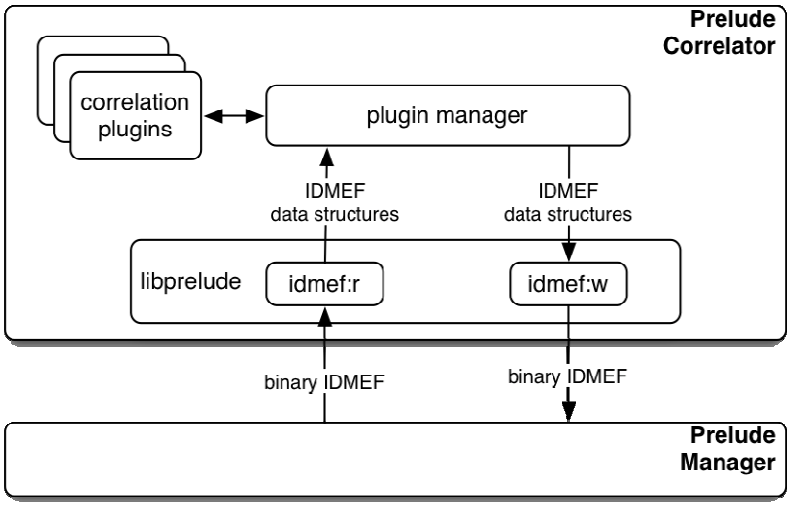


Figure 19 - Prelude Correlator architecture

At a high-level, the MASSIF CEP engine and the Prelude Correlator have similar functions: they take events as input, process (correlate) them and provide new correlated events as output.

Prelude does not provide any kind of decision support mechanisms. As the alerts are stored in the Prelude database, an external component such as MASSIF decision support component could be provided access to the alerts, and use these for threat response.

6.3 Potential MASSIF improvements

From the systems comparison done in previous sections, the main improvements provided by MASSIF to existing open-source systems can be summarized in:

- Generic Event Translator: adding pre-correlation capabilities and automatic recognition of the input data source.
- Resilient Bus: contributing to the secure dissemination of events from the monitored systems to the core SIEM.
- Complex Event Processing Engine: improving the scalability and elasticity of the correlation engine.
- Decision Support and Reaction: enhancing the adaptation capabilities of the SIEMs through the automatic selection and implementation of countermeasures.

Moreover, the complete set of alternatives are analysed in detail in [28] .

7. References

[1] The MASSIF Consortium. Deliverable 3.1.1: Event processing engine architecture. Technical report, 2011.

[2] The MASSIF Consortium. Deliverable 3.1.2: Design of the Distributed event processing operators. Technical report, 2011.

[3] The MASSIF Consortium. Deliverable 3.1.4: Design of elastic computing component. Technical report, 2011.

[4] The MASSIF Consortium. Deliverable 5.3.1: XML-based languages and repository. Technical report, 2011.

[5] Hervé Debar, David Curry, and Benjamin Feinstein. The Intrusion Detection Message Exchange Format (IDMEF). RFC 4765, March 2007.

[6] The MASSIF Consortium. Deliverable 4.3.1: Analytical attack modeling. Technical report, 2011.

[7] Java Compiler Compiler."Homepage".https://7.dev.java.net/7/ retrieved 2011-12-20.

[8] Project MASSIF. "D3.3.1 – Generic External Event Translation PlatformDesign".April 2011

[9] Project MASSIF. "11 – Generic External Event Translation Platform". January 2011

[10] Project MASSIF."10 - Preliminary Resilient Framework Architecture". Technical report, Fundaçao da Faculdade de Ciencias da Universidade de Lisboa, 2011.

[11] Project MASSIF. "11 - Security probes for Service Infrastructures". Technical report, ConsorzioInteruniversitarioNazionale per L'informatica, 2011.

[12] Gustavo Gonzalez-Granadillo et al., "Specification of decision support, simulation, and deployment software components", MASSIF project deliverable D5.2.1, June 2012. Work-in-progress.

[13] 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.

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

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

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

[25] 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.

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

[27] The MASSIF Consortium. Deliverable D2.1.1: Scenario Requirements. Technical report, 2011.

[28] The MASSIF Consortium. Deliverable D5.4.1: Integration specifications. Technical report 2012.

Annex A Detailed Mapping of Modules and Interactions vs. Workpackages