MAnagement of Security information and events in Service InFrastructures

MASSIF

FP7-257475

D5.1.2 - Preliminary Defense Services and Protocols

Activity	A5	Workpackage	WP5.1
Due Date	M24	Submission Date	2012-09-28
Main Author(s)	Nuno Neves (editor) (FFCUL), Alysson Bessani (FFCUL) Cesario Di Sarno (CINI), Nikolai Kuntze (Fraunhofer)		
Contributor(s)	Luigi Romano (CINI), Miguel Garcia (FFCUL) Paulo Verissimo (FFCUL), Ricardo Fonseca (FFCUL) Roland Rieke (Fraunhofer)		
Version	v1.0	Status	Final
Dissemination Level	PU	Nature	R
Keywords	Resilient SIEM operation; Resilient communication and nodes; Resilient event storage; Attacks and accidental faults;		
Reviewers	Luigi Coppolino (Epsilon) Carlos Arce (Atos)		

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

Version history

Rev	Date	Author	Comments
V0.5	2012-07-27	Nuno Neves	First integrated version of the deliverable
V0.6	2012-09-07	Nuno Neves	Second integrated version of the deliverable, which includes improvements in all chapters
V0.8	2012-09-25	Nuno Neves	Text improvements due to comments by internal reviewers
V1.0	2012-09-28	Elsa Prieto	Final review and official delivery

Glossary of Acronyms

AH	Authentication Header
AIK	Attestation Identity Key
API	Application Programming Interface
BFT	Byzantine Fault Tolerance
COTS	Component of the Shelf
CWE	Common Weakness Enumeration
DOS	Denial of Service
DoW	Description of Work
EC	European Commission
EG	Evidence Generator
ESP	Encapsulation Security Payload
EU	European Union
FEC	Forward Error Correction
FIFO	First-In First-Out
FP7	Seventh Framework Programme
FPR	Full Physical Replication
FVR	Full Virtual Replication
GPS	Global Position System
HTTP	Hypertext Transfer Protocol
ID	Identifier
IDS	Intrusion Detection System
IMA	Integrity Measurement Architecture
IP	Internet Protocol
IPsec	Internet Protocol Security
ISP	Internet Service Provider
IT	Intrusion Tolerant
LAN	Local Area Network
MAC	Message Authentication Code
MASSIF	MAnagement of Security information and events in Service InFrastructures
MIA	MASSIF Information Agent
MIS	MASSIF Information Switch
MTU	Maximum Transmission Unit
NVD	NIST National Vulnerability Database

OS	Operating System
PCA	Privacy Certification Authority
PFR	Physical Filter Replication
PII	Personal Identifiable Information
PR	Proactive Recovery
PU	Public Usage
R&D	Research & Development
REB	Resilient Event Bus
RDF	Resource Description Framework
RSA	RSA algorithm for public-key cryptography
RTT	Round Trip Time
SIEM	Security Information and Event Management
SMR	State Machine Replication
SOA	Service-Oriented Architecture
SSL	Secure Socket Layer
TA	Time Authority
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TOM	Total Order Multicast
TPM	Trusted Platform Module
UDP	User Datagram Protocol
USB	Universal Serial Bus
VFR	Virtual Filter Replication
VM	Virtual Machine
VMM	Virtual Machine Manager
WAN	Wide Area Network

Executive Summary

Security Information and Event Management (SIEM) systems offer various capabilities for real-time event collection by monitoring a network at a diverse set of locations, with the correlation and analysis of the obtained data to find out if attacks/intrusions are in progress, so that alarms and remediation actions can be initiated. Since the different components of a SIEM system (e.g., the sensors and the correlation engine) may be geographically dispersed, there is the need for a communication subsystem capable of transmitting data (e.g., events and reconfiguration commands) not only in LAN settings, but also over large scale networks such as the Internet. Some of the inherent problems associated with this setting are delays, routing and node misconfigurations, event integrity violations and denial of service attacks, which can all affect the correctness of the event analysis.

This deliverable describes the *preliminary* design of a set of solutions that can significantly improve the resilience of the operation of a SIEM system. These solutions were developed based on the current understanding of the MASSIF resilient architecture [16], but in most cases they can be easily generalized to be of use to other SIEM systems. The document covers four fundamental areas: The Authenticated Component Event Reporting addresses mechanisms that can be employed to give evidence that produced event data has not been tampered with; The Resilient Event Bus (REB) enforces a resilient communication among the edge-MIS and core-MIS devices, ensuring reliable and timely data delivery in various failure scenarios; Node defense mechanisms explains a set of techniques that can be applied in an incremental way to make nodes increasingly more resilient, and they were applied to a design of a highly resilient core-MIS; Resilient Event Storage (RES) presents a solution for the secure archival of event data.

In the next deliverable of WP5.1, D5.1.4, these solutions will be further refined, in order to provide the complete description of the resilient architecture, services and protocols of MASSIF.

Contents

List of Figures

List of Tables

1 Introduction

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

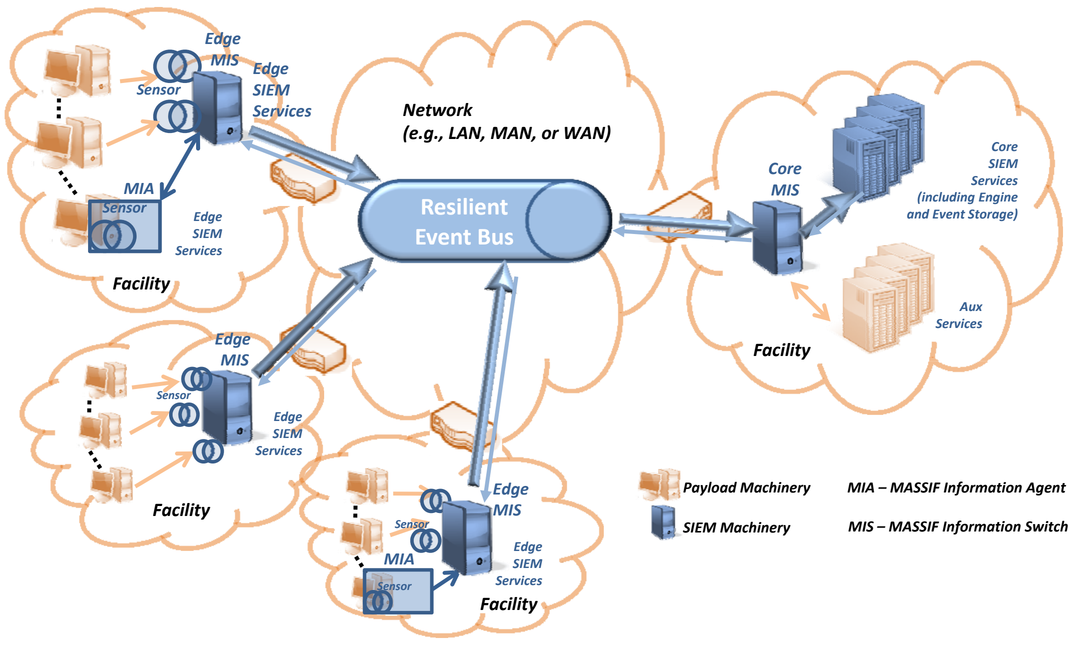

Figure 1.1: SIEM general architecture.

A SIEM system operates by collecting data from the monitored network and applications, which is then forwarded towards a correlation engine for processing. The engine then generates alarms and other information for post-processing by other SIEM components, which provide a complete view on the relevant state of the monitored system (called the payload machinery in Figure 1.1). For a more complete discussion on the MASSIF architecture and services see [17].

Sensors are the components responsible for the periodical event generation based on real-time monitoring of selected network components (e.g., routers, firewalls, intrusion detection systems) and applications (see left part of Figure 1.1). Events are first forwarded to special nodes called edge-MIS, which are placed in the vicinity of the sensors, so that they can be aggregated to other events collected locally and some pre-processing can be applied, such as format normalization. The edge-MISes then cooperate to transmit the events towards the core-MIS, which controls the traffic going in or out of the network where the SIEM processing components are located (see the right side of the figure). The core-MIS next sends the data to the final destination, such as the correlation engine.

Due to its fundamental role in the security management of an organization, the SIEM should be made resilient to faults and attacks. In this deliverable, we present the *preliminary* design of a set of solutions that can significantly improve the resilience of the operation of the MASSIF SIEM system. The document covers four fundamental areas:

Authenticated Component Event Reporting addresses mechanisms that can be employed to give evidence that produced event data has not been tampered with, and therefore, that can effectively be used to make decisions with regard to the monitored systems;

Resilient Event Bus (REB) enforces a resilient communication among the edge-MIS and core-MIS devices. The REB is based on an overlay superimposed on top of the existing SIEM infrastructure, which uses several mechanisms, such as coding algorithms, multihoming and multipath data transmission, to ensure timely data delivery in various failure scenarios;

Node defense mechanisms explains a set of techniques that can be applied in an incremental way to make nodes increasingly more resilient to different forms of faults, of either accidental nature or malicious attacks. Since the core-MIS plays a fundamental role in the protection of the core SIEM services, acting as a gatekeeper and preventing external attacks from entering in the network, we provide a design of how it could be built to be highly resilient by taking advantage of the proposed techniques;

Resilient Event Storage (RES) presents a solution for the secure archival of event data. RES is located together with the core SIEM services, and gives support for the forensic analysis of an incident based on the stored data, and enforces security policies that ensure that events are only made available to authorized entities according to the regulations.

1.1 Guidelines Analysis

According to requirements analysis and guidelines in deliverable D.2.1.1 [15], this deliverable contributes to the satisfaction of the following:

Guideline	Description
G.T.1. Resilience of the infrastructure	The various mechanisms presented contribute is different ways to this objective.
G.T.2. Security of event flows	The Authenticated Component Event Reporting, the REB, and the resilient design of the core-MIS all contribute to protect the event flows.
G.T.3. Protection of the nodes	Specific mechanisms are proposed for increasing the robustness of the nodes by introducing intrusion tolerance capabilities.
G.T.4. Timeliness of the infrastructure	The design of the REB and core-MIS have this guideline as one of their objectives.
G.T.5. Data authenticity	The various mechanisms presented contribute is different ways to this objective.
G.T.6. Fault and intrusion-tolerant stable storage	The RES design enforces this guideline.
G.T.7. Least persistence principle	The RES design enforces this guideline.
G.T.8. Privacy of forensic records	The RES design enforces this guideline.
G.S.6 Securing the evidence progressed by the MASSIF components	The Authenticated Component Event Reporting contributes to this guideline.

Table 1.1: Guidelines covered by this deliverable

1.2 Glossary Adopted in this Deliverable

As agreed by the MASSIF Consortium, the main reference of security glossary is provided by the National Institute of Standard and Technologies (NIST) [41].

1.3 Structure of the Document

The rest of the document is organized as follows: Chapter 2 presents the Authenticated Component Event Reporting solutions. Chapter 3 describes the design and mechanisms employed by the REB to secure the communication among the MIS. Chapter 4 provides an explanation of techniques that can be used to improve the resilience of the nodes by making them intrusion-tolerant. These techniques are applied to one of the fundamental components of our architecture, the core-MIS. Chapter 5 presents the RES for secure event archival. The concluding remarks are provided in Chapter 6.

2 Authenticated Component Event Reporting

Various equipment, including the sources of event data, relevant for the operation of the overall infrastructure is placed in non-protected environments. This recent development can be observed for example in smart grids for energy distribution or approaches in the area of facility management. It is therefore possible for attackers to acquire access to equipment with relative ease, and then initiate fake event reporting. This chapter studies the impacts of this problem and suggests solutions to address it.

2.1 Problem Description

We will start by analyzing some possible misuse cases, which have been reported in the scenario deliverable [15] of the MASSIF project.

Water level sensor compromise The attacker takes control of the water level sensors and uses them to send spoofed measurements to the dam control station. This hides the real status of the reservoir to the dam administrator. In this way, the dam can be overflown without alarms being raised by the monitoring system.

From this, we get the requirement that the water level measures have to be authentic for the administrator when they are displayed at the dam control station.

Tiltmeter compromise The attacker takes control of the tiltmeter sensors and uses them to send false measurements to the dam control station, thus hiding the real status of the tilt of the dam's walls to the dam administrator. An excessive tilt may lead to the wall's failure.

Crackmeter / jointmeter compromise The attacker has access to one of the crackmeters or jointmeters deployed across the dam's walls and takes control of it. So the attacker can weaken the joint or increase the size of the crack at the wall's weak point without any alarm being raised at the monitoring station.

These examples show the need for respective authenticity requirements. In general, however, information flows between systems and components are highly complex, especially when organisational processes need to be considered. Hence, not all security problems are discoverable easily. In order to achieve the desired security goals, security requirements need to be derived systematically [18].

In summary, the analysis of the use case and misuse cases of this critical infrastructure scenario shows that the overall function of the system requires authenticity of measurement values for several sensors. In that sense, the dam scenario is a prime example for the relevance of devices which satisfy respective authenticity requirements.

Authenticity requirements and devices A data record can be considered secure if it was created authentically by a device for which the following holds:

- The device is physically protected to ensure at least tamper-evidence. The data record is securely bound to the identity and status of the device (including running software and configuration) and to all other relevant parameters (such as time, temperature, location, users involved, etc.[3])
- The data record has not been changed after creation.

Digital Evidence according to this definition comprises the measured value (e.g., water level sensor measurement) and additional information on the state of the measurement device. This additional information on the state of the measurement device aims to document the operation environment providing evidence that can help lay the foundation for admissibility. As in the case of calibration of breathalyzers, for example, if the measurement device is modified, such information should also be recorded as part of amassing information supportive of admissibility. This will permit, at a later date, the linking of the software version used to collect the evidence in question. This information would permit an expert witness to testify to the known vulnerabilities of that particular software version and thus the likelihood of attacks.

Forensically Ready. By incorporating requirements into device design that focus on 1) potential admissibility of data records created by the device and 2) creating additional documentation that would support arguments for admissibility, we establish devices that are 'forensically ready'. Subsequent transport and secure storage of digital evidence are not part of this discussion, although they must be considered by anyone responsible for operating a network in a manner that ensures collection of competent legal evidence. However, for the purposes of this deliverable, we assume that digital evidence is created and stored in the device in question, and that there exists reliable mechanisms to maintain authenticity and integrity of the data records and also to provide non-repudiation for any steps of handling or changing the data, perhaps relying on some kind of digital signature which is often the case. For long-term security, archiving schemes can be used where digital signatures are replaced with some other security mitigations, anticipating that employed cryptographic algorithms will become unreliable due to increasingly sophisticated attacks or evolving computing capabilities. Physical attacks on devices are also not included in the discussion. We are assuming that, as in many cases, it will be sufficient to install tamper-evident devices (e.g., by using sealed boxes, installing devices in physically controlled rooms, etc.). Constructing real tamper-proof devices is expensive and difficult. Thus we are focusing on security at the mechanism level– how we develop and implement requirements for forensic readiness. Digital evidence requires that additional security mechanisms be implemented into the hardware that will render them impossible to be manipulated without physical access to the device.

The aim of this task is to integrate industry approaches to the attestation of event reporter states and how to integrate these measurements to gain a certain degree of trustworthiness and non-repudiation for events collected.

2.2 Technical solutions for the creation of digital evidence

The legal requirements towards the creation of digital evidence as discussed above imposes strong requirements on the security of individual technical devices as the evidentiary records but also on the

[3]The actual set of parameters and the protection levels depend on the scenarios and on the type of data record

processes involved in the processes for validating the device and software running on the device, for transmitting and storing evidence records, for linking evidence records to a chain of evidence, and also for verifying evidence records in the case of a dispute. The following subsections provide an overview of existing technical approaches starting from securing the actual creation on the individual device, looking at the infrastructure, and finally the processes involved.

2.2.1 Individual Device

Devices with various interfaces pose particular problems. Besides typical communication network interfaces, direct or close-range access via USB, for example, increase the complexity of protecting devices from physical access, let alone network attacks. As discussed previously, the complexity of current state-of-the-art devices presents a challenge for constructing a secure device that is both efficient and useable. Therefore, taking a pragmatic approach to securing digital evidence on these devices, we suggest to focus on establishing assurance that the device was not manipulated at the time of the creation of the evidentiary record.

One approach might be to establish a cryptographic binding of evidence to the status of the device [67]. This can be achieved by using the existing technology of Trusted Computing [57] as specified by the Trusted Computing Group. The so-called Trusted Platform Module (TPM) can establish a hardware root of trust in the device. The security-level of a specialized security chip can be compared to SmartCard security. In combination with a first trusted step in the boot process, the TPM can be used to store, and securely report, measurement values documenting all software that was loaded after the current boot started. Further, the TPM provides the functionality to sign data records combined with these measured values and also to time-stamp data records to reliably reflect time relationships. The first prototypes of traffic cameras secured by this technology are available [84, 83]. In addition to the so-called attestation of the current boot process of a device as established by Trusted Computing, there also exist approaches that go beyond attesting to only one boot cycle. The cumulative attestation proposed by LeMay and Gunter [46] provides additional records and attests to the history of the boot process.

In contrast to the Trusted Computing approach, measurement values are not completely deleted for each re-boot, but a cumulative measurement chain is generated over several boot processes. This approach ensures that the device has not been booted in an insecure start after the cumulative measurement has started. It should be noted that, by using hardware-based roots of trust protection, this also prevents some types of insider attacks where insiders try to produce false evidence. The trust in the status reporting of a particular device is rooted in certain core roots of trust. The TPM is one prominent example of an available root of trust for reporting. These roots of trusts are built and certified by public bodies to be tamper-proof, or at least hard to tamper. This reduces the possible attack vectors that could result in modification of the reported status of the device, even to authorized insiders like administrators.

2.2.2 Infrastructure

It should be noted that securely creating a data record is not sufficient to establish secure digital evidence. The device producing the record must be integrated into an appropriate infrastructure that can be structured into two parts: 1) elements that collect the data that then is stored in the evidence record and 2) securely transmitting and maintaining long-term storage of that data. Data collection is not only about maintaining the integrity of the data. Correctness of sensor data depends on many other factors, such as physical parameters of the environment (e.g., temperature or humidity), the location of the device and the physical integrity of the sensor itself. Some of these factors can be controlled by additional sensors; the status of these could be included in the reporting from the hardware-based attestation mechanisms.

Nevertheless, physical manipulation of the sensors is always possible. Threat modeling and risk analysis can provide analysis of residual risks remaining after Trusted Computing is implemented. Integrity and authenticity of data records can be maintained through use of public key cryptography. Since the private key can be stored exclusively inside a hardware security chip, this aspect of the infrastructure can be secured in this manner. Also solutions for long-term archiving exist (e.g., by renewing digital signatures before their algorithms are broken and signatures become useless). The mechanisms for this type of protection are well-established and can be efficiently implemented. However, digital evidence can contain personal identifiable information (PII), requiring application of privacy enhancing technologies to digital evidence. Additional infrastructure is needed if several individual evidentiary records are linked to a *chain of evidence* [43].

2.2.3 Process

In addition to technical solutions for securely creating and storing digital evidence and digital evidence chains, organizational processes must enable the correct implementation and reproducibility of these technical solutions. Verification and checking of digital evidence cannot be restricted to checking a single digital signature per evidence record. It also needs to include additional checks on cryptographic key certificates and validation of the status of the devices involved in the creation of evidence records. Various types of digital certificates for cryptographic keys or software measurement values will be necessary. Additional checks can be required such as certification of the platforms involved in the creation of evidence records. A chain of evidence (or most probably a tree or several linked trees) would require going through this process for each type of digital evidence and to establish all necessary links between evidence records.

The following describes a proposed procedure required in advance of actually producing signed digital evidence:

1. Produce hardware security anchor (e.g., TPM): The hardware security anchor must be produced at a high security level.

2. Certify hardware security anchor: Security properties of the hardware security anchor should be documented in a security certificate with an appropriate security level.

3. Certify platform: In addition to the single security chip, the means of its integration into the platform and the properties of the root of trust for measurement are relevant and should be verified and certified.

4. Produce software: Relevant infrastructure software such as operating system, drivers, and application are produced and validated.

5. Installation, initialization and certification of software: It must be ensured, that software installation and initialization has occurred properly, has not been manipulated, and that security certification does indeed cover all relevant aspects.

6. Define location, valid temperature, etc.: Certify reference measurement values for calibrated devices.

7. Generate and certify signing keys: Since the scheme described above relies heavily on cryptography, and therefore on secure generation, distribution and storage of keys, these processes require verification and certification. Because of the range of possible use cases, it is difficult to find and recommend one single algorithm.

8. Define location, valid temperature, etc.: Parameter ranges for correct use of the system must be established and then, either the occurrence of lower or higher temperatures prevented, or the infrastructure design changed to avoid problems. As an example, perhaps temperature control could be included in the device in order to satisfy temperature requirements.

9. Installation of device: The installation an initialization process is critical as this is the phase where keys can be generated and exchanged.

10. Establish communication with server: The establishment of client server communication is in principle well-understood. However, there is no efficient solution currently for binding SSL keys to underlying attestation values and also the platform the key owner claims it belongs to.

11. Reference measurement record: For attestation to make any sense, reference values for the correct state of the device must be established in order to check for manipulation.

12. Document and store reference records and transfer to server: In addition to reference methods, it can also be useful to store a number of data records on the server side in order to enable sanity checks.

13. Start the boot process and time synchronization. The conditions to begin operation have been met.

14. Evidence collection: finally, sensor data can become data records that potentially can become evidence. For this reason, data records are time-stamped using the TPM.

2.3 A high-level architecture for collecting secure digital evidence

The previous section has introduced a notion of secure digital chains of evidence. Before identifying possible building blocks for actually realising the creation and collection of data for such secure digital chains of evidence, this section introduces a high-level architecture for the collection of secure digital evidence.

Obviously, it is infeasible for many real-life systems to identify, create, collect and store all possible or potentially useful digital chains of evidence explicitly. It is more feasible to identify critical events to be documented together with parameters linking events. Thus, the goal of a pro-active collection of digital evidence should be to create and store a graph of linked secure evidence records in a way that a path through the graph can represent a secure chain of evidence. Note that not all paths will represent a useful chain of evidence. Such a system as depicted in Figure 2.1 consists of the following elements:

- *Evidence generators* create data records and securely bind them to relevant parameters e.g. by digital signatures using hardware-based security [68].

- *Evidence collectors* can add semantic information to the evidence record and make it available for distribution and storage [66, 64].

- A *Forensic data-base* stores all secure evidence records as a graph structure representing the links between different events.
- Actual creation of chains of evidence is an interactive process using the *Interactive forensic data-base explorer*.

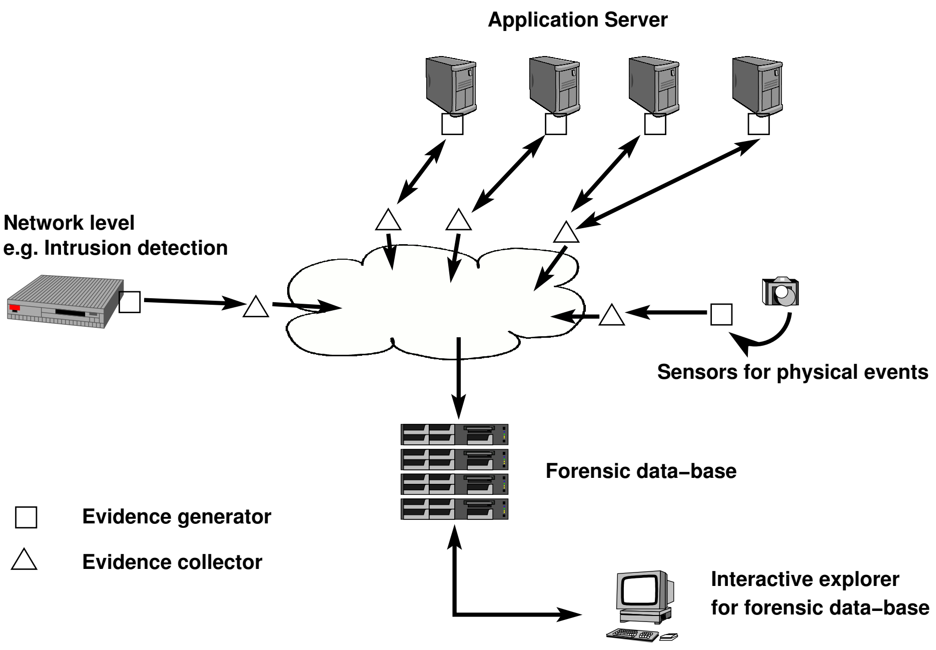

Figure 2.1: High-level architecture for collecting secure digital evidence

2.4 Building-blocks for secure evidence generation

In [68] an approach for the generation of individual secure evidence records was presented. This approach is based on established hardware-based security mechanisms and is applicable to special devices producing data records with possible forensic use. The architecture presented includes a sketch of the process needed to ensure the security of the evidence record. Figure 2.2 shows the different steps of this process.

Even in individual data records (e.g. images taken by a digital camera), it becomes clear that the security of the collected evidence records depends on a number of steps in the process that also need to be documented. Thus, even in this relatively simple scenario, a number of events can produce additional digital evidence, thereby creating a digital chain of evidence consisting of evidential data representing events of very different types. The following paragraph introduces several building blocks that can be used to build a system supporting the construction of secure digital chains of evidence. The roles of the different building blocks correspond to the different components of the architecture for collecting secure digital evidence.

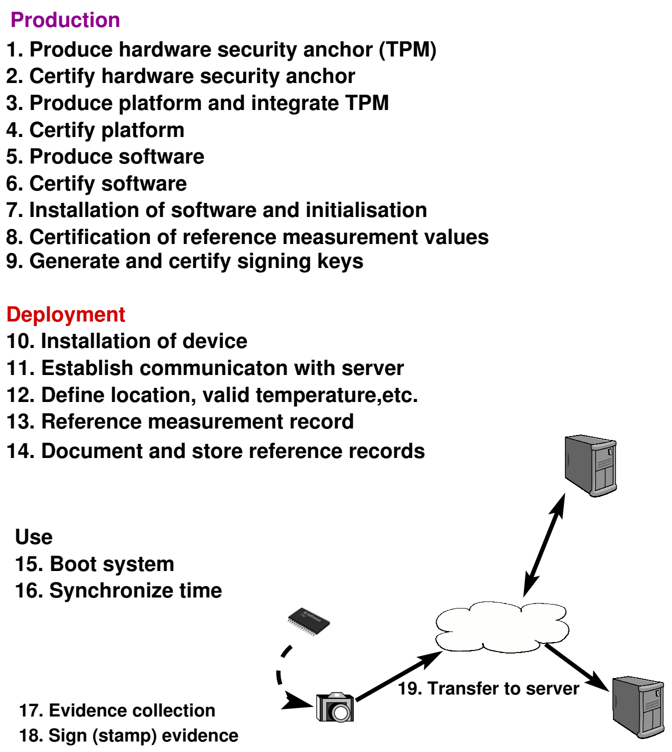
Production
1. Produce hardware security anchor (TPM)
2. Certify hardware security anchor
3. Produce platform and integrate TPM
4. Certify platform
5. Produce software
6. Certify software
7. Installation of software and initialisation
8. Certification of reference measurement values
9. Generate and certify signing keys
Deployment
10. Installation of device
11. Establish communicaton with server
12. Define location, valid temperature,etc.
13. Reference measurement record
14. Document and store reference records
Use
15. Boot system
16. Synchronize time
17. Evidence collection
18. Sign (stamp) evidence
19. Transfer to server

Figure 2.2: Process to establish secure evidence records

2.4.1 Secure evidence generator using Trusted Computing technology

The core part of the architecture is the actual generation of secure digital evidence. One possible approach is the use of hardware-based security mechanisms in particular Trusted Computing and the Trusted Platform Module (TPM) as specified by the Trusted Computing Group (TCG). A TPM provides a variety of security functionality. For secure evidence generation, those parts of the TPM that identify the device, bind data to the identity of the device, and provide authentic reports on the current state of the device are essential. In the context of digital cameras, the feasibility of the use of TPMs for the protection of digital images has already been proposed [68] and demonstrated [84]. The following paragraphs revise the most important parameters to be secured.

Proof of software and configuration

One important aspect of the generation of digital evidence is the status of the device used in the process. The software and configuration used to produce evidence needs to be presented and linked to the individual record. One simple scheme hereby is to include software name and version number as a simple string of text in each evidence record. This first (and often used) approach allows for uncertainties with respect to updates and various attacks on the evidence records. Just naming the software is not sufficient if the device can be manipulated. Stronger means of protection are therefore required to reliably document the software and configuration of the particular evidence generator.

To provide proof on the actual state of the evidence generator, trustworthy reporting in the device is required. The Trusted Computing standard introduces a core root of trust for measurement which establishes the foundation to report on the status by creating a chain of trust [35]. This chain of trust can be reported to external entities to allow for a verification of the evidence generator. This verification process is called Remote Attestation.

Application of remote attestation allows for a session based or per record scheme. The session based approach relies on an initial attestation of the system and a session bound to the individual evidence generator and status. Every evidence record is then cryptographically bound to this session and therefore to a particular system state. The second per record scheme involves an attestation process for each evidence record. As in the basic remote attestation, an external random number generator is involved, and longer delays as well as higher bandwidth utilisation are to be expected. As presented in [77], more advanced schemes allowing for scalable attestation schemes are to be applied.

Lightweight Infrastructure. One important feature of the proposed incorporation of Trusted Computing is the lightweight infrastructure necessary during run-time compared with a traditional Public Key Infrastructure system. Given the assertions of the hardware, a single key will not be revealed. Therefore, it is not required to maintain certificate revocation lists and to check them before a certificate is accepted. It is also not possible to move a certain identity of an evidence generator (represented by its key) from one device to the next. These inherent features allow for typical deployment scenarios like embedded, resource constraint environments.

Evidence record order

Time is a very important parameter in the forensic evaluation of evidence records. In most cases, it is absolutely necessary to have more or less precise but reliable information on when a particular event, such as the generation of an evidence record, has happened. Therefore, evidence generators need to bind evidence records to timing information. For digital chains of evidence representing a particular process this time information is essential to reconstruct the order of events in the process.

In the case of a single device, a monotonic counter (e.g. a clock) can be used to issue a time stamp for each record to ensure the order. To ensure the probative force of the time stamp, the time needs a cryptographically strong binding to the evidence record; further, it needs to be bound to time information to be issued by a trustworthy time authority. Especially the latter proves to be a strong requirement and is mostly solved by trusted third parties producing time stamps in specially secured and certified installations. However, direct online time-stamping by a trusted time source is far too inefficient for most reasonable evidence generators. Particularly in the case of embedded systems and/or high numbers of records, such a remote time stamping would create a bottleneck. Thus, a secure evidence generator should be able to produce time stamps on its own. Of course, the remote and probably more reliable time-stamping service can be used to synchronise the local time of the evidence generator with an official time source.

A feasible approach is to introduce a certified monotonic and timed tick-counter and a mechanism for digital signatures and secure key storage to provide for time-stamps. The tick-counter as well as the cryptographic functionality should be protected by hardware-means in order to prevent manipulations by malicious software. To achieve a trustworthy local time stamp authority, the hardware being protected has to provide for a shielded monotonic counter incremented in a certain interval. To ensure this particular interval, a monitoring of the accuracy of the external clock of the hardware incrementing the counter is also required. Such techniques are available in many standard PC architectures equipped with a Trusted

Platform Module (TPM) and can also be provided via Smart Cards [76]. The TPM identifies each session starting with the power up of the device with a new random number created within the TPM and is then able to time stamp arbitrary data. These time stamps can then be used to identify the order of the evidence records generated.

Considering distributed evidence generators, it is required to establish a link between the individual monotonic counters. Linking two counters results in a measure to translate between the respective local counter value into the other, which can be denoted as $n_1 := n_2 + -offset$. The offset is the expected uncertainty in the association due to delays on the network and computational overhead. Figure 2.3 depicts a scheme to associate one counter to the other. Hereby generator g_1 sends to g_2 a tick stamp on a random value TS. TS is then tick stamped by g_2 and sent back to g_1. The returned $stamp_{g_2}(TS)$ is then again stamped by g_1 and the resulting evidence is stored. Due to differences between the initial stamp of g_1 and the latter one, the maximum offset can be calculated and an attack on the response time of g_2 can be recorded and documented. To extend this scheme to a mutual link, the stamped result of g_2 is to be sent back to g_2. g_2, then it can stamp the actual message received to document the delay between g_1 and g_2.

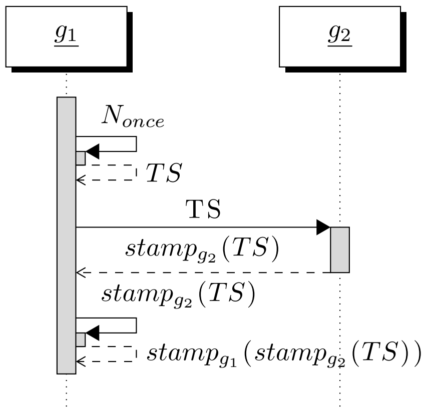

Figure 2.3: Trustworthy counter linking

Depending on the particular infrastructure, it can be necessary to link the time between several nodes belonging to one process. It can be efficient to use one central node to establish bilateral links between each node and the one central node; by doing this indirectly, all timing ticks of all nodes can be linked. Nevertheless, in highly distributed systems it can also be necessary to establish a peer-to-peer structure without any central node. In these cases, more intelligent management procedures need to be set-up in order to ensure that all events in a process can be ordered by their time tick information. Such an approach is particularly important in networks with critical functionality as, e.g., industry controls systems used for example in the dam scenario. In such networks the synchronisation of time ticks can be combined with other existing security mechanisms distributed [47].

Real Time Association

The previous paragraph described that it must be possible to associate the correct order of events represented in a digital chain of evidence. For this requirement, it is sufficient to know the time an event happened in relation to other events. Another stronger requirement for valid evidence can be to know the real time an event has happened. In principle, real time information can be established in the same way. However, synchronisation can be quite loose when only the order of events is important. For real time associations there are two major differences:

- First, the time synchronisation needs to be between the evidence generator and a reliable time source, such as a certified time-stamping service. Indirect synchronisation via other nodes increases the delay and thus the inaccuracy of the time synchronisation.
- Second, the accuracy of the time synchronisation becomes relevant.

Several protocols have been proposed for the use of the TPM tick counter to represent real time information [68, 84] and also the general properties of time synchronisation protocols and algorithms have been analysed [34]. Common to all approaches is that within the digital chain of evidence also information on the time synchronisation has to be recorded. This information contains the original time stamp of the time authority but also information on the accuracy of the synchronisation, the time intervals associated with the tick counter in the evidence generator and also information to keep track of resets of the tick counter. The TPM provides support for all these parameters. One example is the tick counter in the TPM that comes with a tick nonce that identifies tick counter sessions. Tick stamps with the same nonce belong to the same session without a reset of the counter. Thus, once the tick nonce has changed, a new synchronisation with the authentic time source is necessary. It should be noted that in contrast to the proposed use of the tick counter in [84], the change of the tick nonce cannot reliably identify a re-boot of the device. As long as the TPM has power, the tick counter will not be explicitly reset during a re-boot of the device.

Other parameters

Various other parameters can become relevant for forensic use of data records. However, not all of them are readily available and can be easily or efficiently included in the secure digital chain of evidence. As an example, we briefly discuss the geographical location of the device at the time of evidence generation. Different techniques exist to determine the location. Depending on the technology used, the accuracy of the location information differs. More and more devices support the Global Position System (GPS). If adequate GPS signals are available, the GPS localisations can be in the range of 5 meters for consumer-grade GPS devices under open skies. The results within buildings under trees or with other obstacles range from 10 meters to no position information at all [82]. Other approaches, such as triangulation in wireless LAN or localisation within the GSM network are usually less accurate although they can be quite accurate in special scenarios, such as indoor localisation [40]. Parameters can have very different characteristics; and as a means of supporting digital chains of evidence in developing systems, one has to make sure all relevant parameters are covered and maybe additional sensors are installed to enable the collection of these parameters. Examples can include the temperature of the device (very low or high temperatures can lead to corrupted evidence data), the orientation of a camera, or the names of users currently active on a multi-user device. Determining relevant parameters and their validity ranges as well as their meaning for the chain of evidence is a very important step in the engineering of such systems.

Evidence records

In addition to the different parameters related to the evidence records, it is also important to generate and secure the evidence records themselves. Obvious security measures such as digitally signing the evidence records and binding these signatures to identity the other parameters described above can be used to guarantee the authenticity and integrity of the data records in a way that these security properties are not violated by distributing evidence records and storing them in the forensic data-base. However, there can also be additional security requirements. One important factor is privacy. Evidence records can potentially contain information in individual persons or other secret information, e.g. that which is business related. Therefore, the confidentiality of evidence records shall not be neglected either. Suitable encryption should be used and other best practises for dealing with confidential data shall be applied.

2.4.2 Event Collection and Event Correlation

In the majority of application scenarios, a certain decision by the system is not based on a singular event but on the correlation of several factors defining a certain high-level event which is defined by a set of low-level events and a process for the correlation of the low level events. Low level events are simple occurrences, like fire wall incidents and configuration changes as well as photographic evidence from the speeding camera. The process defines how these events have to interact based on an operational model where a certain high level event is to be produced.

In non-complex use cases, as presented in digital cameras [84], the evidence is generated by a single measurement agent and only the evidence records of the particular device are required. To achieve a certain probative force also in complex scenarios, it is also required to provide data on other aspects of the IT system not directly related to the evidence in question but documenting the trustworthy state of the infrastructure. For scalability reasons in bandwidth or computationally restricted applications, it is also required to split one event into a set of corresponding events.

The task of correlating events in order to construct digital chains of evidence is closely related to the task of security information and event management (SIEM) in IT networks. In particular, correlation of events from different levels and contexts, it is still very difficult in the SIEM context. In SIEM systems the correlations need to be explored at run-time to be able to induce appropriate reactions on misbehaviour. The situation is different for forensic use. Digital chains of evidence usually don't have to be created at run-time. In the case of the forensic use of event information, it is sufficient to collect the event information and maybe add additional semantic information at run-time. The actual evaluation of the correlations between events in order to produce a chain of evidence only occurs in the case of disputes or other forensic evaluations. For forensic use, a bigger effort therefore needs to be made in carefully choosing event information to be stored and in defining parameter related events.

To correlate data it is required to provide for an infrastructure supporting the processing of events from various sources in a unified structure representing the relations between the individual events.

2.4.3 Forensic data-base

For the forensic data-base, two main characteristics can be identified:

- The data-base can potentially contain huge numbers of more or less related evidence records representing graph structures where paths through the graphs can be chains of evidence also using semantic information on the events. Thus, the data-base needs to be scalable and it needs to support the exploration of large graphs with semantically enriched information.

- Evidence records need to be securely stored for a potentially long time. Storage of evidence needs to comply to regulations for long-term archiving.

For the first characteristics, the so-called triplestore [1] seems to be particularly suitable. Evidence records can consist of relatively short statements about what has happened. Triplestore is a special purpose database type developed for the use in semantic web frameworks. For a system supporting the proactive generation of secure digital chains of evidence, semantics of evidence records in terms of events that happened need to be known already at design time. The resulting structure is very similar to what can be expressed as resource triples within the Resource Description Framework (RDF) specified by the W3C (http://www.w3.org/RDF/). Some triplestore databases are very powerful with support for billions of triples loaded at a speed of more than 1.000 triples per second. Further, they support a variety of graph representations and rule-based exploration.

In addition, a variety of solutions exist for the area of secure long-term archiving. According to national regulations regarding long-term aspects of the probative force of a certain evidence, record archiving is the last step in the creation process. During the time of an evidence record in the archive, the cryptographic means used can wear out, resulting in a decreased level of trust in a specific evidence record. Existing work (e.g. [44]) shows approaches to maintaining the probative force of digital evidence in long term archives. There are also products on the market supporting long-term archiving and re-signing archived data records.

However, combining long-term archiving with a high-speed triplestore without losing the advantages of the triplestore seems to be very difficult. Therefore, it is probably necessary to follow a dual strategy for the forensic data-base where the long-term archiving and the triplestore are not fully integrated. All evidence records will go into the triple-store, but probably only a small subset really requires secure long-term storage. During the creation of evidence records, the record has to be marked in a way that the forensic data-base can decide whether long-term archiving is necessary or not. Then, a digital chain of evidence can be created using the triplestore and after completing this step long-term secured representations of the evidence records are retrieved from the long-term archive in order to produce the complete, secure digital chain of evidence.

2.4.4 Exploring the forensic data-base

This final part of the architecture for secure digital chains of evidence strongly depends on the format and data model of the forensic data-base. If the data-base is implemented as a triplestore with a good meta model for evidence records, a variety of tools and languages, such as Jena [2], SWI-Prolog [3] and AllegroGraph [4], can be used to develop interactive tools to explore the data-base. Relations between evidence records (and thus between events) can be graphically visualised, queries can be used to find matching evidence records or Complex Event Processing can be used to search for evidence records belonging to a particular chain of evidence.

[1] `http://en.wikipedia.org/wiki/Triplestore`
[2] `http://jena.apache.org/index.html`
[3] `http://www.swi-prolog.org/pldoc/package/semweb.html`
[4] `http://www.franz.com/agraph/allegrograph/`

2.5 Trusted MASSIF Information Agent

The usefulness of monitoring large systems clearly depends on the observer's level of confidence in the correctness of the available monitoring data. In order to achieve that confidence, network security measures and provisions against technical faults are not enough. As stated above, unrevealed manipulation of monitoring equipment can lead to serious consequences. In order to improve the coverage of this type of requirements in a SIEM framework, we now describe a concept and a prototypical implementation of a Trusted MASSIF Information Agent (T-MIA) [18].

2.5.1 Trust Anchor and Architecture

The protection of the identity of the device for measurement collection is necessary. Furthermore, the lack of control on the physical access to the sensor node induces strong requirements on the protection level.

By a suitable combination of hardware- and software-level protection techniques any manipulations of a sensor have to be revealed. In addition to the node-level protection, network security measures are needed in order to achieve specification-conformant behaviour of the sensor network, e.g., secure communication channels that protect data against tampering. This work is not intended to discuss network security, neither protection of hardware components. We rather concentrate on the important problem of clandestine manipulations of the sensor software.

A commonly used technique to reveal manipulation of a software component is software measurement: Each component is considered as a byte sequence and thus can be measured by computing a hash value, which is subsequently compared to the component's reference value. The component is authentic, if and only if both values are identical. Obviously, such measurements make no sense if the measuring component or the reference values are manipulated themselves. A common solution is to establish a chain of trust: In a layered architecture, each layer is responsible for computing the checksums of the components in the next upper layer. At the very bottom of this chain a dedicated security hardware chip takes the role of the trust anchor or "root of trust".

Trusted Computing [57] offers such a hardware root of trust providing certain security functionalities, which can be used to reveal malicious manipulations of the sensors in the field. Trusted Computing technology standards provide methods for reliably checking a system's integrity and identifying anomalous and/or unwanted characteristics. A trusted system in this sense is build on top of a Trusted Platform Module (TPM) as specified by the Trusted Computing Group (TCG). A TPM is hardened against physical attacks and equipped with several cryptographic capabilities like strong encryption and digital signatures. TPMs have been proven to be much less susceptible to attacks than corresponding software-only solutions.

The key concept of Trusted Computing is the extension of trust from the TPM to further system components [37]. This concept is commonly used to ensure that a system is and remains in a predictable and trustworthy state and thus produces authentic results. As described above, each layer of the chain checks the integrity of the next upper layer's programs, libraries, etc. On a PC, for example, the TPM has to check the BIOS before giving the control of the boot-process to it. The BIOS then has to verify the operating system kernel, which in turn is responsible for the measurement of the next level. Actually, a reliable and practically useful implementation for PCs and systems of similar or higher complexity is not yet feasible. Sensoring and measuring devices, however, typically have a considerably more primitive architecture than PCs and are well-suited for this kind of integrity check concept. Even for modern sensor-equipped smartphones, able to act as event detectors, but having the same magnitude of computing power that PCs had a few years ago, an implementation of the presented concept is possible. A prototypical implementation is presented in more detail now.

2.5.2 Proof of Concept: Base Measure Aquisition

Figure 2.4 depicts the architecture of the T-MIA.

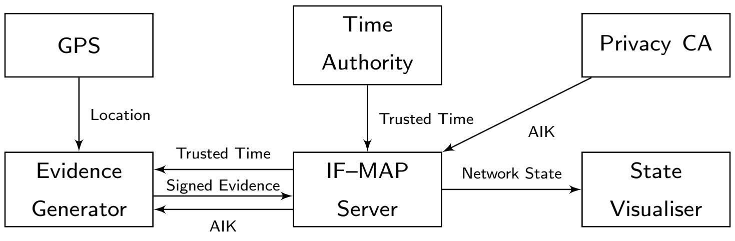

Figure 2.4: Trusted MASSIF Information Agent architecture

The main component of the T-MIA is the *evidence generator* (EG), which collects base measures and provides the measurement functions used to produce derived measures. Furthermore, the EG supports the processing of measures from external sensors, e.g., location data from a GPS module. The EG is expected to operate in unprotected environments with low physical protection and externally accessible interfaces such as wireless networks and USB access for maintenance. A necessary precondition to guarantee authenticity of the measures, is a trustworthy state of the measurement device. To meet this requirement, the EG is equipped with a TPM as trust anchor and implements a chain of trust [43]. As explained above, revelation of software manipulations is based on the comparison between the software checksums and the corresponding reference values. This comparison may be done locally within the node (self-attestation) or by a remote verifier component (remote attestation) [57].

The EG submits the collected measures digitally signed to an IF-MAP [36] server, which acts as an event information broker. During initialisation, the EG obtains two credentials from trusted third-party services for signature purposes. Figure 2.5 depicts the boot-time interaction between the EG and those services, and the role of the TPM in this interaction.

An Attestation Identity Key (AIK) is used to sign measurement results in a manner that allows verification by a remote party. The Privacy Certification Authority (PCA) issues a credential for the TPM-generated AIK. The certified AIK is, henceforth, used as an identity for this platform. According to TCG standards, AIKs cannot only be used to attest origin and authenticity of a trust measurement, but also, to authenticate other keys and data generated by the TPM. However, the AIK functionality of a TPM is designed primarily to support remote attestation by signing the checksums of the EG's software components, while signing arbitrary data is, in fact, not directly available as a TPM operation. We have shown elsewhere, how to circumvent this limitation [42]. Hence, we are able to use TPM-signatures for arbitrary data from the EG's sensors.

Any TPM is equipped with an accurate timer. Each event signature includes the current timer value. However, the TPM timer is a relative counter, not associated to an absolute time. A *time authority* (TA) issues a certificate about the correspondence between a TPM timestamp (tickstamp) and the absolute time. The combination of tickstamp and TA-certificate can be used as a trusted timestamp. Alternatively, another trusted time source, such as GPS, could have been used.

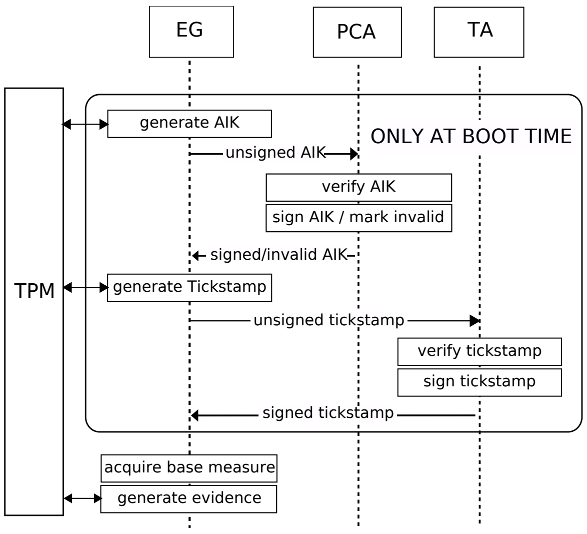

Figure 2.5: Process model

Putting it all together, a measurement record includes arbitrary sensor data, a TA-certified time stamp, and a hash value of the EG's software components. The record itself is signed by the TA-certified AIK.

Figure 2.6 shows a prototype EG, which has been implemented based on the Android smartphone platform. This platform has been selected for various reasons. Modern smartphones are equipped with a variety of sensors such as GPS, gyro sensor, electronic compass, proximity sensor, accelerometer, barometer, and ambient light sensor. Furthermore, photos, video and sound can be regarded and processed as event data. Moreover, Android is well-suited as a software platform for future embedded devices.

The TPM-anchored chain of trust is extended to the linux system and linux application layers by using the Integrity Measurement Architecture (IMA), which is integrated into any stock linux kernel as a kernel module. The Android application layer is based on libraries and the Dalvik Virtual Machine (VM). While the linux kernel layer can check the Android system libraries and the VM, Android applications run on top of the VM and are invisible to the kernel. Thus, we built a modified VM, which extends the chain of trust to the Android application level by computing the applications' checksums. A timestamp-based variant of remote attestation provided by the TPM is used for the verification of the node authenticity. All communication is based on the Trusted Network Connect (TNC) [36] protocol suite, which offers advanced security features, such as dedicated access control mechanisms for TPM-equipped nodes.

2.5.3 Use of a Trusted MIA in MASSIF

From the architectural perspective, the T-MIA implements a specific MASSIF Information Agent (MIA). A MIA is a software appliance residing in edge payload nodes. A MIA in MASSIF terminology [17] 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.

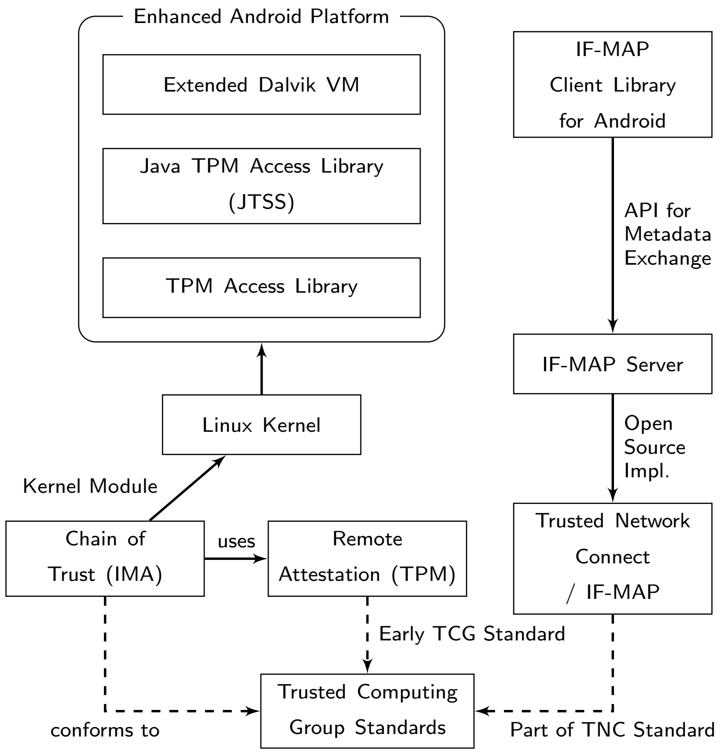

Figure 2.6: Technical building blocks

3 Resilient Event Bus

This chapter presents the design and some implementation details of the Resilient Event Bus (REB). The REB is mainly in charge of disseminating the events collected by the edge-MIS, after being pre-processed by the local Data services, towards the core-MIS. The core-MIS then uses some alternative communication mechanism to forward the events to the processing engine (or other core services). Less often, the REB may also need to transmit messages in the opposite direction, from the core network to the edges, for instance when a Reaction component needs to convey reconfiguration commands to the sensors. Consequently, the REB must provide a bidirectional communication path among the MIS, with the attribute that most of the traffic goes from the edge to the core.

The REB is organized as an overlay network built among MIS nodes, which are named generically as *REB nodes*. The communication in the overlay network is performed on top of the UDP/IP protocols, allowing the support of different network settings. REB uses application-level one-hop source routing to send messages towards the destination, instead of simply following the routes imposed by the network-level routing. It also takes advantage of coding techniques and the available redundancy of the network, such as when a node has multiple network connections (i.e., multihoming), to ensure that messages arrive securely and timely with a very high probability.

3.1 Overview

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. Accordingly, the REB design is influenced by the way SIEM systems are deployed, which are usually distributed over several facilities. A facility corresponds to a subset of the overall network, where a set of sensors provide event data to a local edge-MIS (left side of Figure 3.1) or where a group of machines implement the engine (and other supporting services) connected through the core-MIS (right side of Figure 3.1). A facility can therefore be modeled as a LAN, and the associated MIS can be seen as a routing device that receives the data produced locally and forwards it towards the final destination facility. The interconnection among the facilities can be abstracted as a WAN. One however should keep in mind that LAN and WAN are modeling artifacts, since in practice they will depend on the actual deployment of the SIEM. In one extreme case, the WAN can be the Internet, if the SIEM collects information from various offices of an organization that are located in different regions (of the same country or different countries). In the other extreme case, the WAN could be a set of switches with virtual LANs that interconnect a few PC racks on a data center. This organization has the virtue that entails no meaningful modification to the existing SIEM system and only requires the introduction of a REB node in each MIS at the border of a facility.

The communication among REB nodes is through the UDP/IP protocols. REB nodes can define

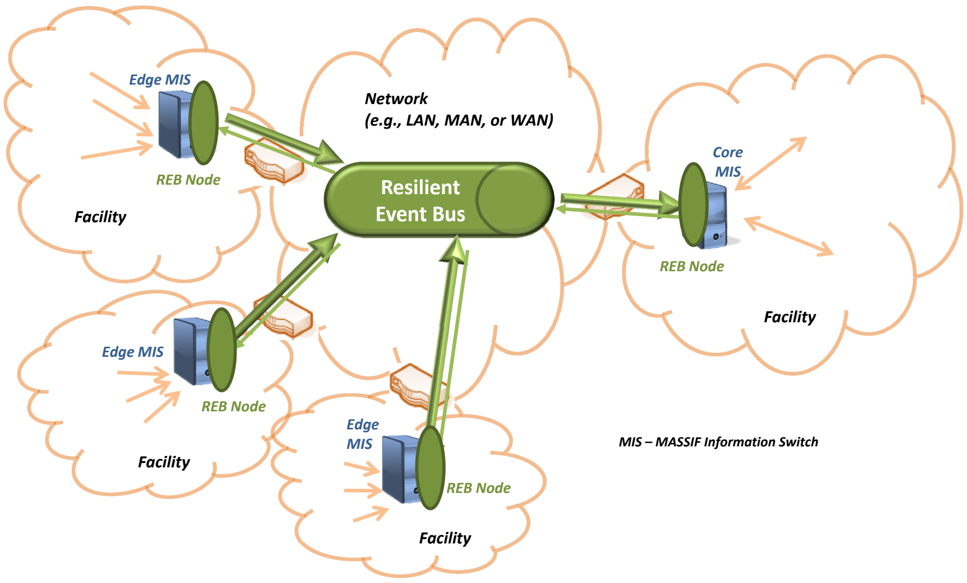

Figure 3.1: REB topological view.

an overlay network atop the IP network, and run application-level routing strategies to select overlay channels that are (expectedly) providing correct communication. Overlay networks have been used as mechanisms to implement routing schemes that take into account specific application requirements [4]. In MASSIF we want to employ overlay networks to create redundant network-agnostic channels for robust and timely communication, namely for event transport from the edge sensors to the core event correlation engine.

Depending on the network setting, we envision different kinds of faults that might preclude the transmission of data unless appropriate measures are enforced among the REB nodes. For instance, accidental or/and malicious faults can cause the corruption, loss, re-order and delay of packets. An adversary might try to modify the event data carried in a packet or add new events to prevent the detection of an intrusion. Congestion or denial of service (DOS) attacks can also make certain IP routers between specific REB nodes unresponsive, causing significant packet loss or the postponement of their delivery. A strong adversary might also be able to take control of one of the IP routers, allowing her (or him) the capability to perform sophisticated forms of man-in-the-middle attacks. REB nodes may also suffer failures, such as a crash or a compromise by an adversary. In this case, the REB per se will not solve the specific problem of the failed node[1], and consequently a facility might become disconnected. However, as a whole, the REB should continue to work properly allowing the remaining nodes to continue to exchange data.

Many of the above threats can be addressed with well known security mechanisms that have been applied to standard communication protocols, such as SSL/TLS [25] and IPsec [61]. For example, each pair of REB nodes shares a symmetric secret key. Leveraging from this key, one can add to every packet a Message Authentication Code (MAC) [56] that allows the integrity and authenticity of the arriving packets to be checked in an efficient way. Consequently, any packet that deviates from the expected, either because it was modified or was transmitted from a malicious source, can be identified and deleted.

[1]In the project we are addressing these failures in the context of node protection mechanisms, and later on in the deliverable we describe a solution for a highly resilient core-MIS.

This means that insertion attacks are simply avoided, and corruptions are transformed into erasures that can be recovered with retransmissions (or in our case, also with coding techniques). Optionally, one can also encrypt the packet contents, which guarantees confidentiality and prevents eavesdropping of event data.

Based on these mechanisms, the REB is able to reduce substantially the attack surface that might be exploited by an adversary. However, they only put the REB at the same level of resilience as standard solutions, which are insufficient to prevent certain (sometimes more advanced) attacks. For example, an adversary might thwart the correlation of two sources of events, if she (or he) is able to delay the packets that contain the events from one of the sources. This occurs because events are correlated within a predefined time window, and if they arrive in different windows the associated rule might not be activated. Alternatively, the adversary might perform a DOS in one of the IP routers that forwards the packets from one of the sources, causing very high transmission losses and the continuous retransmission of packets (to recover from the packet drops). The overall effect of the attack is once again the delay of the certain event packets, or eventually the temporary disconnection of one of the sources. Traditional solutions for secure communication, such as those based on SSL over TCP, are also vulnerable to other more specific attacks [11]. For instance, this approach is fragile to an adversary with access to the channel between the two communicating parties, since she (or he) can continuously abort the establishment of any TCP connection by sending Reset packets, creating in fact unavailability on the communications.

This sort of problems are addressed by the REB by employing some techniques that exploit distinct forms of redundancy. In particular, spatial and temporal redundancy is explored to attain high levels of robustness and timeliness.

Robustness The overlay network created among the REB nodes allows for multiple distinct routes (or paths) to be taken to transmit data to a specific destination. The source can, for instance, send the data directly or ask one of the other REB nodes to forward it to the destination. It is expected, in particular in large scale SIEM deployments, that these two paths will go through distinct physical links, and therefore, localized failures will only disrupt part of the communication. Based on this insight, the REB uses *multipath communication* to send data concurrently over several different paths in the overlay network. These routes consist either of direct paths between the source and destination nodes, or paths in which an intermediary node receives data from a source and redirects it to the destination. REB resorts to a one-hop source routing scheme. The overlay route of each message is defined at the sender (source routing), based on the local knowledge of the state of the links, and is composed of at most one intermediate relaying REB node (one-hop). The option of having a single hop, i.e., a single intermediate node, is due to the conclusion of Gummadi et al. [38] that there is no considerable benefit in using more hops.

A REB node can be connected to the WAN through one physical link or through multiple links. The later case is often observed in organizations that operate in critical sectors, such as in electrical production and distribution, as a way to increase their resilience to accidental failures by ensuring that the links are physically separated (they contract the network service to multiple Internet Service Providers (ISPs), and in some cases they go to the extent of using diverse network technologies, one wired and another wireless). If available, the REB also takes advantage of this form of redundancy by employing *multihoming communication*. Here, a node can either send or receive packets over any of the physical connections, thus increasing the number of possible paths that can be chosen when disseminating data to a certain destination. For example, if the sender and the receiver have two connections, then there are four alternative direct links between the two nodes instead of a single one. It is expected that at least in part these links fail in an independent way, which means they can be used to provided correct packet delivery under somewhat adverse network conditions.

Given a certain application message m, the REB could send a copy of it over k distinct paths. This would allow $k-1$ path failures to be tolerated, at the cost of the transmission of $k-1$ extra message replicas — the overhead is $(k-1)*m$. Depending on the message size, this cost can be significant especially when the network is behaving correctly (which is the expected normal case). Additionally, it also requires the receiver to process and discard $k-1$ duplicates, which can be a limitation in a SIEM system given the asymmetric data flow (remember that the core-MIS needs to receive all traffic coming from the edges). To address this difficulty, the REB employs *erasure coding techniques* to ensure that packet loss can be recovered at the receiver, at a much lower replication overhead. Basically, the sender needs to do some processing on the message m to produce some extra repair information r, and then $m+r$ is divided in several packets that are transmitted over various links. Even if only a subset of the message and repair data arrives, the receiver can still recover the original message.

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 REB 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. As we have seen, the overlay network can provide the necessary path redundancy to provide for timing fault-tolerance. Unfortunately, all overlay networks proposed in the past did not have this objective[2], and therefore, a specific solution had to be developed for the REB.

The REB uses fundamentally two mechanisms to achieve timeliness with high probability, despite being built on top of a best effort time-agnostic protocol such as IP. A sender REB node periodically *probes* each of its most promising paths to a given destination to derive a quality metric to be associated with the path. This metric is currently calculated based on the estimated latency and loss rate, but in the future we are considering other criteria such as the level of independence of this path in relation to the other ones (i.e., to what extent they share the same IP routers). Based on this metric, the sender can determine at each moment which are the best paths for a destination, and select them to disseminate a message based on its deadline.

On average, the above mechanism is able to address most timeliness problems. However, since the metric is calculated based on actual data collected from the network, it may require some period of time for the value of the metric to adjust after sudden changes in the network. During this period the sender could still think that a specific path is good, and consequently continue to use the path for transmissions, when in fact most packets could be lost. Moreover, a malicious REB node could attack the measurement process, for instance by making its paths look particularly good, and then suddenly start dropping all packets. In the REB, these failures end up being tolerated automatically by the erasure codes together with a sufficiently high level of multipath communication. The codes are able to recover from a reasonable number of arbitrary packet losses (in the original and/or

[2]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., [3, 73]).

repair data). Therefore, if one assumes that the failure only affects a limited number of channels, the remaining ones still deliver enough packets for the original message to be reconstructed.

3.2 Communication properties

SIEM systems are often built directly on top of the TCP protocol, or resort to SSL/TLS to augment TCP with security capabilities. Consequently, SIEM developers expect a communication substrate that provides a set of properties that matches those of TCP, since they greatly simplify the implementation of the system. For this reason, in addition to robustness and timeliness, REB was designed to grant most of TCP properties.

REB exports a relatively simple interface, offering point-to-point communication channels between nodes. Data is transmitted as a stream of bytes, and is delivered reliably in first-in first-out (FIFO) order. The arrival of duplicate data is identified and removed, and flow control is enforced at the senders to prevent receivers from being overwhelmed with too much information. REB also ensures data integrity and authenticity, something that TCP does not provide by itself, but that can be attained, for instance, with SSL/TLS. Confidentiality can also be guaranteed on an optional basis, depending on an indication by the applications.

Since the REB is implemented on top of UDP/IP, which does not have any of these properties, it is necessary to devise ways to implement them with a group of mechanisms. In the rest of this section, we explain generically how these properties are attained.

3.2.1 Authentication and error-free (and optional confidentiality)

SIEM applications call the REB interface to send messages to a certain destination. REB treats these messages as sets of bytes that are stored in a queue for transmission. Depending on the amount of queued data and on the maximum transmission unit (MTU) of the underlying network, it might be necessary to break the data first in segments and next in several packets that are forwarded independently (see Section 3.3). REB nodes disseminate the packets using multiple concurrent routes, which can be based on a single direct channel between the source and the destination, or can have a channel from the source to an intermediary node and then another channel to the final receiver. Assuming that the sender and the receiver are both correct, then attacks can occur both on the network and on the intermediary node (in case this node was compromised).

The initialization of the REB creates two shared cryptographic keys between every pair of nodes. The keys are used to protect the communications from attacks, supporting the authentication of nodes and the integrity/authenticity of the data. One of the keys is used to generate a MAC that is appended to every packet. MACs are verified at the receiver before packet delivery, and packets are discarded if their MACs do not match the expected values.

In alternative, one could choose to append a MAC to the whole segment and verify it only after the full reception. This solution would have the virtue of saving some MAC calculations, both at the sender and receiver, whenever segments are large. However, it suffers from a few drawbacks, making it less appealing in practice. First, the verification would be postponed, allowing corrupted packets to occupy space on the receiver buffers until much later. Second, the receiver node cannot separate good from bad packets, and therefore, a single damaged packet would cause the whole segment to be dropped (and then later retransmitted again).

On direct channels, only one MAC is generated per packet by the source, using the key shared with the destination node. On two-hop channels, a pair MACs is created, the first for the intermediary node and the second for the destination node. After receiving a packet, the intermediary validates and removes the second MAC and only then it forwards the packet to the destination.

Here, again, one could save a few MAC calculations if a single MAC was added independently of the type of route (i.e., direct or two-hop). However, since MACs are obtained relatively fast, we decided that it was better to have the capability to immediately identify and delete modified packets, both at the intermediary and final nodes. This feature is also helpful to determine if certain channels are under attack, since we can pinpoint with good precision where the fault occurred.

SIEM systems exchange mainly events and security information collected by the sensors. This data might be considered confidential in some organizations, while in others it can be of no concern. Since encrypting/decrypting every packet can introduce a reasonable performance penalty, and thus create delays that might affect timeliness, we decided to offer data confidentially on a optional basis. The application using the REB can indicate if it desires packets to be transmitted encrypted. In this case, the other key shared between the source and destination is employed for the encryption.

3.2.2 Reliable and timely data delivery

TCP messages are delivered to their destinations if both the source and destination processes are correct and if the underlying communication network provides a *fair-loss* delivery. The Internet is one of such networks, in the sense that it makes an effort to transmit a packet through a route and then delivers it to the destination. It may happen that a packet is lost in-transit due to congestion or a crash of a network router. TCP solves this issue by segmenting the input stream and by transmitting one segment at a time, while waiting for the arrival of an acknowledgment of their reception. A retransmission occurs if it appears that the segment was lost, e.g., when the retransmit timer expires at the sender. As each segment is received and acknowledged, the receiver delivers them to the application. TCP does not offer any guarantee about when a particular message will arrive, though, it tries to optimize the communication to enforce the property of eventual delivery (without any upper time bound).

In REB, the messages input by the application are saved in a queue, and then they are split into several segments for dissemination. Unlike TCP, retransmissions based on timers are avoided when possible so that the timeliness of a message is not affected. This is accomplished by preprocessing each segment with an encoder that applies an erasure code (also called a Forward Error Correction (FEC) code) to produce a number of packets. Depending on the code that is applied, if it is systematic or not, the resulting packets may contain the original data plus some repair information, or they may just have encoded data (see Section 3.5.3). The overall sum of the packets lengths is typically larger than the original segment, but it becomes feasible to reconstruct the segment even if some of the packets are lost.

The sender node disseminates the packets as they are produced by the encoder. It also starts a timer that should expire in case retransmissions have to be performed. However, retransmissions are typically avoided by adjusting the amount of repair capability of the code to the observed loss rate of the network. It may happen nevertheless that: (1) more packets end up being discarded, and therefore, the receiver is incapable of recovering the segment, or (2) the acknowledgement returned by the receiver is lost. When a retransmit timer expires, the sender encodes a few more packets based on the original segment (or selects some of the original packets), and forwards them to the destination. The expectation is that some of these packets will arrive at the receiver, allowing it to decode the segment. Once again, we need to initiate a new retransmit timer, now with a larger value.

The receiver accumulates the arriving packets in a receive queue. When enough packets of a given segment are available, the receiver attempts to decode them. In case of success, it returns an acknowledgement back to the sender. Otherwise, it waits for the arrival of an extra packet for this segment before trying again to decode. This process is repeated until the recovery of the original segment is accomplished. Depending on the network conditions, packets/segments may arrive and be decoded out of order. To address this issue, REB utilizes a selective acknowledgement scheme, to convey to the source information about which packets/segments have already arrived.

Communication in REB tries to be as timely as the network allows. By timely we mean that REB makes an effort to deliver data within the application defined deadline. This is important because events generated by the SIEM sensors maybe only useful for correlation if they arrive within their deadlines at the engine. The timeliness property of data delivery is provided by making use of multiple routes and some synchrony assumptions about the underlying network. Through the usage of a probing mechanism, it is possible to infer a quality metric about the overlay routes, which include estimated latency values. These latencies help selecting, based on the deadline of a transmitted message, the channels that are used to transmit the packets.

It can happen during particularly bad network conditions that it is not possible to deliver an application message within the specified deadline. Here, one could abort the transmission of this message, since it would probably be no longer of use to the receiver. This approach is however not the applicable to a SIEM. Consider the scenario where a sensor produces an event that should be processed within a certain time period. If it arrives late, the event might not be properly correlated, but in any case its delivery can be useful from a perspective of forensic analysis. Therefore, even if a deadline is violated, the REB will still attempts to deliver the message.

3.2.3 Ordered and duplication-free data delivery

The Internet does not ensure an ordered delivery of packets. This occurs because different packets, sent by one source, may experience distinct delays when transmitted through diverse routes. Additionally, there is the possibility of spurious transmission of duplicate packets, which often happens due to rerouting algorithms in intermediate nodes. Despite these difficulties, TCP provides FIFO ordering at the delivery and also removes duplicates. This is accomplished by assigning to each segment a sequence number, which is used on the receiver side to order the segments. Furthermore, the sequence numbers are used to detect and discard duplicate segments.

In REB, the transmission of a segment corresponds to the dissemination of a fixed number of packets, which encode part of the original segment data and some redundant bytes. At the destination, the receiver decodes the original segment from a subset of the received packets.

Each segment has an associated sequence number (with 24-bits) that is incremented monotonically. These sequence numbers are employed to keep track of lost segments and to detect data duplication (accidental or from replay attacks). Packets also have a distinct sequence number (16-bits), which increases monotonically per segment (starting with 1). Therefore, a packet is univocally identified by carrying in the header a pair composed of the sequence number of the segment it belongs to plus its sequence number. Packets that arrive with the same identifiers are detected and removed as duplicates.

REB enforces FIFO order with the segment sequence number. A receiver node can deliver data in the right order by buffering complete unordered segments until all their predecessors have been given to the application. The amount of unordered data that is maintained in a REB node is managed through a receiver sliding window flow control mechanism. This mechanism prevents the receiver from accumulating too much unordered data, something that could be used by an malicious REB node to overflow the memory of the receiver. The sender is informed of how much empty space is still available inside the window, and packets that arrive beyond this space are discarded. The flow control mechanism is addressed in detail on Section 3.5.6.

We realize that there is a potential for an adverse effect caused by the simultaneous use of FIFO ordering and the assignment of deadlines to messages. The issue lies in the fact that a message with a deadline shorter than the one from its predecessor message, could eventually fail to be delivered on time due to the FIFO discipline. For example, it could happen that all segments comprising the second message are fully received before the segments of the first message. Since the receiver node buffers the unordered complete segments until all their predecessors arrive, the second message would stay waiting in the queue and eventually have its deadline expire. One possible solution for this problem could be to use the deadlines to order the messages (which would become message priority values), but it is not entirely clear how issues such as message starvation would be resolved in a setting were REB nodes can be malicious. Our current approach is to keep the strict enforcement of the FIFO discipline, and as we get more experience with using REB, we may need to revisit this issue later on.

3.3 Sending and receiving data

The procedure of sending/receiving messages in REB encompasses a number of steps in order to enforce the various properties discussed in the previous sections, namely robust and timely guaranties with a simple to use application level interface. Since REB nodes are organized as an overlay network, the sender typically has several routes available for data transmission, which in most cases are expected to correspond to disjoint physical links. It is anticipated that most of these routes will fail independently with a reasonable probability. By forwarding data over a subset of the routes concurrently, the REB is able to tolerate some kinds of failures in a transparent way, and at the same time support the distribution of the network load over the alternative paths. Routes are selected using a metric that takes into consideration the observed Round Trip Time (RTT) and loss rate in the recent past. The REB also uses erasure coding to create multiple blocks of data from the messages, which have in total a slightly larger size than the original message, and that are transmitted individually through the chosen routes.

The transmission process, which implements how individual messages are delivered from a source to a destination, is illustrated in more detail in Figure 3.2. When an application needs to send a message, it calls the REB interface and provides a buffer with the data, indicates the destination node and gives a deadline for the delivery (see also Table 3.4). Multiple calls can occur concurrently if the application has more than one thread. When the call returns, the application can re-use the buffer because it either has been transmitted or its contents have been locally copied to a send queue (named *segment queue*).

The REB maintains multiple segment queues, one per each active destination. Messages are copied to the queues in FIFO order. The procedure for selecting data from a queue for transmission involves the following steps:

1. If there is data to be transmitted, search the various segment queues for the one with the message with the shortest deadline. Otherwise, keep on performing the other tasks (such as probing the links);

2. Extract from the queue a segment of data:

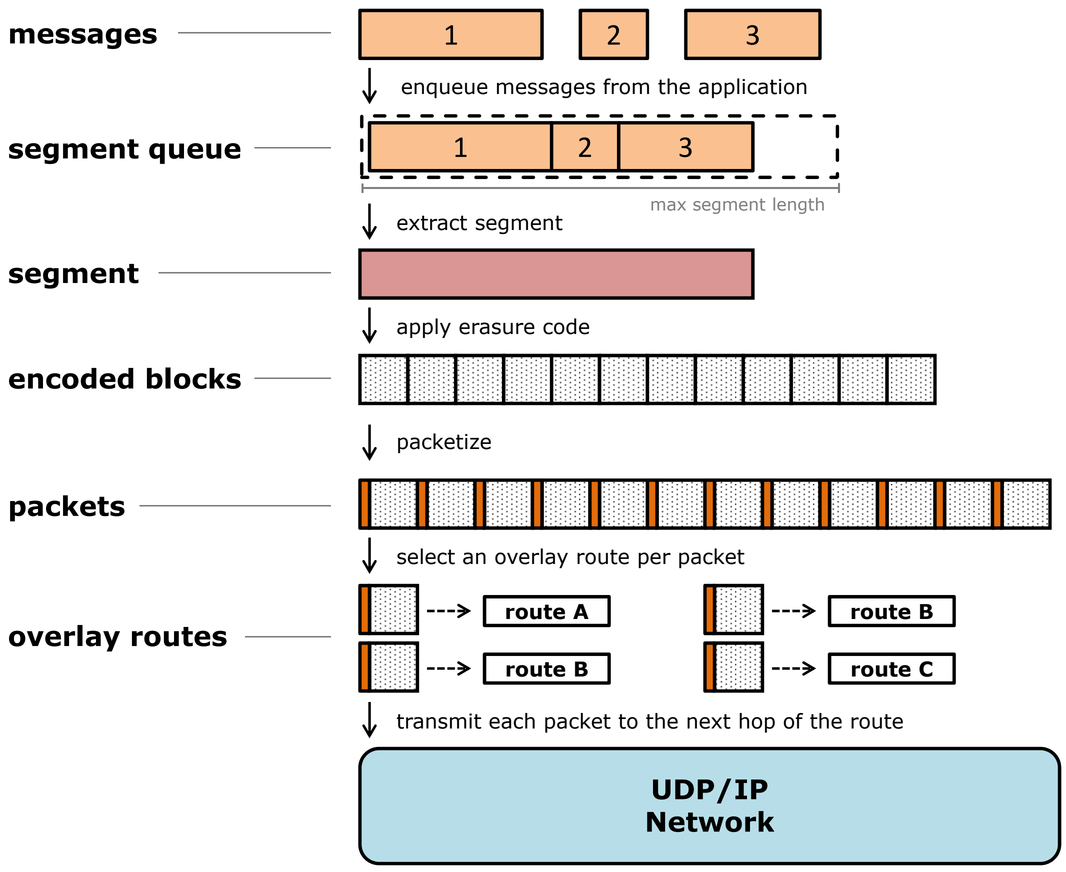

Figure 3.2: The data transmission process at a REB node.

(a) If the queued data has a size larger than the *maximum segment length*, then create a segment with this length;

(b) If the queued data has a size larger than the MTU (minus the headers) but less than the maximum segment length, then create a segment with the available data;

(c) If the queued data is small (less than one MTU minus the headers), then (i.) use all data from this queue if there is no previously transmitted segment for the same destination that is still waiting for an acknowledgement; (ii.) otherwise, skip this queue and move to the next one.

The queue is mainly used for two purposes. First, in case the network is occupied with the transmission of older messages, a local copy is created so that the send operation can return, allowing the application to continue to run. However, if a message is too large and does not fit entirely inside the queue, it is split so that part of it fills the queue and the rest is scheduled to be inserted in when there is space again. Second, if the application normally sends small messages, the queue is utilized to accumulate more bytes. Ideally, one would like to increase efficiency by transmitting packets with a size approximately equal to the MTU of the underlying network. Additionally, the coding algorithms require a reasonable amount of data for optimal execution. This amount of data is named the *maximum segment length* and its value has to take into consideration the size of the headers, the type of code and the network MTU between the sender and receiver. To prevent the creation of many small segments, but at the same time avoid delaying a message indefinitely, the procedure only allows a small segment to be transmitted concurrently to a certain destination (rule 2.c).

The extracted segment is then encoded by applying an erasure algorithm. The algorithm divides the segment in multiple blocks, and then processes them to produce the encoded blocks. If the algorithm is systematic (e.g., with Raptor codes [71]), then the first encoded blocks correspond exactly to the original segment and the remaining blocks contain repair data. When needed, the repair data is used at the receiver to reconstruct the missing blocks. On the other hand, in a non-systematic code (e.g., the LT code [48]) all encoded blocks have a mix of several blocks of the original data. If the segment is very small, it might be difficult to apply the most sophisticated algorithms, since often they are optimized for larger data sizes. In this case, it is more efficient to place the whole segment in a block, which is then replicated enough times to tolerate a certain number of failures.

Encoded blocks are then packetized by adding a header that contains, among other fields, the identification of the packet (sequence numbers of the segment and packet) and the final receiver. In order to maximize the reconstruction capability of the code, each encoded block should be placed in a distinct packet. This ensures that a packet drop in the network only affects one block, allowing the remaining ones to recover the lost data. In some cases, for efficiency reasons, it makes sense to include more than one encoded block per packet. This is only valid if two (or more) blocks fit inside the MTU of the network. This optimization should be utilized however with some care because it weakens the failure independence assumption on which erasure codes are based.

The sender next looks at the available routes to the destination, and selects a few of them that have a good figure of merit, allowing the segment to arrive within the deadline. A MAC is added to the header to let the final receiver detect integrity violations. In case an intermediary node needs to forward the packet, then a second MAC is also appended. The packet is then sent over the corresponding overlay channel, which translates in a transmission though the underlying network using UDP over IP.

On the receiver side, the node collects the packets arriving from the various channels as represented in Figure 3.3. The node is prepared to receive only a subset of the packets, since some of them may be lost in the network, while others can be corrupted. This second problem is detected with the included MAC, and the corresponding packet is deleted. The MAC is also employed for the protection of attacks where an external adversary (not controlling a REB node) may generate malicious packets, for instance, to confuse the segment reconstruction procedure or to make a DoS. Since the adversary does not share a key with the REB node, she (or he) is incapable of producing packets with the right MAC.

DoS attacks performed by a malicious REB node are addressed with a flow control mechanism (see Section 3.5.6 for details). When requested or piggybacked in the acknowledgements, the receiver indicates to the sender the amount of bytes that can be transmitted and that fall within the associated receive queue. Packets arriving when the queue is full are simply discarded. Additionally, packets with identifiers outside the expected range are also dropped, therefore, averting attacks that try to exhaust the memory by extending the local queues.

Packets can be received for the local node or to be forwarded to some other destination. In the first case, a few correctness checks are carried out, including duplication removal, and then the header is removed to obtain the encoded block. Next, the block is stored in the receive queue associated with the sender. Alternatively, when the node acts as a two-hop router, the packet is also checked and then enqueued to be transmitted to the final destination. To thwart starvation attacks caused by a malicious sender node, where it could try to delay the transmissions of this node, the packet is put at the end of the queue to wait for its turn.

Each encoded block is assembled accordingly to the particular segment that it belongs to. Since we are using erasure codes, we try to decode as soon as enough blocks have arrived. However, sometimes, it

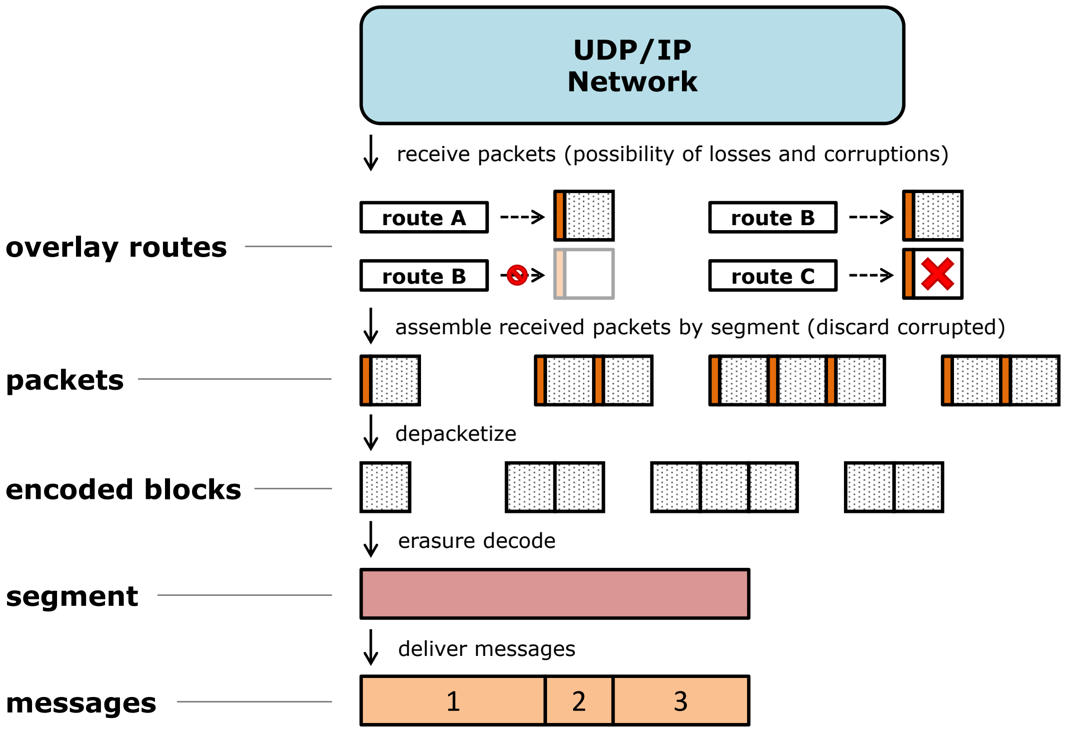

Figure 3.3: Scheme with the data receiving process.

may happen that it is impossible to reconstruct the segment with the available blocks. When this happens, the receiver needs to wait for the arrival of more blocks, and then perform the decoding operation again. As segments are decoded, they are stored in the receive queue waiting for the application to read them.

Initialization/Finalization calls	
init(local_id)	Initializes the REB object for the local node, on which the remaining calls are invoked. The parameter `local_id` indicates the textual identifier of the local node.
destroy()	Closes all communications and releases resources used by the REB object.
Communication calls	
send(buffer, size, destination, deadline)	Sends a message with `size` bytes from the data `buffer` to the specified `destination` node. The `deadline` value is a time value that is used to indicate the urgency of the message delivery.
receive(buffer, size, source)	Receives a message with `size` bytes and stores it in the provided `buffer`. The `source` value may indicate receipt from a specific node or from any source node (when a wildcard is provided).
setEncryptedMode(mode)	Sets the encryption mode for the communication. The `mode` value may be either *on* or *off*. The mode is changed when all pending data has finished being sent.
Information calls	
getLocalNodeID()	Returns the ID of the local node.
getLocalNodeAddresses()	Returns a list with all the IP socket addresses from the local node.
getRemoteNodesIDs()	Returns a list with the IDs of the remote nodes.
getRemoteNodeAddresses(id)	Returns a list with all the IP socket addresses from the remote node with the specified ID.
getRemoteNodeID(address)	Returns the ID of the remote node with the specified IP socket address.

Figure 3.4: Interface to the applications offered by the REB.

3.4 REB interface

REB is implemented as a Java library that can be linked with an application. It offers a relatively simple interface that contains the fundamental operations for transmitting data and some auxiliary methods for the application to collect information about the system. Figure 3.4 gives an overview of the main operations of REB.

3.5 Communication mechanisms

This section describes some setup aspects and various mechanisms utilized by the REB. It offers an explanation about the erasure codes, as well as how multihoming contributes to the overall robustness of the communication. A mechanism for route probing is presented, supporting the inference of a quality metric for individual routes. This quality metric is then used by a route selection algorithm that is also discussed. To finish, we explain how the REB manages flow control.

3.5.1 Overlay network configuration and setup

The current version of the REB uses a static configuration for the overlay that defines the set of nodes that may participate in the communications (some of them may be down or disconnected). When a REB node starts up it is assigned an unique identifier, which is referred to as the *local ID*. The identifier is provided by the application that calls the startup interface of the REB (see Figure 3.4).

Based on this local ID, a node can get the information about the whole overlay network by reading a few configuration files. The files are put in a predefined place in the local machine by the administrator of the SIEM. The following information can be obtained:

Network addresses A REB node receives packets in a specific IP address and UDP port. The ports can be different across the overlay, depending on the machine where the node is located. If a machine has multihoming, then several IP addresses are assigned, one for each physical connection. When this happens, the configuration file has the list of IP addresses that can be used;

Pair-wise shared keys Every pair of nodes shares a secret cryptographic key for secure communication. Each key is stored in a separate file that is statically distributed to the relevant nodes and not shared by any third party.

At start-up, a REB node is connected to no one. A connection is only established when the application requests a message to be transmitted to a specific destination node. Establishing a connection consists in resetting any previous state and in setting up two session keys. The former is necessary because nodes are allowed to crash and then later to be reinstated in the overlay. It could occur that the destination node had been exchanging packets with the source, and then the source had to reboot. In this case, during the connection establishment, the destination node would become aware of the problem, and therefore, it would clean information maintained on behalf of the source[3].

The handshake protocol that is executed between the two nodes is relatively simple. It consists of three messages protected with the shared key (with a MAC and encrypted). In the first message, the initiator indicates that it wants to create a connection by including a nonce N1 and the local IDs of both parties ($< INIT, ID_{initiator}, ID_{destination}, N1 >$). The destination node responds with a second message, which has a new nonce N2 ($< INIT\_RESP, ID_{initiator}, ID_{destination}, N1, N2 >$), and that allows the initiator to authenticate the destination. The last messages is transmitted by the initiator, to support its authentication at the destination ($< RESP, ID_{initiator}, ID_{destination}, N2, N1, conf\_info >$). The destination node only resets the connection when the $RESP$ is correctly received, and it uses information in $conf\_info$ to determine exactly how this operation should be performed. Nonces are created by concatenating two values, a locally generated random number with a high resolution timestamp (up to the microsecond).

It can happen that one of the handshake messages is lost in the network. To address this problem, the sender of the message is responsible for its retransmission until the other side responds. So, for instance, if message $INIT$ is dropped, the initiator should periodically resend it until the following handshake message arrives. If the destination node sees a duplicate $INIT$, this could either indicate that $INIT\_RESP$ was lost or delayed. In this case, it simply waits for its retransmission timer to expire,

[3]The cleaning includes discarding pending packets and out of order segments, and other management data. Completely reconstructed segments that are stored in order in the receive buffer cannot be deleted because the sender might have the expectation that they will eventually be delivered to the application. This is important in the case that the connection needs to be reestablished when the segment sequence number reaches its maximum value.

which would cause $INIT\_RESP$ to be resent, or for the arrival of $RESP$ that would conclude the handshake.

The loss of the last message, $RESP$, is recovered in a slightly different manner. From the point of view of the destination node, it cannot distinguish the case when $INIT\_RESP$ or $RESP$ are dropped by the network. Therefore, it keeps retransmitting $INIT\_RESP$ until a $RESP$ arrives. At the initiator, when a duplicate $INIT\_RESP$ is received, it resends the $RESP$ message. In all situations, handshake messages are only retransmitted a certain pre-defined number of times. When the limit is reached, and the connection attempt is aborted and an error is stored in a log file.

When the overlay is being setup, it may occur that two nodes attempt to start a connection simultaneously. In this case, both would send concurrently the $INIT$ message, and both would receive the peer's $INIT$ as the response. To solve this issue, we use a simple arbitration procedure, where the node with the largest ID aborts its connection attempt, and follows the handshake launched by the other side by responding with a $INIT\_RESP$.

An adversary could take advantage of the handshake protocol to attack the REB communications. For example, she could replay an old $INIT$ message to fake a reboot of the initiator to force a connection reset. Since in general the destination node cannot distinguish a replay from a valid connection attempt, it starts the handshake protocol by responding with the $INIT\_RESP$. However, it continues to process the packets arriving from the initiator as usual, ignoring for now the $INIT$. Since the adversary does not know the shared key, she can not produce a valid $RESP$. Therefore, after a number of $INIT\_RESP$ retries, the destination node forgets about the connection.

In another example attack, the initiator could send an $INIT$ message, and then the adversary could replay an old $INIT$ of the node. As a consequence, the destination would receive two valid but different $INIT$ messages from the same node. In this case, we again use a simple arbitration procedure, where the handshake corresponding to the $INIT$ carrying the nonce with the largest timestamp is the one that is executed, and the other is disregarded[4].

As a final attack example, when a node initiates a connection to a destination, where the $ID_{initiator} > ID_{destination}$, the adversary could replay an old $INIT_{old}$ message from the destination to the initiator. When this happens, the initiator applies the arbitration procedure for a simultaneous connection, and stops its handshake. Then, one of two things can happen. First, the destination node receives the original $INIT_{original}$ and responds accordingly with the $INIT\_RESP$. As the initiator gets the $INIT\_RESP$, it checks that the message carries the $N1$ from its $INIT_{original}$ and a $N2$ with a timestamp larger than the one in $N1$ from $INIT_{old}$. This provides evidence that an attack occurred, and therefore, the initiator returns to the original handshake and completes it with a $RESP$. Second, the adversary could also remove the $INIT\_RESP$ and all its retransmissions, meaning that it has complete control of the routes between the two nodes. In this case, the initiator will retransmit the $INIT\_RESP$ corresponding to $INIT_{old}$ a number of times and then abort the connection, which an appropriate action given the attack power of the adversary.

A node simply ignores a message for which the corresponding previous handshake message was not received (if a $INIT\_RESP$ or a $RESP$ arrives without having sent the $INIT$ or $INIT\_RESP$, respectively). Additionally, since a node only resends a message a predefined number of times, this prevents denial of service attacks where an adversary keeps replaying the most recent handshake messages (either $INIT$ or $INIT\_RESP$) to force the destination to perform retransmissions.

[4]Here, we are working under the assumption that the adversary can not change the clock of the initiator to a later time, and then collect an $INIT$ message with a larger timestamp. If this assumption is violated, then a new shared key needs to be setup between the two nodes by the SIEM administrator. Notice, however, that even in this case the adversary cannot complete the handshake protocol and deceive the two parties.

At the end of the handshake, the two session keys are produced to secure the communication between the nodes. One key provides authentication and data integrity by being used to generate the MACs included in every transmitted packet, and the other key provides confidentiality by encrypting data. The formula for producing the keys is the following:

$$K_{MAC} = hash(SharedKey, N1, N2, ID_{smaller}, ID_{larger}, ''M'')$$

$$K_{Encryption} = hash(SharedKey, N1, N2, ID_{smaller}, ID_{larger}, ''E'')$$

In the formulas, $SharedKey$ is the preconfigured shared key between the nodes, $N1$ and $N2$ are the nonces from the handshake, and ID are the identifiers of the nodes. The identifiers are placed in a deterministic order, first the smaller ID, so that both sides produce the same keys. Strings “M” and “E” are used to differentiate the two keys.

The new keys substitute the old ones when the handshake finishes. Packets from the previous connection may still be in transit when this occurs. These packets will be discarded upon arrival because the MACs are no longer valid. This mechanism has the benefit that prevents packets from an older session from confusing the receiving algorithms.

3.5.2 Multiple paths and multihoming

A REB node can typically reach a destination through many different overlay routes. If the REB is able to determine which routes are behaving erroneously, and picks alternative paths for data transmissions, it is possible to tolerate failures in the network. Of course, these measures are only effective if the failures do not completely cut all communications.

Since REB uses one-hop source routing, the available paths are the following: first, there is the direct link from the sender to the receiver; second, since any other node can act as an intermediate router, each one of them defines an extra path. Overall, in a REB deployment with n nodes, there are at least $n - 1$ paths connecting every pair of nodes.

Facilities may be interconnected via multihoming, i.e., by two or more distinct physical links (e.g., a REB node could have a pair of network interfaces and would be connected through two different ISPs). Since these links usually only share a minimum amount of resources, their failure can be considered independent in many scenarios (e.g., a DoS is performed in one of the ISPs). REB takes advantage of multihoming to increase the number of available overlay paths, allowing a node to overcome the failure of one of its links by exploring alternative routes.

The number of network interfaces directly influences the quantity of overlay paths a local node may have at its disposal. Figure 3.5 shows an example overlay network configuration that makes use of multihoming in order to provide a diversity of links. REB nodes with identifiers ID1 and ID3 have a single network interface, while nodes ID2 and ID4 have two interfaces. Therefore, node ID2 can reach node ID1 through two direct links, and can send packets to node ID4 over four direct paths.

The exact number of overlay routes that exist can be calculated in the following way. Let y be the number of interfaces on a certain local node and w the number of interfaces on another remote node; in total, there are $y \times w$ available direct paths between the nodes. Two-hop paths include an extra intermediary node and make use of the same interfaces as in the direct paths, as well as the interfaces of the intermediate node (the packet can arrive in one interface and then leave from any of the available network interfaces). Let z_i be the number of interfaces on a certain intermediary node. Then, the number

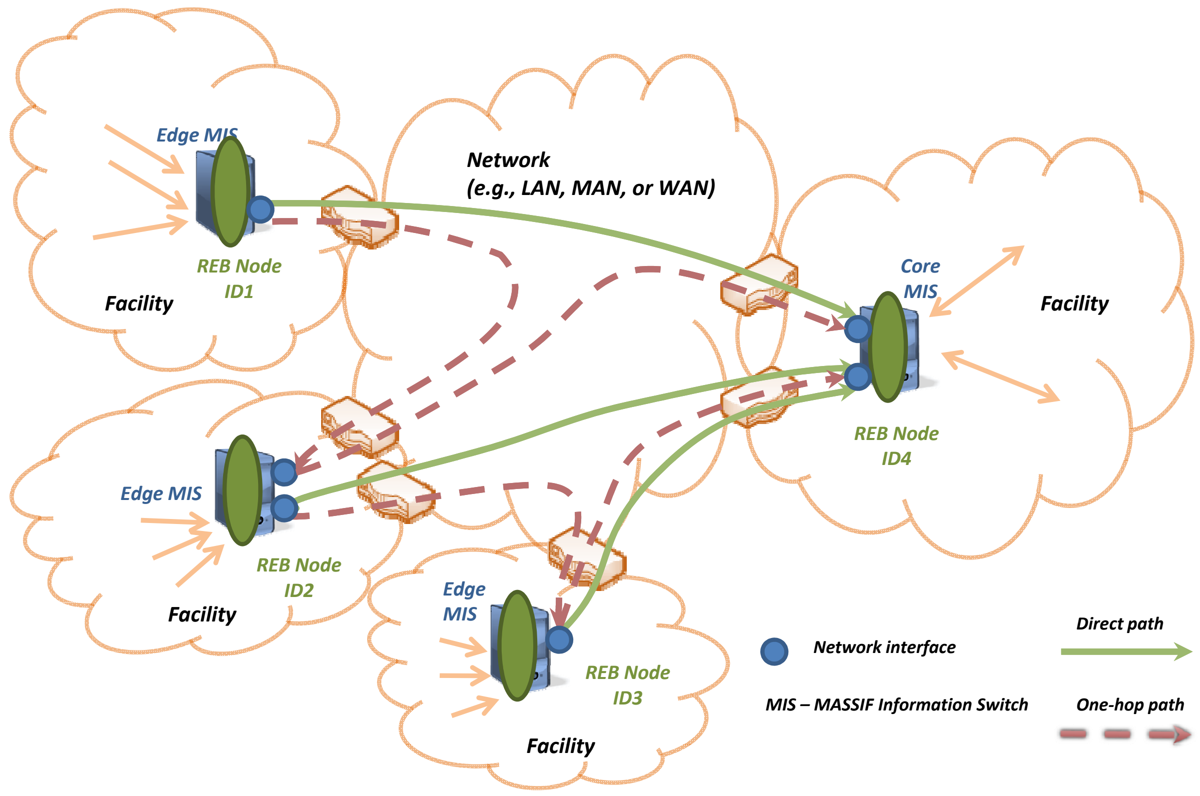

Figure 3.5: Multihoming in REB.

of paths that can go through this intermediate are $y \times w \times z_i^2$. In total, the number of available paths between the two nodes are:

$$Total\ paths = \sum_{i=1}^{n-2} y \times w \times z_i^2 + y \times w$$

Returning to the example from the figure, the total number of paths between nodes ID1 and ID4 is equal to 12. There are two direct paths; eight two-hop paths through node ID2; and two two-hop paths through node ID3. For overlays with many nodes, the growth in number of paths could become challenge if all of them were used in the communications. However, only a subset of the paths is actually employed by a node to transmit data, and these are picked based on their quality metric.

3.5.3 Multipath transmission and erasure codes

A REB node sends packets concurrently over a few of the available channels to the destination. However, data transmission through multiple channels per se is not sufficient to achieve robustness in the communications. In fact, even a single channel behaving erroneously (e.g., losing packets) is enough to prevent the original data from being reconstructed. Therefore, using multiple concurrent channels can actually degrade the reliability of the whole communication.

Two possible approaches to recover from losses are (1) the retransmission of the packets at a later time, or (2) the concurrent transmission of several copies of the packets over different channels. The first solution has the advantage of minimizing the amount of data that is send, at the cost of delaying the delivery of the packets (since retransmissions occur after a timeout). The second approach has the opposite characteristics.

In REB, we use erasure codes to both decrease the amount of transmitted data and to minimize the delays in case of losses. Before disseminating a segment, it is split into k equal sized blocks. These blocks are then encoded to generate N encoded blocks. The encoding process ensures, with very high probability, that the reception of any K blocks is enough for the recovery of the full data, where K is slightly larger than k (and less than N). Therefore, if there is a limited quantity of losses in the network (less than $N - K$ blocks are dropped), then it is possible to reconstruct the original segment in a timely manner, without requiring retransmissions. However, it may happen that the network is behaving worse than expected, and only a less than K blocks arrive. In this case, further communication will be required, either by retransmitting some of the encoded blocks, or by producing and sending a few extra encoded blocks[5].

REB currently uses Fountain Codes [51, 49], or rateless erasure codes, that can infinitely encode the data. In our current implementation, we resort to LT Codes [49], but in the future we intend to replace these by the newer and more efficient RaptorQ Codes [50]. In LT codes, an encoded block is created by XORing a few of the original blocks. Two pseudo-random functions are called, one to determine the number of blocks that should be XORed, and the other to select the actual blocks. Decoding is performed gradually, as the encoded blocks arrive at the receiver. In each step, it is checked if the new encoded block allows the recovery of an original block, and if this is the case, this knowledge is further propagated to decode other blocks. Eventually, when enough encoded blocks reach the receiver, it is possible to reconstruct the segment.

At the source, if sufficient data is available in the send queue, then the segment size is selected in such a way that every encoded block completely fills a packet. This means that an encoded block should have EB_{lenght} bytes, which is equal to MTU minus the size of the headers (added by the REB, UDP and IP). Since the original blocks have the same dimension as the encoded ones, then the segment size should be $k \times EB_{lenght}$. This ensures an efficient utilization of the network, and also that a packet drop only affects a single encoded block, something that is typically assumed by the codes.

Erasure codes are, however, able to recover from bursts of encoded block losses. Therefore, if the segment is small, one can put a few encoded blocks in the same packet to reach a dimension similar to the MTU (note that the pseudo-random functions used by the LT codes are devised for particular values of k, and therefore it is not possible to simply decrease k without affecting the properties of the code). This way one can keep the network efficiency, at the cost of potentially losing more encoded blocks. For tiny segments that fit in a single MTU, there is no advantage of employing coding algorithms. In this case, we simply replicate the segment over a few packets and send them concurrently.

At the destination, the arrival of the encoded blocks triggers the decoding process, and when the whole data is decoded, a confirmation is sent back to the source node. If decoding is unsuccessful, then the source is responsible for retransmitting some of the missing encoded blocks. This makes the delivery reliable since it is assured to happen in the presence of correct processes and a best-effort network (like the Internet).

[5]The exact solution depends on the erasure code being used. The second approach that generates new encoded blocks has the virtue that the sender does not have to know which blocks arrived correctly.

3.5.4 Segment and packet identification

Segments have to be delivered in FIFO order to the application, so it is necessary to univocally identify them to allow a proper organization at the receiver, in case they arrive or are decoded out of order. When transmitting a segment, it is also required to identify each packet to determine which encoded blocks have been received. As a result, each packet carries a unique ID on its header that identifies the segment the enclosed blocks belong to, as well as their position inside the segment.

A packet ID comprises a concatenation of two sequence numbers, resulting in a total length of 40 bits. Out of those 40 bits, the first 24 represent the segment sequence number and the remaining 16 correspond to the packet sequence number inside the segment. Both sequence numbers start at 1 and are incremented monotonically, having an upper bound of $2^{24} - 1$ and $2^{16} - 1$, respectively.

The first upper bound, for the segment sequence number, is expected to overflow eventually if the REB is used over long periods of time. When that happens it is necessary to reset the connection between the source and destination nodes, so that these sequence numbers may be safely reused without the danger of introducing corruption of data caused by replay attacks. Basically, after the connection is reestablished, new shared keys are derived, and therefore packets carrying the previous sequence numbers will not be accepted as the MAC is invalid. The second upper bound, for the packet sequence number, is not expected to overflow because REB limits the number of encoded blocks generated from a segment.

The space size for the segment sequence number was defined using the information provided by the MASSIF use case scenarios [15]. For example, the Olympic Games scenario, at its peak load, produces around 12 million events per day before aggregation. If these events were to be transmitted in separate segments, something highly unlikely because there is typically aggregation of events at the collectors, then the connection would only have to be reestablished with a frequency less than once a day.

3.5.5 Acknowledgments and Retransmissions

During a transmission, the destination node receives the packets and stores the data in memory until enough encoded blocks are available for decoding. After the successful segment reconstruction, the node sends an acknowledgment back to the source and makes the segment readable to the application.

The usage of erasure coding and multiple overlay routes offers a good level of robustness against lost packets. Despite this, the achieved reliability has a probabilistic nature, meaning there is always a (small) chance that the segment cannot be rebuilt at the destination. One reason for such failure is an insufficient number of encoded blocks reaching the destination node, caused by too many packets being dropped in transit. Another reason is that even when K encoded blocks arrive, the decoding algorithm might be incapable of recalculating the original data (with a small probability). To overcome these problems, the source needs to retransmit packets if it does not receive an acknowledgment within a predetermined time.

The network may reorder the packets being disseminated. Additionally, the use of multiple routes in the overlay may also contribute to the arrival of out of order packets, as the paths can have diverse transmission times. Therefore, as unordered packet arrival will potentially occur often, it is advisable that the destination node keeps these packets in buffers instead of dropping them. In REB, a separate buffer is managed for each source at the receiver, and is named *receive queue*.

The receive queue has a fixed length, and allows for the storage of several segments. It can contain segments completely decoded and packets belonging to various segments that have been partially received (see an example in Figure 3.6). As the application reads the stored data (in FIFO order), the

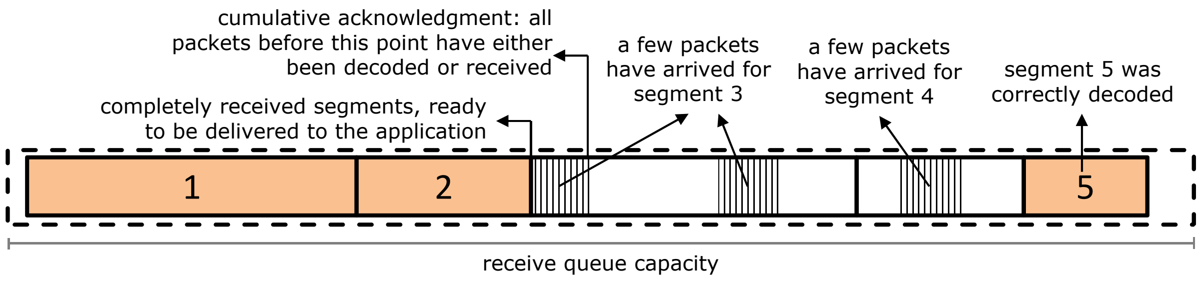
cumulative acknowledgment: all packets before this point have either been decoded or received
a few packets have arrived for segment 3
a few packets have arrived for segment 4
segment 5 was correctly decoded
completely received segments, ready to be delivered to the application
1
2
5
receive queue capacity

Figure 3.6: Example of a receive queue.

corresponding space is freed so that further packets can be accommodated. Packets are placed in the queue in the right place accordingly to their identifier (segment sequence number + packet sequence number), and consequently there might holes in the queue so that out of order packets can be added at a later time. Packets that would need to be stored beyond the queue limit are simply dropped (as the queue is not extended to save them; however, as the flow control mechanism is used exactly to prevent these packets from being transmitted, this can only occur if the sender is malicious).

REB informs the source about which packets/segments have been correctly received with a *selective acknowledgment (SACK)* (a mechanism somewhat inspired in TCP [55]). A SACK contains for each partially received segment a list of pairs of packet IDs, which define continuous intervals of stored packets (e.g., if only packets five to nine have arrived of a certain segment S, then the range is $[S.5, S.9]$). To avoid always having to explicitly define the intervals for previously completed segments, the first pair of IDs represents a *cumulative acknowledgment* that confirms the reception of all previous segments and packets (including the indicated packets itself). This optimization is possible because segment sequence numbers are incremented monotonically and cannot be reused during a session. In the queue of the example figure, the cumulative acknowledgement appears at the end of the first packets of segment 3.

REB implements the following few simple rules with regard to the management of the receive queue and the return of acknowledgements:

1. The receive queue places the packets/segments in their expected position with respect to their identifiers.
2. The receive queue does not store packet/segments beyond its maximum size. Therefore, out of bound packets are dropped.
3. The receive queue never garbage collects packets of the segment that is currently being received, and that have been acknowledged to the source.
4. The receive queue can garbage collect packets and segments that are beyond the segment currently being received, even if they had been acknowledged to the source.
5. Selective acknowledgments may only confirm the reception of part of packets/segments in the queue as there is limited space in a SACK packet. The packets/segments that should be acknowledged are the ones that appear first in the receive queue.

In the example of Figure 3.6, the segment that is currently being received is 3. Therefore, its packets will never be garbage collected. The saved packets for segment 5 or even segment 5 may be removed if the node needs to reclaim their space.

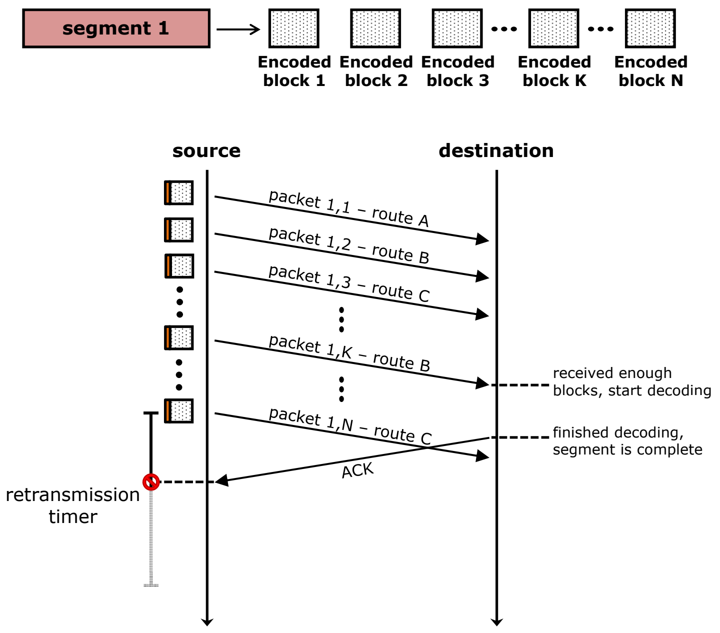

Figure 3.7: The common scenario for transmissions.

The source node starts a timer with a predetermined duration after the transmission of the last packet of the segment (in the flow control section, we will look into the case where there are more segments to be send). Ideally, the duration is defined in such a way that allows for the receiver to decode the segment, and return the acknowledgement. Figure 3.7 shows an example of a transmission in this normal scenario.

If the timer expires before an acknowledgment is received, then the network or the receiver had a problem. For example, some of the data packets were dropped and the segment could not be reconstructed, or there were delays in the network or/and receiver that made the acknowledgment arrive later. Since the source does not know the cause of the problem, it sends a special packet with no data (referred to as a *ping packet*) to the destination node to ask about which packets have been received so far. When the destination node receives a ping, it immediately sends a SACK back to the source. The ping packets are transmitted periodically, until either an acknowledgment arrives or the sender gives up (and returns an error). With the reception of the SACK, the source node starts to retransmit all missing packets. Figure 3.8 show an example scenario of a retransmission caused by lost packets.

Following the example of TCP, the value of the retransmission timer of REB is calculated based on the estimated RTTs and variations [63]. However, in REB one needs to account for multiple overlay routes, which have independent quality of service metrics. Since a node keeps information about each overlay route RTT as well as an average of the RTT variation, it is possible to use this information to select a reasonable value for the timer.

In REB, it was decided to take a conservative approach for the calculation of the timeout. We use the worse values for the expected RTT and RTT variation of all the paths were packets were transmitted. Additionally, since the decoding process at the receiver may take a non-negligible interval, the calculation of the retransmission timer also takes into account an estimation of this period. The decoding time depends on the type of erasure code being used, and specifically on its decoding algorithm complexity. To simplify the estimation, we assume that it takes approximately the same interval to decode as to encode, and therefore we measure its value at the sender. As we get more experience in using the REB,

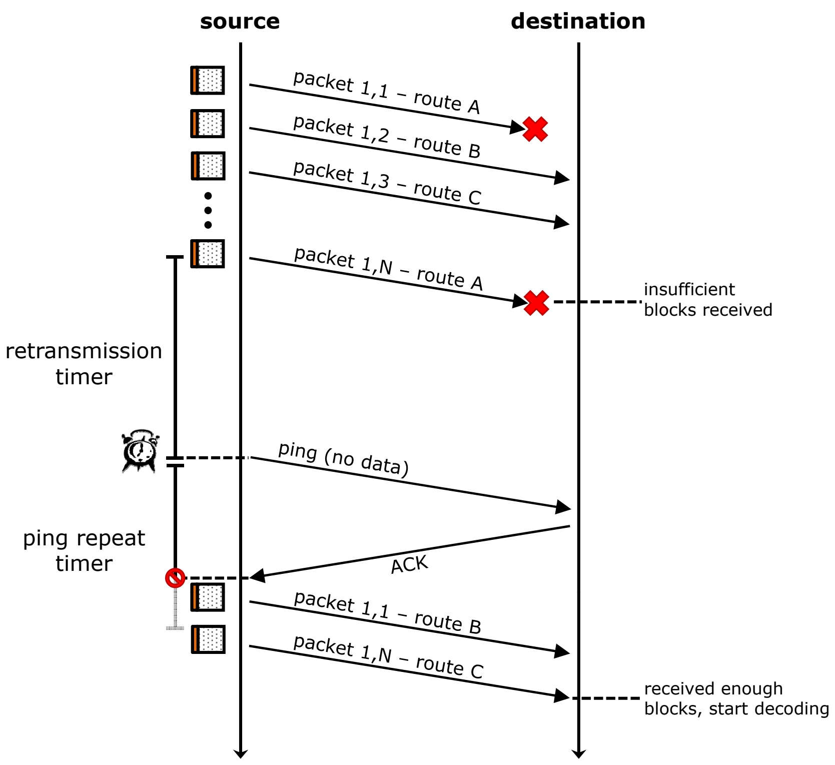

Figure 3.8: A retransmission caused by a high packet loss.

one may need to devise a more accurate way to make this estimation.

Let $RTTest$ and $RTTvar$ be the values for the estimated RTT and RTT variation, respectively, of the route with highest expected RTT ($RTTest + RTTvar$), and let δ be the estimated processing time for decoding. The retransmission timer value is calculated as:

$$T_{ret} = RTTest + max(1, 4 \times RTTvar) + \delta$$

.

Three important issues must be considered when addressing retransmissions. The first is that they can potentially cause a delivery of a segment to expire its deadline. In general it is not possible to solve this problem because the network is lossy and might be under attack. However, to minimize the probability of this event, the REB combines the usage of multiple routes and erasure codes to make the need for retransmissions very unlikely.

The second issue is related to the reliability of SACK delivery. SACK can be lost due to an accidental corruption in the network or because of an attack that causes its deletion. In order to maximize the probability of SACK arrival, it is sent through all the routes that were used by the source node when transmitting the data packets. A destination node must therefore keep a record of these routes.

The third issue concerns acknowledgment loss, duplication and/or reordering, either of accidental or malicious nature. For example, an attacker may try to replay previous acknowledgments to a source node, trying to force unnecessary retransmissions. Even in a non-malicious scenario, the re-ordering of two acknowledgements could make the source think that some of the previously received packets were garbage collected, therefore, they would eventually need to be retransmitted. To solve this sort of problems, the source applies the following procedure when an SACK arrives:

- Compare the acknowledgment information in the new SACK with the last accepted SACK with regard to (a) the value of the cumulative acknowledgement, and (b) the ranges of correctly received packets in the segment that is currently being received.
  - If any of the two indicates that less packets have arrived, then discard the new SACK;
  - Otherwise, update the current knowledge of what has been received accordingly to the new SACK.

3.5.6 Flow control

When a source node sends data to a destination, it may have several segments ready for transmission. In this case, it would be interesting if the node could disseminate packets from multiple segments before receiving an acknowledgment for the initial one. This has the benefit of allowing source nodes to do useful work while the destinations are busy decoding packets, or while the SACK is being forwarded through the network. However, the increased transmission capability needs to be bounded, otherwise too many packets may reach the receivers, which can cause memory exhaustion on their machines (or too many packet drops). To address the problem, REB implements a flow control mechanism to limit the transmission rates between nodes.

The rate of a transmission is dictated by the capacity of the receive queue at the destination, but more importantly by the rate of data consumption by the application. The receive queue holds (in order) the latest completely received segments until they are consumed by the application. As a result, the queue must have at least a capacity large enough to hold one encoded segment with maximum length. To tolerate out-of-order reception, blocks from subsequent segments are also buffered inside the queue. If those segments are received before the previous ones, they are decoded but stay in the queue until the earlier segments are read (in order) by the application.

The portion of the receive queue that starts immediately after the last in order complete segment (or the whole queue if it is empty) can be used to store further packets. When managing this space, we give priority to the next transmitted segment because it will be read by the application right after the existing segments. Therefore, the packets of this segment, which we call as the *segment currently being received* (see Section 3.5.5), are never reclaimed by the garbage collector in case of lack of memory in the node. The packets from the following segments can, however, be deleted.

The *receive window* is defined as the free space in the receive queue. We include in this space all packets/segments that are more recent (i.e., with higher segment sequence number in the session) than the segment that is currently being received. Therefore, the storage of these packets/segments is considered provisional.

A source node keeps an informed view of the receive window, based on information that is returned by the destination. The current view sometimes does not reflect the actual window state because packets may be lost, but in normal network conditions it tends for the right value. This view is required by the source node because the data transmission rate is constrained by its value (i.e., the the source can only send packets that fit within the available space). To keep the value accurate, the destination node includes the size of the receive window in each acknowledgment.

When a source node has outstanding encoded segments for a given destination, it starts transmitting them block by block in a FIFO order, (ideally) each one inside a distinct packet. The transmission procedure is implemented in such a way that the initial outstanding segment is given a higher priority than the rest. The procedure steps are as follows:

1. Let $winSize$ the currently perceived value for the receive window size;
2. For the first outstanding segment, obtain the constants: N is the total number of encoded blocks; K is the typical number of blocks required by the destination node to decode the segment with high probability; and l is the length in bytes of each block;
3. Set variable D to the total number of delivered blocks from the first segment so far (0 at the beginning of the transmission);
4. Obtain the number of undelivered blocks from the first segment that fit inside the window, as $Z = winSize/l$ (integer division);
5. If the receive window size is insufficient to hold all the undelivered blocks from the first segment, that is, if $Z < N - D$, then:
   (a) Send Z undelivered blocks from the first segment and keep a record of those transmissions;
   (b) Begin a periodic transmission of ping packets (packets with a data of size 0, as described in Section 3.5.5), to force the destination node to respond with an acknowledgment;
   (c) Try to send segments to other destinations.
6. Otherwise, if there is space in the receive window:
   (a) Send the remaining undelivered blocks from the first segment, that is, send $N - D$ blocks;
   (b) Start the retransmission timer (as described in Section 3.5.5);
   (c) Update the receive window size, as $winSize = winSize - l \times (K - D)$ (note that it is K instead of N);
   (d) If there are outstanding segments beyond the first one, apply a similar procedure and send as many blocks as the receive window allows; otherwise, try to send segments to other destinations.

Notice that when sending extra blocks from the first segment (beyond the initial K), we take an optimistic approach to update the receive window (rule 6.c). This is done because in most cases K blocks are enough to decode a segment. As a result, the destination node needs to manage the receive queue accordingly, being prepared for the fact that K blocks might not be enough and that extra blocks might need to be stored ($N - K$ blocks). This means that in some cases, a destination node may discard received blocks from the next segments (and possibly even decoded segments), if not enough space is available in the queue. This is reflected on the way acknowledgments are understood by the source, where a newer SACK may "rollback" the delivery status of subsequent blocks/segments. Figure 3.9 shows an example of this scenario. Note that thanks to the efficiency of the erasure codes, it is expected that such scenarios occur rarely in practice.

On the subject of window size updates, it is possible for a duplicate acknowledgment (spurious or forced by an attacker) to give an incorrect view of the window, forcing unnecessary retransmissions. This can happen when a destination node sends acknowledgments which roll back the information about the delivery status of subsequent segments (see above). This situation arises because even though it

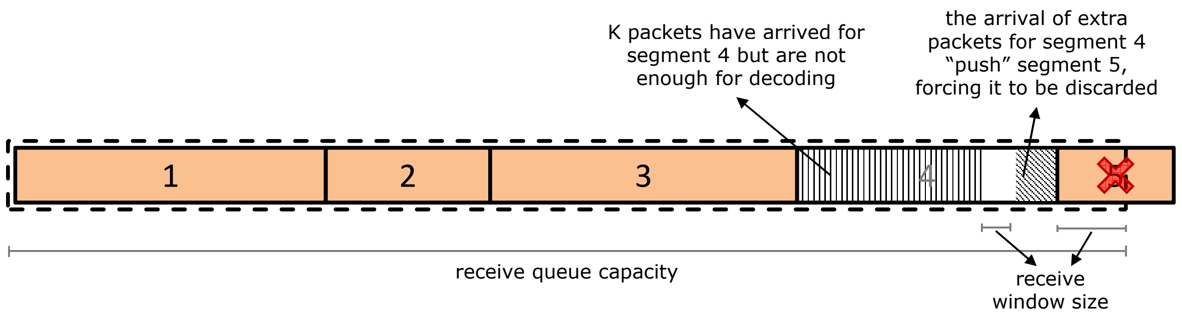
K packets have arrived for segment 4 but are not enough for decoding
the arrival of extra packets for segment 4 "push" segment 5, forcing it to be discarded
1
2
3
4
5
receive queue capacity
receive window size

Figure 3.9: Example scenario where a segment has to be discarded from the receive queue to get space for storing packets of the segment that is currently being received.

is possible for a source node to detect an ordering on the acknowledgments by analyzing the delivery status of the first segment (which never rolls back), different acknowledgments that only differ on the subsequent segments cannot be consistently ordered. This roll back behavior however is expected to occur only on rare occasions, so this particular scenario does not significantly affect the communication. Besides, note that it only has a small impact on the communication progress because the first segment continues to be delivered correctly.

On the subject of ping packet transmissions, there seems to be a possibility for a DoS attack on destination nodes causing them to transmit a large number of acknowledgments in a row. To circumvent this problem, destination nodes only respond to ping packets if their period is greater than a fixed amount of time.

It is also worth mentioning that when receiving an acknowledgment, the advertised window size could be too small (smaller than the size of a packet). To avoid this problem, the receiver only advertises windows with a capacity that allows the transmission of packets with a considerable size. Currently, this minimum capacity is MTU.

3.5.7 Route probing and selection

The timeliness of the communication is based on the assumption that the appropriate selection of overlay routes will, with high probability, result in the delivery of segments before specified deadlines. Therefore, it is required to periodically estimate the quality of service of the routes, in order to have a continuous informed and updated knowledge about the best available paths.

Concerning metrics of quality of service, the fundamental one of an overlay route is the expected Round Trip Time (RTT). For a source node, it is both important to know how much time will take for a given segment to reach its destination, and also how long it will take before the respective acknowledgment arrives back. One should not forget that a source is only capable of continuing the transmission of buffered segments after the acknowledgment is received, with the respective update to the window size. Another relevant metric is the loss rate of a route, which influences the effective time necessary for a packet to arrive to a destination (if a packet is lost, then the transmission time can become the interval for the timer to expire plus the retransmission through the network). Therefore, we do not use the loss rate directly in the choice of the best routes, but as a way to penalize negatively the RTT value of a route. This way, routes with high loss rates are viewed as being slower and so fall behind others with better latencies and/or lower loss rates.

In order to estimate the RTT values of its routes, a source node utilizes a probing mechanism that is activated periodically, causing the destination node to return back information about the delivery delays and packet losses. Given the current size of SIEM deployments, the REB overlay may have a few hundred nodes. To keep the overhead of probing small, we adjust the probing frequency to the usefulness of the routes – routes that are used regularly are probed more often. Furthermore, if a node normally communicates with a certain destination, then the routes towards that node are checked more frequently.

REB uses two different approaches to manage probing traffic efficiently: a) a source node transmits probe requests through unused routes to trigger immediate replies with probing data by the destination nodes (a *pull approach* by the senders); b) a destination node also initiates the transmission of probing data through some routes whenever a segment is fully received (a *push approach* by the receivers). Consequently, a route that is used recurrently will have more probing traffic being conveyed by the receiver. Routes that are never utilized are only checked when the source decides to send a probe through them. This allows for a fast recovery of the routes being currently employed for the main communication (should they suddenly become attacked), while keeping the knowledge about idle routes updated over time.

The probing mechanism is intrinsically connected to the transmission of acknowledgments, that is, probing information is delivered to a source node inside acknowledgment packets that are returned from a destination node. As we have seen before, acknowledgments are transmitted when a segment is completely received, as well as when a source node explicitly requests them through the use of packets with zero-length data (referred to as ping packets). We can see an immediate parallel with the push and pull approaches defined earlier. Probing requests are nothing more than ping packets and probing information is immediately transmitted per receipt of these packets.

However, sending probing data only when segments are completely received, as opposed to transmitting this data when packets arrive, could at first seem to affect the precision of the RTT estimation at the source. Adding such transmission of acknowledgments every time any packet is received through some route, though, would increase the network traffic, which could interfere with the main communication by delaying it and causing processes to stall. It was thus necessary to find a mechanism that kept the extra traffic to a minimum, but at the same time offered a fine granularity on the probing of individual packets (information about their loss rates and individual latencies).

To achieve the desired granularity of having probing information about individual packets during a normal transmission, a destination node stores the arrival times of every received packet and includes those times inside the acknowledgment. At the same time, a source node stores the departure times of each packet it transmits, as well as the identification of the route that was used. When an acknowledgment arrives, the source node calculates the RTT value of each packet using the saved departure time and the announced arrival time. It then uses that sample RTT value to estimate the expected RTT value of the route that was used to transmit the packet. Since REB nodes might not have their clocks synchronized with very high precision, a source node cannot simply calculate the latency of a packet by subtracting its departure time from its arrival time. Instead, a source node obtains the time of the acknowledgment arrival, and for each reported packet inside:

1. Measures the elapsed time between the moment the packet was transmitted from the source *PacketDeparture* and the moment the acknowledgment arrived back *ACKArrival*;

2. Obtains the elapsed time between the moment the packet arrived at the destination node *PacketArrival* and the moment the acknowledgment was transmitted from there;

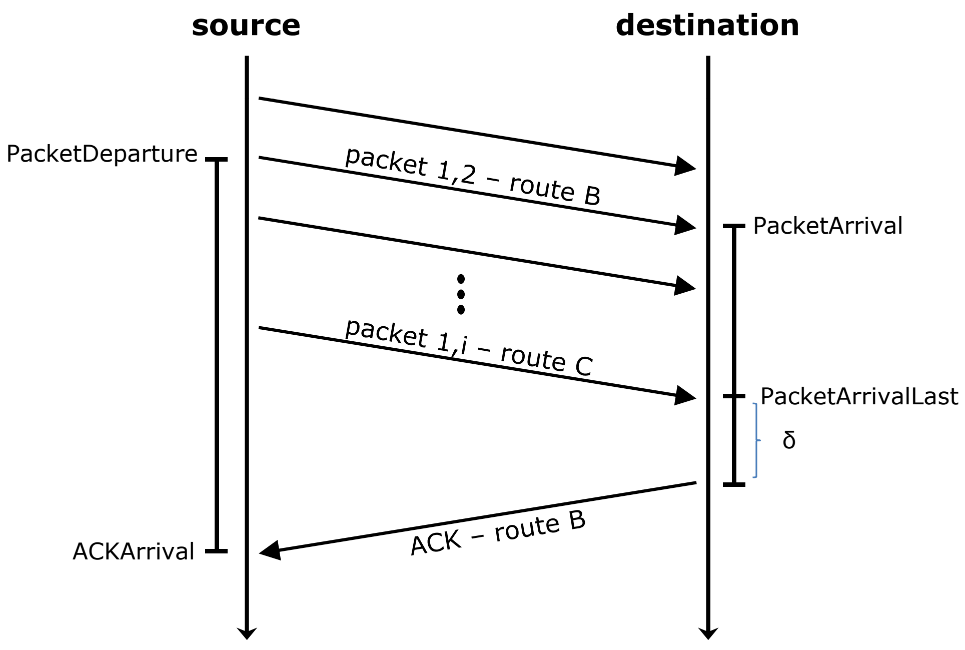

Figure 3.10: Example scenario where the RTT is estimated for packet with ID (1,2).

3. Subtracts the first elapsed time from the second and hence obtains the sample RTT value.

Note that in the second step, the source does not receive the instant when the acknowledgement was received, but gets the moment when the last packet arrived *PacketArrivalLast* that allowed the segment to be decoded. Since it may take some time for the decoding operation to conclude, the instant of the acknowledgement transmission is estimated by adding δ to *PacketArrivalLast* (recall that δ is approximately equal to the decoding time[6]). Presenting it in a formula, the sample RTT value is taken as (see Figure 3.10 for a graphical representation):

$$RTTsamp = (PacketDeparture - ACKArrival) - (PacketArrivalLast + \delta - PacketArrival)$$

It is important to refer that each acknowledgment must be transmitted through all the routes from where the reported packets were received. In turn, for every acknowledgment copy it receives, the source node must only measure the sample RTT values from the packets that were transmitted through the same route as the acknowledgment. This is required because otherwise it would be possible for the latency of a route to affect the estimation of the RTT of another.

The way REB calculates and keeps an expected RTT value per route is built upon the well tested algorithms employed by TCP [63]. As mentioned before, the expected RTT value corresponds to the sum of an RTT estimation and an RTT variation. Both metrics are averaged over time following an exponentially-weighted moving average, which keeps an history of past values but gives more weight to recent values over old ones. For each route, both metrics are computed as follows:

[6]As we get more experience with REB, we will find out if δ is a good estimate for the delays at the receiver. In any case, as an alternative solution, the destination node could add to the message the instant when the ACK is sent *ACKDeparture* (not represented in the figure), and then δ could be calculated as $ACKDeparture - PacketArrivalLast$.

- If no sample RTT values have been taken, then:
  - $RTTvar = 0$
  - $RTTest =$ a predefined large value
- When the first sample RTT value has been taken, then:
  - $RTTvar = RTTsamp/2$
  - $RTTest = RTTsamp$
- When an additional sample RTT value has been taken, then:
  - $RTTvar = (1 - \beta) \times RTTvar + \beta \times |RTTest - RTTsamp|$
  - $RTTest = (1 - \alpha) \times RTTest + \alpha \times RTTsamp$
  - Note that here, $RTTvar$ is updated using the value of an $RTTest$ from the previous update.

Following the advice in [63], α is set to $^1/_8$ and β to $^1/_4$. However, these values presuppose a modus operandi for transmitting data that comes from TCP, and may not be ideal in REB because of the different transmission semantics. This subject will require a further understanding in order to ascertain whether different values for α and β make more sense in the context of REB.

Loss rate is another metric which is used to affect the perceived quality of the overlay routes. In order to detect lost packets, source nodes inspect the IDs from the packets reported in the probing information. If there are "holes" within the listed IDs, a source node assumes the respective packets were lost in transit. The loss of an acknowledgment can also affect the perceived loss rate of a route. In order to identify the loss of such acknowledgments, each one carries in the probing information the ID of the last packet reported in previous acknowledgments. A source node assumes then that every unacknowledged packet which has an ID inferior to the one indicated above is lost. However, since acknowledgments are transmitted through all multiple routes, the chance of all being lost is reduced. Every time a packet is deemed lost, the $RTTest$ value of the route the packet was sent through is affected negatively in a fixed amount λ.

The algorithm for selecting the routes that should be used for transmitting the packets uses the paths with best quality of service (i.e., $RTTest$). At this point, this algorithm has been kept relatively simple. For a given destination, the source picks R routes for transmission. Of these routes, some of them should be direct links to the destination and the remaining should have an intermediary node. This ensures a reasonable level of diversity among the chosen routes, which can beneficial in case of attacks. The quota for direct links (up to $R/2$) is first filled in with the best direct routes. For the remaining slots are picked the two-hop paths that have also the best quality of service.

4 Node Defense Mechanisms

This chapter addresses defense mechanisms that can be employed to enhance the dependability and security of the nodes. These mechanisms start to be presented in a generic way, since some of them might be appropriate for certain nodes while others to other nodes. Moreover, given the various criticality levels of the SIEM nodes and the cost of using some of these techniques, it is worth to consider a hierarchy of designs that are incrementally more resilient. In order to ensure correct node behavior in the presence of component failures, either of accidental nature or forced by an adversary, we resort to replication techniques. These techniques support the construction of *Intrusion-Tolerant Systems*, allowing failures to be automatically masked (and recovered) as long as enough good replicas remain operational. Consequently, by introducing more replicas one can typically improve the overall dependability of the system (of course, with some added costs).

To put these ideas in practice, the chapter also presents the design of a highly resilient core-MIS, a solution that can eventually be extended to other MASSIF nodes like the edge-MIS. The core-MIS role is to protect the most critical part of the SIEM, the core facility, where the events' are processed, correlated and archived. Since all traffic to and from the core facility needs to go through the core-MIS, it can perform various filtering operations to ensure that external malicious messages are prevented from entering. This is particularly true for traffic originating from nodes other than the edge-MIS, which normally can be simply dropped, but also some checks can be carried out on the messages from the edge-MIS (in case one of them was compromised and started to operate in a malicious way). Therefore, it is fundamental to ensure correct functioning of the core-MIS on adverse conditions, such as when this node becomes the target of attacks or/and experiences accidental faults (e.g., a replica component crashes).

4.1 Resilient mechanisms support

This section starts by providing an overview of intrusion tolerance, and then describes the four main mechanisms that can be used to enhance the resilience of a node: Byzantine fault tolerance, diversity, proactive recovery, and a combination of proactive-reactive recovery. A more general rationale for these mechanisms was presented in [16], namely in sections "Communication Support" and in "Runtime Support".

4.1.1 Intrusion tolerance

The advancements in software development have provided us with an increasing number of useful applications with an ever improving functionality. These enhancements, however, are achieved in most cases

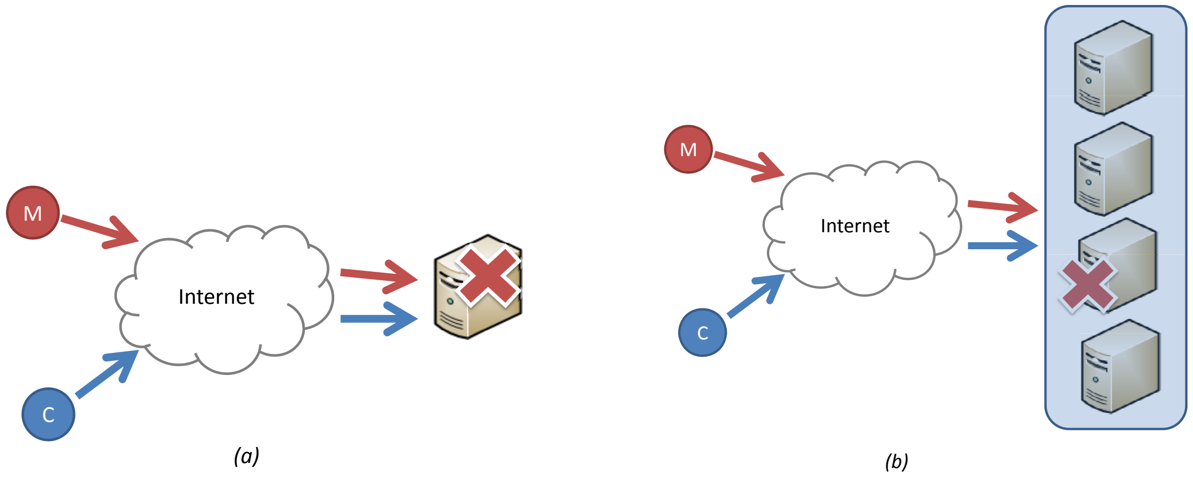

Figure 4.1: Remote service accessed by a client C (and potentially by an adversary M) that suffers a fault, where *(a)* is a simplex system, and *(b)* is a replicated system.

with larger and more complex projects, which require the coordination of several teams. Third party software, such as components off-the-shelf (COTS), is frequently utilized to speed up development, even though in many cases it is poorly documented and supported. In the background, the ever-present trade-off between thorough testing and time to deployment affects the quality of the software. These factors, allied to the current development and testing methodologies, have proven to be inadequate and insufficient to construct dependable software. Everyday, new flaws (including vulnerabilities) are found in what was previously believed to be secure applications, unlocking new risks and security hazards.

When the software is deployed, these flaws can result in accidental faults, such as when a boundary condition is violated and creates a process crash, or in intentional (or malicious) faults. An intentional fault can be caused by an external entity that performs a successful attack to exploit a vulnerability, and the impact is various, for instance, the corruption of data or the termination of a service. In any case, in the presence of faults the system can start to behave abnormally, i.e., stop to provide the expected service.

One of the most efficient and transparent ways to deal with faults is to tolerate them. Replicated systems have been used to solve this problem over the last decades, in the majority cases only addressing accidental faults. These solutions are thus suitable for settings relatively benign (or where intrusions can be dealt with by some other means, e.g., with insurance), but they easily fall prey of an adversary that is able to compromise a single replica.

A system is said intrusion-tolerant when a subset of the components (i.e., replicas) fail without harming the offered service (for an overview of the area see [81]). Redundancy is the cost attached to intrusion tolerance, for instance, by replicating the client's request in several replicas. Hence, each replica executes the same request and the system can assure that a number of failures can be tolerated with a (relatively simple) voting on the produced results. The number of required replicas is related to the number of faults one wants to tolerate, and its value depends on the type of service/algorithm that is implemented and on the strength of the assumptions that can arguably be made on the execution environment. Typically, the number of replicas n that is needed to tolerate f faults is $n \geq 3f + 1$, in an environment with very weak assumptions [45] .

To illustrate these points, Figure 4.1 shows two versions of a service that is accessed remotely by clients C: *(a)* a non replicated (or simplex) system, and *(b)* a replicated system. If the simplex system suffers a fault, such as an intrusion, then the service stops from being correct and the client might start to receive wrong responses (or no answers).

On the other hand, the intrusion tolerant replicated system offers a service to the clients under very weak failure assumptions. Typically, it is assumed that an adversary M can perform various attacks on the network (e.g., replay, delay, re-order, corrupt messages), and that he or she controls an undetermined number of clients and up to f replicas. Nevertheless, even under these challenging conditions, the system needs to provide correct responses to the good clients.

The design and implementation of the intrusion tolerant system has to address two main concerns. First, the protection of the communication between the client and the service, where it is necessary that all correct replicas execute the request and then that the client is able to select a correct response. By employing re-transmissions and cryptographic methods, it is possible to secure the messages from the network attacks and ensure their delivery. The selection of the response is a bit more delicate because some of the replies might be produced by malicious replicas, and therefore contain erroneous data. Consequently, the usual procedure is to wait for $f+1$ equal responses, and only then choose this answer (this ensures that at least one correct replica vouches for the reply).

Second, all correct replicas should start to execute from an identical initial state, and then they should evolve through the equivalent states as they process the requests (by assuring that requests are executed in a deterministic way). For this reason, a service with these characteristics is said to implement *State Machine Replication (SMR)* [70]. This is achieved by running a Byzantine fault-tolerant replication protocol among the replicas, which associates a processing order number to each request and makes sure the correct replicas carry out the same requests.

In MASSIF, SMR brings however a difficulty due to its programming model — the assumption that there is a client that invokes an operation on a replicated service and waits for replies — since it does not match the way some SIEM components operate. For example, the core-MIS acts as a *forwarder* of message traffic to and from the core facility, and therefore, arriving messages need to be transmitted to a node other than the original sender. Nevertheless, as we will explain in Sections 4.2 and 4.3, the implementation of a replicated forwarder can be done with small modifications to the basic SMR model. The idea is to replicate a deterministic forwarder and use a total order broadcast (see next section) to make all replicas process the same sequence of messages (or requests). The difference from the SMR model is that the result of the processing is delivered to the final destination of the message, and not the sender (see Figure 4.2).

4.1.2 Byzantine Fault tolerance

A key building block of intrusion-tolerant systems is Byzantine fault-tolerant (BFT) protocols, which guarantee the correct behavior in spite of arbitrary faults (also called Byzantine faults), provided that a minority of the components are faulty.

In our implementation of a BFT replication protocol, we are considering two forms communication support: reliable and atomic broadcast.

Reliable Multicast. A reliable multicast protocol is used to ensure that if a message is sent to a group of processes (or replicas), either all correct processes deliver this message or none will do that. Formally, a reliable broadcast protocol can be defined in terms of the following properties [12, 21, 39]:

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

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

Total order multicast. A total order multicast (TOM) protocol is similar to a reliable broadcast protocol, but it ensures an additional property [21]:

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

This additional property makes the implementation of a total order broadcast much harder than reliable broadcast. More precisely, the problem of total order broadcast has been shown equivalent to the well known consensus problem [39, 14, 21], and thus cannot be solved deterministically in asynchronous systems with crash failures (and therefore, cannot also be solved in systems with Byzantine failures) [31]. Over the years, several approaches have been proposed to circumvent this limitation, i.e., to slightly modify the system model in such a way that consensus becomes solvable. Examples include randomization [65, 9], failure detectors [14, 52], partial-synchrony [28, 26], and hybridization/wormholes [19, 60].

On the other hand, the up side is that a significant amount of research has been carried out on the design of new consensus protocols (for a survey see [22]), and this work can be leveraged to build better atomic broadcast protocols. For example within MASSIF, we have recently proposed an optimization to a well known randomized BFT consensus protocol [80], which uses a speculative execution to reduce the number of communication rounds from three to two in the normal case.

4.1.3 Replica diversity

Works concerning BFT protocols tend to assume that replica nodes fail in an independent manner [13, 85, 10, 20, 58, 2]. To respect this condition, system components need to exhibit failure diversity. However, when security is considered, the possibility of simultaneous attacks against several components cannot be dismissed. If multiple components exhibit the same vulnerabilities, they can be compromised by a single attack, which defeats the whole purpose of building an intrusion-tolerant system in the first place. Moreover, this problem is also relevant from an accidental fault perspective. Since all replicas are executing similar tasks, if the software contains a bug that is activated when processing a request, then every replica will progress into a faulty state.

To circumvent this limitation one must substantiate the fault independence assumption by construction. Diversity allows one to build safer replicated systems based on the assumption that different components exhibit independent failure modes with high probability. Replicas should execute distinct software that offers similar functionality (e.g., think about the operating system), but since the code was developed by different teams no common flaws should occur. Moreover, every diverse replica should be fully patched, clean from known vulnerabilities, to increase the difficulty of attacks.

One can find several opportunities to use diversity when setting up a system. For example, various hardware platforms could be employed and/or the software could be configured in a different way (e.g., using randomization of the placement of the programs in memory), which often is enough to limit heavily the success of attacks that are performed with automated tools. If higher levels of security are desired, then one could use multiple operating systems (OS) and other support software. As a last step, even different implementations of the applications can also be employed, especially if the replicated component has a standardized interface.

Nowadays, nearly all software systems rely on COTS, i.e., third party software components readily available for use, like graphic packages, mathematical libraries, operating systems and database management systems. This is mostly due to the sheer complexity of such components, coupled with benefits such as the perceived lower costs from their use (some of the components may be open-source and/or freely available), faster deployment and the multitude of available options. Consequently, leveraging on COTS to implement diversity is less complex and more cost-effective than actually developing variants of software.

Within MASSIF, we have been exploring the idea of using diversity at the operating system level. Realistically, people will resort to a COTS operating system rather than build their own. Given the variety of operating systems available and the critical role played by the OS in any system, diversity at the OS level can be a reasonable way of providing good security against common vulnerabilities at little extra cost.

In a recent study, we analyzed the likelihood of common vulnerabilities on operating systems [32]. We analyzed more than 15 years of vulnerability reports from the NIST National Vulnerability Database (NVD) totaling 2120 vulnerabilities of eleven operating system distributions. The results suggest substantial security gains by using diverse operating systems for intrusion tolerance. Some of the more specific findings were: the number of common vulnerabilities on the studied operating system pairs was reduced by 56% on average if the application and locally-exploitable vulnerabilities could be avoided; more than 50% of the 55 OS pairs studied have at most one non-application, remotely exploitable common vulnerability; that there are some setups for a four-replica diverse system that have experienced very few (or no) common vulnerabilities over the years.

We have also been researching ways to build diversity at the application level. In particular, we have looked into the problem of incompatibilities that are created when using different implementations of the same application in the replicas [6]. Various kinds of incompatibilities were analyzed, and a new methodology was proposed to evaluate the compliance of diverse server replicas. The methodology collects network traces to identify syntax and semantic violations, and to assist in their resolution. A tool called DiveInto was developed based on the methodology and was applied to three replication scenarios. The experiments demonstrated that DiveInto was capable of discovering various sorts of violations, including problems related with nondeterministic execution.

4.1.4 Proactive Recovery

Individual replicas, even if fully patched, are still vulnerable to attacks that exploit flaws unknown to the security community (normally called *zero-day vulnerabilities*). Typically, these vulnerabilities are kept secret by the hackers until an suitable exploit is developed, and they are only discovered when eventually they start to be used to compromise systems. Although the number of zero-day vulnerabilities observed per year is relatively small, since they require specialized knowledge, effort and time to be found, Symantec stated that 14 zero-day vulnerabilities for operating systems were discovered in 2010 [78] and 8 in 2011 [79]. Therefore, they cannot be disregarded in highly resilient solutions, and proper techniques should be employed to address them.

One however should keep in mind that the intrusion tolerant system remains correct as long as the adversary only controls up to f replicas. The problem occurs if a powerful attacker could silently discover flaws in $f + 1$ replicas, and then run the exploits at the same time taking over the system. *Proactive recovery (PR)* is a way to avoid this scenario [69, 74, 13, 53], by forcing replicas to be periodically rejuvenated with a diverse software configuration, which has a (hopefully) different set of vulnerabilities.

PR allows for: *i)* the cleaning of the faulty state in case the replica was silently compromised; *ii)* if a replica is correct but is being probed by an attacker, then a recovery will force the attacker to restart over because previously acquired knowledge is no longer of use. The system stays intrusion-tolerant as long as the recoveries occur faster than the $f + 1^{th}$ fault. Moreover, the recovery must modify the replica in such a way that it is not trivial for an attacker to compromise it again.

In previous works, the event that triggers the rejuvenation of a replica is based on time (e.g., every hour the replica is recovered). This can bring non trivial problems to the actual implementation of the PR mechanisms, since it needs to ensure that the trigger can not be delayed accidentally (e.g., due to heavy load in the system) or maliciously, and that the rejuvenation terminates within a bounded time [75]. To address this difficulty, we are developing within MASSIF other triggering actions, which at this point are based on an assessment of the risk level associated with a given replica in operation [33].

4.1.5 Reactive Recovery

A limitation of proactive recovery is that a malicious replica can execute any action to disturb the system's normal operation (e.g., flood the network with arbitrary packets) until its recovery time, and there is little or nothing that a correct replica (that detects this abnormal behavior) can do to stop the faulty one. The observation is that a more complete solution should allow correct replicas to force the recovery of *detected or suspected* faulty replicas. This solution is called *Proactive-Reactive recovery* [74], and it can improve the overall performance of a system under attack by reducing significantly the amount of time a malicious replica has to disturb the normal operation.

If $f+1$ different replicas suspect and/or detect that replica R_j is failed, then this replica is recovered. This recovery can be done immediately, without endangering availability, in the presence of at least $f + 1$ detections, given that in this case at least one correct replica detected that replica R_j is really faulty. Otherwise, if there are only $f + 1$ suspicions, the replica may be correct and the recovery should be coordinated with the proactive recoveries in order to guarantee that a minimum number of correct replicas is always alive to ensure the progress of the system. The quorum of $f + 1$ in terms of suspicions or detections is needed to prevent recoveries triggered by malicious replicas – at least one correct replica must detect/suspect a replica for some recovery action to be taken.

Notice that a proactive-reactive recovery service is completely orthogonal to the failure/intrusion detection strategy used by a system. The proposed service only exports operations to be called when a replica is detected/suspected to be faulty. In this sense, any approach for fault detection [7, 14, 27], system monitoring [23] and/or intrusion detection [24, 59] can be integrated in the system. For example, the observation of certain malicious actions performed by faulty replicas of the resilient core-MIS, such as altering the content of messages to be forwarded, can be used for this propose.

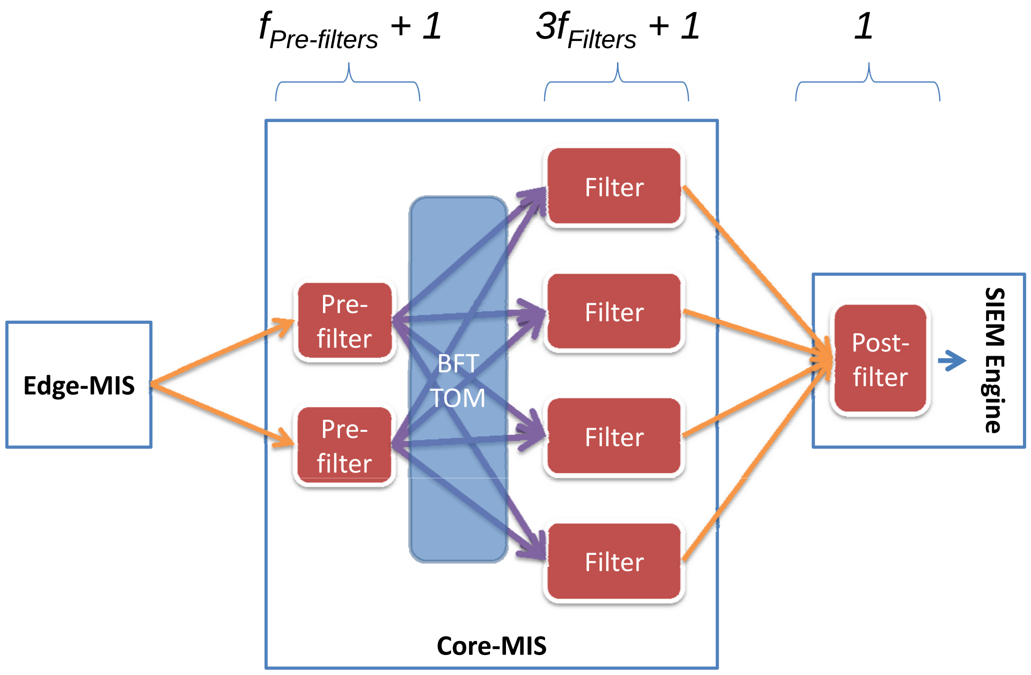

Figure 4.2: Architecture of the core-MIS.

4.2 Resilient core-MIS architecture

The core-MIS architecture and protocols takes advantage of the mechanisms described in the previous section to achieve a high level of resilience. Its main role in a SIEM system is to forward messages coming from the edge-MIS, usually containing the event data collected by the sensors, towards one of the core services. Occasionally, the core-MIS may also need to return data to the edge facilities, for instance, when reconfiguration actions need to be performed in some device. For simplicity, we will focus on the description of the processing of messages coming from the outside of the core facility, since these messages can have a malicious origin. Additionally, we will assume that the destination of these messages is the correlation engine of the SIEM.

The core-MIS provides a message forwarding service that mimics the properties of the Resilient Event Bus (REB) (see Chapter 3). For example, it ensures reliable delivery of the messages that arrive to its interface. For messages transmitted from a particular source, it also guarantees FIFO order in their delivery. Additionally, it attempts to minimize the operations that need to be performed with the messages to avoid delays and increase throughput.

The core-MIS is a replicated system, built from two main components called the *Pre-filters* and *Filters*, and a third component named *Post-filter* that is located in the SIEM Engine. In order to make this system intrusion-tolerant up to f faults, the core-MIS needs at least $f + 1$ Pre-filters and $3f + 1$ Filters. Post-filters are assumed to be correct because they run together with the engine. The communication between Pre-filters and Filters is made by a total order multicast channel, ensuring that every Filter replica processes the messages in the same order. Filters run diverse operating systems to avoid common flaws, and when a recovery is made, proactively or reactively, a different OS is loaded.

Figure 4.2 shows the general architecture of the core-MIS. The communication between the edge-

MIS and the core-MIS is implemented by the REB. The REB delivers the messages to the Pre-filters, which then retransmit them to the Filters via a group communication library (that is being built with some support from the project, and is called BFT-SMaRt[1]) that offers a total order communication primitive. At the end, the Filters use authenticated reliable point-to-point channels to send the messages to the Post-filter for voting. An illustration of the communication among the various components if presented in Figure 4.3.

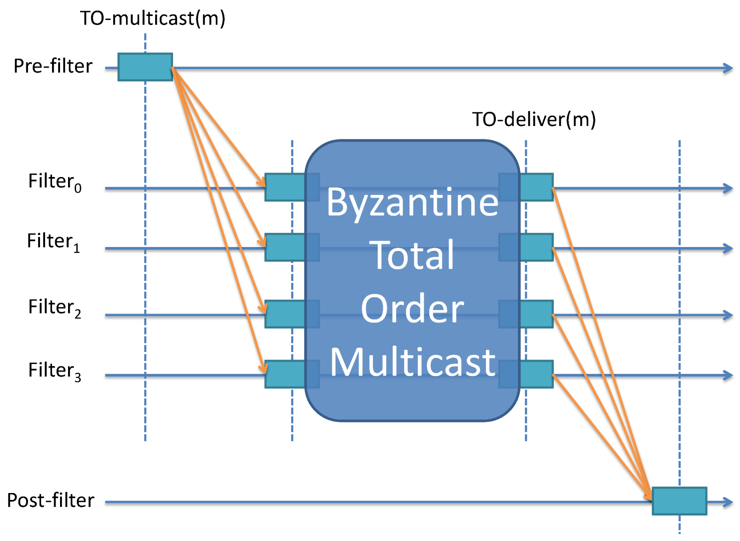

Figure 4.3: Communication among the various core-MIS components.

Each of the three components has its specific function regarding the resilience of the system:

Pre-filter: receives messages from the REB, then verifies their source to accept only messages from known edge-MIS. External messages are simply dropped. To avoid denial of service (DoS) attacks carried out by compromised edge-MIS, Pre-filters implement a very simple (and efficient) message flow control based on grants that limits to acceptable levels the amount of information that can be transmitted from each source. Messages are discarded if their arrival rate is higher than what was granted.

Filter: makes two fundamental verifications on the arriving messages. First, it checks a MAC added by the edge-MIS, using a key that is shared between the edge-MIS and the Filter (different keys are used for each pair of components). Second, it makes a sequence number verification to detect for instance duplicates. Besides that, the Filter can also enforce more complex security policies, e.g., it can check if the events included in a message are acceptable for its source.

Post-Filter: must vote the Filters' replies in order to deliver only correct messages to the SIEM Engine. Therefore, it prevents compromised Filters from influencing the correlation processing with malicious data.

4.2.1 Denial-of-service mitigation

Denial-of-service (DoS) is one of the important attack vectors that has to be considered in the core-MIS design. In general, there is no solution to prevent DoS attacks if a system can be accessed remotely. DoS attacks can be very simple (e.g., just send many SYN packets from the TCP protocol), and their strength relies often on the number of machines that perform the attack. The power of multiple attackers can lead to the exhaustion of resources, compromising the system's availability.

A DoS can be roughly divided in two categories depending on the resource that is exhausted, either the network or the application/node (but sometimes, both resources are the target). In the first one, the state-of-the-art mitigation solutions propose techniques applied to the network to isolate the attack, via for instance quarantine links or by closing ports. These measures are to a large extent orthogonal to the core-MIS design, since they require reconfiguration operations in the network devices on the path to the adversary machines. Therefore, we would like to count with this protection, but its availability depends on the existing support in the network where the core-MIS is deployed.

The second category of DoS attempts to exhaust the resources of the system implementing the service, in this case the various replicas of the core-MIS. In order to minimize the effects of the DoS, the core-MIS employs two ideas: 1) when processing a message over the various layers (Pre-filter, Filter, and Post-filter), apply first the simplest (and more efficient) tests to identify malicious traffic; 2) discard as much as possible the malicious DoS traffic in the earlier layers of processing (ideally in the Pre-filters). Together, these ideas ensure that DoS messages are dropped soon, without loading the later stages of processing.

One of the mechanisms that is implemented is inspired on the use of capabilities that have been proposed in previous works [5, 62]. Capabilities are a grant/token based communication, in which it is needed permission to send messages, in order to prevent abusive flows of requests. However, our solution does not make assumptions on the network, i.e., the network routers do not analyze or drop packets. Assuming a reliable authenticated channel, as the one offered by the REB, we can provide a grant system to control the amount of message that can be transmitted. The core-MIS periodically provides grants to the edges, allowing a certain level of communication. Once the contract is violated, which indicates that the edge is malfunctioning, the packets start to be dropped and an alarm is raised.

4.3 Core-MIS protocols

This section gives an explanation of the core-MIS protocols that are run among the components. We also discuss some of the attack scenarios, and describe how they are addressed in order to prevent malicious behaviors.

4.3.1 Failure-free execution

The protocol for exchanging data in a normal execution, i.e., without faults, is performed in the following way among the components.

1. Sensors in the edge facilities collect data that is transmitted to the Data services (normalization, aggregation, and local correlation) in the edge-MIS.

2. The edge-MIS creates a message with the received data, a sequence number and a vector of MACs that ensures the integrity to the whole message. Overall, the vector contains a separate MAC for every Filter. Each MAC is created using a shared key between the edge-MIS and one of the Filters.

3. The edge-MIS sends the message to $f + 1$ Pre-filters.

4. Every Pre-filter does some validations on the message, and then forwards it by invoking a BFT total order multicast to the $3f + 1$ Filters.

5. Once the messages are ordered and delivered, the Filter application verifies the MAC and the sequence number. A message with an incorrect MAC is dropped. If the message has a sequence number higher than the expected, then it is archived until all previous messages have been processed. A message is discarded if has a sequence number lower than the expected. If the sequence number is the right one, the message is ready to be go to the next stage, and therefore it is forwarded to the Post-filter (on the SIEM Engine).

6. The Post-filter receives messages transmitted from the Filters until it collects $f + 1$ matching messages. This ensures that the message is correct, since at least one correct Filter vouched for it. Then, the Post-filter delivers the message to the SIEM Engine.

   The voter in the Post-filter has a buffer to store messages until it collects $f + 1$ matching messages for a given sequence number. In the normal case, i.e., without faults, the first $f + 1$ messages will be equal, and therefore, only one of the copies needs to be stored plus a counter on the number of received votes. Since the core-MIS is intrusion tolerant, it must deal with eventual compromised parts, and therefore our solution must guarantee that malicious Filters cannot send wrong events to the SIEM Engine.

4.3.2 Behavior under attack

This section describes how the core-MIS behaves in presence of accidental faults or attacks. In particular, we consider attacks that may affect the events integrity, namely message content modification. The core-MIS also tolerates re-ordering attacks performed for instance by malicious Pre-filters. In the attacks description, we assume that the messages have already arrived to the Pre-filters, something that is enforced by the REB. We also assume $f = 1$, therefore, we can have one malicious Pre-filter and/or one malicious Filter. The objective is to keep the forwarding service of the core-MIS correct, despite the attacks and intrusions.

To give more power to the adversary, we assume in the following scenarios that the correct Pre-filters are slower than the malicious one but eventually send messages. Otherwise, our solutions trivially overcome the attacks.

Content modification attack. A malicious Pre-filter has the ability to modify the content of a message. Since the malicious Pre-filter is faster, it is the first to send a message to the Filters. Then each Filter verifies the integrity of the message with the included vector of MACs. By checking its corresponding MAC, the Filter confirms that the message was modified and therefore can discard it.

This attack does prevent the correct delivery of the message. Since a copy of the message also arrives to a correct Pre-filter, it gets to be forwarded to the Filters. Now, the checks will be all valid, and the message is transmitted to the Post-filter for final delivery.

MAC modification attack. This attack is more sophisticated as the malicious Pre-filter does not modify the part of the message with the events, but only a subset of the MACs in the vector, leaving the remaining unaltered. The objective of this attack is to prevent a certain message from being delivered to the engine, by averting the necessary quorum of votes to be reached in the Post-filter.

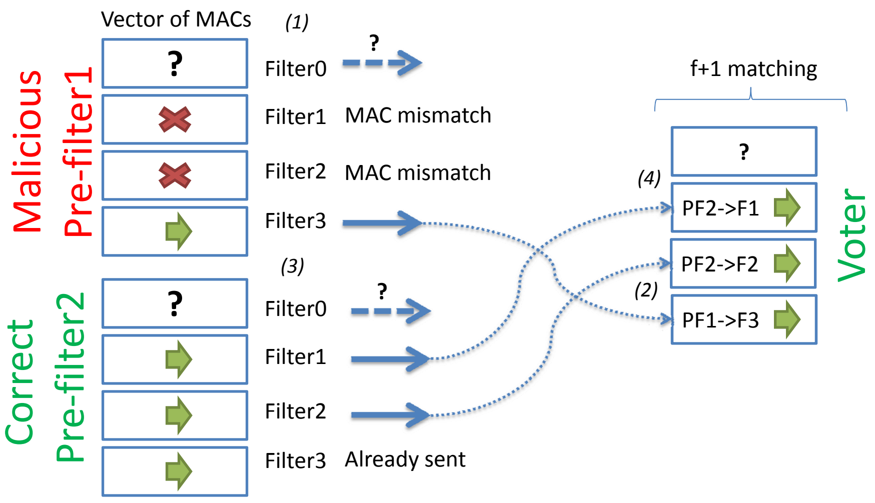

Figure 4.4: core-MIS MAC attack.

Figure 4.4 illustrates how this attack is performed in case there is both a malicious Pre-filter (named Pre-filter1) and a malicious Filter (identified as Filter0). The malicious Pre-filter starts by changing the MACs for Filter1 and Filter2, and then forwards the message. Then, the Filters check their own MAC in the message, and the following occurs (step *(1)* in the Figure): Filter0 is compromised and can do anything; Filter1 and 2 see the MAC verification fail and drop the message; Filter3 checks correctly the MAC verification, and therefore, it sends the message to the Post-filter (step *(2)*).

Eventually the correct Pre-filter (Pre-filter2) also gets and sends the message. Then, every MAC is correctly validated (step *(3)*), and the following happens: Filter0 is compromised and can do anything; Filter3 abstains from transmitting the message because this was previously done; Filter1 and 2 send the message to the voter. To finish, the Post-filter collects $f + 1$ matching messages (step *(4)*), one from the malicious Pre-filter through Filter3 and one from the correct Pre-filter either through Filter1 or 2, and delivers the data to the engine.

Re-ordering attack. A malicious Pre-filter can try to do two possible re-ordering attacks: 1) modify the message's sequence number; 2) delay some of the messages, and then send them out of order. Regarding the first attack, since the sequence number is protected with the MAC, the modification is detected and the packet is dropped as explained above. The second attack is addressed by having the Filters check the sequence numbers. Messages are only forwarded to the Post-filter in order. If a message's sequence number is larger than the expected value, then the message is stored until the previous ones are received.

A malicious edge-MIS could also send messages out of order, for example messages with sequence numbers much larger than the expected. This could have an impact on the Filters, as they would need to store them locally. To avoid this problem, the Filters only save messages with larger sequence numbers up to a certain threshold. Messages with sequence numbers above the threshold are discarded, and an alarm is raised.

DoS attack. There are various forms of DoS that can be performed, but they result in extra traffic/computation, and therefore, they can be detected. For example, a Pre-filter could receive messages from an edge-MIS at a rate above the grant. In this case, the edge-MIS is performing abnormally and an alarm is generated by the Pre-filter. In another example, the Pre-filter could constantly send messages with wrong MACs, something that does not occur with the rest of the Pre-filters. This justifies the generation of an alarm because it indicates that either the edge-MIS or the specific Pre-filter is malicious. As a last example, the Post-filter could receive correctly authenticated messages from a Filter but with a content different from the other Filters. In this case, the first Filter can be detected as malicious, and therefore should be rejuvenated.

4.3.3 Pre-Filter protocol

Algorithm 1: Pre-filter$_i$ protocol

```
input: M: <DATA, EdgeMIS_ID, SEQ_NUMB,[MAC$_0$, ... ,MAC$_{3f}$] >
1 begin
2     if ( verify_Source (EdgeMIS_ID) == TRUE ) then
3         if ( check_message_flow (EdgeMIS_ID) == TRUE ) then
4             Total_Order_Multicast (M);
5 end
```

Protocol 1 presents the tasks performed by a Pre-filter when it receives a message M from the REB (see *input*). First, the Pre-filter checks if the source of the message in authorized to send traffic through the core-MIS (line 2). Each Pre-filter has a pre-configured list of authorized sources (typically edge-MIS identifiers). Then, the Pre-filter checks if the message is within the limits of the offered grant to edge-MIS, to prevent the execution of DoS attacks (line 3). Last, the Pre-filter uses the total order multicast to disseminate the message to the Filters (line 4).

4.3.4 Filter protocol

Algorithm 2: Filter$_i$ protocol

```
input: M = <DATA,EdgeMIS_ID,SEQ_NUMB,[MAC_0, ... ,MAC_{3f}] >
1  Initialization, executed only once;
2  SEQ_NUMB_EXPECTED ← 0;
3  WaitingQueue ← ∅;
4  begin
5      if (( verify_MAC(M) == TRUE and semantic_Validation(M) == TRUE ) ) then
6          M = M \[MAC_0, ... ,MAC_{3f}];
7          if ( SEQ_NUMB == SEQ_NUMB_EXPECTED) then
8              send_Voter(M);
9              SEQ_NUMB_EXPECTED ++;
10         else if ( SEQ_NUMB > SEQ_NUMB_EXPECTED and SEQ_NUMB <
           (SEQ_NUMB_EXPECTED + THRESHOLD) ) then
11             WaitingQueue = WaitingQueue ∪ {M};
12             if ( MinSeqNumber(WaitingQueue, SEQ_NUMB) == TRUE ) then
13                 MIN_SEQ_NUMB = SEQ_NUMB;
14     while ( MIN_SEQ_NUMB == SEQ_NUMB_EXPECTED) do
15         M = retrieveMessage(WaitingQueue );
16         SEQ_NUMB_EXPECTED ++;
17         MIN_SEQ_NUMB = getMinSeqNumber(WaitingQueue );
18         send_Voter(M);
19 end
```

Protocol 2 presents the tasks carried out by the Filter. After the total order multicast is executed, the message is delivered to each Filter in the same order. Every message has the identifier of the source, the sequence number, and a MAC vector with $3f + 1$ MACs (*input*). To simplify the presentation, we have excluded from the variables the identification of the source — a separate copy of every variable is need for each source. The first three lines of the algorithm are executed only once, when this source started to send messages.

First, the Filter verifies the corresponding MAC$_i$ in the message using the shared key (line 5). Then, based on a pre-defined security policy, it can also perform other semantic checks on the content of the message, such as verifying that the events were produced by a reasonable sensor (line 5). Next, the Filter removes the MAC vector from the message, since it will not be needed anymore.

Second, the Filter compares the message's sequence number with the one that is expected (line 7). If is the same sequence number, then the message is sent the Post-filter voter and the expected sequence number is incremented.

If the message's sequence number is greater than the expected and less than a pre-defined threshold (line 10), then the message is stored in a waiting queue. It also checks if the message just stored is the one with lowest sequence number, and if it is true, then the minimum sequence number is updated to that value.

To complete the procedure, the Filter checks if there are enqueue messages that need to be transmitted. Therefore, if the minimum message sequence number is equal to the expected sequence number (line 14), then the enqueued message is retrieved and sent to the Post-filer voter.

4.3.5 Post-Filter protocol

Algorithm 3: Post-Filter's protocol

```
input: Filter_ID: the Filter that sent the message
input: M: <DATA,EdgeMIS_ID,SEQ_NUMB >
1  Initialization, executed only once;
2  SEQ_NUMB_EXPECTED ← 0;
3  WaitingQuorom ← ∅;
4  begin
5      if ( SEQ_NUMB == SEQ_NUMB_EXPECTED) then
6          WaitingQuorom ← WaitingQuorom ∪ {<M, Filter_ID>};
7      else
8          if (SEQ_NUMB > SEQ_NUMB_EXPECTED) and (SEQ_NUMB <
           (SEQ_NUMB_EXPECTED + THRESHOLD) ) then
9              WaitingQuorom ← WaitingQuorom ∪ {<M, Filter_ID>};
10     while ( existsQuorom(SEQ_NUMB_EXPECTED) == TRUE ) do
11         aux = getQuorom(SEQ_NUMB_EXPECTED);
12         deliver_SIEM_ENGINE(aux);
13         garbageCollection(aux);
14         SEQ_NUMB_EXPECTED ++;
15 end
```

Protocol 3 presents the actions performed by the Post-filter. Fundamentally, it needs to ensure that correct, new, and ordered messages are delivered. To simplify the presentation, we have excluded from the variables the identification of the original source — a separate copy of every variable is need for each edge-MIS.

First, the Post-filter checks if the sequence number is the expected one (line 5), and in the affirmative case it stores the message in the *WaitingQuorom* set. Otherwise, it sees if the message has a sequence number larger than the expected (line 8), and in this case, the message is also stored in the same set. To prevent a malicious edge-MIS from sending large sequence numbers, which could overflow the Post-filter queue, we bound the accepted sequence numbers with a threshold (line 8). If the sequence number is under the expected sequence number, then the message is discarded because it was already delivered.

Second, the Post-filter verifies if there is a sufficiently large quorum of messages for the expected sequence number, i.e., if there are $f+1$ or more equal messages for the expected sequence number (line 10). Then, it gets the corresponding message and delivers it to SIEM Engine. Next, the set is garbage collected from the information related to this message and the expected sequence number is incremented.

4.4 Core-MIS deployment decisions

The intrusion-tolerant core-MIS requires several replicas for effective deployment. However, costs are a major concern of any administrator. Therefore, we present a few deployment alternatives that can be made based on our solution. In any case, one must keep in mind that resilience usually has associated costs.

Figure 4.5 presents the rational for making deployment decisions. The considered solutions try to tradeoff costs with the use of virtualization [8]. In all options, different replicas run in separate virtual machines and/or physical machines. A solution with more physical machines is desirable for more critical systems, due to the higher fault isolation and also because of the potentially better performance. With virtualization, each physical machine supports several virtual machines, which means that there might be less machinery costs but performance can be reduced as resources are shared. Although virtual machines provide some level of isolation among the different components, preventing in most cases intrusions from propagating from one replica to the others, hardware faults may have an impact in all replicas.

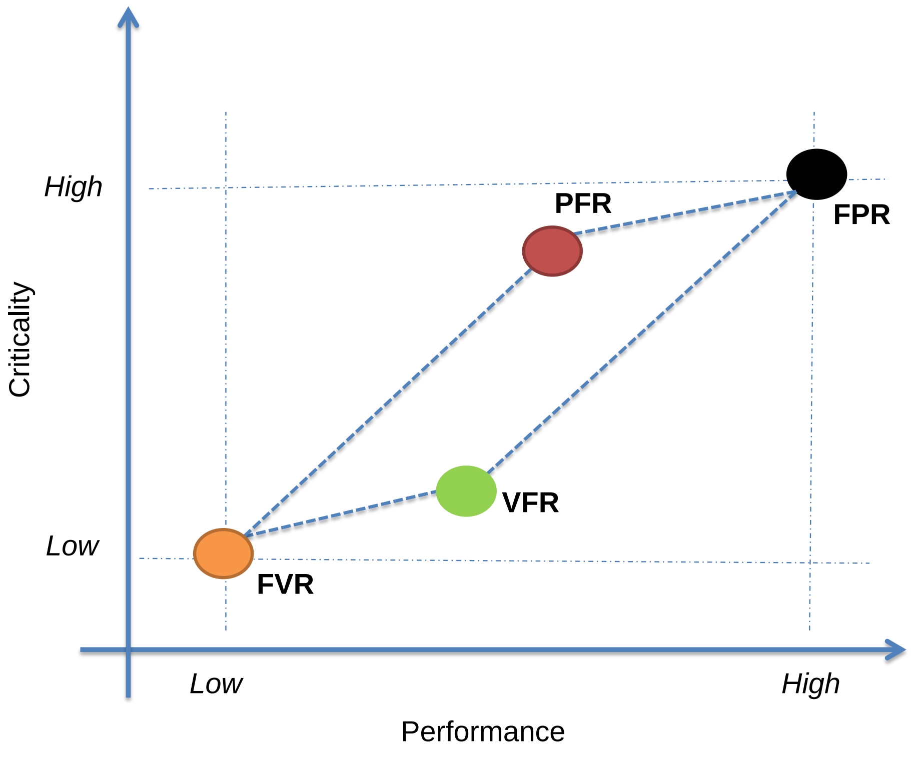

Figure 4.5: Two-axis deployment decisions: four deployment options based on Criticality and Performance.

Some possible deployment scenarios are:

Full Physical Replication (FPR) Every Pre-filter and Filter runs in a different physical machine.

Full Virtual Replication (FVR) Every Pre-filter and Filter runs in different virtual machines in the same physical machine.

Virtual Filter Replication (VFR) Pre-filters and Filters run in virtual machines, but the Pre-filters are in a physical machine and the Filters are in another physical machine.

Physical Filter Replication (PFR) Pre-filters run in virtual machines of the same physical machine, and each Filter runs in a separate physical machine.

5 Resilient Event Storage

The system under scrutiny of MASSIF generates massive amount of events per second. There are some practical algorithms for enforcing privacy on data stored on databases [30, 29, 54], storing no sensitive data at all would remove the problem altogether. But in Critical Infrastructure protection systems where security violation may result in terrible effects and put human life in danger, the reasons and responsible persons of this terrible incident would require to be disclosed. So MASSIF will have this constraint in great consideration when designing security enhancing solutions. By relying on Complex Event Processing (CEP) technology and on a stream-based computing paradigm, MASSIF solutions make enhancements in this ”zero storage” principle and makes use of what is called ”Least Persistence” principle. According to Least Persistence principle the only data which will be permanently stored by resilient storage is the one generated by processing engine at the time of security breach and will be helpful to identify the reasons of this breach. This stored data will be used for forensic analysis of the incident and will be made available only to authorized entities according to the regulations.

5.1 Architecture

A resilient architecture for forensic storage is a facility that allows secure storage of data by implementing a set of techniques which make data alteration difficult to achieve. These techniques may include usage of cryptographic functions and storage media equipped with tamper resistance/detection techniques. The proposed architecture of the Resilient Event Storage (RES) is shown in Figure 5.1. This architecture will take the security events as input and store them after digital signatures. The incoming data carry information about the security breach which can be used for forensic purposes later on. In order to be able to use this data for forensic purposes, it should be stored in raw format without any kind of processing.

The security events will be digitally signed prior to storage in order to ensure their authenticity and integrity for forensic purposes. Digital signatures will be performed using cryptographic algorithm RSA but instead of using classical RSA, it will be used in threshold cryptographic mode. The reason behind using RSA threshold signatures scheme instead of RSA classical is to enforce intrusion and fault tolerance in the architecture and to make sure that the system is available to sign the security events when a security breach occurs.

The first component starting from the left in Figure 5.1 is the Processing Engine in MASSIF which serves as an input source for the RES. Each event generated from Processing Engine related to security breach that requires to be stored forensically, will be sent to the controller. As described earlier, the events related to a security breach should be stored in raw format, so the security event “m” shown in Figure 5.1 will be in raw format. We suppose that we have n participants in the threshold signature scheme (a detailed study about threshold signature scheme can be found in [72]) and each participant holds the secret key share necessary to sign the events. These participants are represented by Node 1,..., Node n in Figure 5.1. In the initialization phase, the secret key is distributed among n shareholders in such a way that at least a threshold number of shares, let's say t where $t < n$, are necessary to calculate the full secret key and not less that t shares reveal even partial information about the secret key. After the secret key is divided into shares and distributed to all shareholders or nodes, the incoming message (containing information about the security breach) is sent in parallel by controller to all nodes. Each node computes a hash of the same message (represented as h in the Figure 5.1) and it uses its secret key share to perform a partial digital signature. This hash is unique for each distinct message. A component called Combiner is responsible for assembling all signature shares received from the shareholders in order to generate a full signature which is attached to the original message thus forming a signed security record i.e. a forensic record. This signed security event will be stored for future forensic analysis. The digital signatures achieved in this distributed way will be exactly similar to the digital signatures achieved by classical RSA with an advantage of intrusion and fault tolerance and enhanced availability of system. In fact, the Combiner is also responsible for verifying the signature shares it receives from all nodes. Each secret key share has a corresponding verification key share which can be used to verify the signature share generated with the secret key share. All these verification key shares are places in Combiner, so when a signature share arrives at Combiner, it will verify the validity of the signature share. After verification, the signature share will be used with other verified signature shares to generate a full signature otherwise the sender node is marked as compromised.

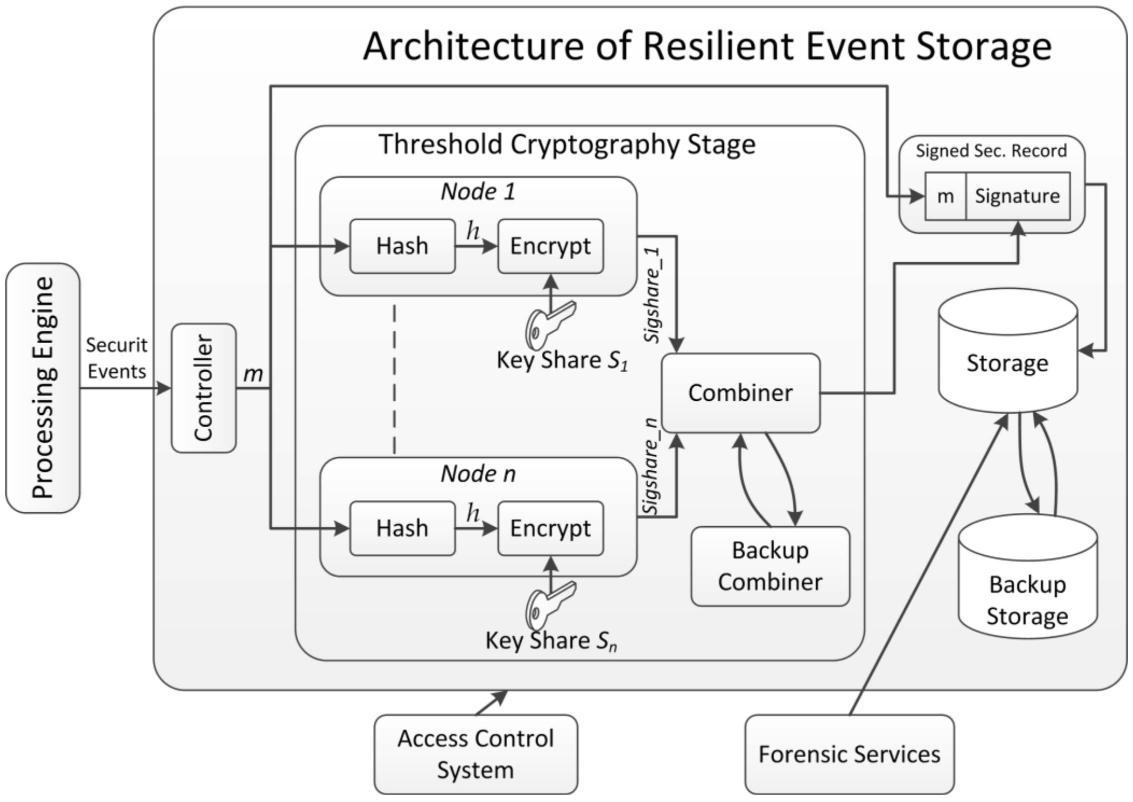

Figure 5.1: Architecture of Resilient Event Storage.

Forensic Services block represents the forensic analysis services which will be able to determine the altered data and intruder. This will be performed by analysing the information obtained by recorded events. The Access Control is the service that will control the access to the resources. Due to their importance in the architecture, Combiner and Storage medium will be replicated to avoid single point of failure. Diversity will be employed in the replication of these components to further improve the intrusion and fault tolerance in terms of using different programming languages and techniques for their implementation.

The described architecture will be implemented as a distributed application. The overall organization will follow the SOA model to offer services. Any new service is published by its provider in a registry component. The registry can be searched by users requiring specific services as shown in Figure 5.2.

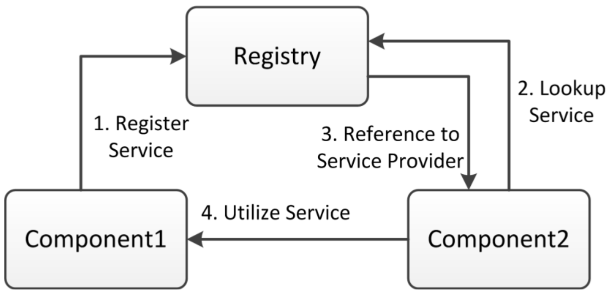

Figure 5.2: An example of registration and search of a service in Registry.

The registry provides a directory service and the application will be written as a choreography of services. In order to ensure a good level of security when the different services communicate with each other, only communication through SSL will be allowed. The authentication mode accepted will be bilateral.

5.2 Feature of Intrusion and Fault Tolerance

The proposed architecture will be able to function even when the system is under attack or some hardware faults occur. Usage of RSA threshold signature scheme instead of RSA classic guarantees that if some secret shares, less than the threshold, are compromised or some shareholders face some hardware faults or DoS attack, rest of the shareholders can do the job well and generate and send the signature shares to the Combiner which will generate full signatures. If Combiner is compromised in some way or bears some hardware faults, the Backup Combiner will start assembling the signature shares into full signatures so that the critical security events are not lost. If the storage medium is corrupted because of hardware faults, the Backup Storage will be available to store the signed security events. It is very important to mention that in parallel to Storage, also the Backup Storage stores all the signed events coming from Combiner or Backup Combiner so that if first level storage medium fails due to some reason, forensic records stored in Backup storage will be available for forensic purposes.

5.3 RES Availability

The security events generated by a critical infrastructure at the time of a security breach should be stored for forensic purposes. The intruders would try to compromise the RES in order to prevent the secure storage of security events. They can prevent some of the shareholders from signing the events or an DoS attack can be performed upon them and Combiner. The presence of Backup Combiner and a certain threshold of the shareholders to sign the events ensure the availability of the RES to sign and store the events. If the attacker is successful in compromising some of the shareholders and Combiner, the remaining shareholders and Backup Combiner continue to perform their job of signing security events.

6 Conclusions

In this report, we described the preliminary specification of services and protocols for the MASSIF resilient SIEM system. Our solution intends to reply to the challenge of how to achieve high levels of security and dependability in a system that is potentially distributed across several facilities and that has some strict requirements in terms of timeliness and performance. The document addresses the following main areas:

Authenticated Component Event Reporting discusses mechanisms that can be employed to give evidence that produced event data has not been tampered with, and therefore, that can effectively be used to make decisions with regard to security problems that are observed in the monitored systems.

Resilient Event Bus (REB) presents a solution for data dissemination from the edge-MIS towards the core-MIS. This solution achieves high levels of resilience by resorting to a number of mechanisms, such multipath data transmission, multihoming, and coding algorithms.

Node defense mechanisms explains a set of techniques that can be applied in an incremental way to make nodes increasingly more resilient to different forms of faults, of either accidental nature or malicious attacks. The design of a highly resilient core-MIS that takes advantage of the proposed techniques is also described.

Resilient Event Storage (RES) presents a solution for the secure archival of event data. RES gives support for the forensic analysis of an incident based on the stored data, and enforces security policies that ensure that events are only made available to authorized entities according to the regulations.

In the next deliverable of WP5.1, D5.1.4, the specification of the resilient architecture will be further refined, and the complete description of services and protocols will be provided.

Bibliography

[1] Project BFT-SMaRt. http://code.google.com/p/bft-smart/, 2012.

[2] M. Abd-El-Malek, G. Ganger, G. Goodson, M. Reiter, and J. Wylie. Fault-scalable Byzantine fault-tolerant services. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 59–74. ACM, 2005.

[3] 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.

[4] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 131–145, 2001.

[5] T. Anderson, T. Roscoe, and D. Wetherall. Preventing internet denial-of-service with capabilities. *SIGCOMM Comput. Commun. Rev.*, 34(1):39–44, January 2004.

[6] J. Antunes and N. Neves. Diveinto: Supporting diversity in intrusion-tolerant systems. In *Proceedings of the 30th IEEE Symposium on Reliable Distributed Systems*, October 2011.

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

[8] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.

[9] M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, pages 27–30, 1983.

[10] A. Bessani, E. Alchieri, M. Correia, and J. Fraga. DepSpace: a Byzantine fault-tolerant coordination service. In *Proceedings of the ACM SIGOPS/EuroSys European Conference on Computer Systems – EuroSys'08*, pages 163–176. ACM, 2008.

[11] D. Brumley and D. Boneh. Remote timing attacks are practical. In *Proceedings of the 12th Conference on USENIX Security Symposium*, pages 1–14, 2003.

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

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

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

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

[16] MASSIF Consortium. *Deliverable D5.1.1 - Preliminary Resilient Framework Architecture*. Project MASSIF EC FP7-257475, September 2011.

[17] MASSIF Consortium. *Architecture Document*. Project MASSIF EC FP7-257475, April 2012.

[18] L. Coppolino, M. Jäger, N. Kuntze, and R. Rieke. A Trusted Information Agent for Security Information and Event Management. In *Proceedings of the International Conference on Systems*. XPS, 2012.

[19] M. Correia, N. Neves, Lau Lung, and P. Verissimo. Low complexity byzantine-resilient consensus. *Distributed Computing*, 17(3):237–249, 2005.

[20] M. Correia, N. Neves, and P. Veríssimo. How to tolerate half less one Byzantine nodes in practical distributed systems. In *Proceedings of the 23rd IEEE Symposium on Reliable Distributed Systems*, October 2004.

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

[22] M. Correia, G. Veronese, N. Neves, and P. Verissimo. Byzantine consensus in asynchronous message-passing systems: a survey. *International Journal of Critical Computer-Based Systems*, 2(2):141–161, 2011.

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

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

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

[26] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77–97, January 1987.

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

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

[29] A. Evfimievski, J. Gehrke, and R. Srikant. Limiting Privacy Breaches in Privacy Preserving Data Mining. In *Proceedings of the Symposium on Principles of Database Systems*, pages 211–222, 2003.

[30] A. Evfimievski, R. Srikant, J. Gehrke, and R. Srikant. Privacy Preserving Mining of Association Rules. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pages 217–228, 2002.

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

[32] M. Garcia, A. Bessani, I. Gashi, N. Neves, and R. Obelheiro. OS diversity for intrusion tolerance: Myth or reality? In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 383–394, June 2011.

[33] M. Garcia, N. Neves, and A. Bessani. Diversys: Diverse rejuvenation system. In *Proceedings of the INFORUM - Simpósio de Informática*, September 2012.

[34] P. Gladyshev and A. Patel. Formalising event time bounding in digital investigations. *International Journal of Digital Evidence*, 2005.

[35] D. Grawrock. *Dynamics of a Trusted Platform: A Building Block Approach*. Intel Press, 1st edition, 2009.

[36] Trusted Computing Group. TCG Trusted Network Connect – TNC IF-MAP Binding for SOAP Version 2.0. www.trustedcomputing.org, 2010.

[37] Trusted Computing Group TPM Working Group. TCG Specification Architecture Overview. http://www.trustedcomputinggroup.org/resources/, 2007.

[38] K. Gummadi, H. Madhyastha, S. Gribble, K. Levy, and D. Wetherall. Improving the Reliability of Internet Paths with One-hop Source Routing. In *In Proceedings of the Symposium on Operating Systems Design & Implementation*, 2004.

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

[40] F. Izquierdo, M. Ciurana, F. Barcelo, J. Paradells, and E. Zola. Performance evaluation of a toa-based trilateration method to locate terminals in wlan. In *1st International Symposium on Wireless Pervasive Computing*, 2006.

[41] R. Kissel. Glossary of key information security terms. NIST Interagency Reports NIST IR 7298 Revision 1, National Institute of Standards and Technology, February 2011.

[42] N. Kuntze, D. Mähler, and A. U. Schmidt. Employing trusted computing for the forward pricing of pseudonyms in reputation systems. In *Axmedis 2006, Proceedings of the 2nd International Conference on Automated Production of Cross Media Content for Multi-Channel Distribution, Volume for Workshops, Industrial, and Application Sessions*, 2006.

[43] Nicolai Kuntze and Carsten Rudolph. Secure digital chains of evidence. In *Sixth International Workshop on Systematic Approaches to Digital Forensic Engeneering*, 2011.

[44] T. Kunz, S. Okunick, and U. Pordesch. Data Structure for Security Suitabilities of Cryptographic Algorithms (DSSC)-Long-term Archive And Notary Services (LTANS). Technical report, IETF Internet-Draft, 2008.

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

[46] M. LeMay and C. Gunter. Cumulative attestation kernels for embedded systems. *Computer Security–ESORICS 2009*, pages 655–670, 2009.

[47] J. Liu, F.R. Yu, Lung C.-H., and H. Tang. Optimal combined intrusion detection and biometric-based continuous authentication in high security mobile ad hoc networks. *IEEE Transactions on Wireless Communications*, 8(2), 2009.

[48] M. Luby. Lt codes. In *Proceedings of the 43rd Symposium on Foundations of Computer Science*, pages 271–280, 2002.

[49] M. Luby. LT Codes. In *Proceedings of the IEEE Symposium on the Foundations of Computer Science*, pages 271–280, 2002.

[50] M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, and L. Minder. RaptorQ Forward Error Correction Scheme for Object Delivery. IETF RFC 6330, August 2011.

[51] D. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, New York, NY, USA, 2002.

[52] D. Malkhi and M. Reiter. Unreliable intrusion detection in distributed computations. In *Proceedings of the 10th Computer Security Foundations Workshop*, pages 116–124, June 1997.

[53] M. Marsh and F. Schneider. CODEX: A robust and secure secret distribution system. *IEEE Transactions on Dependable and Secure Computing*, 1(1):34–47, January 2004.

[54] D. Martin, D. Kifer, A. Machanavajjhala, J. Gehrke, and J. Halpern. Worst-Case Background Knowledge for Privacy-Preserving Data Publishing. In *Proceedings of the International Conference on Data Engineering*, pages 126–135, 2007.

[55] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. IETF RFC 2018, October 1996.

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

[57] C. Mitchell. *Trusted Computing*. Iet, 2005.

[58] H. Moniz, N. Neves, M. Correia, and P. Veríssimo. Randomized intrusion-tolerant asynchronous services. In *Proc. of the Int. Conf. on Dependable Systems and Networks*, jun 2006.

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

[60] N. Neves, M. Correia, and P. Verissimo. Solving vector consensus with a wormhole. *IEEE Transactions on Parallel and Distributed Systems*, 16(12):1120–1131, 2005.

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

[62] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, and Y.-C. Hu. Portcullis: protecting connection setup from denial-of-capability attacks. In *Proceedings of the 2007 conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '07, pages 289–300, 2007.

[63] V. Paxson, M. Allman, J. Chu, and M. Sargent. Computing TCP's Retransmission Timer. IETF RFC 6298, June 2011.

[64] M.M. Pollitt. Report on digital evidence. In *13th INTERPOL Forensic Science Symposium*. Citeseer, 2001.

[65] M. O. Rabin. Randomized Byzantine generals. In *Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science*, pages 403–409, 1983.

[66] M. Reith, C. Carr, and G. Gunsch. An examination of digital forensic models. *International Journal of Digital Evidence*, 1(3):1–12, 2002.

[67] J. Richter, N. Kuntze, and C. Rudolph. Securing digital evidence. In *Proceedings of the Fifth International Workshop on Systematic Approaches to Digital Forensic Engeneering*, pages 119–130, 2010.

[68] J. Richter, N. Kuntze, and C. Rudolph. Security Digital Evidence. In *2010 Fifth International Workshop on Systematic Approaches to Digital Forensic Engineering*, pages 119–130. IEEE, 2010.

[69] T. Roeder and F. Schneider. Proactive obfuscation. *ACM Transactions on Computer Systems*, 28(2):4:1–4:54, 2010.

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

[71] A. Shokrollahi. Raptor codes. *IEEE Transactions on Information Theory*, 52(6):2551–2567, June 2006.

[72] V. Shoup. Practical Threshold Signatures. In *Proceedings of the International Conference on Theory and Application of Cryptographic Techniques*, pages 207–220, 2000.

[73] A. Snoeren, K. Conley, and D. Gifford. Mesh-based content routing using XML. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, pages 160–173, 2001.

[74] P. Sousa, A. Bessani, M. Correia, N. Neves, and P. Veríssimo. Highly available intrusion-tolerant services with proactive-reactive recovery. *IEEE Transactions on Parallel Distributed Systems*, 21(4):452–465, April 2010.

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

[76] G. Starnberger, L. Froihofer, and K.L. Goeschka. Using smart cards for tamper-proof timestamps on untrusted clients. In *International Conference on Availability, Reliability and Security, ARES*, pages 96–103. IEEE Computer Society, 2010.

[77] Frederic Stumpf, Andreas Fuchs, Stefan Katzenbeisser, and Claudia Eckert. Improving the scalability of platform attestation. In *Proceedings of the Third ACM Workshop on Scalable Trusted Computing (ACM STC'8)*, pages 1–10, Fairfax, USA, October 31 2008. ACM Press.

[78] Symantec. Symantec internet security threat report vol. 16. http://msisac.cisecurity.org/resources/reports/documents/SymantecInternetSecurityThreatReport2010.pdf, April 2011.

[79] Symantec. Symantec internet security threat report vol. 17. http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_2011_21239364.en-us.pdf, April 2012.

[80] B. Vavala and N. Neves. Robust and speculative Byzantine randomized consensus with constant time complexity in normal conditions. In *Proceedings of the 31th IEEE Symposium on Reliable Distributed Systems*, October 2012.

[81] P. Veríssimo, N. Neves, and M. Correia. Intrusion-tolerant architectures: Concepts and design. In *Architecting Dependable Systems*, volume 2677 of *LNCS*. 2003.

[82] M.G. Wing, A. Eklund, and L.D. Kellogg. ”consumer-grade global positioning system (gps) accuracy and reliability”. *Journal of Forestry*, 103, 2005.

[83] T. Winkler and B. Rinner. Applications of trusted computing in pervasive smart camera networks. In *Proceedings of the 4th Workshop on Embedded Systems Security*, page 2. ACM, 2009.

[84] T. Winkler and B. Rinner. TrustCAM: Security and Privacy-Protection for an Embedded Smart Camera based on Trusted Computing. In *Proceedings of the Conference on Advanced Video and Signal-Based Surveillance*, 2010.

[85] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin. Separating agreement from execution for Byzantine fault tolerant services. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 253–267, 2003.