Eindhoven University of Technology
Department of Mathematics and Computer Science

Passive Network Audit Framework

Master Thesis

Javier Santillan

Supervisors:
Dr. Jerry den Hartog (TU/e)
M.Sc. Barry Weymes (Fox-IT)
Dr. Tanir Ozcelebi (TU/e)

Public Version

Eindhoven, August 2014

Abstract

"Passive Network Audit" technology includes network discovery and monitoring techniques in which network packets are captured, processed and analyzed in order to gather information about the network. It relies on passive analysis of network traffic activities, meaning that no single active interaction takes place between auditor and network environment. The goal of passive network audit is to assess the *security level* of the network environment, based on predefined baselines (e.g policies, rules). By using different detection techniques such as protocol behaviour analysis, network enumeration, signature-based Intrusion Detection Systems (IDS), Network Flow Analysis (NFA) and Deep Packet Inspection (DPI), it is possible to identify network threats taking into account network elements, patterns and known vulnerabilities. This research aims to identify the gap between Network Security Monitoring (NSM), Security Information and Event Management (SIEM) and Passive Network Audit (PNA) by analyzing some taxonomies and existing frameworks. Findings of this background study are taken into consideration to define a security framework which is intended to take features of different technologies in order to provide a very flexible, automated and reliable assessment framework that can be implemented as passive audit technology. Hence, this framework aims to provide complementary features to SIEM and NSM approaches in order to get a better context of security events. Furthermore, this thesis defines the prototype that provides the capabilities of an automated passive network audit engine. It was developed as internal project of Fox-IT.

Release Notes

The present document is the ***Public version*** of a research developed as part of an internal Fox-IT project. This master thesis contains the foundations of *Passive Network Audit Framework* research, omitting specific information (e.g. internal design features, datasets, etc). For further information you may contact Fox-IT.

Fox-IT B.V.
Olof Palmestraat 6
P.O. box 638
2600 AP Delft
The Netherlands

Phone: +31 (0)15 284 7999
Fax: +31 (0)15 284 7990
E-mail: info@fox-it.com
Internet: http://www.fox-it.com

Preface

This Master's thesis is the result of my graduation project executed at Fox-IT in The Netherlands. The graduation project is part of my Master in Computer Science and Engineering - *Information Security Technology* at the Eindhoven University of Technology. This security-focused master program is coordinated by the Kerckhoffs Institute, a collaboration program between the University of Twente, Radboud University Nijmegen and the Eindhoven University of Technology.

I would like to express my gratitude to all people who helped to finish my thesis. I would like to thank to my supervisor at TU/e, Jerry den Hartog, who provided valuable guidance to write my thesis and find a better way to conduct my research. Also, I would like to thank to my technical supervisor at Fox-IT, Barry Weymes, who opened the doors of an awesome security company, and who always provided me valuable feedback, time and resources to improve my research.

Of course, I would like to thank to my family, Guillermina, Magdaleno, Daniel. Without their trust and support I would never get that far, they have played a key role to achieve all my goals. My sincere thanks to all my friends in Mexico and Netherlands who always have been there to help, work together and live invaluable moments.

Finally, I am very thankful to three organizations. First, CONACYT (Consejo Nacional de Ciencia y Tecnología) for its trust and financial support that allowed me to study abroad during these two years. Also, Fox-IT, a great security company that allowed me to develop and reinforce security skills within an awesome environment. Finally, UNAM-CERT, a great place where I learnt and gained experience in information security, specially intrusion detection skills that allowed me to develop my thesis in a better way.

Javier Santillan

August, 2014

Contents

List of Figures

List of Tables

Listings

List of Algorithms

Acronynms

PNAF	Passive Network Audit Framework
PNA	Passive Network Audit
PND	Passive Network Discovery
NSM	Network Security Monitoring
IDS	Intrusion Detection Systems
IPS	Intrusion Prevention Systems
NIDS	Network Intrusion Detection Systems
NFA	Network Flow Analysis
DPI	Deep Packet Inspection
SIEM	Security Information and Event Management
SIM	Security Information Manager
SEM	Security Event Manager
ML	Machine Learning
CIDF	Common Intrusion Detection Framework
IDWG	Intrusion Detection working Group
CISL	Common Intrusion Specification Language
IDXP	Intrusion Detection eXchange Protocol
IDMEF	Intrusion Detection Message Exchange Format
IDR	Intrusion Detection and Response
TCP	Transmission Control Protocol
EC	Event Calculus
SME	Small and Medium Enterprises
DCM	Data Capture Module
DPM	Data Processing Module
DVM	Data Visualization Module
NTCE	Network Traffic Capture Engine

NTPE Network Traffic Processing Engine

NPEE Network Profiling and Enumeration Engine

IDSE Intrusion Detection System Engine

NFAE Network Flow Analysis Engine

DPIE Deep Packet Inspection Engine

IDCE Intermediate Data Correlation Engine

NSAE Network Security Audit Engine

GSVE Graphic Security Visualization Engine

SARE Security Audit Report Engine

CIDR Classless Inter-Domain Routing

OS Operating System

HTTP Hypertext Transfer Protocol

DNS Domain Name System

TLS Transport Layer Security

JSON JavaScript Object Notation

API Application Programming Interface

CPAN Comprehensive Perl Archive Network

CVE Common Vulnerabilities and Exposures

CSV Comma-Separated Value

IP Internet Protocol

UDP User Datagram Protocol

Chapter 1

Introduction

Intrusion Detection is a field of Information Security introduced some decades ago. Some of the very first versions of Intrusion Detection Systems (IDS) prototypes were introduced in the early 70's by Anderson [4], [5], and over the last decades new models have been used as a part of the security strategies within organizations. IDS aim to identify and detect security issues within computer systems, network environments or information systems in general. According to Debar et al. [21], the task of IDS is to monitor the usage of such systems and to detect the apparition of *insecure states*. Moreover, IDS detect attempts and active misuse by legitimate users or external parties to abuse privileges or exploit security vulnerabilities.

Depending on the way security issues are detected, IDS can be classified in two main categories: *misuse-based* and *anomaly-based* detection. In general, the triggering of misuse and anomaly based security alerts depend on predefined patterns and behaviours exceeding *normal* thresholds, respectively. Moreover, as it will be addressed in detail on Chapter 2, there are different taxonomies that describe standard features of IDS and the capabilities that they can provide. Complex prototypes can combine these models in order to implement IDS with larger scope. In fact, features of modern IDS have evolved over the years in such a way that, novel and improved detection models are used in order to detect new security threats. However, the fact that not only data networks are more complex at present days, but also the way network protocols are implemented and the amount of information that needs to be analyzed, imply that security threats are more difficult to identify as well.

The incursion of detection complexity implies that IDS detection mechanisms might not be enough to identify security threats and their full context. Hence, new technologies have emerged in order to complement detection systems. These technologies implement new approaches in detection methods based on different techniques such as patterns analysis, data logs processing, network statistics, structured traffic analysis, data aggregation, etc. However, it is important to mention that they are not IDS themselves, but they provide complementary information based on additional data sources. In practice, some of the aforementioned approaches are represented by Network Security Monitoring (NSM), Security Information and Event Management (SIEM) and Passive Network Audit (PNA). These three approaches aim to get awareness of security context with slight differences in methods and focus, sharing features that can be used in different ways depending on the context of analysis. Figure 1.1 presents an overview of this relationship. Moreover, these relatively recent technologies have played an important role within security mechanisms of enterprises, since they are now able to merge a set of threat identification datasets from different sources (e.g. IDS, Firewalls, OS, Applications, Network devices, etc.) into one single point of analysis and provide an structured platform for data analysis.

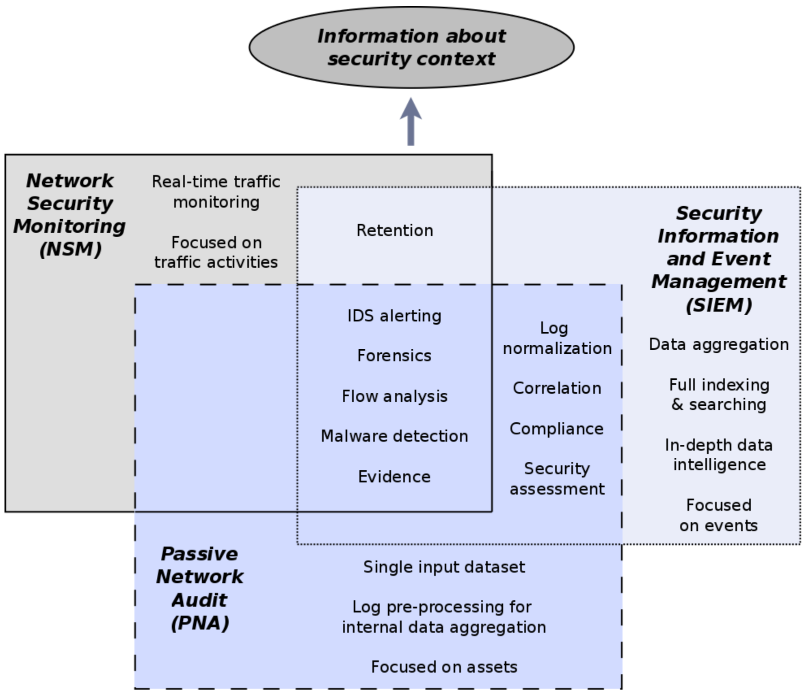

Information about security context
Network Security Monitoring (NSM)
Real-time traffic monitoring
Focused on traffic activities
Retention
Security Information and Event Management (SIEM)
Data aggregation
Full indexing & searching
In-depth data intelligence
Focused on events
IDS alerting
Forensics
Flow analysis
Malware detection
Evidence
Log normalization
Correlation
Compliance
Security assessment
Passive Network Audit (PNA)
Single input dataset
Log pre-processing for internal data aggregation
Focused on assets

Figure 1.1: Overview of analysis models for security events awareness.

1.1 Network Security Monitoring (NSM)

NSM is a detection and monitoring model focused on the analysis of network traffic activity. This concept has been developed at the same time new detection technologies have emerged to be applied into security tools. NSM includes a set of processes and data analysis techniques. It defines guidelines and methods of how communication and security devices can be deployed over the network in such a way that security monitoring can be performed in an effective way. Furthermore, it defines how the information generated by such devices is processed and how it can be interpreted in order to create meaningful information.

Depending on diverse factors such as network characteristics, protocols and even budgets, among others, NSM describes the general issues to define a monitoring framework. In this context, different tools and techniques can be used on similar environments while the same monitoring requirements are met. For instance, there is a set of open source tools that can be used in order to deploy a NSM infrastructure, some of them working as middle layer between detection engines and front-ends for data analysis. It has to be emphasized that NSM describes not only the tools that can be used, but also how, when and why to use them.

Tools like Sguil [88] have been widely deployed among network monitoring infrastructures. Sguil is an open source tool that combines some passive detection techniques and provides to network analysts an interface to perform a deep packet inspection and signature-based alert analysis. Moreover, open source IDS such as Snort [70] and Suricata [61] are also widely used on NSM tools. The key factor that should be emphasized is that NSM is a multilayer process in which, for example, IDS are just part of the whole monitoring environment, since other tools with different approaches such as Argus [68], Pads [73], Sancp [20], etc, use passive analysis techniques as well, in order to complement data generated by IDS.

Figure 1.2 shows an approach of the *Network Incident Response Process*, in which NSM takes place within the *Detection* phase, playing a role in two different ways:

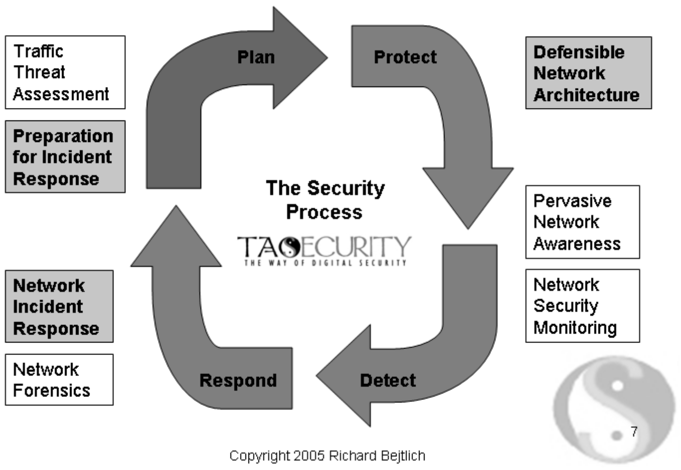

Traffic Threat Assessment
Preparation for Incident Response
Plan
Protect
Defensible Network Architecture
The Security Process
TAOSECURITY
THE WAY OF DIGITAL SECURITY
Pervasive Network Awareness
Network Security Monitoring
Network Incident Response
Network Forensics
Respond
Detect
7
Copyright 2005 Richard Bejtlich

Figure 1.2: Network incident response process described in [10]

1. **Short term incident containment**: Steps taken immediately upon confirmation that intrusion has occurred.
2. **Emergency**: Operations that look for additional intrusion patterns. It is a validation mechanism able to scope the extent of the incident, in other words it provides the evidence of malicious activity.

It is very important to mention that NSM takes into account not only a set of analysis techniques, but also deployment considerations including threats models, monitoring zones, deployment locations, risk, incident response procedures, etc. as crucial part of NSM successful implementation. Thus, in order to gather data that eventually will trigger alerts and produce statistical information, NSM sensors should be placed into specific locations within the network infrastructure. Furthermore, the actual useful data that can be generated depends on the amount and kind of traffic collected by sensors, which also are directly linked to the deployment location. The *Reference Intrusion Model* proposed in [10] includes the following types of data:

1. *Full content data*: Logging of every single bit of network traffic to perform forensics analysis. In practice this is difficult to achieve due to performance issues, specially on large networks.
2. *Session data*: Network traffic aggregates, protocol breakdown and distributions.
3. *Statistical data*: Connection pair records as part of conversation between two hosts. It takes the concept of *flow* as filtering parameter (i.e. proto, src ip, src port, dst ip, dst port)
4. *Alert data*: Generated by Intrusion Detection Systems. Provides alerts as indication of anomalous activities. It serves as base point of further investigations.

For each type of data, a set of tools and techniques can be used in order to extract useful datasets. The goal is to correlate the information generated on each stage in order to produce meaningful interpretations about security events. Thus, depth of NSM depends on the requirements of the monitoring infrastructure. Bejtlich [11], [10] presents two complete guides of aspects that should be taken into account in order to deploy a monitoring infrastructure, including techniques and tools that can be used, and further considerations to get the best detection results.

Certainly, intrusion detection is a sub process part of Network Security Monitoring. In order to understand the context, on the one hand, by using NSM it is possible to describe network security activity either as a general picture (summary of network security-related activity) or in a very specific way for certain purposes and environments (structured traffic analysis, forensics, etc). On the other hand, *intrusion detection* is focused on the way how these security events are identified. In fact, NSM may use intrusion detection techniques such as signature-based and anomaly-based IDS (which will be discussed in Chapter 2). Some discussions[1] emphasize the fact that IDS developers want "*Immaculate Detection*" whereas NSM practitioners want "*Immaculate Collection*". Certainly there is no product that provides 100% of accuracy and effectiveness, hence further investigations need to be performed. For example, despite the fact that a very complex IDS can trigger security alerts, depending on the situation, an additional investigation might be needed, hence NSM is applied in order to provide evidence and to interpret the incident with a more detailed or full context.

1.2 Security Information and Event Management (SIEM)

Process Mining is a research area focused on the analysis of event logs that aims to get knowledge about the business processes. Moreover the purpose of *process mining* is to develop methods and tools that provide the capabilities for discovering process and to exploit the data recorded into event logs. In order to achieve that purpose, techniques of *Data mining*, which refers to the process of extracting descriptive models from large stores of data [25], are applied in different ways through the use of approaches such as *statistical analysis* and machine learning, both described in Chapter 2. In the field on Information Systems, specifically within Information Security, *process mining* can be applied in order to analyze log datasets generated by security sources such as Operating Systems, Firewalls, IDS, Intrusion Prevention Systems (IPS), communication devices, etc.

There are different approaches to apply process mining for intrusion detection purposes. The so-called Security Information and Event Management (SIEM), Security Event Manager (SEM), Security Information Manager (SIM) are examples of such process mining approaches. In practice, there exist implementations based on open source tools [86], as well as commercial solutions [85], [49], [53] that provide log management and SIEM capabilities for both, general and security-related events. In fact, in order to provide a wider framework, SIEM combines features of both SIM and SEM which in general are focused on real-time analysis and long-term storage respectively. Thus, among the most important capabilities that SIEM provides are:

- *Data aggregation*: Data from different sources is used as *input data* that feeds log analysis engines. Such data may contain not only security related event information, but also general system logs, operational system trails, etc.
- *Correlation*: Based on different data sources, a meaningful interpretation can be generated by linking events taking into account common parameters and attributes.
- *Alerting*: When an anomaly or predefined behaviour is detected, then an alert can be triggered in order to notify a possible security issue. This can be done by using either IDS datasets or by post analysis of other security logs (e.g. firewalls).
- *Compliance*: Auditing techniques can be applied in order to assess whether specific procedures are followed within a system or network environment.
- *Retention*: Historical data is stored in order to feed datasets for data correlation over the time. This is specially useful for forensics and incident response purposes.
- *Forensics analysis*: Analysis of different datasets are correlated based on a timeline in order to reconstruct events that happened as part of security incidents.
- *Indexing*: Uses special techniques to facilitate indexing and searching information on aggregated data. This is a vital task since data aggregation and retention imply huge amount of data to analyze.

[1]Post by Richard Bejtlich's on TaoSecurity blog. `http://taosecurity.blogspot.nl/2007/03/nsm-and-intrusion-detection-differences.html`

- *Intelligence*: By combining the aforementioned capabilities, a better, detailed and meaningful context can be described in order to feed a knowledge database for high-level interpretations suitable for decision making processes.

Furthermore, it is important to mention that log analysis processes imply the use of passive methods of data processing since the source dataset contains logs of events that already happened, and there is no any active interaction with the network environment at all.

1.3 Passive Network Audit (PNA)

There is a similar approach in which both log analysis and data correlation are involved as well, the so-called PNA. Although typically some SIEM implementations [85], [49], [53], [7] include PNA as part of their capabilities, PNA itself includes additional processes and it can be applied independently without all the features of SIEM. Thus, despite the fact that PNA can be considered as a part of unified SIEM system, in this thesis PNA will be distinguished as independent term, emphasizing the key characteristics that PNA provides as individual component.

PNA is focused on network traffic analysis captured as source dataset, mainly as *offline-analysis*. Thus, a network traffic sample of fixed period is analyzed by characterizing network components and tracking their activities. The goal of the analysis is to get an assessment of network events performed within certain period of time, as well as assess policy compliance based on predefined baselines. Nevertheless, a real-time analysis is possible, turning PNA into a hybrid system with some NSM and SIEM features, or in the other way around, by enabling PNA within SIEM implementation. As it is shown in Figure 1.1, there are shared properties among the three approaches. In the case of PNA and SIEM, capabilities such as log normalization, correlation, compliance and security assessment are common, however the key differences are described as follow:

- Unlike normal aggregation process in SIEM, where the system is fed by logs generated by multiple devices or systems such as Firewalls, OS, IDS, etc, in PNA the input dataset is the raw network traffic itself. The data aggregation process is performed as internal task in such a way that different kind of decoded data sources feed analysis engines that PNA may include to achieve data correlation, compliance and security assessment.

- The single input dataset imply the use of *log pre-processing* as prior additional task for data aggregation. This means there are no logs or traces that can be aggregated as input data directly, but rather raw traffic contains all the information that can be used to produce *logs*, in the same or similar way other devices fed normally SIEM systems. This *log pre-processing* task may involve traffic decoding, payload extraction and some specific data correlation tasks at network level. Figure 1.3 shows the general information flow in PNA where additional pre-processing task can be appreciated before the actual log analysis.

- The focus of PNA is on *Assets* identification, characterization and tracking, since these are the entities that will be evaluated as part of an *auditing* process. Such information can feed SIEM in order to produce a better intelligence about events in a wider context.

PNA may also be referred as Passive Network Discovery (PND). Arkin [8] describes an overview about PND and monitoring systems, discussing its limitations and giving a criticism about some current weaknesses on this technology. Usually, the goal of passive technologies is to answer the questions *"What is on the enterprise network?"*, *"Who is on the enterprise network?"* and *"What is being done on the enterprise network?"*. To this end, new technologies have emerged, providing capabilities to describe (a) Active network elements and their properties, (b) Active network services and their versions, (c) Distances between active network elements and the monitoring point, (d) Active client-based software and their versions, (e) Network utilization information, (f)

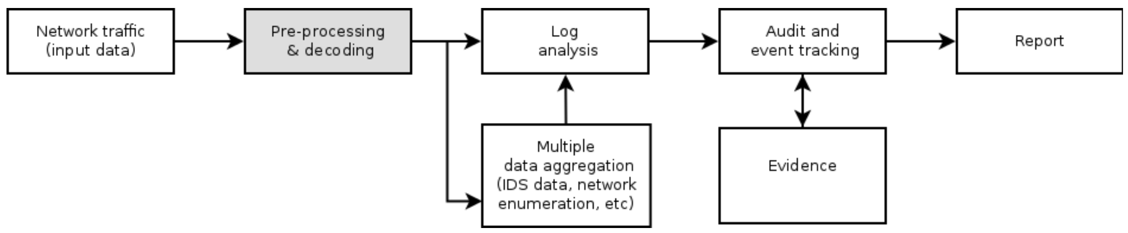


Figure 1.3: General workflow of PNA.

Vulnerabilities found for network elements residing on the monitoring network. In general terms such information can be used for the following purposes:

- Network Audit
- Network utilization information
- Network forensics
- Vulnerability discovery
- Network operation profiling

1.3.1 Strengths of PNA

- *Passive technology*: This feature represents one of the best advantages of PNA, since behaviour, integrity or characteristics of the monitored network are not affected by the monitoring infrastructure. This is possible since there is not any single active interaction between monitors and monitoring targets.
- *Real-time operation*: The monitoring process can be performed in real-time as the network activities occur.
- *Zero performance impact*: No additional workload to the monitored network is involved. In fact, workload implied by the monitoring infrastructure is independent of the working network environment.
- *Full data processing*: It is possible to perform analysis over all network layers.
- *Detection of active network elements*: It is possible to identify network elements (e.g. hosts, communication devices, etc.) participating on network communications.
- *Granular network communication*: PND is able to provide the knowledge needed to generate information about network infrastructure utilization.
- *Abnormality Detection*: By using the collected data, it is possible to perform statistical analysis in order to identify abnormal activities within the network.

1.3.2 Weaknesses of PNA

- *Limited analysis*: the analysis depends on the data that has been collected. If any data traffic related to certain event is not collected, then event identification might be limited to partial interpretations and even out of the scope of the analysis.
- *Not everything can be passively determined*: Some network features can not be determined by just analyzing the traffic if there is no enough data for interpretation. For instance, data for vulnerability discovery or system's attributes identification.

- *Performance issues*: In large networks, passive monitoring becomes a hard task due to the amount of traffic that needs to be collected. Furthermore, there are important issues on full capture analysis schemes (e.g. performance).
- *Reliability and accuracy*: Passive analysis is not completely accurate since some identification processes are based on assumptions that may lead to wrong interpretations, for instance, OS detection might be ambiguous, hence inaccurate.

Additionally, De Montigny-Leboeuf et al. [56] mention three categories of passive packet monitoring: (1) *Singleton*: Tests conducted on a single packet emitted by a host. It typically looks for default values of header fields such that information about the sender is retrieved. The general algorithm to perform a singleton is by monitoring traffic with specific filter and then derive information once packets are captured. (2) *Sample*: This test involves samples of packets generated by a host. Its purpose is to analyze how packets change while they are transmitted. The general algorithm consists in traffic monitoring with specific filters, then hold in memory all received packets by source address, to finally derive information from the sample. (3) *Stimulus-Response*: This test consists in listening communication in both directions, creating a *stimulus-response* pair. The analysis takes into account facts such as stimulus with no response, as well as response with no stimulus. The general algorithm consist as follows: monitoring the traffic with specific filter satisfying either the stimulus or response, then hold the stimulus and look for its corresponding response pair (conversely hold the response and check it corresponding stimulus), to infer information about the event.

The important aspects of applying PND cited in [56] are: (a) Discover active nodes, (b) System uptime, (c) OS identification, (d) Network elements roles, (d) Services offered, (e) protocols supported and (f) IP network configuration.

Moreover, Gula [33] describes two categories of vulnerabilities that can be identified by using PND. On the one hand, *Server based vulnerabilities*: that are based on the monitoring traffic focused on the data generated by servers within the network. On the other hand, *Client based vulnerabilities*: that can be used to identify security flaws based on information from clients, for instance, by checking web browser user-agents, email agents, etc. On the other hand, vulnerabilities over communication devices can be discovered, for example, policies violation or misconfiguration of firewalls, routers, and so on, by checking how the communications between hosts are being performed. A complementary guide of vulnerability correlation with IDS is presented in [34].

There are some existing PNA based implementations such as *Real-Time Network Awareness (RNA)* by Sourcefire [74], *Network Vulnerability Observer (NeVO) (old)* [22] and Passive Vulnerability Scanner (PVS) [81] and *ArcSight ESM* [6]. Moreover, services such as Fox-IT ProtAct [29] are available on the market as well. All these technologies implement passive network monitoring techniques.

1.4 Problem Statement

There are different approaches aimed to get awareness about security context within organizations to different extents. On the one hand a variety of intrusion detection techniques can be used through existing IDS implementations such as Snort, Suricata and Bro. On the other hand, commercial products and services are available on the market to get information about security events that happen within the network. Such technologies provide different capabilities and imply different levels of complexity and flexibility, since they may combine features of NSM, SIEM and PNA. When an enterprise needs to get awareness of security events, it can choose among a set of solutions which indeed may comply with its requirements, however not necessarily in the most effective and suitable way. Furthermore, despite the fact that solutions may provide powerful analysis capabilities, they may involve additional tasks which may turn the whole process inconvenient or even unfeasible.

Since detection phase may involve automated, semi-automated and even manual processes, in which data is processed in different ways, then an extensive analysis is needed to propose a combined technique that could impact in a positive way the *ease*, *suitability* and *effectiveness* of Passive Network Audit models without the need to be implemented as a part of complex SIEM environments.

Short descriptions of these notions (NSM, SIEM and PNA) have shown that information retrieval of security context is not perfect. Hence, it is necessary to determine exactly what is the actual gap and what technologies and methods can be used to fill it. In this regard, taking into account the capabilities of existing detection techniques and aiming the definition of an improved *Passive Network Audit* framework, the research question of this work is:

> ***What is the actual gap between NSM, SIEM and PNA capabilities and how it can be filled in order to deploy a streamlined, flexible and effective PNA framework capable to get information about security context?***

To answer this, the following support questions have been defined, which will be addressed in this research:

1. *What are the most relevant and suitable characteristics to define an accurate detection framework?* In order to answer this question, some detection taxonomies and existing detection frameworks will be analyzed to describe useful and common characteristics that general detection and analysis framework should include.

2. *How can detection models be combined to define a Passive Network Audit Process?* To answer this question, an analysis of existing detection frameworks will be performed and findings will be taken into consideration in order to define specific tasks and features that apply on PNA context.

3. *What kind of detection techniques can be used taking into account network environment data?* This question will be answered by analyzing a set of tools that are able to process network traffic. A selection criteria will be defined such that specific-purpose tools are used in specific ways within the PNA framework in order to produce useful datasets that can be analyzed for PNA purposes.

4. *Are current widely deployed IDS technologies good enough to get awareness of security context?* In order to answer this question, this work will evaluate some detection approaches that rely on pure IDS alerts (misused-based), including some NSM implementation that use IDS as core detection engine. Thus, a comparison will be performed in order to identify differences between information from IDS and passive tools.

5. *How can anomaly detection algorithms be used in PNA in order to identify patterns accurately?* This question will be tackled by testing an anomaly detection algorithm based on data mining. The main purpose is actually to test not only whether *anomaly-based* methods can be applied for auditing processes, but also to find specific features that can provide criteria for selection of *anomaly-based* techniques in PNA context.

1.5 Motivation

In general terms, the need of an efficient and simple *Passive Network Audit* technology is motivated by the fact that this technique addresses different issues in security threats detection such as:

- There are no absolute guarantees of the correct compliance of the security controls within the network.
- Conventional detection technologies may not be enough to detect some threats such as zero-day attacks since they are unknown.
- Real-time monitoring and interpretation is very hard to achieve, thus security vulnerabilities might be identified after an incident.
- Some active detection technology such as penetration testing may imply disruptive impacts on production systems. In this sense, unsuitable security mechanism for both prevention and reaction may be placed over the network.
- Some existing technologies implement PNA features, however they are part of complex SIEM systems that may increase the complexity of the auditing process as well.

In spite of the existence of some commercial and open *Network Audit* technologies, it is intended to create the aforementioned framework with specific capabilities to gather information from network traffic, not only in an efficient way but also with less complexity and acceptable reliability of security assessment.

1.6 Outline

In this thesis, a *Passive Network Audit Framework*, based on an integration of different detection techniques, will be presented. It is intended to provide not only a feasible way to get an assessment of the general security status of the network, but also to address the issues of conventional deployments, such as complexity and long processes, by creating an automated functional model. In this section has been tackled a basic approach to concepts that will help to understand the motivation of this research. The remainder of chapters of this thesis are structured as follows. Chapter 2 presents an overview intrusion detection taxonomies used as reference of background context of PNA. Moreover, this chapter presents some existing frameworks that involve passive detection processes and passive technologies. These existing approaches are presented in order to show features that can be taken for the proposed PNA framework. Chapter 3 presents the PNA framework definition, including functional model and detailed modules features. Chapter 4 presents a practical implementation, its characteristics and a case study that evaluates the features of the framework. This Chapter also includes a discussion about suitability, accuracy, performance, limitations and issues identified during experiments. Finally, Chapter 5 presents final conclusions that clarify the findings of this work with regard to the research question.

Chapter 2

Network Audit and Intrusion Detection Taxonomy

2.1 Overview

The goal of this chapter is to address some intrusion detection taxonomies that describe common characteristics of intrusion detection models. This background will be useful to take some considerations about detection and analysis approaches for the proposed PNA framework. Thus, the workflow of this Chapter is as follows: first are presented intrusion detection taxonomies that define *common IDS characteristics*. Taking this into account, as well as some existing classifications that are be presented in Section 2.2, then a summarized *classification of intrusion detection models* are described. Finally, some examples of *existing intrusion detection frameworks* are presented in such a way that specific detection and analysis features can be considered, in regard with the intended PNA framework that is described in Chapter 3.

2.2 Taxonomy of Intrusion Detection Models

Despite the fact that Intrusion Detection Systems have evolved over the years, there are common model specifications such as *architecture*, *components* and *measurable properties* that state the general characteristics that IDS should meet. In addition, IDS classifications may be defined from different points of view (e.g. *functional*, *location*, etc.), splitting intrusion detection models according with the principles they follow and features that they provide. In order to describe a base point of common IDS properties, **three taxonomies** will be tackled in this section. Each taxonomy describes three key aspects about IDS specification: *architecture*, *components* and *measurable properties*.

The **first taxonomy** studied on this thesis is presented by Debar et al. [21]. It describes IDS **architecture** as an environment that includes a *detector* that processes information coming from the system that is to be protected. This *detector* uses three types of information: *Database* (long-term information: knowledge base of attacks); *configuration* (defines the current state of the system) and *Audits* (events description). The goal of this detector component is to filter the information from the audit trail in order to identify

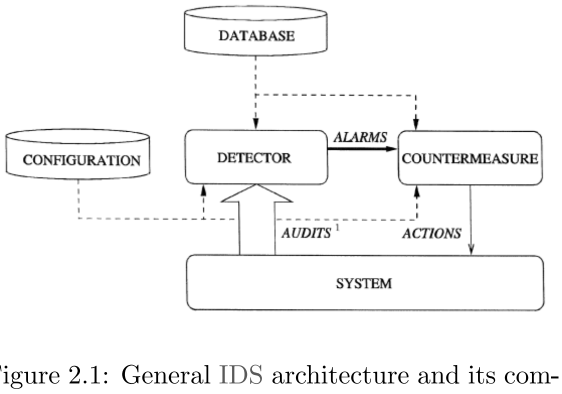


Figure 2.1: General IDS architecture and its components presented in [21]

security-related interpretations.

Moreover, in order to evaluate IDS, this taxonomy defines a set of **measurable properties**: *Accuracy, Performance, Completeness, Fault Tolerance and Timeliness.*

Furthermore, four main **components** are described: (1) *Detection method*: Describes the characteristics of the analyzer. (2) *Behaviour on detection*: Describes the response of the intrusion detection system. (3) *Audit source location*: Distinguishes the kind of input data. (4) *Usage frequency*: Describes the monitoring way of the analyzer. In addition, based on these components, IDS can be classified into eight sub-types grouped in two main sets: (a) *functional* and (b) *non-functional* characteristics. In Figure 2.1 is depicted the architecture of a simple IDS described in [21], whereas Figure 2.2 shows the resulting classification.

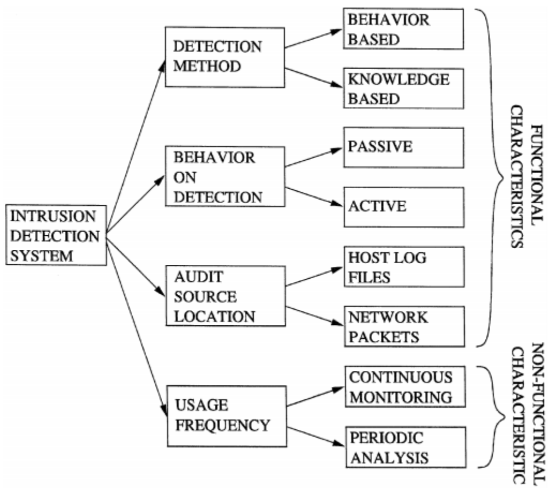


Figure 2.2: Characteristics of Intrusion detection systems presented in [21]

The key aspects of this approach that can be considered are: first, the modular design in which a system feeds the detector based on some configuration parameters. Since there is a *long-term* storage database, this taxonomy implies both NSM and SIEM capabilities. However, the countermeasure module can be considered just partially, since it may imply *active* actions as feedback from the IDS model, thus in the PNA context this feedback would be a different output data such as *recommendations*. Second, the measurable properties can be used to define the evaluation criteria for the PNA framework. Finally, the proposed classification provides functional and non-functional characteristics, so the summarized classification can take this into account.

The **second taxonomy** is presented by Catania et al. [14]. It describes the general **architecture** of IDS as a set of modules:

- *Traffic Data Acquisition* : Module for data collection.
- *Traffic Features Generator* : Traffic features extractor. It includes different layers: *low-level features* (IP header), *high-level features* (traffic information). Furthermore, additional feature layers can be defined according with the source of input data: *packet features* (raw packet headers), *payload features* (payload data)
- *Incident Detector* : Data processing module intended to identify intrusive activities.
- *Traffic Model Generator* : Uses data from *Incident Detector* as a reference in order to perform a comparison for incident identification purposes.

- *Response Management* : Module to execution actions in response of a possible intrusions.

Whereas the set of **measurable properties** is comprised by *Prediction accuracy* (detection quality), *Processing time* (event processing rate), *Adaptability* (detection of new attacks) and *Resource consumption* (memory and storage usage). Finally, this taxonomy describes the main **components**, which according with the authors is a summary of elements *"commonly accepted in the intrusion detection research community"*. These elements are:

- *Detection Method*: Determines the two main detection approaches: *anomaly-based* and *misuse-based*
- *Model acquisition*: The way knowledge is produced: *human-based* or *automatic generation*
- *Usage Frequency*: Detection execution: *real time* or *batch* (periodic analysis)
- *Architecture*.Data collection and processing perspective: *centralized* or *distributed*

In this taxonomy, again the modular architecture is described in order to perform the information flow through focused tasks. This can be mapped into PNA context in such a way that modules perform specific tasks over input dataset (i.e. network traffic decoding, pre-processing, post-processing, model generation, interpretation). Moreover, similar measurable properties (from the first taxonomy) are described, hence they are taken into account to consolidate the summarized list of properties for the framework. Finally, components are consistent as well, in the sense that similar concepts such as detection method and usage frequency are considered, thus a summarized classification is consolidated as well.

The **third taxonomy** is presented by Axelsson [9] and Patcha et al. [62]. It describes a generalized model of IDS. Figure 2.3 shows this approach in which there are diverse components within IDS architecture with similarities related to the aforementioned taxonomies. In this case, the elements have the following roles: *Audit data collection* (module used to collect data in order to be analyzed by detection algorithms), *Audit data storage* (module to store audit data for further reference), *Analysis and detection* (module where all the detection algorithms are implemented), *Configuration data* (defines operation parameters of the whole IDS), *Reference data* (stores parameters of operation of the detection module e.g signatures, thresholds, baseline datasets), *Active/processing data* (stores intermediate results about the detection dataset), *Alarm* (module to report and handle the identified suspicious events).

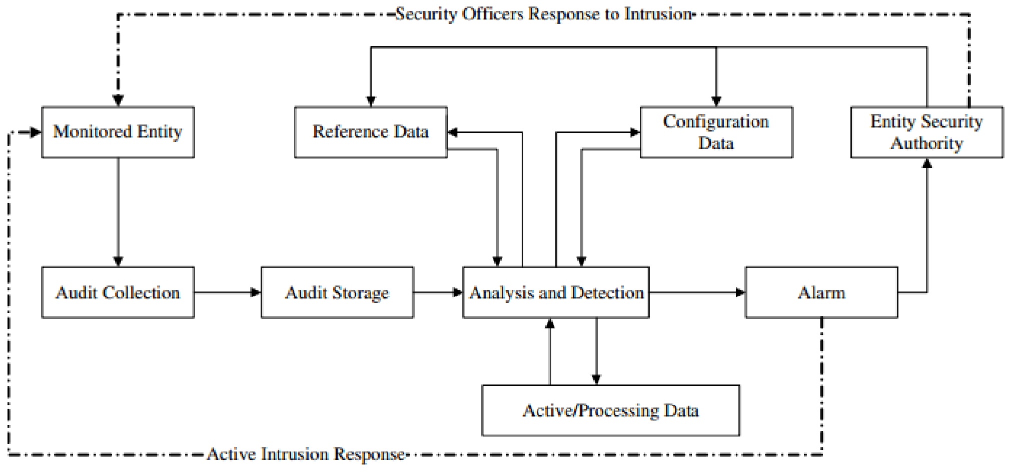


Figure 2.3: Organization of a generalized Intrusion detection systems presented in [9] and [62].

In summary, these approaches define similar components which represent features of both NSM and SIEM. First, the whole system depends on predefined configuration and reference modules, which are shared features mentioned in all presented taxonomies. Moreover, the core detection engine is represented by different modules that manage audit data which is analyzed to trigger alarms and produce feedback. Taking all these aspects into account, then a summary of properties to consider into the PNA framework specification is described as follows in Table 2.1.

Architecture	Measurable properties	Components
Modular design, storage and configuration modules	Accuracy, Performance, Timeliness	Detection Method, Usage Frequency, Behaviour of detection, Audit source location

Table 2.1: Summary of taxonomy definitions to consider into PNA framework

2.3 Classification of Detection Models

The aforementioned taxonomies have described the common features of IDS as a generalized intrusion models. The purpose of this section is to describe a general picture and background of intrusion detection techniques, some of them intended to be used within the framework as core detection engine.

Intrusion detection models, with respect to the way the anomalous activities are detected, are commonly grouped into two main detection approaches: *Misuse-based* and *Anomaly-based.* Similar nomenclatures refer such methods as *Knowledge-based* or *detection by appearance* [77] (misuse) and *Behaviour-based* (anomaly). Furthermore, *misuse-based* can be considered as *black listing* since it defines what is malicious or anomalous in advance, whereas *anomaly-based* can be considered as *white-list*, since it defines the accepted or normal behaviour.

According with Catania et al. [14], in the context of *misuse-based* models, detection itself relies on three different techniques *Pattern recognition*, *Implication rule-based* and *Data mining.* It is which summarized in Table 2.2. In regard with *anomaly-based* models, Arca-Teodoro et al. [30] present a classification of *anomaly-based* detection models. Thus, a summarized classification is described as follows in Figure 2.4

2.3.1 Misuse-based detection

This detection method applies the previous knowledge that has been accumulated about specific events in such a way that information about attacks, vulnerabilities, and threats in general can be retrieved and applied to describe a detection base.

Pattern recognition : It creates the so-called *signatures* generated from a base knowledge by looking for specific patterns that match from incoming network packets or command sequences. On the one hand, the advantage of such a model is the fact that since the signatures contain very specific matching patterns, then detection for well known attacks is achieved in a reliable and accurate way. On the other hand, since the detection is limited to the detection parameters defined within the signature, then a drawback is represented by the fact that signatures for all known attacks need to be contained within the detection engine.

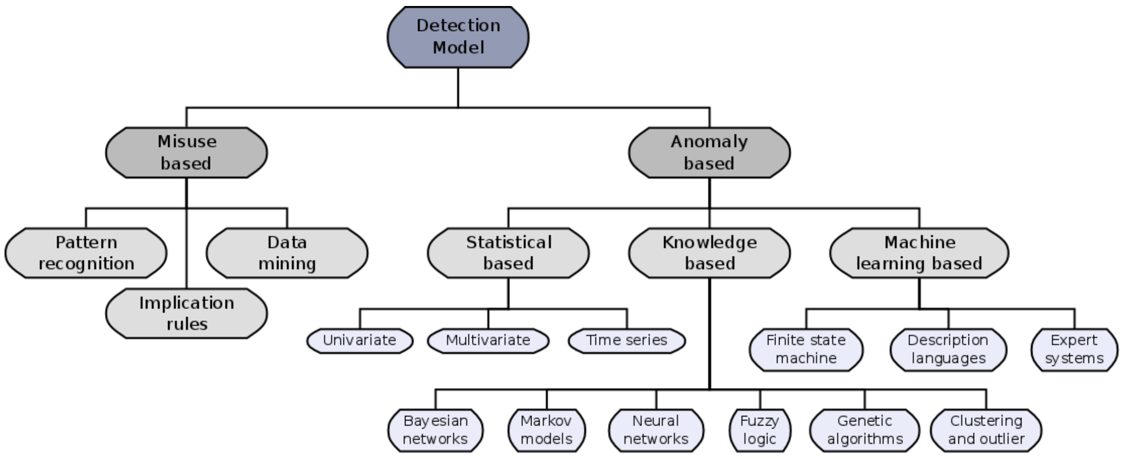


Figure 2.4: Classification of detection models in IDS.

Furthermore, signatures need to be updated and aggregation of new signatures for new attacks is needed as well, otherwise new attacks will not be detected. It has to be emphasized that although signatures describe very specific attacks, the granularity can be changed in order to generalize the matching pattern for attacks that share some detection patterns. However, this generalization process implies the triggering of *false positives*, meaning that events that actually did not happen are reported by the IDS. Listing 2.1 shows an example of IDS signature described by a Snort rule that allows to detect successful exploitation of a critical OpenSSL vulnerability found in April 2014, CVE-2014-0160 [1]. Most of the NSM infrastructures are based on *signature-based* detection model since they are usually easier to deploy and manage. In fact, two of the most widely used IDS, Snort and Suricata, implement this type of detection technique. Since these tools are intended to be part of PNA framework, then it is important to take into account the aforementioned features, in order to define the scope of the framework. Further details, of how these tools are used within the framework, are described in Chapter 3.

```
alert tcp any any -> any any (msg:"FOX-SRT - Flowbit - TLS-SSL Client Hello";
    flow:established; dsize:< 500; content:"|16 03|"; depth:2; byte_test:1, <=, 2,
    3; byte_test:1, !=, 2, 1; content:"|01|"; offset:5; depth:1; content:"|03|";
    offset:9; byte_test:1, <=, 3, 10; byte_test:1, !=, 2, 9; content:"|00 0f 00|";
    flowbits:set,foxsslsession; flowbits:noalert; threshold:type limit, track
    by_src, count 1, seconds 60; reference:cve,2014-0160; classtype:bad-unknown;
    sid: 21001130; rev:9;)
```

Listing 2.1: Example of IDS signature (Snort/Suricata rule published by Fox-IT)

Implication rule-based : This technique defines *event-rules* that describe a chain of network events, so that detection engine may infer an intrusion when a series of network events matches with such description. Emerald [65] is an example of approach using implication rules. This technique is used by Bro IDS (also part of PNA framework), and it is a feature that can complement signature-based detection in order to extend the context of security events.

Data mining : This technique aims to eliminate the need of manually created *traffic models* by automatically building them based in on some references such as protocol and event specification. Hybrid approach uses *data mining* to extract meaningful information about security events. It

[1]Critical security vulnerability reported on April, 8th 2014. `http://heartbleed.com/`. Fox-IT published a set of Snort/Suricata rules to detect this attack. `http://blog.fox-it.com/2014/04/08/openssl-heartbleed-bug-live-blog/`

has to be emphasized that, although data mining is typically used in *anomaly-based* detection, according with Catania et al. [14], it can be applied on *misuse-based* as well. Approaches based on as *Artificial Neural Networks* (ANN) [13], [48] for feature extraction, and *Genetic Algorithms* (GA) [1] for model structure designing. Both examples use data mining to analyze, classify and extract meaningful information related to security incidents.

Table 2.2 shows a comparative analysis of some existing approaches of *misuse-based* intrusion detection based on taxonomy presented in [14]. In order to provide an overview of the scope of such approaches, the following features are described: detection technique, usage frequency, model acquisition and type of analysis. In the list, it has to be noted that the work presented by Roesch [69] is the practical implementation of Snort IDS, which is also intended to be part of the core detection engine within PNA framework and it will provide data aggregation of IDS alerts. The characteristics of the signatures used for pattern identification are described in Chapter 3.

Authors	Misuse detection technique	Usage Frequency	Model acquisition / adaptability	Scope of analysis
Lindqvist and Porras [47]	Implication rules	Real-time	Human: rules written by experts	Low and high-level: packet, payload, flow
Roesch [69]	Pattern Signature	Real-time	Human: signature patterns written by experts	Low: packet, payload
Cannady [13]	Data mining:ANNs	Batch	Automatic: retraining with new attack patterns	Low and high-level: packet
Li [46]	Data mining:GA	Batch	Automatic: retraining with new attack patterns	Low and high-level: packet, flow
Gomez et al. [31]	Data mining: fuzzy / GA	Batch	Automatic: retraining with new attack patterns	Low and high-level: packet, flow, payload

Table 2.2: Comparative analysis of misuse-based intrusion detection approaches described in [14].

2.3.2 Anomaly-based detection

Anomaly detection methods are based on the analysis of the *normal traffic behaviour* in such a way that baseline profiles, described by *legitimate* network traffic activities, are defined in order to identify potential *anomalous* activity. The main feature of this type of detection model is that they provide the capability to identify unforeseen attacks, such that, unlike misuse-based detection, there is no the need of specifying detection parameters about attacks themselves. It has to be noted that the central premise of anomaly detection is that *intrusive activity is a subset of anomalous activity* [40]. The ideal case considers all anomalous activities as intrusive activities, however this is not always the case. In fact there are be four possibilities: (1) *Intrusive but not anomalous*, (2) *Not intrusive but anomalous*, (3) *Not intrusive and*

Although there are different anomaly detection approaches, a generic functional model is described in Figure 2.5. It includes the following main components: (a) *Parameterization*: Observed instances are represented in a pre-stablished form. (b) *Training*: the normal or abnormal behaviour is characterized and the corresponding model is generated. (c) *Detection*: the generated model is compared with the observed traffic such that if certain deviation exceeds the threshold, then an alert is triggered.

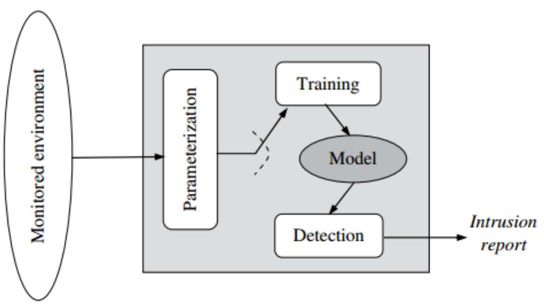


Figure 2.5: Functional model of generic anomaly based IDS described in [30].

It is important to emphasize the fact that, *data mining* is used in all the techniques under *anomaly-based* category presented as part of the classification of detection models (Figure 2.4). In this category, *anomaly-based* detection techniques are grouped into three main subcategories: *statistical*, *knowledge based* and *Machine learning*. The following subsections describe a background of such subcategories in order to understand what type of techniques can be used in the context of PNA. In fact, not all of them are intended to be used within the proposed PNA framework, nevertheless, by taking into consideration the modular design mentioned in Section 2.2, it is possible to deploy different *anomaly-based* detection approaches as part of the framework. The type of dataset that can be used for anomaly detection purposes are described in Chapter 3.

Statistical-based

Garcia-Teodoro et al. [30] mention that the idea of statistical-based detection model is to generate profiles which describe the stochastic behaviour of the network traffic. The profile includes properties such as activity intensity measures, audit record distribution measures, categorical measures and ordinal measures. Typically two profiles are involved playing different roles: the *current* profile which correspond to the network traffic observed over the time, and the *stored* profile that corresponds to the previously stored and trained statistical profile. These profiles are compared using a function of abnormality of all measures described within the profile. This comparison results into a score that indeed indicates the degree of irregularity of the event. Here, a predefined threshold is set, thus, an alert is generated in case of the score exceeds the threshold. The main advantage of statistical-based technique is that prior knowledge about the normal behaviour of network activity is not required. The drawback is given by the fact that this approach is susceptible to be trained by attackers, moreover, the process of parameters and metric definition might be cumbersome.

Knowledge-based

According with Garcia-Teodoro et al. [30], this approach applies the knowledge that has been accumulated about specific attacks. The so-called *Expert systems* classify audit data involving three phases and considering a set of rules. First, attributes and classes are identified from training data, then classification rules are generated to finally classify audit data based on such rules. This provides a systematic way to find evidence of vulnerabilities exploitation attempts within the audit trail. A similar approach uses the concept of *specifications* to define a set of rules that describe the normal behaviour. This type of specification can be generated using *Finite State Machine* (FSM) methodology that can be implemented by using standard description languages such as N-grammar, LOTOS and UML. On the one hand, potential advantages of this technique is that it provides high level of flexibility. On the other hand, the implementation implies cumbersome processes and time consuming.

Machine learning-based

In general terms, Machine Learning (ML) is a research field in which the purpose is to develop systems that can improve automatically by themselves with the experience they have gained from past events, meaning an iterative learning process. It tries to address issues that statistics and data mining try to solve. As a background [55], [2] about ML concepts, a very general description of different types of learning methods is presented below:

Supervised learning: It is based on examples as training dataset in order to find a shared descriptions between positive examples which are not in negative examples. This allows to identify unforeseen instances. The components involved are the inferred function (classifier) and an instance (vector). Furthermore two tasks can be performed. *Classification*: the classifier tries to determine the class or label that an input vector belongs to. *Regression*: task intended to predict an inferred function from training dataset (e.g. predict the network protocol given a set of network packet header attributes or traffic parameters).

Unsupervised learning: Unlike supervised learning, unsupervised methods do not rely on training datasets, but they just take the input data in order to describe regularities.

Reinforcement learning: If the output of a system is comprised by a set of actions, then a single action itself is not relevant, but what matters is the way such an output sequence is formed, (i.e. policy that defines a sequence of correct actions to reach a goal). Thus, the goal of this method is to assess whether the policy is correct and to learn from past actions in order to generate a new policy.

ML is a strong research are within Information Security field, specifically when it is applied for intrusion detection processes. Features of ML coincide in some cases with statistical approaches, however ML not only is intended to build and describe a model, but also to improve the performance or execution strategy based on newly acquired information or previous results. Based on the classification presented on Figure 2.4, the subcategories of ML techniques used in anomaly-based detection models are describes below:

Bayesian Networks: Bayesian networks represent *models that encode probabilistic relationships among variables of interest* [30] and they are used to represent knowledge about uncertain domain. Bayesian networks are in fact probabilistic graphical models that contains *nodes* (random variable) and *edges* (probabilistic dependencies among certain random variables). For intrusion detection purposes, Bayesian networks are combined with statistical methods in order to develop detection schemes capable to predict events as consequence of actions based on prior knowledge and data. Moreover, Bayesian techniques are used for data classification and false alarm suppression. The accuracy of this method depends on assumptions based on the behavioral model of the target system, thus any deviation from these assumptions leads to inaccurate detection. This means that the key factor is to build an accurate behavioural model, which in fact is not an easy task since it depends on the complexity of the network as well.

Markov Models:This category comprises two main techniques: On the one hand is *Markov chains*: a set of states that are interconnected through transition probabilities which define the capabilities of the model and its topology. The training phase includes an estimation of probabilities associated to the transitions, based on the normal behaviour of the target system. On the other hand, *Hidden Markov*: a statistical model in which the system being modeled is assumed to be Markov process with unknown parameters. Such hidden parameters needs to be determined from observable parameters. This detection approach usually has been applied to Host IDS, in which system calls are analyzed, however it can be applied to network traffic as well.

Neural Networks: This method, applied to network intrusion detection, has been usually applied in order to predict the behaviour of the user, daemon, processes and so on within the system based on network traffic patterns. Due to their flexibility and adaptability to changes within the systems environment, neural network have been studied within the anomaly detection research area. The main advantage is that they can deal with imprecise data and uncertain information without the need of prior data describing the regularities of the system. However, there is a set of drawback with this approach, such as they may fail due to the lack of data to analyze and they may be slow and hard to train as well.

Fuzzy Logic: These techniques are based on fuzzy set theory and they have been used in computer security area since the early 90's. In fuzzy set theory, reasoning is not precise, but rather an approximation is deducted from classical predicate logic. Fuzzy logic can be applied for intrusion detection processes because it is possible to consider diverse measurable parameters as CPU usage, connection intervals, and so on, in such away they can be described as *fuzzy variables*. This is in fact a very effective detection method, specially for recognition attacks such as port scans, probing, etc. Its main disadvantage is that the rule creation is a very intensive task.

Genetic Algorithms: Genetic algorithms are described as global search heuristics. It is a technique used to find approximate solutions for optimization and problem searching. They are a particular class of *evolutionary algorithms*. Since genetic algorithms are derived from *evolutionary biology* such as inheritance, mutation, selection and recombination, then it is consider a ML-based technique. In the field of intrusion detection, genetic algorithms are used in order to distinguish normal traffic from anomalous connections. The main goal is to provide the capability of deriving classification rules and select the appropriate parameters for detection processes. The main advantage in IDS is that it is a flexible search method that uses uses probabilistic methods.

Clustering and outlier detection: Clustering techniques allow to find patterns in unlabeled data with many dimensions, by grouping the observed data into clusters based on the similarity or distance measure. By using clustering techniques it is possible to learn from audit data (e.g. network traffic), to eventually identify intrusions based on the generated knowledge without the need of explicit description provided in advance. There are different approaches to apply clustering into intrusion detection models. One of them uses two datasets, normal and anomalous. The other approaches uses only normal dataset, performing a training process in order to characterize a model of normal behaviour. Some of the most common used distance metrics on clustering methods are *Euclidean distance* and *Mahalanobis distance*. In this kind of methods, IDS tuning process is considerably since the clustering process identifies intrusion events by itself based on raw audit data.

Table 2.3 shows a summary of some existing *anomaly-based* IDS including their main features and related work. The anomaly detection engine that is deployed within the framework, uses the basics concepts addressed in the work presented by Lee and Stolfo [43], covering packet and payload analysis. It uses data mining through frequent pattern analysis. In-depth explanation will be tackled in Chapter 3. Moreover, as an additional reference, Garca-Teodoro et al. [30] presents a summary of existing IDS that provide both misuse and anomaly detection capabilities, as well as hybrid features.

Authors	Anomaly detection technique	Usage Frequency	Model acquisition / adaptability	Scope of analysis
Porras and Valdes [66]	Statistical: chi-square-like	Real-time	Automatic: readjust model with new attack-free patterns	Low and high-level: packet, flow, payload
Staniford et.al [79]	Statistical: Naive bayes networks	Real-time	Automatic: readjust model with new attack-free patterns	Low-level: packet, flow
Lakhina et al. [41]	Statistical: PCA	Real-time	Automatic: readjust model with new attack-free patterns	Low-level: flow
Mahoney and Chan [51]	Machine Learning: rules learning and Markov models	Batch	Automatic: retraining with new attack-free patterns	Low and high-level: packet, flow, payload
Lee and Stolfo [43]	Data mining: rules learning and frequent pattern count	Batch	Automatic: retraining with new attack-free patterns	Low and high-level: packet, flow
Portonoy [67]	Data mining: unsupervised clustering	Batch	Automatic: retraining with patterns containing a reduced amount of attacks	Low and high-level: packet, flow, payload

Table 2.3: Comparative analysis of anomaly-based intrusion detection approaches described in [14]

2.4 Intrusion Detection Frameworks useful for PNA

At this point, both *IDS general taxonomy* as well as *detection model classification* have been defined in the context of this thesis. This section describes six different intrusion detection frameworks in order to provide a background about some existing approaches that implement different data analysis techniques with features related to PNA. The purpose of presenting these frameworks is not only to present a panorama of existing framework references, but also to extract some useful features and reinforce considerations about the scope and capabilities that the proposed PNA framework is intended to provide.

2.4.1 Common Intrusion Detection Framework (CIDF)

There are different approaches of intrusion detection frameworks. One of the formal references is the so-called Common Intrusion Detection Framework (CIDF) , *"an effort to develop protocols and application programming interfaces so that intrusion detection research projects can share information and resources and so that intrusion detection components can be reused in other systems"* [15]. As part of the CIDF research, the Intrusion Detection working Group (IDWG) [38] has defined a framework that includes (1) *Requirements documents*, which describes high-level functional requirements for communications between intrusion detection systems and management systems; (2) Common Intrusion Specification Language (CISL) which describes the data formats that satisfy requirements; and (3) *Framework Document* which identifies protocols best used for communication between intrusion detection systems. As a result, the Intrusion Detection eXchange Protocol (IDXP) [37] and Intrusion Detection Message Exchange Format (IDMEF) [16] have been defined.

Porras et al. [39] describes the main approach of CIDF in which the core definition of *Interoperating Intrusion Detection and Response (IDR) Components* is described as: *"Two intrusion detection and response systems are interoperating when they exchange data automatically, and as a result achieve some goal which neither could have achieved alone"*. Such a definition is very important to emphasize the core idea of the proposed PNA framework, where different IDS will be working together exchanging and correlating information. CIDF defines conditions for interoperability: (a) *Configuration interoperability*: condition in which two systems can find each other and successfully send data back and forth. (b) *Syntactic interoperability* implies that both systems can parse the syntax of the exchanged data correctly. (c) *Intercomprehension* implies that both systems agree on the data as well as its syntax.

Moreover, a set of scenarios where interoperation between IDR component is involved:(a) *Analyzing*: A gathers raw data and may analyze a small amount of data at a time. Then B takes a larger or longer view, analyzing reports from A, producing a single report. (b) *Complementing*: Components A_1 and A_2 complement each others' coverage, while a third component M merges their input. (c) *Reinforcing*: components A_1 and A_2 may reinforce each other's findings in order to reduce false positives. Then J judges their output accepting only when both A_1 and A_2 agree. (d) *Verifying*: A_1 detects and attack and reports it to a boundary controller J. This controller then asks its resident detector A_2 whether it also see the attack, if so then J considers the report verified. (e) *Adjust monitoring*: A adjusts monitoring depending on the kind of warnings it receives. Then it sends instructions to E about the monitoring target such that A receives information from E, analyzes it and passes it on. (e) *Responding*: A sends a prescription about certain attack to R in the meantime between the human response and the system response. Figure 2.6 shows the aforementioned scenarios.

Furthermore, CIDF defines a formal language specification focused on the following features: (1) *Expressive*: Components should be able to express a wide range of intrusion related prescriptions. It should be able to express causal relationships among events, the roles of objects in events, properties of objects, relationships ob objects, response prescription and contingent prescriptions. (2) *Unique in expression*: Components should be able to express a given sentiment in one or small

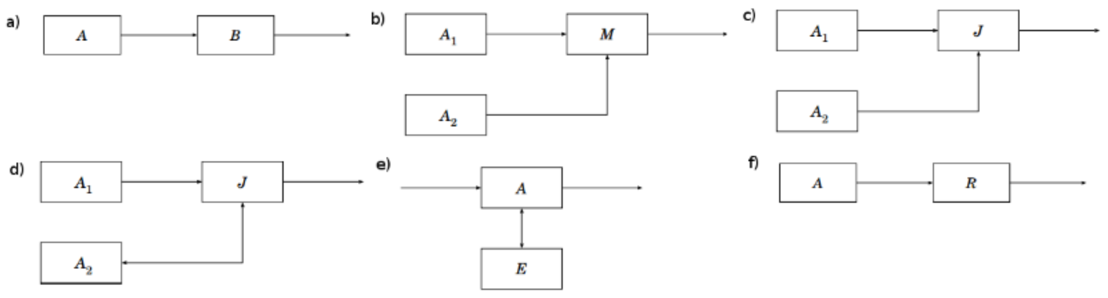


Figure 2.6: Interoperating IDR scenarios described [15].

number of *natural* expressions. (3) *Precise*: Two receivers reading the same message must not draw mutually contradictory conclusions from it. (4) *Layered*: There should be a mechanism to express specific concepts in terms of more general ones. (5) *Self-defining*: Consumers that receive a report should be able to interpret messages to the degree they need, without the recourse to out-of-band negotiation. (6) *Efficient*: Messages should consume as little of the system resources as possible. (7) *Extensible*: It should be able to express additional information, not defined previously, such that it remains the compatibility with other CIDF components. (8) *Simple*: Producers should be able to encode information quickly, whereas consumers should be able to extract the information in an easy way without extensive processing tasks. (9) *Portable*: Language should support a variety of platforms and transport mechanism. (10) *Easy to implement*: This is a crucial practical requirement.

The aforementioned features are integrated as part of the CISL [36]. Data exchanged among CIDF components is described in terms of the so-called *generalized intrusion detection objects (gidos)* which encode an occurrence of a particular event that occurred at specific time, a conclusion about a set of events, or an intrusion to carry out in action. Figure 2.8 shows (a) an example of CISL message and (b) an example with its corresponding encoding based.

According with Garcia-Teodoro et al. [30], IDWG defines the general IDS architecture based on four type of functional models described in Figure 2.7.

1. *E-blocks (Event-boxes)*: It is comprised of sensor elements that monitor the target system in order to gather information about events to be analyzed by other blocks. Depending on the information source considered, it might be either *host-based* (system calls, processes, etc.) or *network-based* (traffic volume, IP addresses, protocol, ports, etc.)

2. *D-blocks (Database-boxes)*: These elements store information from E blocks for subsequent processing by A and R blocks.

3. *A-blocks (Analysis boxes)*: Processing modules for analyzing events and detecting potential hostile behaviour. It might trigger alarms if necessary. Depending on the type of analysis, A boxes can be either *misuse-based* or *anomaly-based.*

4. *R-blocks (Response-boxes)*: Executes a response procedures if any intrusion occurs and the corresponding action is previously defined. Ning et. al. [58] propose an extension to CISL that allows IDR components to specify requests for particular information from other IDR components.

The important idea of CIDF that can be taken into consideration in the PNA context is that, a language can be used to define *events*, either as intrusion detection parameter or as audit policy parameter. In practice, such a language can be encoded as well to be managed easily from the coding point of view (practical implementation). Moreover, since both NSM and SIEM involve tasks such as storing, indexing, alerting, etc., such a language can define specific tags such as

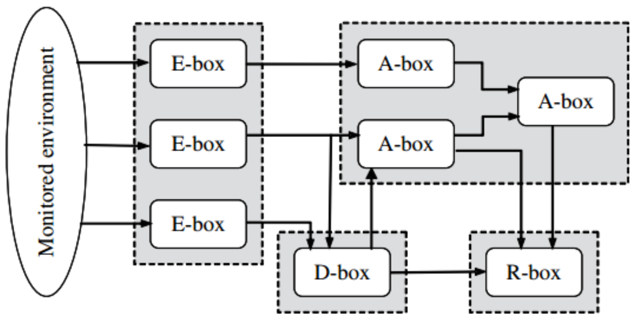
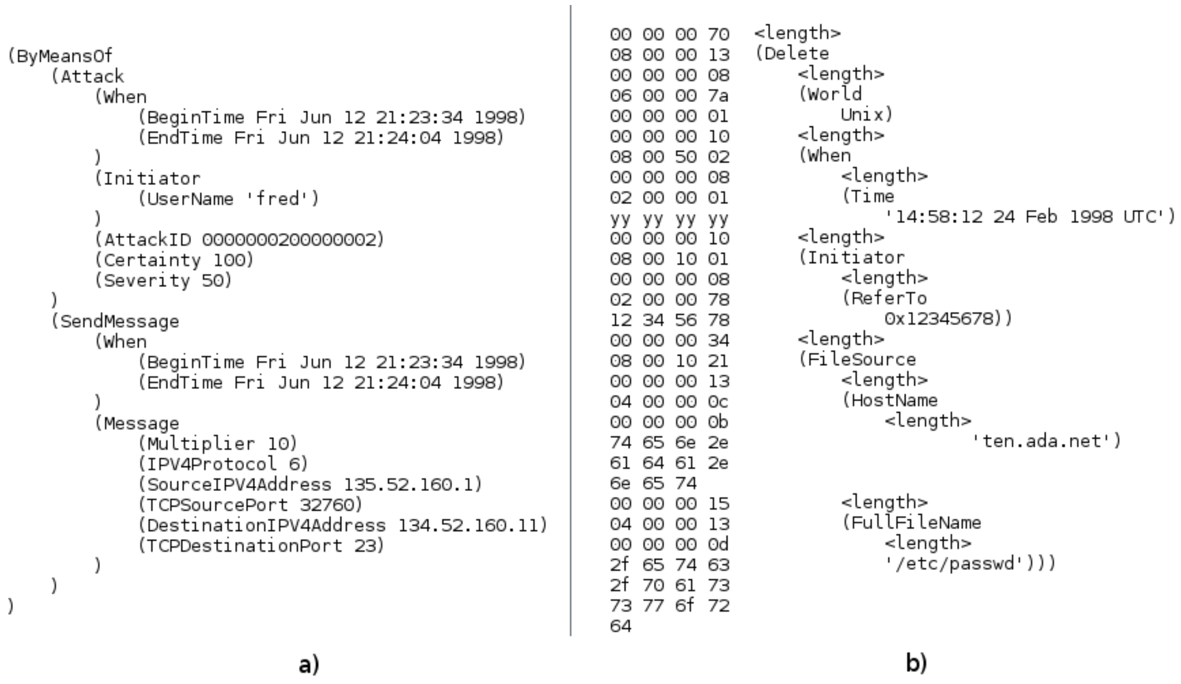


Figure 2.7: General CIDF architecture for IDS described [15]



```
(ByMeansOf
    (Attack
        (When
            (BeginTime Fri Jun 12 21:23:34 1998)
            (EndTime Fri Jun 12 21:24:04 1998)
        )
        (Initiator
            (UserName 'fred')
        )
        (AttackID 0000000200000002)
        (Certainty 100)
        (Severity 50)
    )
    (SendMessage
        (When
            (BeginTime Fri Jun 12 21:23:34 1998)
            (EndTime Fri Jun 12 21:24:04 1998)
        )
        (Message
            (Multiplier 10)
            (IPV4Protocol 6)
            (SourceIPV4Address 135.52.160.1)
            (TCPSourcePort 32760)
            (DestinationIPV4Address 134.52.160.11)
            (TCPDestinationPort 23)
        )
    )
)
```

a)

```
00 00 00 70   <length>
08 00 00 13   (Delete
00 00 00 08       <length>
06 00 00 7a       (World
00 00 00 01           Unix)
00 00 00 10       <length>
08 00 50 02       (When
00 00 00 08           <length>
02 00 00 01           (Time
yy yy yy yy               '14:58:12 24 Feb 1998 UTC')
00 00 00 10       <length>
08 00 10 01       (Initiator
00 00 00 08           <length>
02 00 00 78           (ReferTo
12 34 56 78               0x12345678))
00 00 00 34       <length>
08 00 10 21       (FileSource
00 00 00 13           <length>
04 00 00 0c           (HostName
00 00 00 0b               <length>
74 65 6e 2e                       'ten.ada.net')
61 64 61 2e
6e 65 74
00 00 00 15           <length>
04 00 00 13           (FullFileName
00 00 00 0d               <length>
2f 65 74 63               '/etc/passwd')))
2f 70 61 73
73 77 6f 72
64
```

b)

Figure 2.8: Example of CIDF messages and the corresponding encoding

timestamp, index, even type, initiators, etc, in order to provide an easy way to deploy all the intended features.

2.4.2 Data mining framework

Lee et al. [45], [44] propose a framework to develop intrusion detection models based on data mining. They state that *"a basic premise for intrusion detection is that when audit mechanisms are enabled to record system events, distinct evidence of legitimate activities and intrusions will be manifested in the audit data"*. Thus, due to the amount of data and the features of the environment, an intelligent data analysis tools are required. The framework then is focused on learning classifiers and meta-classification, association rules for link analysis, frequent episodes for sequence analysis and a support environment to evaluate detection models. Hence, the three main aspects are:

1. *Classification*: Maps a data item into one of several predefined categories. The related algorithms usually create *classifiers* as a form of decision trees of rules. Given a set of

records, classification algorithms can compute a model that uses the most discriminating feature values to describe each concept. The accuracy depends on the amount of features provides as training dataset.

2. *Link analysis*: The goal is to determine multifeature (attribute) correlation from a database table. So, given a set of items, $support(X)$ is defined as the percentage of records that contain item set X. Association rules is defined by the expression $X \to Y, [c, s]$ where both X, Y are items sets and $X \cap Y = 0, s = support(X \cup Y)$ is the support of the rule, and $c = \frac{support(X \cup Y}{support(X)}$ is the confidence.

3. *Sequence analysis*: Models sequential patterns so that time-based sequences of audit events that occurs frequently can be identified. Given a set of time stamped event records, where each record is a set of items, an interval $[t_1, t_2]$ is the sequence of event records that starts from timestamp t_1 and ends at t_2. The width of the interval is defined as $t_2 - t_1$. $support(X)$ is the radio between of minimum occurrences that contain X and the total number of event records. A frequent episode rule is the expression $X \to Y, [c, s, w]$ where X, Y, Z are items sets and they together define an episode $s = support(X \cup Y \cup Z)$ and $s = \frac{support(X \cup Y \cup Z}{support(X \cup Y)}$ is the confidence. Figure 2.10 shows an example of such a specification: in (a) an example of association rule and (b) an example of frequent episode. Figure 2.9 shows the data mining process (described in [42]) applied to build IDS models.

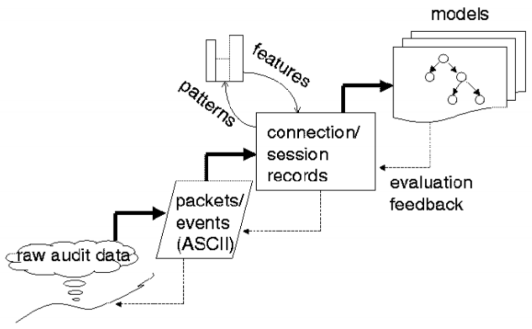
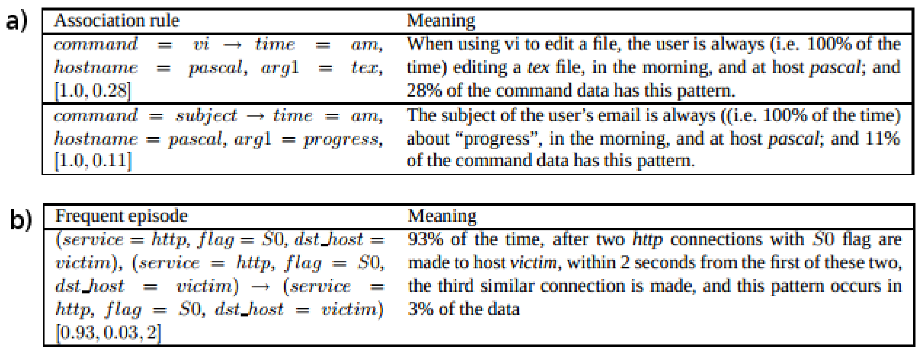


Figure 2.9: Data mining process of building IDS models described in [44].



a)

Association rule	Meaning
$command = vi \to time = am, hostname = pascal, arg1 = tex,$ [1.0, 0.28]	When using vi to edit a file, the user is always (i.e. 100% of the time) editing a *tex* file, in the morning, and at host *pascal*; and 28% of the command data has this pattern.
$command = subject \to time = am, hostname = pascal, arg1 = progress,$ [1.0, 0.11]	The subject of the user's email is always ((i.e. 100% of the time) about "progress", in the morning, and at host *pascal*; and 11% of the command data has this pattern.

b)

Frequent episode	Meaning
$(service = http, flag = S0, dst\_host = victim), (service = http, flag = S0, dst\_host = victim) \to (service = http, flag = S0, dst\_host = victim)$ [0.93, 0.03, 2]	93% of the time, after two *http* connections with *S0* flag are made to host *victim*, within 2 seconds from the first of these two, the third similar connection is made, and this pattern occurs in 3% of the data

Figure 2.10: Example of association rule and frequent episode

In a similar way than CIDF, this data mining framework defines a language specification to associate events and identify frequent episodes. Again, since all NSM, SIEM and PNA define parameters such as timestamp, service, hostnames, etc. that can be used for data correlation, then by using such a specification language, event identification can be performed in a suitable way. In this approach, parameters related to frequency measurement are expressed as a vector, however, since the key idea to consider is to add parameters of measurement, the format can be expressed using tags or any other way that links attributes with their frequency. In the case of PNA framework, a language specification is intended to be defined in such a way that this type of tags can be used to identify frequent episodes. Details about it are addressed in Chapter 3.

2.4.3 Event Calculus-based framework

Rouached et al. [71] present a framework based in Event Calculus (EC) that formally analyze the process of detecting network intrusions. EC is a language for representing and reasoning about dynamic systems whose ontology is based on (a) a set of *time-points* isomorphic to the non-negative integers, (b) a set of *time-varying* properties called fluents and (c) a set of *event types.* There are some approaches [50], [75], [3] where EC is used to specify security policies and requirements, as well as to map policies and system behaviour into formal notation.

This approach addresses the challenges for building IDS such as reliability on attacks detection, ability to analyze large amount of data and the ability to correlate alerts with the actual security incidents. This framework is based on four main tasks: (1) Design and development of an underlying scalable and adaptive parallel and distributed IDS architecture for high speed networks. (2) Design and development of algorithms and techniques to improve the accuracy of Network Intrusion Detection Systems (NIDS) alerts generation and correlation. (3) Design and development of an efficient and integrated management platform. (4) Testing and performance study of the system on large scenarios.

Moreover, this framework has four main research pillars: (1) *Modeling the brain* to improve accuracy and prioritize attack alerts. (2) *Making the mind* to add intelligence to the system by reasoning on alert logs. (3) *Architectural theme* to design distributed architectures, traffic load and balancing algorithms. (4) *Management theme* to manage the overall defense process in an efficient way. The functional model is comprised of the following elements:

- *Security requirements*: Define properties that network environment should fulfill.
- *Assumptions*: Define additional properties to facilitate verification process (e.g. security properties with respect to security policies).
- *Network specification*: Describes an abstract model for security critical entities within the network.
- *IDS rules*: Set of intrusion detection misuse-based signatures (Snort based rules).
- *Event logs*: Set of audit trails from operating systems, application and network components.
- *Verification engine*: Set of verification to test and fix design errors and verify whether the process design does have certain desired properties.

Security requirements, assumptions, IDS rules and system specification can be described using EC predicates and axioms. For instance, Listing 2.2 shows the mapping of a Snort rule into EC-based specification. This snort rule triggers an alert when a Transmission Control Protocol (TCP) packet with *Acknowledge* flag is set containing a value of 0 and the destination is a host within the network 192.168.1.0/24. With this regard, the PNA framework involves a mapping process as well, in which a set of *high-level rules* are used to define how IDS rules, in lower analysis layer, are interpreted to identify a security event. The full specification of such a high-level rules is presented in Chapter 3 and in Appendix.

```
alert tcp any any -> 192:168:1:0/24 any (flags : A; ack : 0;msg : "TCP ping
    detected")

Happens(receive(TCP,A,ACK: 0; 192:168:1:0/24), t) -> HoldsAt(e(Alert;,msg : TCP
    ping detected ), t)
```

Listing 2.2: Mapping of Snort rule into EC-based specification

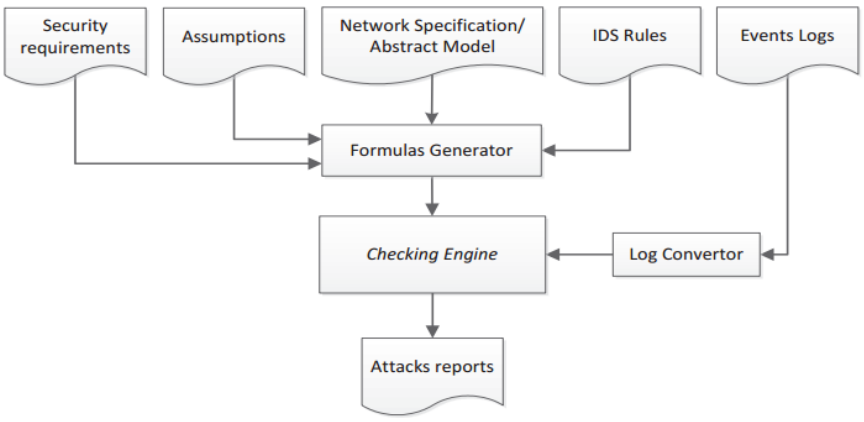


Figure 2.11: EC framework functional model proposed in [71]

As it can be appreciated in Figure 2.11, the functional model of EC presents shared features with the PNA context. First, data aggregation is present in such a way that a central point of analysis engine is used to generate the context of security events (attacks). It is done based on data aggregation of five sources: security requirements, assumptions, network specification, IDS rules and event logs. Second, there exists a pre-processing task for event logs, which supposes data interpretation and correlation tasks. Finally, control parameters such as security requirements and assumptions are used in order to define the scope of the checking engine. Hence, this approach reinforces the use of the following concepts: *control parameters, data aggregation, pre-processing task*.

2.4.4 Passive testing framework based on security rules specification

Mallouli et al. [52] propose a passive testing approach that provides the ability to check whether a system respects its security policy. This framework is focused on system events analysis, nevertheless, a mapping into network events is possible by generalizing the concept of *event*. This framework relies on a *Nomad Formal Language* [19], which is based on deontic and temporal logics, such that by collecting execution traces it is possible to deduce automatically a verdict about the compliance of security policies within the system. To achieve this goal, three main steps can be described below, and the functional architecture is depicted in Figure 2.12.

1. *Definition of passive architecture*: execution traces collection.

2. *Description of the system security policy using a formal specification language*: description of security rules in Nomad language.

3. *Security analysis*: A passive tester deduce a global verdict with three possible options: *PASS, FAIL or INCONCLUSIVE.*

The formal language presented in this framework provides the ability to express privileges on *non-atomic* actions. Here, a set of basic definition are described as follow:

1. *Atomic action*: the emission or the reception of a message between two system entities (or components) using the following syntax $Entity_1 ?or! Msg(Par_1, Par_2, ..., Par_n) Entity_2$ where $Entity_1$ and $Entity_2$ represent the source or the destination of the message. $'?'$ and $'!'$ define a reception and emission of a message by $Entity_1$. $Msg(Par_1, Par_2, ..., Par_n)$ represents the message exchanged between $Entity_1$ and $Entity_2$ with its parameters. $Entity_1$, $Entity_2$, Msg and Par_i can be replaced by the symbol $*$ to represent any entity, any message or any parameter.

2. *Non-atomic action*: If α and β are actions, then $(\alpha;\beta)$, which means "α is followed immediately by β" and $(\alpha;*;\beta)$, which means "α is followed by β" are non-atomic actions.

3. *Formula*: If α is an action then $start(\alpha)$ (action α is being started) and $done(\alpha)$ (action α is done) are formula.

4. *Deontic modalities*: If A is a formula, then a modality $\mathcal{O}$ ('A' is mandatory, $\mathcal{F}$ ('A' is forbidden) and $\mathcal{P}$ ('A' is permitted) are formula.

Interpretation	Rule definition
Permission granted to usr_1 to write on $file1.doc$ which is managed by $Server_A$, if earlier, the user usr_1 was authenticated and not disconnected	$\mathcal{P}(start(usr_1!Msg(ReqWrite, fich1.doc)Server_A)\| \ominus (done(usr_1!Msg(AuthReq)Server_A)) \wedge done(usr_1?Msg(AuthOK)Server_A) \wedge \neg done(usr_1?Msg(DisconnectReq)Server_A))$
$Server_A$ can not accept more than two authentication request from the same user in the same second	$\mathcal{F}(start(Server_A?Msg(https, AuthReq)user)\| \mathcal{O}_{\leq -1s} done(Server_A?Msg(https, AuthReq))user) ;*; Server_A?Msg(https, AuthReq)user)$

Table 2.4: Example of rule specifications for policy checking

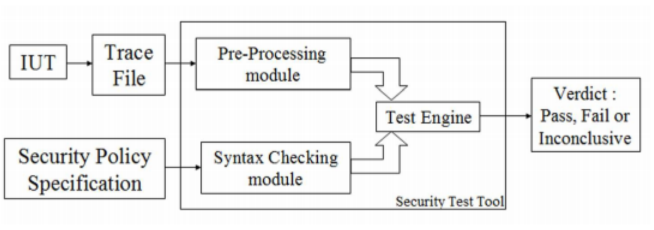


Figure 2.12: Security rule specification architecture of framework presented in [52]

In order to perform the security checking process, an algorithm is presented as well in [52] to deal with obligation rules and retrieve the verdict.

Table 2.4 shows an example of security policies mapped into specification rules. As it can be appreciated on it, the type of security policies that can be mapped into a formal definition rules is based on the use of attributes (e.g. *DisconnectedReq, AuthReq*), elements (e.g. *file1.doc, $Server_A$*) and actions (e.g. *start, done*). In the context of PNA framework this can be mapped into *assets* with their corresponding attributes. It is important to emphasize the fact that the use of logic as part of security policy specification, provide a way to link events in order to create contexts. With regard to the functional model, in this framework there is no multiple data aggregation, however there are other shared components with NSM, SIEM and PNA contexts, such as control parameter (security policies), single input (trace file) that mapped into PNA it would be the raw network traffic. In addition, the actual result are not IDS alerts, but a verdict about a security policy which is mapped into a verdict of network event audit. Thus, the key concepts from this approach that can be taking into consideration within the PNA framework is the implementation of a mapping process that may use logic for event correlation.

2.4.5 Alert post-processing framework

Spathoulas et al. [76] propose a framework focused on alert pre-processing. In this approach, the input of the system is the set of alerts generated by multiple IDS, thus, by using different algorithms, the system generates a higher level interpretation about the security events that have occurred on the network. This framework defines the following data formats in order to handle inputs and outputs for each component: *Alert* (A), *Aggregated Alert* (AA), *Aggregated Alerts' Cluster* (AAC), *Alert Set* (AS), *Aggregated Alert Set* (AAS), *Aggregated Alert Cluster Set* (AACS). In addition, the following fields from IDS alerts are used: *Attack ID* (AID), *Attack Class* (ACL), *Timestamp* (T), *Source IP* (SIP), *Destination IP* (DIP). Moreover, three algorithms are defined in order to aggregate components, merge AAS, and clustering. Figure 2.13 depicts the operational architecture of this framework which is based on three phases.

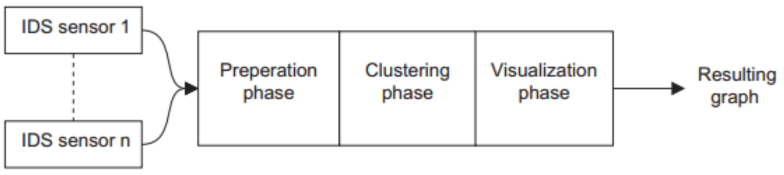


Figure 2.13: Alert post-processing framework architecture proposed in [76].

1. *Preparation phase*: the system is fed by the set of alerts generated by the IDS. The input is passed to the *Aggregation Component* in order to convert groups of alerts, belonging to a single security events, into an aggregated alert that contains information about relevant events. Then, an additional component, the so-called *Merging Component* (MC), filters aggregated alerts that already have been referred. Figure 2.14 shows the functional model of this phase.

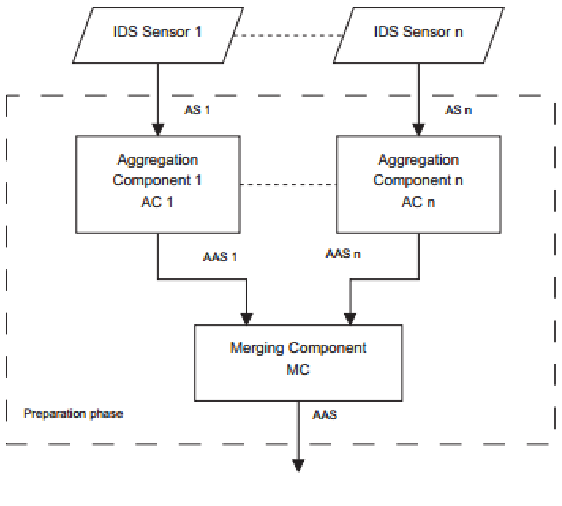


Figure 2.14: Alert post-processing framework proposed in [76]. Preparation phase.

2. *Clustering phase*: this phase has two operation modes, simple and advanced. On the one hand, *Simple mode* outputs the *MC* data to a single *Cluster Component* (CC) in order to

create inter-related clusters of aggregated alerts, based on similarity clustering proposed by Valdes et al. [84]. On the other hand, *Advanced mode* proposes an hypothesis about missed security events, before the clustering process. The intermediate clustering is performed by *CC* and it passes the generated small cluster to the *Cluster Generator* (CG), in order to create artificial clusters. These clusters are used to produce high clustering as the final set of clusters. Figure 2.15 shows the functional model of this phase.

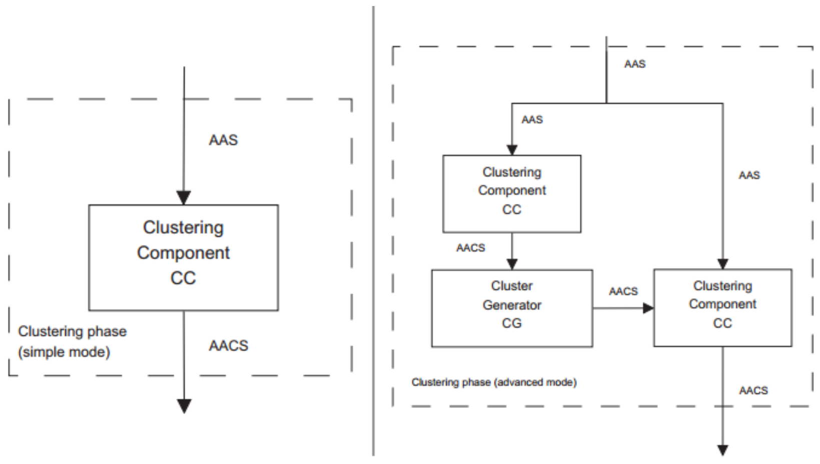


Figure 2.15: Alert post-processing framework proposed in [76]. Clustering Phase: Simple and advanced modes.

3. *Visualization Phase*: This component creates a graphical representation of the produced clusters, in order to create a high-level interpretation. Figure 2.16 shows the functional model of this phase.

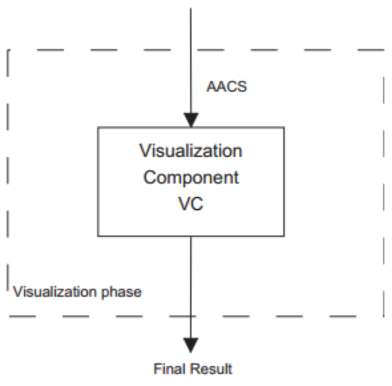


Figure 2.16: Alert post-processing framework proposed in [76]. Visualization phase.

The similarity value is calculated by combining the four similarity components with respect to T, AID, SIP and SIP such that:

$$sim(a, c) = w_{aid} * sim_{aid} + w_t * sim_t + w_{sip} * sim_{sip} + w_{dip} * sim_{dip}$$

The weights are values between 0 and 1 so that

$$w_{aid} + w_t + w_{sip} + w_{dip} = 1$$

The AID similarity is calculated with regards to the attack class so that:

$$sim_{aid}(AA_1, AA_2) = \begin{cases} 1 & \text{if } AID_1 \equiv AID_2 \\ 0 & \text{otherwise} \end{cases}$$

From a general perspective, this framework describes a functional model very similar to the intended PNA framework. First, the model includes an input that passes through preparation, analysis and visualization phases. In this sense, very similar phases are intended to be defined within the proposed PNA framework as well, just using with different components and internal workflow. Second, data aggregation is performed with IDS data from multiple sensors and it is merged into a single point of analysis. Here, this approach describes an additional task done by the *Merging component* producing classified data. In fact, the key idea to take into account from this framework is that data aggregation includes not only a merging process, but also classification processes based on specific network attributes, used to correlate data and to create contexts of single events.

2.4.6 Passive Network Appliance

Schultz et al. [72] describe the so-called *Passive Network Appliance* (PNA, but referred as PNApp in this thesis to distinguish it from Passive Network Audit). It is intended to monitor network activities and its main goal is real-time monitoring, thus NSM features are mostly involved, focusing on measurement purposes. In general terms, the idea of this framework is to provide a cost-effective way of network monitoring based on *real-time packet logging* used to generate *snapshots* of network behaviour through the time. It has a strong focus on performance. Figure 2.17 describes the operational architecture of this implementation, in which there are two main environments: (1) *kernel space*: performs low-level tasks such as packet decoding and network elements tracking. (2) *user space*: performs high-level tasks such as monitoring based on previous packet decoding and alerting process. Since it is a real-time passive approach, the actual output is the same input, thus only decoded attributes are used to analyze and produce meaningful information.

Despite the fact that this framework is mainly focused for high-performance real-time monitoring, and the intended PNA framework is focus on audit of security-related events, there are three key ideas that can be taken into consideration in the context of PNA.

- First, this implementation uses two *stateful real-time monitors*: *IP-to-IP tracker* and *IP-statistics-tracker*. This can be used to track single IP activities in order to create meaningful contexts for network audit purposes.
- Second, the idea of *snapshot* of network behaviour is in fact the way *real-time audit* can be performed within the PNA framework. In practice, this can be done by retrieving, in real-time, samples of network traffic of certain period (input dataset) and then to enqueue them on the audit engine. Details of this process are described in Chapter 3.
- An API for real-time monitoring, that in PNA context such an API is mapped into network audit capabilities, providing a scalable implementation.

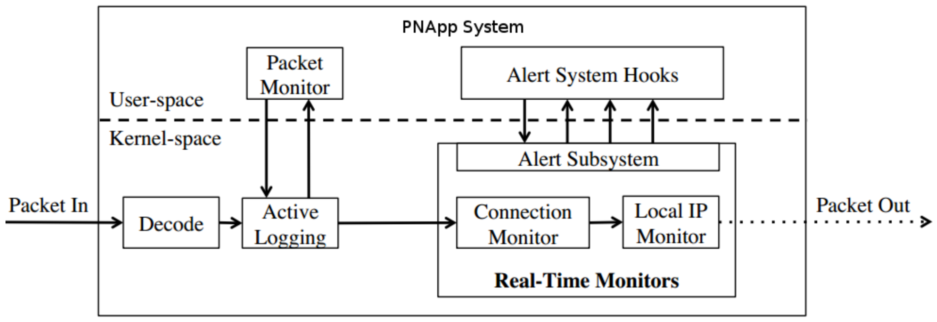


Figure 2.17: Passive Network Appliance functional model.

2.5 Summary of gathered knowledge and features

Taking into account characteristics of existing frameworks presented in this section, as well as the capabilities that current NSM and SIEM technologies provide, a list of useful and suitable features that will be part of PNA framework are described in Table 2.5. This information addresses the first two support questions that try to set the context to answer the research question of this thesis (Section 1.4).

Framework	Focus	Suitable features for PNA	Other considerations
Common Intrusion Detection Framework [15]	Specification language for intrusion detection	Language format and encoding for event description, timing tags for event correlation and timeliness, tagging granularity for data classification	Syntax of language and encoding
Data mining framework [45], [44]	Specification language and frequency analysis	Language for event description, timing tags for event correlation and timeliness, frequency tags for identification of frequent episodes	Syntax of language, type of frequency analysis
Event Calculus Framework [71]	Specification language	Specification language for event description, multiple data aggregation, control parameters: configuration and security requirements for compliance verification	Syntax of language, attributes of configuration parameters
Passive testing framework based on security rules specification [52]	Security compliance verification	Specification language for event identification and policy checking, Logic for event correlation, Control parameters: configuration and security policies, single result: verdict	Syntax of language, verdict validation
Alert Post-processing Framework	IDS rules preprocessing	Modular architecture, IDS data correlation method	Analysis of classification methods
Passive Network Appliance	Real-time monitoring and packet logging	General architecture, Tracking modules, Application Programming Interface (API)	Modularization

Table 2.5: Summary of useful features from some existing detection/monitoring frameworks

Chapter 3

Framework Specification

3.1 Overview

Previous sections have addressed basic concepts about Intrusion Detection technologies, as well as some existing approaches related to PNA (summarized in Table 2.5). The methodology used to define the framework specification including four main aspects:

- *Findings of background study* that were taken into consideration to define features and characteristics of the architecture.
- *Definition of framework's capabilities* based on PNA context discussed in Chapter 1. This was necessary to determine specific purposes of framework's components.
- *Characterization* of detection features and suitable design taking into account literature review presented in Chapter 2, as well as intended features part of the research question (i.e. to provide streamlined, flexible and effective framework).
- *Analysis of capabilities* of a variety of traffic analysis tools in order to identify useful features on PNA context and their integration with framework's components.

Thus, this section will describe the framework specification including architecture, functional model and in-depth description of its components.

3.1.1 Purpose

The present work aims to develop the so-called Passive Network Audit Framework (PNAF), which is a framework that uses PNA techniques for network traffic analysis. The main goal of PNAF is to provide the capability to get a security assessment of network traffic and provide a high-level interpretation of security events in an automated way. It is done by combining different analysis techniques, algorithms and technologies, all specifically based on PNA approaches. Furthermore, the practical implementation or the framework, designed as Fox-IT technology, aims to define an API that provides the aforementioned capabilities in a flexible and structured way.

3.1.2 Scope

In order to generate useful information within a network security context, PNAF is intended to provide the capability of getting the following information:

a. Summary of the Security Level of the network

b. Findings of anomalous activities

c. Findings of security audit policy

d. Findings of impact analysis

e. Summary of security recommendations

f. Reference of evidence

3.2 Main architecture

PNAF is a modular framework in which every component is intended to perform a specific task as part of a serial process. This means that the input of a certain component is the output of past processes. The basic idea of the architecture is based on the general concept of *Audit*, in which an *Auditor* performs a set of tests, checks and verification processes over a *System* which is comprised by a set of *Assets*. In that sense, the aforementioned entities exist as well on PNAF, such that the framework is the working environment that provides audit capabilities. Figure 3.1 shows the main architecture of PNAF, which receives *input* data from the *Auditor* who defines the audit parameters (*configuration* and baselines), hence the framework can perform a set of specific processes in order to generate a meaningful output report, following the purpose and scope presented in Sections 3.1.1 and 3.1.2.

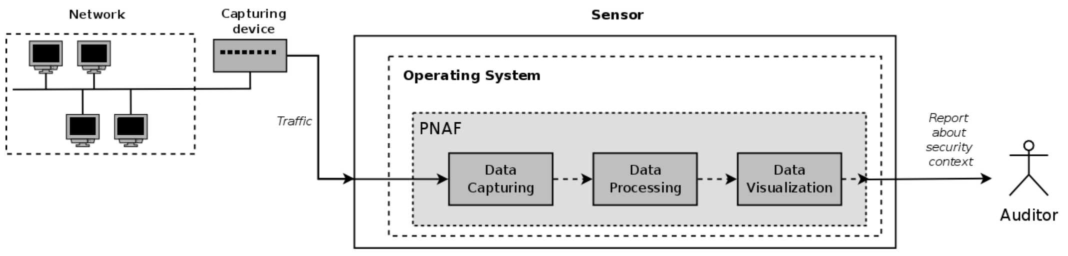


Figure 3.1: PNAF architecture

3.3 Modules

PNAF is comprised by three main modules: Data Capture Module (DCM), Data Processing Module (DPM) and Data Visualization Module (DVM). These modules all together define the so-called PNAF instance, which is the core entity within the *input-function-output* scheme depicted in Figure 3.1. Each module includes *engines* for specific purposes. Figure 3.2 shows a single representation of the components within the framework (i.e. not going into the level of formal representation such as class diagrams, but just giving an overview of the framework general structure).

3.3.1 Data Capture Module (DCM)

The main purpose of this module is to handle the input of the framework, which can be either network traffic from the network interface (*online mode*), or network traffic from a *pcap* capture file (*offline mode*). It includes pre-processing tasks over network traffic performed by the toolset selected within the framework. This module is comprised by two engines and the corresponding sequence diagram is depicted in Figure 3.3

Network Traffic Capture Engine (NTCE)

This engine manages the whole capturing process within the framework when it is used in *online* mode. Its purpose is to schedule the capturing process, to control capturing tools and to handle captured raw traffic.

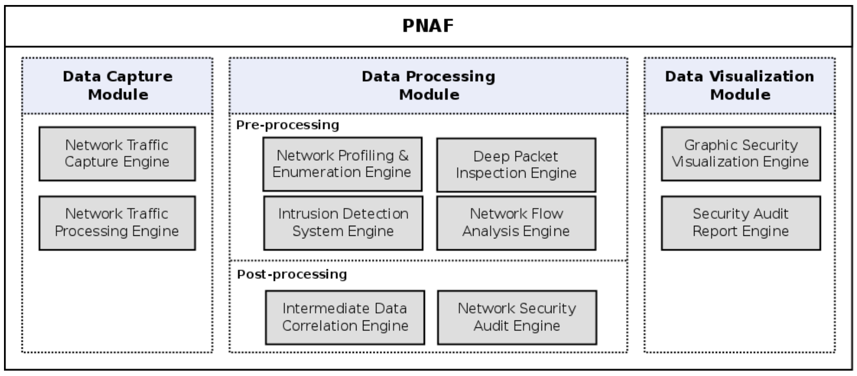

PNAF
Data Capture Module
Network Traffic Capture Engine
Network Traffic Processing Engine
Data Processing Module
Pre-processing
Network Profiling & Enumeration Engine
Deep Packet Inspection Engine
Intrusion Detection System Engine
Network Flow Analysis Engine
Post-processing
Intermediate Data Correlation Engine
Network Security Audit Engine
Data Visualization Module
Graphic Security Visualization Engine
Security Audit Report Engine

Figure 3.2: Passive Network Audit Framework components

Network Traffic Processing Engine (NTPE)

This engine handles all external tools used within the framework (e.g. IDS, flow analyzers, decoders, etc.) that are executed to generate the input data for the DPM. Its purpose is not only to execute the tools in an automated way, but also to apply the framework audit parameters, integrity checking of the actual output and execution errors handling. The output of this engine is the actual log dataset generated by the tools, each of them with specific format readable by the framework (e.g. Comma-Separated Value (CSV), JavaScript Object Notation (JSON), etc).

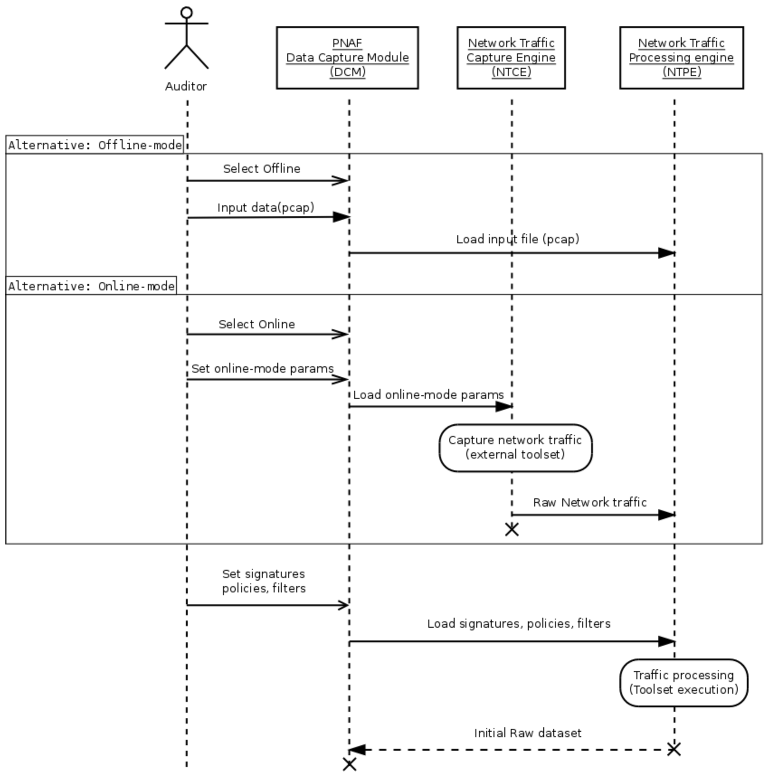

Auditor
PNAF Data Capture Module (DCM)
Network Traffic Capture Engine (NTCE)
Network Traffic Processing engine (NTPE)
Alternative: Offline-mode
Select Offline
Input data(pcap)
Load input file (pcap)
Alternative: Online-mode
Select Online
Set online-mode params
Load online-mode params
Capture network traffic (external toolset)
Raw Network traffic
Set signatures policies, filters
Load signatures, policies, filters
Traffic processing (Toolset execution)
Initial Raw dataset

Figure 3.3: Sequence diagram of Data Capture Module.

3.3.2 Data Processing Module (DPM)

This module performs all processing tasks to translate the raw pre-processed data from the toolset, into a classified, sorted and specifically processed information that can be sent to the DVM in order to be presented as a report. This module includes engines for pre-processing with dynamic aggregation, which provides the flexibility to include additional engines for pre-processing tasks. The current design includes four engines for pre-processing, whereas two fixed modules are defined for post-processing tasks (see Figure 3.2). The corresponding sequence diagram is depicted in Figure 3.4.

Network Profiling and Enumeration Engine (NPEE)

The purpose of this engine is to retrieve a profile of the system by performing an enumeration process of all assets within the environment that it is being audited. The enumeration process includes not only identification of asset themselves, but also identification of their profiles (e.g. type of system, role, software that is being used, active services, etc). Such an information is used afterwards to identify policy violations as well as vulnerabilities and threats within the environment. Tools that can be used to generate the NPEE data set are: P0f[89], Prads[28], Httpry[12], tcpdstat[87], Snort (OpenAppID engine)[70] and Suricata (HTTP parser) [61].

Intrusion Detection System Engine (IDSE)

This engine is intended to execute all intrusion detection engines used within the framework. Its purpose is not only to execute and get the list of alerts triggered by IDS, but also to perform a tracking and decoding process to gather and analyze payloads for further evidence reference and data correlation. Moreover, a classification and prioritization process is done in order to group and identify threats according the severity and possible impact they may represent. Such a prioritization is a dynamic parameter that can be defined as part of a framework's profile. Tools that can be used to generate IDSE data are Snort[70], Suricata [61] and Bro[80].

Deep Packet Inspection Engine (DPIE)

This engine is intended to perform an in-depth analysis over the network traffic, including payload decoding in application layer. Thus, data extracted from protocol communications can be analyzed in order to find potential patterns or anomalies that could represent a threat, policy violation or even compromise evidence. Deep Packet Inspection (DPI) includes not only such a decoding process, but also additional gathering tasks such as file extraction from network traffic, tokens analysis from decoded strings and protocols, etc. PNAF is focused on DPI over Hypertext Transfer Protocol (HTTP), Domain Name System (DNS) and Transport Layer Security (TLS) protocols. Tools that can be used to generate DPIE dataset are: Suricata (HTTP and DNS parsers), Httpry, argus[68], Bro[80], Chaosreader[32], Nftracker[27], Tcpxtract[35], TcpExtract[18] and Tcpflow[24].

Network Flow Analysis Engine (NFAE)

The purpose of this engine is to perform a statistical analysis of network flows in such a way that an overview of network utilization is presented, as well as general statistics for protocols and hosts within the working environment that is being audited. Moreover, Network Flow Analysis (NFA) provides the capability to identify possible anomalies and attacks based either on frequency analysis or simple counters analysis. Tools that can be used to generate NFAE dataset are: Argus[68], Cxtracker[26], Tcpdstat[87] and Silk[57].

Intermediate Data Correlation Engine (IDCE)

The purpose of this engine is to merge all *intermediate data* and perform a correlation process of data produced by NPEE, IDSE, DPIE and NFAE engines.

Network Security Audit Engine (NSAE)

This component is the leap engine within DPM in which all the pre-processed information from other engines are sent before the next stage is reached. The goal of this engine is to perform *auditing*. In fact, the core security audit process of PNAF is done on this stage since the input for this engine is comprised by all necessary pre-processed information where all profiles, roles, policy violations and anomalies are identified from. The behavior of this engine is managed by a set of four audit components:

1. PNAF Rules: Specification to describe audit events
2. PNAF Dictionary: Specification to describe security events using baselines based on well-known security issues (vulnerable software).
3. PNAF Blacklist: Specification of forbidden terms that should not be found on audit trails.
4. Anomaly detection: A simple anomaly detection algorithm that can be used to identify potential threats or unusual activities, based on sequences analysis. If a training dataset is provided, sequence of events can be checked in such a way that anomalies can be identified in a more accurate way, otherwise the training dataset is generated over the time when the framework is used in *online mode*.

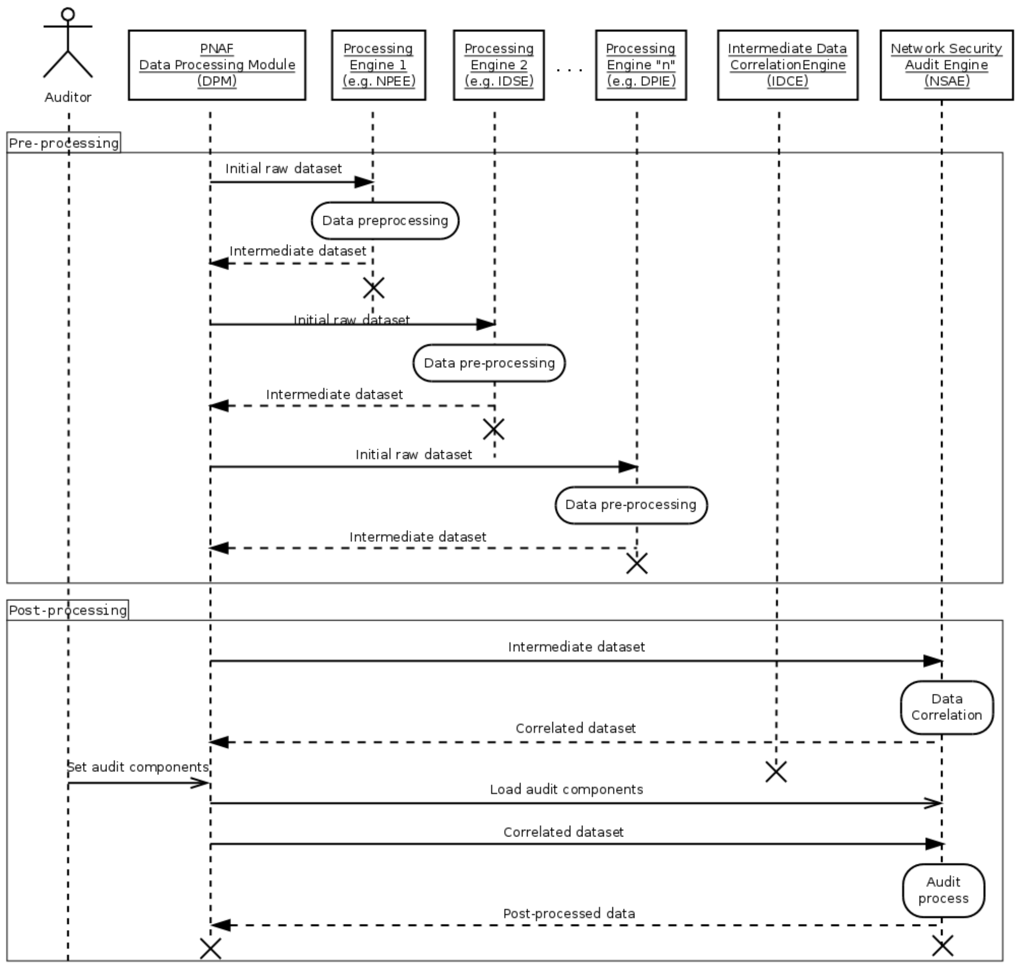


Figure 3.4: Sequence diagram of Data Processing Module.

3.3.3 Data Visualization Module (DVM)

This module is intended to take processed information from the DCM in order to present a meaningful report to the auditor. This module presents the key findings of the audit process such that summary of threats, policy violations, and impacts are reported, emphasizing the most important aspects to take into consideration. This module is comprised by two engines and the corresponding sequence diagram is depicted in Figure 3.5

Graphic Security Visualization Engine (GSVE)

This engine uses simple visualization tools to transform the audit result into some graphical visualization. The first version of the framework includes only web data visualization using JSON format. Further versions may include tools for tracking visualization.

Security Audit Report Engine (SARE)

This engine maps the actual findings of the whole audit process into a human-readable report, emphasizing the key aspects. This is the middle layer between the actual data generated by PNAF and any other external visualization tool used to generate the final report. An important feature of such an engine is that, based on the audit results, it can provide not only some recommendations for the analysts, but also it can generate new rules for IDS (Snort/Suricata format) in order to extend and improve detection capabilities for similar problems in the future.

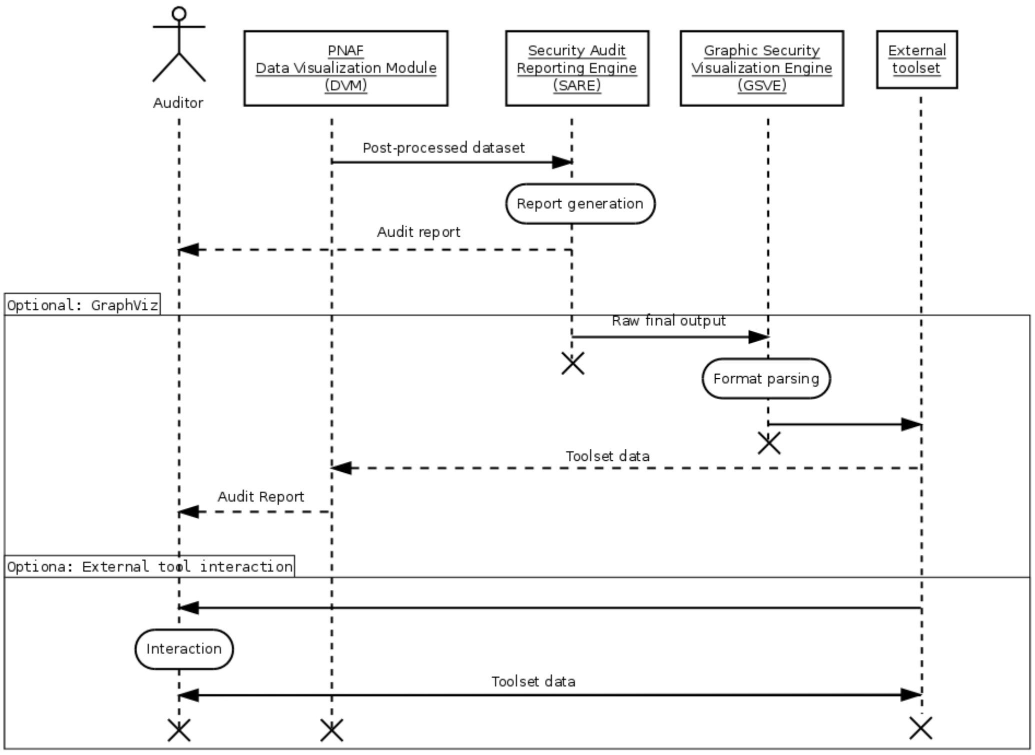


Figure 3.5: Sequence diagram of Data Visualization Module.

3.4 Selection of technologies

Taking into account the purpose and scope of the framework, a set of analysis tools and technologies have been analyzed in order to select the most suitable ones to be implemented as part of the framework. The three main aspects taken into account were:

1. Data collection: The purpose was to find the best way to capture network traffic and select the most suitable tool, also taking into account the fact that PNAF is intended to be deployed within a Small and Medium Enterprises (SME) environment (although not exclusively).
2. Data processing: The purpose was to find a suitable way of data processing. This process includes not only the way data is handled, but also what kind of data is relevant on each stage such that only necessary information is used in order to improve the efficiency of the framework.
3. Data analysis: The purpose was to define the kind of information that can be retrieved, analyzed and eventually interpreted, taking into consideration audit parameters that can be defined (e.g. DPI over HTTP data allows to identify *user-agent* for vulnerability discovery based on software version).

Table 3.1 shows an overview of the tools that have been analyzed, presenting PNAF modules and engines they are used on, their purpose, specific data retrieved and advantages considered for selection. It is important to emphasize the fact that some tools provide the same type of information, however an important feature of the framework is that data correlation process not only links data into a single security events, but also compares and validates similar information retrieved from different tools in order to create more reliable results.

Tool	Module	Engine	Purpose	Data Retrieved	Advantages
Suricata logger	DCM	NTCE	Real time traffic capturing	Raw traffic	Efficiency in traffic capturing using *af_packet* or *pfring* (High-performance network traffic capturing technology)
P0f	DPM	NPEE	Network and service enumeration	Link types, Platform description, Operating System (OS) description and version, system role	Detection by multiple methods: signatures, behaviour
Snort Open AppId	DPM	NPEE	Application identification	Application identified within the network traffic	Application identification by protocol analysis and not by port number checking
Tcpdstat	DPM	NPEE	Protocol enumeration and statistics	Protocol counters	Classification of network traffic by protocol layers
Prads	DPM	NPEE	Network service and enumeration	Link type, Vlan detection, Role, protocols, Platforms, OS description and version	Detection by multiple methods.
Suricata IDS	DPM	IDSE	Misuse-based IDS engine	IDS alerts	Flexible detection by signatures, Application layer parsers, traffic normalization, efficiency, payload gathering

Snort IDS	DPM	IDSE	Misuse-based IDS engine	IDS alerts	Flexible detection by signatures, preprocessors, traffic normalization, efficiency, payload gathering
Bro IDS	DPM	IDSE	Misuse-based IDS engine and policy checker	IDS and policy alerts	Flexible detection by signatures and policies, application layer parsers
Argus	DPM	NFAE	Network Flow analysis	Network Flow statistics	Efficient method to analyze large amount of network traffic
Cxtracker	DPM	NFAE	Network Flow analysis	Network Flow statistics	Efficient method to analyze large amount of network traffic
Silk	DPM	NFAE	Network Flow analysis	Network Flow statistics	Efficient method to analyze large amount of network traffic
Bro HTTP parser	DPM	DPI	HTTP protocol data decoding	HTTP headers, User agents, URL, Domains, Host role, HTTP status code, Mime-types, proxy information, file names, referrer, tags, usernames, payloads	More detailed information of HTTP data
Bro TLS parser	DPM	DPI	TLS protocol data decoding	SSL/TLS fields: issuer, validity, subject, version, cipher, client subject, server name	More detailed information of SSL/TLS data
Bro SSH parser	DPM	DPI	SSH protocol data decoding	SSH client and server version, direction, status	Detailed information of SSH data
Bro DNS parser	DPM	DPI	DNS protocol data decoding	DNS fields: query, qclass, rcode, TTL, AA, RA, rcode, qtype	Detailed information of DNS data
Bro SSH parser	DPM	DPI	SSH protocol data decoding	SSH client and server version, direction, status	More detailed information of SSH data
Suricata File decoder	DPM	DPI	File extraction	Filename, file type, source, hash (md5, sha1, sha256), bytes	File extraction for malware identification and other policy violations
Httpry	DPM	DPI	Http protocol data decoding	HTTP headers, URL, Domains, Host role, HTTP status code, payloads	In-depth analysis of HTTP data not only related with IDS alerts
Suricata TLS parser	DPM	DPI	TLS certificate analysis	IssuerDn, TLS version, Subject, Fingerprint	Pre-processing of TLS certificates info for further analysis
Ssldump	DPM	DPI	TLS data analysis	TLS Protocol workflow	TLS behaviour analysis
Suricata DNS parser	DPM	DPI	DNS data analysis	Rrtype, TTL, Type, Rname	Passive DNS analysis, Domain blacklisting

Passive DNS	DPM	DPI	Passive DNS analysis	DNS full data and stats	Data gathering for further analysis
Suricata File parser	DPM	DPI	File extraction from network traffic	Single files transferred within complete TCP sessions	Malware detection
Tcpxtract	DPM	DPI	File extraction from network traffic	Single files transferred within complete TCP sessions	Malware detection
TcpExtract (python)	DPM	DPI	File extraction from network traffic	Single files transferred within complete TCP sessions	Malware detection
Chaosreader	DPM	DPI	Application layer data decoding	HTTP, DNS, FTP, SMTP conversations	Payload analysis
Tcpflow	DPM	DPI	TCP sessions reassembly and HTTP decoding	Single sessions and basic HTTP data: method, protocol	Evidence gathering and session tracking

Table 3.1: Toolset used within the framework.

In order to retrieve information about security events, PNAF performs data correlation using both IDS schemes: *misuse-based* through PNAF rules and *anomaly-based* through a data mining algorithm. As it was explained in Section 3.3.2, the core audit engine of PNAF is controlled by four detection components. On the one hand, *PNAF rules*, *auditor dictionaries* and *blacklists* involve *misuse-based*. On the other hand, the fourth detection component involves *anomaly-based* techniques based on data mining using *short sequences* analysis.

It is very important to emphasize the fact that PNAF relies mainly on the use of *misuse-based* techniques since *Compliance* is the key process in which *audit* activities are based. This process is of course supported by additional processes such as evidence retrieval and impact analysis. Thus, since as part of the audit process it is necessary to identify any fact that does not correspond to predefined baselines, then *misuse-detection* is perfectly suitable to be used. Nevertheless, as it is proposed in this work, *anomaly-based* techniques can be applied for audit purposes as well.

3.5 Misuse-based features

In order to perform the audit process, a simple algorithm is defined in such a way that all pre-processed information retrieved from data aggregation is used to create meaningful context. This algorithm involves three *misuse-based* approaches: *rules, dictionary and blacklist*, meaning that different layers of correlation are used within the framework. These so-called *Audit components* are defined as follows:

3.5.1 Audit Rules

This is the general specification to identify anomalous events on network activities from audit perspective. The purpose of this specification is to define a default *language* that PNAF is able to understand and apply detection parameters, just like *signatures* are applied on *misuse-based* IDS. Hence, PNAF rules are defined by five main properties showed in Listing 3.1:

```
ID -> Score -> Asset -> Attribute -> Flag
```

Listing 3.1: PNAF rules specification

- *ID*: Unique Id of the rule. It is used for internal identification purposes.
- *Score*: Defines the rule relevance for audit purposes. It is used afterwards to sort the security events according to their impact.
- *Asset*: Description of an entity part of the working environment. Examples of possible values can be single IP address, network subnet, single device explicitly identified, hostname or event. Appendix XX shows the full list of values defined within the framework.
- *Attribute*: Defines a specific attribute of the Asset defined on the rule.
- *Flag*: Defines whether specific value for the attribute is considered either as *valid* as part of *permissive* policy, or as *violation* as part of *restrictive* policy.

Listing 3.2 shows some examples of PNAF rules. Rule with $ID = 1$ audits that OS version on hosts within subnet 192.168.1.0/24 use a Windows based system with minimum version 7. Rule with $ID = 2$ audits that WebBrowser version used by hosts within subnet 192.168.2.0/24 use any software except Internet Explorer. Rule with $ID = 3$ checks whether host 192.168.3.10 has output flows before 18:00hrs. Finally, rule with $ID = 4$ audits that host 192.168.10.10 runs SSH service only on TCP port 22 and the software has to be OpenSSH. The *Asset* description can be defined in Classless Inter-Domain Routing (CIDR) format.

```
1 -> 1 -> 192.168.1.0/24 -> OS -> Platform=Windows, MinimumVersion=7
2 -> 1 -> 192.168.2.0/24 -> WebBrowser -> ForbiddenBrowser=IExplorer
3 -> 5 -> 192.168.3.10 -> Flow -> MaxOutTime=1800
4 -> 2 -> 192.168.10.10 -> Service -> SshValidPort=22, SshServer=OpenSsh
```

Listing 3.2: Example of PNAF rules

3.5.2 Audit Dictionary

It includes a list of predefined and *well-known* issues related to vulnerable versions of software used by assets. Hence, some vulnerabilities can be identified by comparing the OS, Services, WebBrowsers, etc. against such predefined dictionary. This dataset can be updated periodically such that the most recent well known vulnerable versions used within the environment can be identified. Listing 3.3 shows the dictionary specification which is comprised of five fields:

```
ID -> Score -> Software -> Version -> Reference
```

Listing 3.3: PNAF dictionary specification

- *ID*: Unique Id of the rule. It is used for internal identification purposes.
- *Score*: Describe definition relevance for audit purposes. It is used afterwards to sort the security events according to their impact.
- *Software name*: Name of software that is being described ad vulnerable.
- *Software version*: Software version described as vulnerable.
- *Reference*: Reference to related vulnerabilities advisories.

Listing 3.4 show examples of Dictionary terms. Entry with $ID = 1$ looks for assets that use web browser Iexplorer with versions 9,10 and 11 since they are related to vulnerability with CVE-2013-3893. Entry with $ID = 2$ looks for assets that use Apache 2.2 web server related to vulnerability CVE-2014-0098. Entry with $ID = 3$ looks for SSH Server 5.5 related to CVE-2012-5975.

```
1 -> 1 -> Iexplorer -> 9,10,11  -> CVE-2013-3893
2 -> 5 -> Apache -> 2.0.9 -> CVE-2014-0098
3 -> 3 -> SSHTectiaServer -> 6.04 -> CVE-2012-5975
```

Listing 3.4: Example of PNAF dictionary

3.5.3 Audit Blacklist

It is a template that defines a list of forbidden hosts, servers, services, versions, domains, etc. Listing 3.5 shows PNAF Blacklist specification which includes three fields:

```
                ID -> Score -> Term(s)
```

Listing 3.5: PNAF blacklist specification

- *ID*: Unique Id of the rule. It is used for internal identification purposes.
- *Score*: Describe definition relevance for audit purposes. It is used afterwards to sort the security events according to their impact.
- *Term*: Definition of forbidden term to look during audit process.

Listing 3.6 show examples of Blacklist. Entries can be described either with a single term or multiple terms.

```
1 -> 1 -> productX
2 -> 5 -> music,download
3 -> 3 -> snmp, community
```

Listing 3.6: Example of PNAF blacklist

Algorithm 1 shows the way misuse-based techniques are used in PNAF. The three *baselines* used as reference for security event identification should be defined and loaded. Then, the immediate process is to describe all assets involved within network traffic. Thus, based on the information of every asset (e.g. role, description, OS, UserAgent, protocol codes, etc.) a matching comparison can be done against the three references by calling the corresponding procedures *checkBlacklist*, *checkDictionary* and *checkRules*. In order to define relevance of security events, a simple scoring method is used in such a way that every single asset accumulates three scores, one per detection component. Afterwards, a summary of these scores is taken into account in order to sort security events based on relevance.

As it is presented in Algorithm 1, Asset description (*DescribeAssets* procedure) involves *Asset enumeration*, *Asset profiling* and *Asset list*. These tasks involve DPI to analyze the input dataset and retrieve specific data such as software products that are being used within the network, including operating systems, web browsers, service software, web servers, etc. Since the input dataset may not be necessarily in standard formats such as *Common Log Format* (RFC 1413) or *User Agent format* (RFC 2616), which can be parsed easily, then an additional *tokenization* task is included within the framework in order to retrieve the needed information in such cases.

Algorithm 1 PNAF Misuse-based

```
MisuseAudit();
Sort Assets by score
End
procedure MisuseAudit
    Load PNAF Blacklist
    Load PNAF Dictionary
    Load PNAF Rules
    describeAssets()
    auditAssets()
    for every Asset do
        if asset score > threshold then
            trigger Alert on $asset$
    return
procedure AuditAssets(Assets)
    for every Asset do
        checkBlacklist(Asset)
        checkDictionary(Asset)
        checkRules(Asset)
    return
procedure describeAssets
    perform Assets Enumeration
    perform Assets Profiling
    create Assets List
procedure checkBlacklist(Asset)
    for every Asset Attribute do
        if attribute matches BlacklistItem then
            $AssetBLscore = AssetBLscore + BlacklistItemScore$
    return
procedure checkDictionary(Asset)
    for every Asset Attribute do
        if attribute matches DictionaryItem then
            $assetDCscore = assetDCscore + DictionaryItemScore$
    return
procedure checkRules(Asset)
    for every Asset Attribute do
        if attribute matches rule then
            $AssetPR : score = AssetBLscore + RuleScore$
    return
```

A *tokenizer* is used to extract all terms (i.e. tokens) within strings with unknown formats. Therefore, in the context of PNAF, a *token* is an alpha-numeric string that is extracted from a string and that provides special meaning (e.g. software product name or version). It not only filters the tokens, but also takes potential meaningful strings that can be used to create a duple *str1-str2* where *str* is a string that might describe either a product name (e.g. web server name such as "*Apache*" or "*IIS*") or a product version (e.g. mixed version specification including major version, minor version, etc. such as "2.2.1", "1.5", etc). Thus, arbitrary strings are not taken into account to produce the final token that will be passed as *Asset attribute*. Algorithm 2 shows how the tokenizer works in PNAF. Important aspects are the way actual meaningful tokens are extracted using special delimiters (i.e. *,:/&)(=- ?_+;!%$#][*), as well as all possible combinations of both *alpha* and *alpha-numeric* strings. Thus, the actual output of the tokenization process is a list of tokens that represent values that will be compared against PNAF misused-based dictionaries.

Algorithm 2 PNAF Tokenization process

```
for every $string$ containing asset attributes do
    getAuditTokens($string$)
End
procedure getAuditTokens($String$)
    Load delimiters
    tokenize($String$, $delimiters$)
    if tokenizer outputs $alpha$, $numeric$ and $alphanumeric$ tokens then
        for every $alpha$ token do
            for every $alphanumeric$ token do
                create a duple ($alphaToken$-$alphanumericToken$)
            for every $numeric$ token do
                create a duple ($alphaToken$-$numericToken$)
    return list of duples
```

3.6 Anomaly-based features

The reason to use an *anomaly-based* algorithm is that an audit trail can describe the normal behaviour and characteristics of single assets over the time, thus, by analyzing sequences it might be possible to identify changes on this behaviour that otherwise would not be detected unless new definitions on rules, dictionary and blacklist were added. For instance, the fact that an asset has a lot of connections to *usual* hosts and after some time this behaviour changes, (due to malware infection for example), then connections targets may not be as usual even though they are not marked on the blacklists neither as policy violations. Nevertheless, it might be an indication of potential anomalous change. Another example would be the case of an asset that has been identified to be a web server running specific software, then it changes its software version which is not necessarily listed as vulnerable in dictionaries.

It is possible to use different *anomaly-based* approaches such as statistical-based, knowledge-based, machine learning, etc, taking audit trails as input data. Thus, in order to show the feasibility of using this kind of approaches, this section describes a method that involves data mining and that takes features from statistical-based and knowledge-based models. It is intended to provide anomaly detection within PNA context.

Guojun Mao et al. [54] propose an intrusion detection model through the analysis of short sequences. Two algorithms are proposed: Frequency Patterns (FP) and Tree Patterns (TP). Despite the fact that the original idea in [54] is intended to be applied to the analysis of sequences of OS *systems calls* this concept is a quite feasible to be mapped into PNA context. Thus, in PNAF, the so-called *"Audit sequence"* is comprised by a set of elements that are in fact audit trails that describes attributes of assets over the time. The use of this *anomaly-based* technique implies the need of a trainer dataset that is supposed to describe *normal* behaviour of assets involved on the network traffic.

3.6.1 Short sequences analysis

As it is presented by Guojun Mao et al. [54], this anomaly detection model is based on the concept of *short sequences*. Here, system calls that belong to a system process are transformed into a long sequence where each element represents a single system call. In order to get *short sequences*, a *sliding window* of size k is used in such a way that subsets of sequences of k elements are retrieved by shifting the window from the first element to the *n-th* element of the long sequence. In order to understand further definitions within PNA context, original definitions from [54] are summarized below:

Definition 1 (Process trace). Given a process p, a p's trace t is a sequence of system calls conducted by p from the beginning to the end of the process, denoted by $t =< c_1, c_2, ..., c_L >$.

Definition 2 (A short sequence). Given the size of a sliding window K and a process trace $t =< c_1, c_2, ..., c_L >$, if $L > K$, the set of short sequences of t is created through sliding windows, which means t is transformed into a set of short sequences $(s_j)_{L-K-1}$ where $s_j =< c_j, c_{j+1}, ..., c_{j+K-1} >$ is a short sequence with length $K(j = 1, 2, ..., L - K - 1)$

Such definitions can be mapped into the PNA context. To this end, audit trails can be used to describe *sequences*. Thus within PNA context it is possible to define the following concepts:

Definition 4 (Audit trail) Set of data describing a single asset. This data may refer to different contexts (e.g. audit trail of web servers used by certain asset over the time)

Definition 3 (PNA sequence). Given an *audit trail* s, a s' trace t is a sequence of *Assets attributes* contained within s from the beginning to the end of the audit trail, denoted by $t =< h_1, h_2, ..., h_L >$, where L is the number of assets attributes extracted from s.

Definition 4 (PNA short sequence) Given the size of a sliding window K and a PNA trace $t =< h_1, h_2, ..., h_n >$, if $L > K$, the set of short sequences of t is created through sliding windows, which means t is transformed into a set of short sequences $(h_j)_{L-K-1}$ where $s_j =< h_j, h_{j+1}, ..., h_{j+K-1} >$ is a short sequence with length $K(j = 1, 2, ..., L - K - 1)$.

Figure 3.6a shows an example of the concept using *"sc"* (system calls) as elements. Figure 3.6b shows a mapping example using *Audit Attributes* with a window size *k=3*

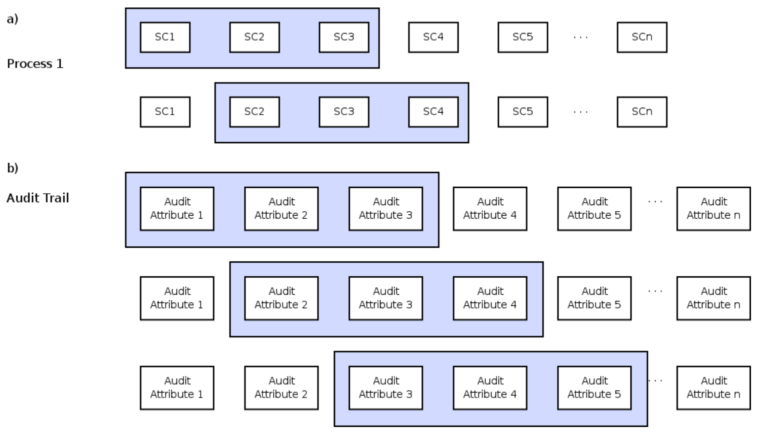


Figure 3.6: Short sequence concept and PNA mapping example

The original detection model proposed in [54] defines two algorithms: *Frequency Patterns* (FP) and *Tree Patterns* (TP). These algorithms are based on different detection methods, however both are intended to detect anomalies by analyzing short sequences. FP is based on a simple and general frequency analysis, whereas TP is based on the definition of a *forest* comprised by a set of *trees* of depth equals to the size of the window *k*, containing all the possible *short sequences* within the trace t of each process.

Moreover, the whole anomaly detection is based on a *trainer-detector* model. For both algorithms FP and TP, a base *Trainer* dataset (which is supposed to be comprised only by *normal* or *valid sequences*) is generated to execute a *Detector* algorithm that can be used in order to identify possibles anomalies within *unproven* short sequences. Furthermore, a threshold value is used to specify the boundary of normal patterns.

3.6.2 Frequency Pattern

This detection method is based on frequency analysis. Taking into account the Definition 4 of *Frequent System Call* presented in [54], the corresponding mapping into PNA context is as follows:

Definition 5 (Frequent asset attribute) Let the number of the *audit trails* be *M*, the size of sliding window *K*, and the user-specified threshold called *minimum support* (percentage) be δ. An asset attribute e is called a frequent asset attribute in the l th position $(l = 1, 2, ..., K)$ related to the i th *audit trail* $(i = 1, 2, ..., M)$ if the ratio of the times that e occurs in the l th position $(l = 1, 2, ..., K)$ related to the i th *audit trail*m to the number of all short sequences in the i th *audit trail* is not less than δ.

Definition 6 (Frequent pattern) Given a set of short sequences $(s_{ij})_{MXN}$ and the size of sliding window K, we can get frequent patterns, denoted by $(f_{il})_{MXN}$, such that each frequent pattern f_{il} is the collection of frequent asset attributes in the l th position related to the i th *audit trail.*

In summary, Frequency analysis need to generate a *Trainer* dataset by retrieving the *Frequent asset attributes* within the threshold and then check the *unproven sequence* by looking for short sequences that contain elements that are not present within the *Trainer* dataset.

3.6.3 Tree Pattern

The second approach, based on the use of *Trees* of depth *K*, is proposed as well in [54]. This scheme describes every single *short sequence* as a path within the Tree's structure. In fact, the whole structure may include a set of Trees or *Forest.* In addition, a *Matching Value* (*MV*) is used as a threshold, such that positive detection might be triggered taking into account the size of the sliding window K as well as the *MV*.

Taking as a basis both Definition 7 (Tree Pattern) as well as Definition 8 (Matching Value) proposed in [54], the corresponding mapping into PNA context defines the following concepts:

Definition 7 (Audit Tree Pattern). Given a training dataset of short sequences $(s_{ij})_{MXN}$ and the size of sliding window K, the tree patterns for anomaly detection trained from $(s_{ij})_{MXN}$, denoted as $(t_{il})_{MXV}$, are a set of trees, each of which t_{il} presents the l th tree in a forest related to *audit trail* st_i. Thus, every path of t_{il} from the root to a leaf constitutes a short sequence in $(s_{ij})_{MXN}$.

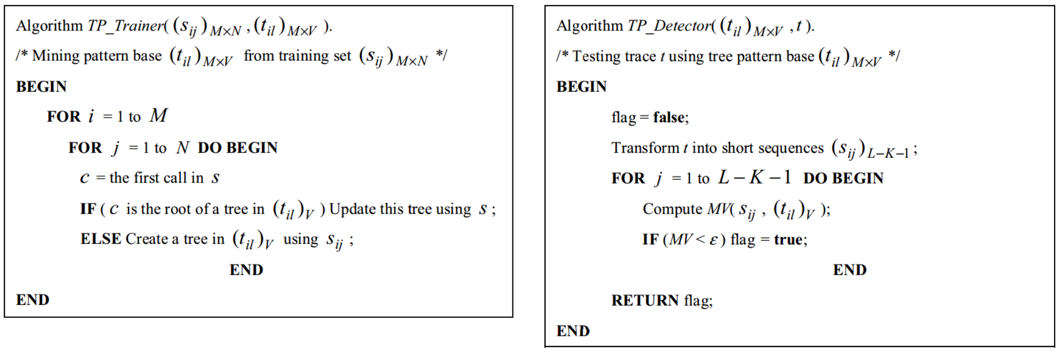
Definition 8 (Matching Value). Given a tree pattern base $(t_{il})_{MXV}$, and a user-specified matching parameter $w(w < 1)$, for a testing short sequence $s =< h1, h2, ..., h_K >$ (where K is the size of sliding window) related to an *audit trail* st_i , a matching degree needs to be evaluated between s and $(t_{il})_V$ that is the forest related to st_i in $(t_{il})_{MXV}$. The computing formula is as follows: $MV(s, (t_{il})_V) = w + w^2 + ... + w^L$, where L is the length of the longest sub-sequence of s that matches tree patterns $(t_{il})_V$.

There are some considerations to use this approach. On the one hand, with *Frequency Pattern* approach, an anomalous sequence should happen several times in order to be identified as frequent pattern and to be marked within the threshold. Therefore, unusual sequences that happen only once or a few times, might not be detected considering a high threshold, whereas using a low threshold the number of false positives may increase significantly turning this approach into unsuitable detection method for PNAF despite the fact that it is a very simple algorithm with computational efficiency.

On the other hand, Tree Pattern algorithm is a more complex technique which is able to characterize every single sequence within the network traffic. In this case, the amount of times that anomalous events happen is independent from the detection result. It just analyzes how the sequences themselves are defined. Hence, PNAF uses *Tree Pattern* algorithm and the corresponding algorithms are presented in Figure 3.7

Given the aforementioned mapping definitions, PNA sequences are defined according to asset attributes. The initial scope of PNAF includes only definitions showed in Table 3.2, nevertheless additional attributes may be used to describe different patterns. Thus, In order to create sequences, different fields are concatenated in such a way that a single sequence is a string of the form *asset-attribute-value* that depending on the kind of anomaly is intended to identify, different attributes are used.

Some modifications to the original algorithm have been performed. First, the original *Tree Pattern* algorithm uses ϵ as additional dynamic user-defined parameter, together with the sliding



```
Algorithm TP_Trainer( $(s_{ij})_{M\times N}$ , $(t_{il})_{M\times V}$ ).
/* Mining pattern base $(t_{il})_{M\times V}$ from training set $(s_{ij})_{M\times N}$ */
BEGIN
    FOR $i$ = 1 to $M$
        FOR $j$ = 1 to $N$ DO BEGIN
            $c$ = the first call in $s$
            IF ( $c$ is the root of a tree in $(t_{il})_V$ ) Update this tree using $s$ ;
            ELSE Create a tree in $(t_{il})_V$ using $s_{ij}$ ;
        END
END
```

```
Algorithm TP_Detector( $(t_{il})_{M\times V}$ , $t$ ).
/* Testing trace $t$ using tree pattern base $(t_{il})_{M\times V}$ */
BEGIN
    flag = false;
    Transform $t$ into short sequences $(s_{ij})_{L-K-1}$ ;
    FOR $j$ = 1 to $L-K-1$ DO BEGIN
        Compute $MV( s_{ij}$ , $(t_{il})_V$ );
        IF ($MV < \varepsilon$) flag = true;
    END
    RETURN flag;
END
```

Figure 3.7: Tree pattern trainer and detectors algorithms proposed in [54].

Anomaly to identify	Sequence definition
Changes on connection targets	Asset-ConectionDestination-Protocol
Changes on software	Asset-SoftwareName-SoftwareVersion

Table 3.2: Attributes used to create the sequences

window size K. Since every single sequence is contained on the tree structure, and a single matching evaluation between unproven and trainer dataset is done by the algorithm itself, then depth $(i = 1, 2, ..., K)$ of a partial short sequence that is contained within the trainer, actually represents the actual matching degree. For instance, taking the short sequence $< a, b, c, d >$, if sub-sequence $< a, b, c >$ is contained on the trainer but the whole $< a, b, c, d >$ is not, then the actual matching degree is $i = 3$ so that it matches in the 75% of the elements and therefore the *Matching value* $[(t_{il})_V) = w + w^2 + ... + w^L]$ defined on the *Tree pattern* algorithm can be calculated with $i = 3$. Thus, the actual implementation defines the threshold $\epsilon = K$.

Furthermore, in order to decrease the false positive rate, the sequences with elements that are not part of the trainer dataset are discarded in such a way that only sequences comprised by *trained elements* are evaluated by the algorithm. In this regard, a support argument to take *untrained* elements out of the detection model is due to the fact that anomalies using unknown attributes can be detected in a simpler ways by using other detection models (e.g. signature based, whitelisting, etc). Thus, detection is focused on misbehavior related to *known* attributes. The actual output of this algorithms is a list of anomalous sequences that afterwards can be correlated to asset security events.

3.7 Functional model definition

This section describes a summary of the general workflow of PNAF in which different types of data are processed to produce the final output of the framework (i.e. a report with information of security events). In fact, the functional model depicted in Figure 3.8 represents an overview of all workflows (full context) performed within sequence diagrams presented in Section 3.3 (Figures 3.3, 3.4, 3.5). The three main modules DCM, DPM and DVM perform *Data collection*, *data processing* and *data visualization* processes respectively, and details of information workflow are described in next subsections.

3.7.1 Data collection

PNAF performs *Data collection* in two main modes: *online* (i.e. real-time network traffic) and *offline* (i.e network traffic capture). Then, taking Figure 3.8 as reference, when *online* analysis is

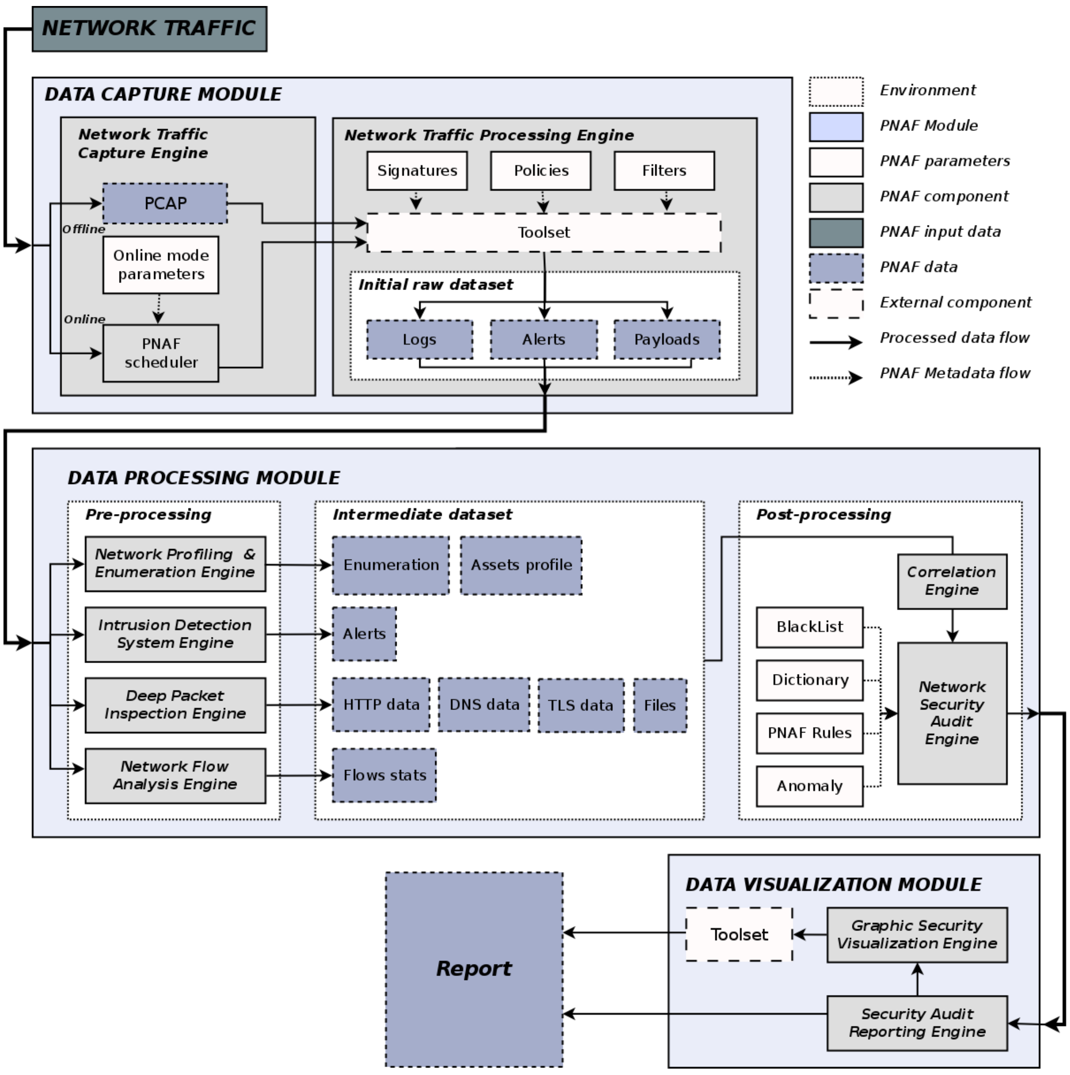


Figure 3.8: PNAF workflow model

needed, *PNAF* uses a *scheduler* that manages the execution of the toolset over the input traffic in such a way that *initial raw dataset* (i.e. *logs, alerts, payloads*) can feed DPM. Otherwise, in *offline* mode, a single traffic capture is passed to the toolset to produce the corresponding *initial raw dataset.* The toolset is comprised by the selection of technologies (described in Table 3.1) and it is controlled by a set of parameters (*signatures, policies, filters*) that define the actual data produced by every single tool. At this point, the *initial raw dataset* contains unprocessed information that, even thought it provides meaningful information by itself (e.g. IDS alerts, list of domains, etc), from PNA perspective it does not provide a meaningful interpretation yet, since it needs to be audited (i.e. check compliance, perform data correlation).

3.7.2 Data processing

DPM receives *initial raw dataset* as input in such a way that pre-processing engines (i.e. *Network Profiling & Enumeration Engine, Intrusion Detection system Engine, Deep Packet Inspection Engine, Network Flow Analysis Engine*) take the corresponding raw logs in order to decode and parse them to extract the *intermediate dataset.* This dataset contains meaningful information that is classified based on *Assets* point of view, thus, a summary of description of the whole environment can be described at this point. In fact, the kind of information that is retrieved is similar that SIEM would receive as input, nevertheless it is not the same context in terms of classification and sorting. For instance, unlike SIEM, (where logs are generated by external tools providing all information of activity such as IDS alerts, Webserver logs, etc.) *intermediate dataset* is comprised by two main subsets focused on assets:

1. Summary subset: It contains the summary of every single attribute found by pre-processing engines. Its purpose is to show the general picture of environment's attributes.
2. Tracking subset: It contains the set of attributes of every single *asset* found by pre-processing engines, hence further tracking investigations (i.e. audits) can be performed in a simple way.

Once *intermediate dataset* is generated, it is ready for *post-processing*, hence it is passed to *Intermediate Data Correlation Engine* to perform data correlation. In order to do so, the so-called PNAF audit data structure is used in such a way that attributes of similar data sources can be described as part of a single *Asset*. Such data structure is depicted in figure 3.9.

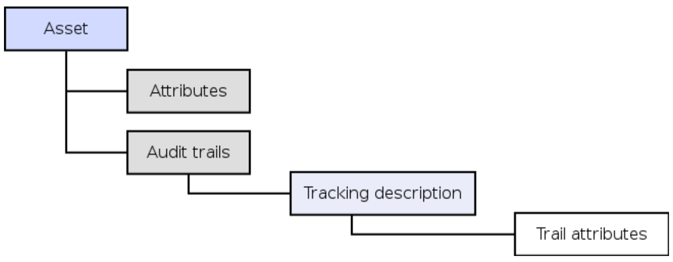


Figure 3.9: *PNAF audit data structure* used for intermediate data correlation

When data correlation is done, information is sent to *Network Security Audit Engine* to perform the core *audit* process, taking into account the four *Audit components* (*Blacklist, Dictionary, Rules and Anomaly*) described in Sections 3.5 and 3.6. As a result, additional attributes are added to this data structure such that correlated data is complemented with audit flags that indicate *Compliance* findings for every single *Asset*.

3.7.3 Data Visualization

The final stage of PNAF workflow is the *visualization output*. Its purpose is to present the findings of the whole audit process. In order to do so, *PNAF audit data structure* is used to feed external visualization tools. The scope of the very first version of the framework includes only a simple visualization through web interface using standard JSON formats. This is done to facilitate the integration with tools such as Kibana[23], Splunk[78], etc, as well as other suites focused on visualization of security events such as the ones included in Davix [64] distribution.

3.8 Remarks

In regards with framework's features related to the research question, this section summarizes and present some remarks about the proposed design:

- *Streamlined process*: The whole process is simplified in general terms. It has a modular design which performs serialized tasks in order to collect, process and produce information through a simple workflow. Thus, this suggest the fact that an auditor may be able to perform security audits over network traffic in a straight forward way.
- *Effectiveness*: An effective way to perform security audits is implied by the fact that a selection of specific-purpose tools are intended to be used within the framework. This not only means that only required specific information is processed, but also that such processes

are performed by different engines that are specifically designed to execute specific pre-processing, post-processing, audit and visualization tasks.

- *Flexibility*: The modular design for the framework provides flexibility to add further capabilities. This can be done either by adding additional modules within the pre-processing phase, or by improving existing modules by adding additional *parsers*. The current design mainly suggest *parsers* for some application layer protocols (i.e. HTTP and DNS) and specific information from protocols of lower layers such as Internet Protocol (IP), TCP and User Datagram Protocol (UDP) for flow analysis. Nevertheless, for instance, if DPI module needs to extract specific information for a new protocol, then a new *parser* can be attached, which can handle datasets that meet specific formats supported by the framework itself (e.g. CSV, JSON). In fact, it is important to emphasize the fact that additional capabilities depends on two different aspects: On the one hand, the external tool that is used to retrieve datasets. On the other hand, the kind of information that need to be parsed.

Thus, the aforementioned features are intended to provide ease for practical implementations. In order to find whether this can be done in a feasible way, next Chapter will address a practical implementation which will be analyzed and validated.

Chapter 4

Implementation and Validation

4.1 Overview

This Chapter describes the implementation and validation of the proposed framework. In order to identify the strengths and the weaknesses of the aforementioned framework's design, a set of tests were performed in such a way that findings can provide evidence to argue whether the intended goals are achieved and to what extent. Moreover, findings are intended to provide support arguments in order to answer the research questions.

4.2 System overview

PNAF implementation has been written in Perl programming language [63] and it has been designed as a functional programming scheme. Moreover, since one of the purposes of the framework is to provide processing flexibility (through an API), then PNAF modules (Figure 3.8) were implemented as their corresponding *Perl modules*, which are independent piece of code that can be correlated and integrated as part of the whole functional model. PNAF has been implemented in Perl programming language due to the following reasons:

- *Easily extensible*: Allows to provide reusable, independent and extensible code that can be used for specific tasks in PNAF.
- *Powerful regular expressions engine*: Since many parsing processes are involved in PNAF, Perl provides powerful capabilities to use regular expressions in order to parse and filter data, taking into account aspects such as performance and flexibility.
- *Text manipulation*: Since DPI is involved in PNAF, a big amount of text data need to be manipulated. Thus, flexible integration of libraries part of the public Perl API, Comprehensive Perl Archive Network (CPAN) [17], can be done to facilitate data extraction tasks.

4.3 Case Study

In order to evaluate the Framework's implementation, a *sase study* has been performed. The running environment includes network traffic samples recorded from a large network environment. This input dataset contains all activities performed within the network traffic. Thus, it is intended to achieve PNA goals taking into account how information is retrieved in order to evaluate whether in a real-world example, PNAF design is indeed suitable to be used as PNA technology. This will provide support arguments to identify possible weaknesses and strengths that may answer research questions as well.

4.3.1 Proof of concept setup

The setup for the Case Study is comprised by the following components. Table 4.1 shows both, physical (server) and logical components (host software) that were used to run the experimental test. Moreover, Table 4.2 presents the input dataset for the experiment that includes a set of network traffic samples (capture files) of different sizes. The purpose of having such an input dataset is to test the behavior, performance and actual output of PNAF, taking into account criteria such as number of packets, type of traffic, protocols, etc. In fact, this set of input samples is comprised by a set of chunks of captures extracted from the biggest sample (i.e. *Capture 10*). Furthermore, Table 4.3 presents the components that define audit baselines used to compare against audit trails extracted from network traffic (e.g. IDS alerts, Flows, Software, Domains, URLs, etc.)

Component	**Type**	**Purpose**	**Characteristics**
Server	Physical	Framework execution	Dell Inc. PowerEdge R510: Intel(R) Xeon(R) CPU E5620 2.40GHz (16 cores), 32Gb RAM.
Operating system	Logical	Manage and host the whole running environment	Gentoo Linux x86_64 hardened/linux/amd64/no-multilib
Perl environment	Logical	Provide the programming language for practical implementation	Perl 5, version 12, subversion 4 (v5.12.4)
PNAF	Logical	Framework implementation	PNAF v 0.1.0

Table 4.1: Case Study. Logical and physical infrastructure

Traffic Sample	**Size (Mb)**	**Millions of packets**	**Duration (seconds)**
Capture 1	3,303.06	6	1,135.20
Capture 2	6,306.58	12	2,521.89
Capture 3	9,155.64	18	4,506.91
Capture 4	11,665.39	24	8,275.16
Capture 5	14,110.29	30	10,870.14
Capture 6	16,527.02	36	11,227.89
Capture 7	18,919.26	42	14,721.09
Capture 8	21,966.28	48	17,450.08
Capture 9	24,594.71	54	19,684.57
Capture 10	26,988.89	60	24,301.63

Table 4.2: Case Study. Input datasets (sample captures) characteristics

Audit component	**Description**	**Details**
PNAF Dictionary	Vulnerability database from National Vulnerability Database [60] publised by NIST[59]. It contains a list of identified vulnerable products and their corresponding versions, classified by Common Vulnerabilities and Exposures (CVE).	CVE entries: 22,121. Vulnerable products: 9,792. Vulnerable versions: 81,308
PNAF Blacklist 1 (IP Reputation)	Reputation database [83] of blacklisted IP addresses, scored according with the type of threat (category).	Scored blacklisted IP addresses: 928,965. Number of categories: 33
PNAF Blacklist 2 (Domain Reputation)	Reputation database [83] of blacklisted domains, scored according with the type of threat (category).	Scored blacklisted domains: 14,294. Number of categories: 33
PNAF Rules	Rules to define audit policies to check compliance.	Number of rules: 10

Table 4.3: Case Study. Audit components

Table 4.4 shows tools distribution and their corresponding versions. In the case of IDS, it shows the number of rules that were used to detect intrusions alerts as audit trails. It has to be noted that IDS signatures were taken from Snort VRT[70] and Emerging Threats [82] rulesets.

Tool	**Version**	**Additional description**
argus	3.0.6	none
bro	2.3	none
cxtracker	github_52318e60d5	none
httpry	github_7dc427196a	none
nftracker	github_c9a920c324	none
p0f	3.06b	none
passivedns	github_fe8f48a3c0	none
prads	github_3c751c869e	none
snort/OpenAppId	2.9.7.0/2014-05-30.205-0	Number of detectors: 2,196
snort IDS	2.9.6.1	Number of signatures: 21,592
ssldump	0.9b3	none
suricata	2.0.2	Number of signatures: 36,351
tcpdstat	github_be5bd28da	none
tcpflow	1.3/1.4.4	none
tcpxtract	1.0.1	none

Table 4.4: Case Study. Network analysis tools

Finally, Table 4.5 shows how datasets were defined to perform anomaly detection over audit trails. Around 20% of the total number of audit trails was taken for the *Trainer* dataset, which is assumed to contain only *normal* trails in such a way that, according the simple algorithm presented in Section 3.6, certain behaviors might be identified. *Unproven* dataset contains audit trails to evaluate. For the same dataset, value of K was variable (i.e. $k = \epsilon = 3$, $k = \epsilon = 6$ and $k = \epsilon = 9$) to check possible differences on the output since bigger k might include longer descriptions paths (*treepatterns*) of anomalous events.

Dataset	**Trainer (audit trails)**	**Unproven (audit trails)**
Change on connections targets	200,000	1,047,433
Change on software	2,000	10,394

Table 4.5: Case Study. anomaly detection dataset

4.3.2 Findings

Overview

Table 4.6 describes some details about specific features that were evaluated on the current implementation.

Feature	**Observations**
Period of analysis	PNAF provided flexibility to specify and report a fixed period of time for the auditing process
Talkers interaction	PNAF provided information about the type of interaction (i.e. flow direction) between talkers within the network. Such an interaction was defined by classifying assets according with the environment they belong to. This means that assets are grouped either within *Internal* or *External* networks. Thus possible interactions can be *Internal-Internal*, *Internal-External*, *External-Internal* and *External-External*
General Bandwidth Stats	PNAF provided information about *Bandwidth usage*, generating accurate stats due to data correlation and the use of multiple tools (e.g argus, tcpdstat, capinfos).

Assets distribution - (Bandwidth)	PNAF provided information about *Assets distribution* classifying by Bandwidth, number of packets, Flow counters, percentage of usage and role identification. This information is assumed to be accurate due to data correlation and the use of multiple tools (argus, tcpdstat, capinfos) that output similar results.
Assets IP sharing behaviour	PNAF provided information about assets that were identified to be either a proxy server or possible NAT gateways with IP sharing. This was detected by using datasets of tools such as p0f and prads and some suricata rules. These tools looked for patterns in which, for instance, a certain IP address had different attributes over the time (e.g. MTU, Window, OS) which meant that potentially other IP addresses were behind of such IP.
Network Link type distribution	PNAF provided information about the type of network links based on datasets generated by different tools (e.g. p0f, prads).
Protocol distribution (Bandwidth)	PNAF provided accurate information of Protocol distribution classifying by Bandwidth,number of packets, flows counters and percentage of usage. This information is assumed to be accurate due to data correlation and the use of multiple tools datasets (e.g. argus, tcpdstat) with same resulting values.
Data correlation	PNAF used data correlation from multiple datasets, which could filter similar information.
Events classification	PNAF took advantages of IDS and DPI engines such as Suricata and Bro. Thus, event classification was based on the internal classification that these tools perform and it provided a picture of network activities related to anomalous or malicious patterns (misused-based).
Events tracking	PNAF took events datasets generated by IDS and DPI engines to process and classify event information. This allowed to track events information in depth, including data of all network layers of TCP/IP model.
DPI over HTTP	PNAF DPI engines provided information about application data from HTTP communications such as URL's, User Agents, Methods, protocols, status codes and response phrases.
DPI over TLS	PNAF DPI engines provided information about unencrypted application data from TLS communications such as certificates and fingerprints.
DPI over DNS	PNAF DPI engines provide information about application data from DNS communications such as Domains, TTL, query types, etc. In the case of Domains, PNAF not only presented domains distribution, but also it performed post-processing tasks in order to use such a dataset for audit purposes (i.e. Domain reputation auditing with PNAF Dictionary). This information provided a meaningful context and turned domain information into a more useful dataset.
Software - Operating System distribution	Deeper inspection in PNAF due to data correlation
Software - Platform distribution	PNAF provided a classification of software platforms (e.g. UNIX family, Windows, Apple, etc.) based on OS distribution. This provided a picture of environments used within the network.
Software - Services distribution	PNAF provided information about OS distribution extracted from generated generated by updated versions of tools (e.g. prads, p0f, suricata, bro) which used more and improved signatures for OS detection.
Software - Product versions identification	PNAF presented accurate information on parsing tasks, as well as flexible data extraction from unknown data formats, due to its tokenization process (explained in Chapter 3).
Audit - Assets with vulnerable software	This feature was one of the most useful capabilities of PNAF since it not only represent one of the core ideas behind auditing and its relationship with security compliance, but also because it turned manual audit into an automated process based only on passive analysis. As an actual output, PNAF produced a list of assets that were identified as vulnerable point within the network infrastructure since they were using vulnerable versions, based on the trusted database [60] published by NIST. The scoring process defined by this database allowed to sort assets according with the risk they represent.
Audit - Assets with anomalous activity (IP reputation)	As part of auditing process, PNAF provided a list of assets that were found targeting IP's marked with bad reputation based on a trusted database [83]. This identification was also done when the asset themselves were blacklisted. The scoring allowed to sort the assets based on the risk they represent.
Audit - Assets with anomalous activity (Domain reputation)	As part of auditing process, PNAF provided a list of assets that were found targeting Domains marked with bad reputation based on a trusted database [83]. The scoring allowed to sort the assets based on the risk they represent.

Audit - Assets with anomalous activity (Policies)	PNAF provided information about policy violations based on PNAF rules as baseline and data correlation dataset as input. Since this feature is intended to define specific policies for specific environments, then unlike Dictionary and Blacklist audit engines, this input is not a standard baseline but rather specific list of policies defining expected behaviors. The actual output showed list of assets violating any of defined policies.
Audit - Assets with anomalous activity (IDS events)	Despite the fact that IDS datasets were used for data correlation (used for post-processing tasks), IDS events were also presented in a separate classification to present the list of assets that triggered security alerts.
Audit - Assets with anomalous activity (mining)	PNAF can perform anomaly detection for certain behaviours, however it was very limited. Audit trails contained on *trainer* dataset were not enough. Even though about 20% of the total number of audit trails was considered, features extraction were not proportional. This means that the actual output of such algorithm depends not only on the size of its *trainer*, but also on the distribution of the number of assets that are actually described on it. Thus, anomaly detection turned out to be inaccurate unless a better *Trainer* dataset is used (i.e. more assets are described as well as more number of audit trails for every of them).

Table 4.6: Case study. PNAF features evaluation.

Performance

Since the research question involves effectiveness as one of the features of the framework, then performance needs to be evaluated in order to find whether a practical implementation is feasible using the proposed design. Thus, different tests were performed using a sample dataset comprised by ten captures of variable sizes. The reason to use multiple files of different sizes was to try to find out the behaviour of the Framework, analyzing relatively small and large files. As it was explained before, the biggest file is actually the original dataset, whereas the other nine (i.e. captures 1 to 9) are aggregated chunks of the biggest one containing the n first millions of packets. Moreover, for every test there were two different schemes:

1. *Parallel execution*: PNAF was executed at the same time over every single file, using the same configuration (described on "*Proof of concept setup*" section).

2. *Single execution*: PNAF was executed as a queue of files, thus every single sample capture was analyzed as a unique task on the system, meaning that no additional tasks were executed on the server during this execution time. PNAF configuration was the same for all the captures as well (described on *Proof of concept setup* section).

Figure 4.1 shows the execution times for every single capture in both schemes *Parallel* and *Single* execution. It has to be noted that, taking into consideration PNAF functional model (Figure 3.8), *full execution* includes tasks of DCM (i.e. initial raw dataset from toolset execution), DPM (i.e. intermediate dataset and post-processing) and DVM (i.e. audit reporting). This test showed that, when multiple files need to be analyzed, *Parallel execution* takes more time per single file (in average a 50% more time). Nevertheless, it finished the processing of all capture files within a time equivalent to the one of the biggest file (i.e. 60 millions of packets [24Gb of traffic] in around 5.45 hours), whereas single execution took the sum of all sequential executions (i.e. around 18.74 hours).

Moreover, it has to be emphasized the fact that behaviour of execution time was also different in both schemes. On the one hand, in *Parallel* execution, time increment was not directly proportional to the size of the file, but it was below a linear behaviour. On the other hand, *Single* execution presented a linear trend, slightly below the actual linear reference. These behaviours are presented in Figure 4.2.

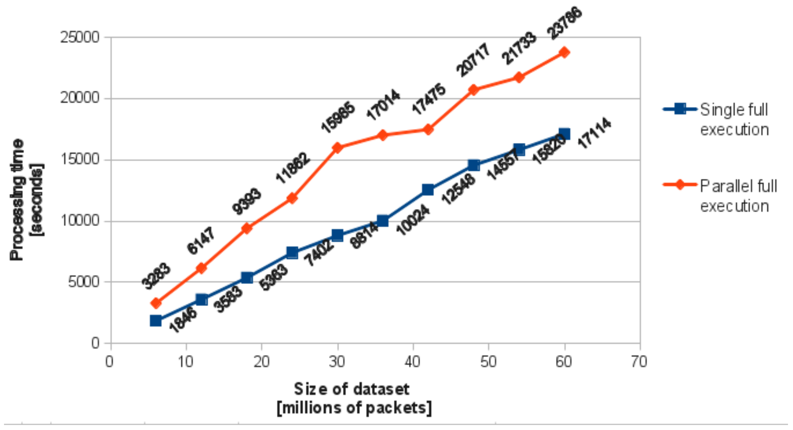


Figure 4.1: Execution times of PNAF. Parallel and Single execution comparison.

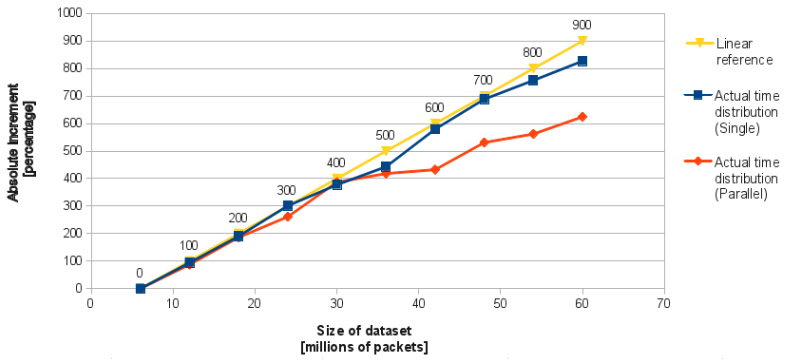


Figure 4.2: Execution times behaviour for Single and Parallel full executions.

PNAF performs different kind of tasks that were independent from each other (nevertheless linked by their input/output datasets according to PNAF functional model). Hence, during the experiment all specific execution times were recorded in order to find possible issues and to define criteria for improvements. Figure 4.3 shows a comparison of execution times between *Instance-only* and *Full execution* processing modes. *Instance-only* processing does not include external toolset execution times, but it includes only PNAF components execution according PNAF functional model (Figure 3.8). Thus, it takes into account DCM (i.e scheduler, parameters loading), DPM (i.e intermediate dataset generation and post-processing) and DVM (i.e. audit reporting) tasks. This basically measures the processing time when initial dataset is already generated, so it is just read as intermediate input dataset.

This test showed that execution of a PNAF instance itself is significantly faster. In fact, on average it took only 29.67% of the whole execution time in *single* mode, whereas in *parallel* it took only 15%. This meant that performance issues were not directly related with actual PNAF tasks themselves, but with the way how initial datasets (i.e. toolset raw logs) are generated by external tools. Figure 4.4 shows relative percentage of time from the whole execution that DPM and DVM take to process all datasets produced by the external toolset (e.g. Snort, Argus, Suricata, Prads, Bro, etc) and to generate the actual output.

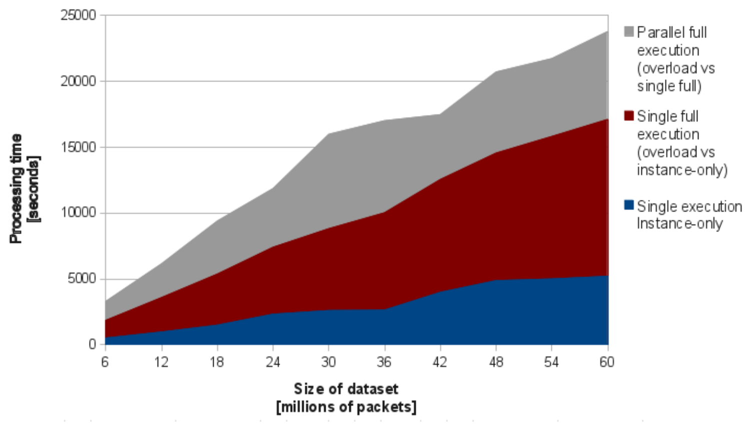


Figure 4.3: Execution times of PNAF. Full and Instance-only execution comparison

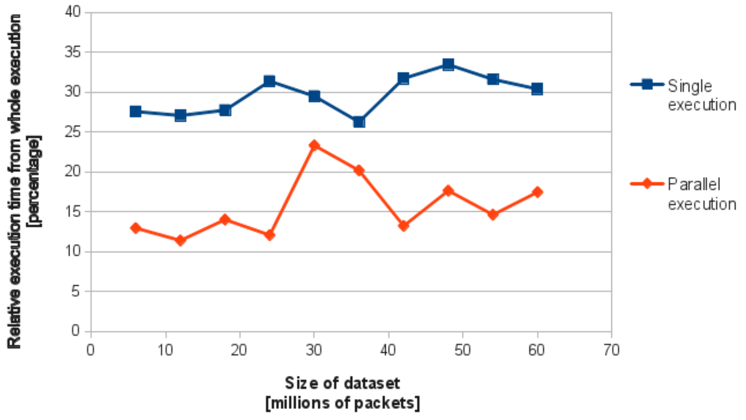


Figure 4.4: Percentage of the whole execution time that PNAF tasks take to process data.

Figure 4.5 shows executions times, emphasizing the ones of core auditing processes (*post-processing*), in which the actual verification and finding of potential anomalies within the network are identified. This graph also shows the difference of times when specific PNAF parameters for the auditing assets are defined (e.g. homenet of assets to audit). This actually found that PNAF parameters for filtering do not affect significantly the performance, since in general terms, execution times were just slightly higher.

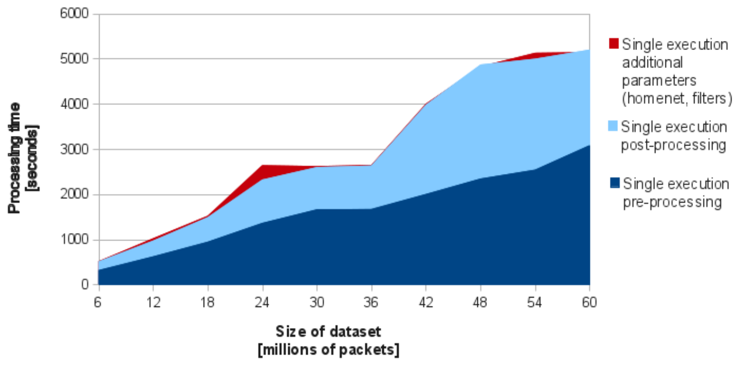


Figure 4.5: Execution times of DPM tasks

Accuracy

Validation of an audit process implies that not only baselines (e.g. trusted sources such as NVD [60]) were known to perform a comparison process based on all information gathered from the raw traffic, but also the fact that previous knowledge about the network was known (e.g. assets subnets, roles, services, etc). This means that, in order to evaluate whether the actual output of the framework is indeed accurate, then previous knowledge about the network was taken into account in order to analyze whether PNAF actual output contained accurate features and issues according to the baselines that PNAF was set to use and the goals that PNAF was intended to achieve.

However, it has to be emphasized the fact that, unlike manual auditing processes, in which security compliance verification can be performed within the full environment to gather real evidence about security events, PNAF auditing process only gathers *relative evidence* from its single input dataset. Hence, there are some assumptions to be considered as part of the validation process:

- All baselines used to perform audit comparisons (i.e. *NVD*, IP and Domain reputation) were considered as trusted base. This assumption was supported by the fact that they all are based on extensive filtering, reputation score and previous knowledge as is explained in their sources [60], [83].

- All information interpreted by PNAF that describes to assets themselves (and not *security events*), represents *valid* evidence since in PNA context, the single input dataset does not contain actual logs, but only network activity used to create equivalent logs. This does not mean validation assumes that actual PNAF output was always accurate, but rather PNAF datasets were considered untrusted (i.e. potential false positives triggering) when assets information was identified from data with unknown or non-standard formats (e.g. datasets different from *Common Log Format* [RFC 1413] or *User Agent format* [RFC 2616]). In this sense, three main classification were defined: (1) *Trusted*: Data meets standard formats, (2) *Untrusted*: completely unstructured formats and (3) *Partially trusted*: format is partially standard.

- From the *Passive* point of view, there is no direct way to perform an actual full validation of findings of the framework, since it would imply to check every single asset found within the network. Even for small environments, hundreds of single assets should be verified either manually or in an automated way, which is also out of the PNA scope since it would involve *active* tasks (e.g. interaction with assets, queries, software verification, additional traffic, etc), as well as time consuming and even unfeasible or inconvenient tasks.

Thus, taking into account the aforementioned assumptions, false positives might be triggered either during log generation (by external toolset) and the tokenization process (processing unknown formats). Nevertheless, since the tokenization process involves generation of all possible combinations of unknown strings (under basic criteria defined on Algorithm 2), then straight forward fixes, based on whitelisting, could be defined to improve the accuracy for this well known issue. This will be explained in the last Chapter (Further Work). Table shows a general evaluation of the actual PNAF output, describing the number of trusted, untrusted and false positives context. A summary about these consideration and its relationship with the PNA context in general will be tackled in the last Chapter as well (Discussion section).

Dataset	**Total identified IDs**	**Trusted**	**Partially trusted**	**Untrusted**	**False positives Number**	**False Positives Description**
Software	1062	500	120	442	6	Invalid combination of product name and version that were formed from data correlation and tokenizer and that was marked to be vulnerable according to *NVD* baseline.
Assets	651	211	384	56	59	Assets linked with CVEs triggered from invalid products or reputation IP that was retrieved from correlation stage and that were marked as vulnerable according to the accumulated total score of their vulnerable products and the threshold set as audit parameter.

Table 4.7: Case Study. Accuracy evaluation

Finally, as it was explained on Table 4.6, anomaly detection presented poor detection results due to the lack of assets features described within the trainer dataset. In practice, PNAF was able to identify the same number of anomalies in all cases for $k = 3, k = 6, k = 9$, however such a detection was not completely accurate since only 1,694 out of 5,647 assets were partially described for *Connection targets*, whereas only 577 out of 2,242 for *Software changes*. Consideration about this will be summarized in last Chapter (Discussion Section).

Dataset	**Assets described on Trainer**	**Actual Anomalies Identified**	**Threshold**
Changes in Software	577	2	3,6,9
Changes in Connection targets	1694	1	3,6,9

Table 4.8: Case Study. Accuracy evaluation

4.4 Discussion

Experiments showed that PNAF was able to analyze data in a passive way with good results, however different kind of issues and considerations were identified in order to deploy such a PNA technology. It has to be mentioned that, in fact, some of them are related with PNAF design itself, whereas others are involved with the practical implementation. In summary, aspects to take into account are:

1. *Reliability*: Unlike similar technologies such as SIEM, PNA has no actual information generated as *trusted* data sources (i.e. actual raw logs from tools), but rather the fact that they have to be generated from raw network traffic (e.g. through DPI), means that reliability is not the same. This actually implies additional tasks and intermediate validation processes (e.g. tokenization and terms whitelisting) that would allow to produce more reliable data. Thus, it has to be emphasized that one of the core assumptions of PNA, as actual *passive* technology with single input data, is to have reliable data in regards with network profiling and enumeration (that would be used to identify security issues). This data is then

interpreted and validated to some extent using baselines (e.g. standard formats, nature of signatures, etc).

2. *Accuracy*: In order to measure the accuracy of a PNA system, different aspects should be taken into account. False positives can be categorized according the stage they might be triggered on (i.e. intermediate dataset or post-processing). On the one hand, all initial security datasets (e.g. IDS alerts, weird patterns, etc) involve certain level of uncertainty about whether all triggered events indeed belong to anomalous or even malicious activities. This is due to the fact that PNA depends partially on how external tools are fed with specific detection parameters (e.g. *VRT* rules, community rules, unreliable OS identification signatures, etc), then it represents an additional layer to be validated and controlled to avoid false positives on intermediate input data. On the other hand, post-processing stage might trigger false positives due to unreliable initial datasets either from previous false positives or from *Untrusted* data that is handled during this post-processing stage (i.e. generating tokenizer false positives).

3. *Performance*: Since PNA technology involves different tasks to capture, decode, filter, compare and produce actual security interpretations, then a set of performance issues has to be tackled to deploy an effective analysis technology (i.e feasible to be used within network environments regardless their size). First, the actual performance problem is mostly related with the way how initial datasets are produced (i.e. raw data logs from external toolset). For instance, with big sample captures, network traffic needs to be read multiple times depending the number of external tools are used within the framework. In addition, reading may take more or less time depending on the tool.

   Moreover, another big problem is the amount of input/output operations performed within the system (in which hard-disk read/write operations are more expensive than CPU/Memory transactions). In fact, the reason why during experiments all parallel schemes presented unstable behaviours in execution times, was due to the fact that expensive read/write operations were performed at the same time, decreasing considerably the efficiency of log generation. Whereas CPU/Memory task were not affected since the system had multiple cores (16 single cores) and relatively big amount of RAM (32Gb), so that it was possible to handle single tasks per core. Nevertheless, even in machines with much less resources, efficiency would be still better as soon they have multiple cores (the more cores the better). Thus, in order to solve performance issues, a set of proposal are tackled in next Section (Further Work).

4. *Feasibility*: One the one hand, for small environments, capabilities such as the ones proposed on the current implementation of PNAF are quite useful to deploy a fully automated audit system. In fact, even taking into account relatively low performance in case of limitations on processing resources, the information that PNAF would provide represent a useful dataset about security events within the network environment.
   On the other hand, in order to turn the framework to be feasible to use in production systems with huge amount of traffic, the main problem to solve is the performance issue, since in general terms, the picture about security auditing is useful even for decision making processes, however, PNA can be combined with NSM and SIEM systems.

5. *Additional features*: Experiments found that anomaly detection approaches on PNA depends on the kind of patterns that are intended to be identified, as well as how meaningful trainer datasets are defined (regardless their size). In this regard, PNA processes normally are not intended to use formal trainer datasets unless an additional and independent process is done. For instance, some security technologies such as IPS, provide optional features such as *learning* capability (i.e. process in which traffic is recorded to identify patterns), that is intended to create profiles and *trainer sets* to improve the accuracy of the system. In the same sense, in order to create meaningful and useful trainer datasets from network traffic in

PNA, datasets with normal events and expected behaviours should be created beforehand based on network traffic but specially handled so that they contain indeed assets descriptions.

6. *Validation*: This process is in practice very difficult to perform. For instance, in small environments a testing framework could be done in such a way that vulnerabilities and other issues reported by PNAF can be checked directly on the assets themselves (checking whether indeed OS, software, system logs, etc, matches with PNA findings). However, even if such a test shows that all reported findings are accurate, it might not be the same behaviour on different testing environments (e.g. large networks in which is not possible to validate thousands of single machines), since a lot of the information depends on the kind of protocols, applications, systems, etc. Thus, the best way to perform validations processes is by taking assumptions and analyze data based on trusted baselines (e.g. trusted sources, standard formats, very specific proven signatures for enumeration processes, etc).

Chapter 5

Final Remarks

5.1 Further Work

Taking into account experiment findings, the following tasks can be done to achieve PNA goals through an improved framework:

- *Performance*: Improve DCM in such a way that initial dataset can be retrieved without the need of multiple reads of raw traffic, as well as improve DPM to avoid read/write operations as possible. This could be done by turning input data set to be raw traffic replayed through network interface and gathered directly by external tools. That means tools would actually listen on the network interface to generate initial dataset at the same time. In addition, as performance test showed about parallel processing and its better performance for multiple files, then a complete parallel execution environment (e.g. using threads) can be deployed in such a way that expensive operations (i.e. hard disk read/write) are avoided as much as possible (e.g. by using local UNIX sockets or pipes instead or temporal log files). This might turn parallel execution faster not only as a whole process, but also for single files (which does not happen in the current implementation).

- *Baselines*: All criteria to filter, control and evaluate information within the framework can be improved. For instance, in order to avoid false positives in post-processing phase, whitelisting of terms retrieved from tokenization can be used to reduce or limit invalid entries. In addition, only well known and trusted rulesets (e.g. from *VRT* and *Emerging threats*) can be used to control false positives during initial dataset generation. Nevertheless, rule verification should be implemented as formal task within the framework.

- *Anomaly features*: Although misused-based is more suitable for auditing purposes, there are some issues that can be tackled in further implementations. On the one hand, there is a need of an improvement of the way how trainer dataset is created. It is necessary to use actual trainer datasets that describe enough information about assets that are being audited. On the other hand, the algorithm used to detect anomalies can be replaced by supervised machine learning techniques, however, from the audit point of view trainer dataset is still the most important key since assets need to be described based on audit trails.

5.2 Conclusions

Taking into consideration existing taxonomies (describing standard features of detection frameworks), existing frameworks (with related PNA capabilities) and results obtained from experiments using the proposed framework, then actual findings of this research suggest that in order to extend in the best way both NSM and SIEM capabilities within a PNA approach, the following aspects should be consider:

- The use of either NSM, SIEM or PNA technologies depends on the characteristics of the networks as well as monitoring, detection and identification goals that need to be achieved. Thus, in order to provide a feasible model based on passive technology, PNA should be considered taking into account two key factors: the use of single input dataset and its focus on assets.
- Definition of a descriptive language that covers the type of output data that it is aimed to be retrieved. For instance, multiple input parameters that control output behaviour (e.g. policies, dictionaries, blacklists) in such a way that granularity can be as depth as the aimed results (e.g. define event types, role types, asset attributes). In the case of PNAF proposed audit language, it has been proven to be useful to describe detection parameters in order to identify vulnerable assets based on trusted baselines.
- PNA can combine different detection and identification models to create data interpretation about security events. Nevertheless, the point is not to use as many network analysis tools as possible, but rather take into consideration the kind of data that can be retrieved from them to use only either specific tools or specific features of a single tool. In PNA it is better to have real meaningful data to describe assets, such that data correlation from multiple datasets can be performed. Thus, after an extensive analysis of multiple technologies, a specific toolset was defined performing not only capabilities and features comparison, but also defining the role they may play within PNA environments in order to be categorized according the type of information of security events aimed to generate. It has to be noted that even if no data correlation nor audit processes are performed, the proposed toolset is a quite useful to be used as a reference in PNA environments.
- Despite the fact that usually IDS are deployed for threat identification (through alert triggering), as well as SIEM for data correlation, these technologies by themselves, specially IDS, are not enough to get awareness of security events within network environments. It has to be emphasized that IDS provide alerting, but this represents an intermediate stage within the data correlation process, in which different kind of datasets (e.g. logs, data enumeration) are correlated to generate *contexts* of security events. In the case of PNA, *contexts* are defined by audit baselines.
- The main gap found between PNA and SIEM that has to be filled, is the capability of PNA approaches to create reliable datasets. Unlike SIEM systems, which already have *reliable* input dataset (i.e. actual raw logs from different tools), PNA need to take assumptions and perfoms additional validation tasks to produce its actual useful output. In fact, this issue is also related with other challenges of PNA such as good performance to produce datasets (through extensive filtering, parsing, comparison, etc.) in order to be feasible technology in large environments. Thus, if this challenge can be tackled in a better way, PNA can cover significant advantages over SIEM as process, in which only a single input is needed with no additional tasks involved, meanwhile the reliability is considerable acceptable.

Thus, this framework proposes a feasible way to take advantage of certain NSM and SIEM capabilities within PNA context, taking into account the gap between these technologies that is mostly related with *reliability*, and in spite of issues it involves, PNAF represent a quite useful technology for passive security audits.

Bibliography

[1] Ajith Abraham, Crina Grosan, and Yuehui Chen. Evolution of intrusion detection systems. *School of Computer Science and Engineering, Chung-Ang University, Korea*, pages 2–3, 2005. 16

[2] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010. 17

[3] N. Amalio and G. Spanoudakis. From monitoring templates to security monitoring and threat detection. In *Emerging Security Information, Systems and Technologies, 2008. SECURWARE '08. Second International Conference on*, pages 185–192, Aug 2008. 24

[4] James P Anderson. Computer security technology planning study. volume 2. Technical report, DTIC Document, 1972. 1

[5] James P Anderson. Computer security threat monitoring and surveillance. Technical report, Technical report, James P. Anderson Company, Fort Washington, Pennsylvania, 1980. 1

[6] Arcsight. Arcsight esm. http://www8.hp.com/us/en/software-solutions/arcsight-esm-enterprise-security-management/index.html. 7

[7] HP ArcSight. Hp arcsight express siem. http://www.ndm.net/siem/arcsight/arcsight-express. 5

[8] Ofir Arkin. Demystifying the myth of passive network discovery and monitoring systems. 2012. 5

[9] Stefan Axelsson. Research in intrusion-detection systems: A survey, 1998. viii, 13

[10] Richard Bejtlich. *The Tao of network security monitoring: beyond intrusion detection*. Pearson Education, 2004. viii, 3

[11] Richard Bejtlich. *The Practice of Network Security Monitoring: Understanding Incident Detection and Response*. No Starch Press, 2013. 3

[12] Jason Bittel. Http logging and information retrieval tool. https://github.com/jbittel/httpry. 34

[13] James Cannady. Artificial neural networks for misuse detection. In *National Information Systems Security Conference*, pages 443–456, 1998. 16

[14] Carlos A. Catania and Carlos Garca Garino. Automatic network intrusion detection: Current techniques and open issues. *Computers and Electrical Engineering*, 38(5):1062 – 1072, 2012. Special issue on Recent Advances in Security and Privacy in Distributed Communications and Image processing. x, 12, 14, 16, 19

[15] CIDF. Common intrusion detection framework. http://gost.isi.edu/cidf/. viii, 20, 21, 22, 30

[16] CISL. The intrusion detection message exchange format (idmef) rfc 4765. http://gost.isi.edu/cidf/drafts/language.txt. 20

[17] CPAN.org. Pcomprehensive perl archive network. http://www.cpan.org. 51

[18] Kyle Creyts. Tcpextract python. extract files from captured tcp sessions. support live streams and pcap files. https://github.com/faust/tcpextract. 34

[19] Frederic Cuppens, Nora Cuppens-Boulahia, and Thierry Sans. Nomad: A security model with non atomic actions and deadlines. In *Computer Security Foundations, 2005. CSFW-18 2005. 18th IEEE Workshop*, pages 186–196. IEEE, 2005. 25

[20] John Curry. Sancp: Security analyst network connection profiler. http://nsmwiki.org/SANCP. 2

[21] Hervé Debar, Marc Dacier, and Andreas Wespi. Towards a taxonomy of intrusion-detection systems. *Computer and Networks*, 31(9):805–822, April 1999. viii, 1, 11, 12

[22] Renaud Deraison, Ron Gula, and Todd Hayton. Passive vulnerability scanning: Introduction to nevo. *Revision*, 9:1–13, 2003. 7

[23] Elasticsearch. Kibana. a browser based analytics and search interface for elasticsearch. http://www.elasticsearch.org/overview/kibana/. 48

[24] Simson L. Garfinkel Elson J. Tcpflow, tcp/ip packet demultiplexer. https://github.com/simsong/tcpflow. 34

[25] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. The kdd process for extracting useful knowledge from volumes of data. *Commun. ACM*, 39(11):27–34, November 1996. 4

[26] Edward Fjellskal. Connection tracker is a passive network connection tracker for profiling, history, auditing and network discovery. https://github.com/gamelinux/cxtracker. 34

[27] Edward Fjellskal. The network file tracker. https://github.com/gamelinux/nftracker. 34

[28] Edward Fjellskal. Passive real-time asset detection system. http://gamelinux.github.io/prads/. 34

[29] Fox-IT. Protact passive audit. https://www.fox-it.com/en/products/protact/protact-passive-audit/. 7

[30] P. Garca-Teodoro, J. Daz-Verdejo, G. Maci-Fernndez, and E. Vzquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers and Security*, 28(12):18 – 28, 2009. viii, 14, 16, 17, 18, 19, 21

[31] Jonatan Gómez, Dipankar Dasgupta, Olfa Nasraoui, and Fabio Gonzalez. Complete expression trees for evolving fuzzy classifier systems with genetic algorithms and application to network intrusion detection. In *Fuzzy Information Processing Society, 2002. Proceedings. NAFIPS. 2002 Annual Meeting of the North American*, pages 469–474. IEEE, 2002. 16

[32] Brendan Gregg. Open source tool to trace tcp/udp/... sessions and fetch application data from snoop or tcpdump logs. http://www.brendangregg.com/chaosreader.html. 34

[33] Ron Gula. Passive vulnerability detection. *Network Security Wizards*, 9, 1999. 7

[34] Ron Gula. Correlating ids alerts with vulnerability information, 2002. 7

[35] Nick Harbour. Tcpxtract. tool for extracting files from network traffic based on file signatures. http://tcpxtract.sourceforge.net/. 34

[36] IDWG. Common intrusion specification language. http://gost.isi.edu/cidf/drafts/language.txt. 21

[37] IDWG. The intrusion detection exchange protocol (idxp) rfc 4767. http://gost.isi.edu/cidf/drafts/language.txt. 20

[38] IDWG. Intrusion detection working group. http://datatracker.ietf.org/wg/idwg/charter/. 20

[39] Clifford Kahn, Phillip A Porras, Stuart Staniford-Chen, and Brian Tung. A common intrusion detection framework. In *Journal of Computer Security*. Citeseer, 1998. 20

[40] Sandeep Kumar and Eugene H Spafford. An application of pattern matching in intrusion detection. 1994. 16

[41] Anukool Lakhina, Konstantina Papagiannaki, Mark Crovella, Christophe Diot, Eric D. Kolaczyk, and Nina Taft. Structural analysis of network traffic flows. *SIGMETRICS Perform. Eval. Rev.*, 32(1):61–72, June 2004. 19

[42] Wenke Lee, Rahul A. Nimbalkar, Kam K. Yee, Sunil B. Patil, Pragneshkumar H. Desai, Thuan T. Tran, and Salvatore J. Stolfo. A data mining and cidf based approach for detecting novel and distributed intrusions. In Herv Debar, Ludovic M, and Shyhtsun Felix Wu, editors, *Recent Advances in Intrusion Detection*, volume 1907 of *Lecture Notes in Computer Science*, pages 49–65. Springer, 2000. 23

[43] Wenke Lee and Salvatore J. Stolfo. Data mining approaches for intrusion detection. In *Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7*, SSYM'98, pages 6–6, Berkeley, CA, USA, 1998. USENIX Association. 19

[44] Wenke Lee and Salvatore J. Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM Trans. Inf. Syst. Secur.*, 3(4):227–261, November 2000. viii, 22, 23, 30

[45] Wenke Lee, Salvatore J. Stolfo, and Kui W. Mok. A data mining framework for building intrusion detection models. In *IEEE Symposium on Security and Privacy*, pages 120–132. IEEE Computer Society, 1999. 22, 30

[46] Wei Li. Using genetic algorithm for network intrusion detection. In *In Proceedings of the United States Department of Energy Cyber Security Group 2004 Training Conference*, pages 24–27, 2004. 16

[47] Ulf Lindqvist and Phillip A. Porras. Detecting computer and network misuse through the production-based expert system toolset (p-best). In *IEEE Symposium on Security and Privacy*, pages 146–161. IEEE Computer Society, 1999. 16

[48] Guisong Liu, Zhang Yi, and Shangming Yang. A hierarchical intrusion detection model based on the {PCA} neural networks. *Neurocomputing*, 70(79):1561 – 1568, 2007. Advances in Computational Intelligence and Learning 14th European Symposium on Artificial Neural Networks 2006 14th European Symposium on Artificial Neural Networks 2006. 16

[49] LogRhythm. Logrhythm siem. http://www.logrhythm.com/siem-2.0/one-integrated-solution.aspx. 4, 5

[50] Davide Lorenzoli and George Spanoudakis. Detection of security and dependability threats: A belief based reasoning approach. In Rainer Falk, Wilson Goudalo, Eric Y. Chen, Reijo Savola, and Manuela Popescu, editors, *SECURWARE*, pages 312–320. IEEE Computer Society, 2009. 24

[51] Matthew V. Mahoney and Philip K. Chan. Learning rules for anomaly detection of hostile network traffic. In *Proceedings of the Third IEEE International Conference on Data Mining*, ICDM '03, pages 601–, Washington, DC, USA, 2003. IEEE Computer Society. 19

[52] W. Mallouli, F. Bessayah, A. Cavalli, and A. Benameur. Security rules specification and analysis based on passive testing. In *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, pages 1–6, Nov 2008. viii, 25, 26, 30

[53] Tenable Log Management and SIEM. Tenable network security. http://www.tenable.com/solutions/log-management-siem. 4, 5

[54] Guojun Mao, Jing Zhang, and Xindong Wu. Intrusion detection based on the short sequence model. In *Intelligent Control and Automation, 2008. WCICA 2008. 7th World Congress on*, pages 1449–1454, 2008. viii, 43, 44, 45, 46

[55] Tom Michael Mitchell. *The discipline of machine learning*. Carnegie Mellon University, School of Computer Science, Machine Learning Department, 2006. 17

[56] Annie De Montigny-leboeuf and Frdric Massicotte. Passive network discovery for real time situation awareness. In *In Proceedings of the The RTO Information Systems Technology Panel (IST) Symposium on Adaptive Defence in Unclassified Networks*, pages 288–300, 2004. 7

[57] CERT Network Situational Awareness Team (CERT NetSA). Silk, the system for internet-level knowledge. https://tools.netsa.cert.org/silk/. 34

[58] Peng Ning, X. Sean Wang, and Sushil Jajodia. A query facility for common intrusion detection framework. In *In Proceedings of the 23rd National Information Systems Security Conference*, pages 317–328, 2000. 21

[59] National Institute of Standards and Technology. Computer security resource center (csrc). http://csrc.nist.gov/. 52

[60] National Institute of Standards and Technology. National vulnerability database. http://nvd.nist.gov/download.cfm. 52, 54, 58

[61] Open Information Security Foundation (OISF). Open source ids / ips / nsm engine. http://suricata-ids.org/. 2, 34

[62] Animesh Patcha and Jung-Min Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks*, 51(12):3448 – 3470, 2007. viii, 13

[63] Perl.org. Perl programming language. http://www.perl.org. 51

[64] PixCloud. The davix live cd: Security visualization. http://www.secviz.org/node/89. 48

[65] Phillip A Porras and Peter G Neumann. Emerald: Event monitoring enabling response to anomalous live disturbances. In *Proceedings of the 20th national information systems security conference*, pages 353–365, 1997. 15

[66] Phillip A. Porras and Alfonso Valdes. Live traffic analysis of tcp/ip gateways. In *NDSS*. The Internet Society, 1998. 19

[67] Leonid Portnoy. Intrusion detection with unlabeled data using clustering. 2000. 19

[68] Delaware corporation QoSient, LLC. Argus. network audit and real time flow monitor tool. http://qosient.com/argus/. 2, 34

[69] Martin Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX Conference on System Administration*, LISA '99, pages 229–238, Berkeley, CA, USA, 1999. USENIX Association. 16

[70] Martin Roesch and The Snort Team. Snort intrusion detection system. http://snort.org/. 2, 34, 53

[71] Mohsen Rouached and Hassen Sallay. An efficient formal framework for intrusion detection systems. *Procedia CS*, 10:968–975, 2012. viii, 24, 25, 30

[72] M.J. Schultz, Ben Wun, and P. Crowley. A passive network appliance for real-time network monitoring. In *Architectures for Networking and Communications Systems (ANCS), 2011 Seventh ACM/IEEE Symposium on*, pages 239–249, Oct 2011. 29

[73] Matt Shelton. Passive asset detection system. http://passive.sourceforge.net/. 2

[74] Sourcefire. Rna, real time network awareness by sourcefire. http://www.sourcefire.com/products/rna.html. 7

[75] George Spanoudakis, Christos Kloukinas, and Kelly Androutsopoulos. Towards security monitoring patterns. In *Proceedings of the 2007 ACM Symposium on Applied Computing*, SAC '07, pages 1518–1525, New York, NY, USA, 2007. ACM. 24

[76] Georgios P. Spathoulas and Sokratis K. Katsikas. Enhancing {IDS} performance through comprehensive alert post-processing. *Computers and Security*, 37(0):176 – 196, 2013. viii, 27, 28

[77] Paul Spirakis, Sokratis Katsikas, Dimitris Gritzalis, Francois Allegre, John Darzentas, Claude Gigante, Dimitris Karagiannis, P Kess, Heiki Putkonen, and Thomas Spyrou. Securenet: A network-oriented intelligent intrusion prevention and detection system. *Network Security Journal*, 1(1), 1994. 14

[78] Splunk. Splunk. operationl intelligence platform. http://www.splunk.com/. 48

[79] Stuart Staniford, James A. Hoagland, and Joseph M. McAlerney. Practical automated detection of stealthy portscans. *J. Comput. Secur.*, 10(1-2):105–136, July 2002. 19

[80] Bro Development Team. Bro network security monitor. http://www.bro.org/. 34

[81] Tenable. Tenable passive security scanner. http://www.tenable.com/products/passive-vulnerability-scanner. 7

[82] Emerging Threats. Etopen ruleset. http://emergingthreats.net/open-source/etopen-ruleset/. 53

[83] Emerging Threats. Iqrisk rep list actionable threat intelligence. http://emergingthreats.net/products/iqrisk-rep-list/. 52, 54, 58

[84] Alfonso Valdes and Keith Skinner. Probabilistic alert correlation. In *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, RAID 00, pages 54–68, London, UK, UK, 2001. Springer-Verlag. 28

[85] Alient Vault. Alienvault open threat exchange. http://www.alienvault.com/open-threat-exchange. 4, 5

[86] Alient Vault. Open source security information management. http://www.alienvault.com/open-threat-exchange/projects. 4

[87] John Adams. Modified version of Dave Dittrich's tcpdstat project. Tcpdstat. protocol statistics. https://github.com/netik/tcpdstat. 34

[88] Robert (Bamm) Visscher. Sguil: The analyst console for network security monitoring. http://bammv.github.io/sguil/index.html. 2

[89] Michal Zalewski. P0f v3. http://lcamtuf.coredump.cx/p0f3/. 34

Glossary

Alerts Events triggered by an Intrusion Detection System when specific pattern occurs within the network traffic..

Anomalous activity Any activity out of the thresholds or specifications of any kind of communication protocol, information exchange or component interaction within the network platform..

Anomalous component Component related to any anomalous activity..

API Application Programming Interface intended to manipulate and develop additional, specific or personalized features within the framework..

Assets Any component of the network that has a value either for the organization or the network platform itself..

Attack Any event that is intended to infringe the security level of the network platform..

Attacker Any person or entity (i.e. internal or external component) which has been identified as the source of an attack..

Audit policy Process in which a set of policies defined either by the organization or by standard security baselines, are compared against the behaviour of the network traffic in such a way it is possible to find possible policy violations..

Data collection Process in which samples datasets are gathered for further analysis..

Data processing Process in which the collected data is parsed, filtered, classified, etc. in order to create meaningful interpretation taking into account the information security context and security baselines..

Deep Packet Inspection Network traffic analysis technique in which different network layers are decoded, analyzed and interpreted in order to characterize the network activities and extract specific information..

Evidence Any data that can prove or support the interpretation of any event identified and reported within the framework..

Impact analysis Process in which the impact (i.e. risk x vulnerabilites) is calculated taken into consideration findings about vulnerability and risk assessment..

Intrusion Detection System System that inspects the network traffic in order to trigger alerts when a specific pattern is identified..

Malicious activity Any activity out of the scope of the thresholds or specification, with explicitly identified malicious behaviour within the security context defined either by the organization or by any general security reference..

Malicious component Component related to malicious activity..

Malware Any software (i.e. piece of code or binary) that is involved either on attacks or malicious activity..

Network Flow Analysis Network traffic analysis technique in which only the network flows are analyzed in order to generate statistics and patterns identification..

Network platform Set of components that comprises the company's logical environment in which information exchange, data processing or any communication protocol is performed..

Network Security assessment Process intended to determine the status of the network within the information security context..

Passive Network Audit Network auditing process performed in such a way that network traffic is collected without any kind of active interaction with the network components. Thus, no changes take place over communication environment..

Policy violation Any event out or against the policies defined by the organization or a security baseline..

Security level Exposure status of the assets within the network platform within the information security context. It is useful to define the level of compliance of certain policies..

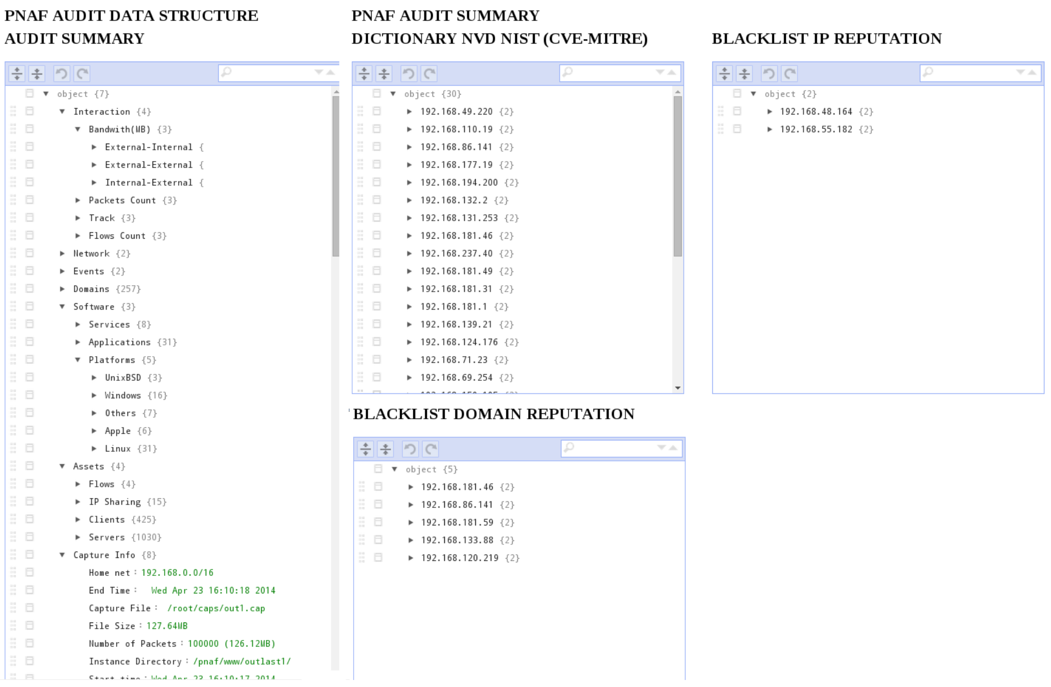
Threat Any event that can cause damage to the network platform. It can be rated as a chance or probability given certain facts such as vulnerabilities, security level, etc..

Vulnerability Any weakness identified within the network platform focused on the information security context. Such a weakness might allow any attacker to reduce the network security level..

Appendix A

Appendix

This appendix shows the actual output of DVM, which includes not only post-processed dataset, but the list of audited assets that were identified with possible anomalous activity or policy violations based on all baselines.



PNAF AUDIT DATA STRUCTURE
AUDIT SUMMARY

```
▼ object {7}
    ▼ Interaction {4}
        ▼ Bandwith(MB) {3}
            ▶ External-Internal {
            ▶ External-External {
            ▶ Internal-External {
        ▶ Packets Count {3}
        ▶ Track {3}
        ▶ Flows Count {3}
    ▶ Network {2}
    ▶ Events {2}
    ▶ Domains {257}
    ▼ Software {3}
        ▶ Services {8}
        ▶ Applications {31}
        ▼ Platforms {5}
            ▶ UnixBSD {3}
            ▶ Windows {16}
            ▶ Others {7}
            ▶ Apple {6}
            ▶ Linux {31}
    ▼ Assets {4}
        ▶ Flows {4}
        ▶ IP Sharing {15}
        ▶ Clients {425}
        ▶ Servers {1030}
    ▼ Capture Info {8}
        Home net : 192.168.0.0/16
        End Time :  Wed Apr 23 16:10:18 2014
        Capture File :  /root/caps/out1.cap
        File Size : 127.64MB
        Number of Packets : 100000 (126.12MB)
        Instance Directory : /pnaf/www/outlast1/
```

PNAF AUDIT SUMMARY
DICTIONARY NVD NIST (CVE-MITRE)

```
▼ object {30}
    ▶ 192.168.49.220 {2}
    ▶ 192.168.110.19 {2}
    ▶ 192.168.86.141 {2}
    ▶ 192.168.177.19 {2}
    ▶ 192.168.194.200 {2}
    ▶ 192.168.132.2 {2}
    ▶ 192.168.131.253 {2}
    ▶ 192.168.181.46 {2}
    ▶ 192.168.237.40 {2}
    ▶ 192.168.181.49 {2}
    ▶ 192.168.181.31 {2}
    ▶ 192.168.181.1 {2}
    ▶ 192.168.139.21 {2}
    ▶ 192.168.124.176 {2}
    ▶ 192.168.71.23 {2}
    ▶ 192.168.69.254 {2}
```

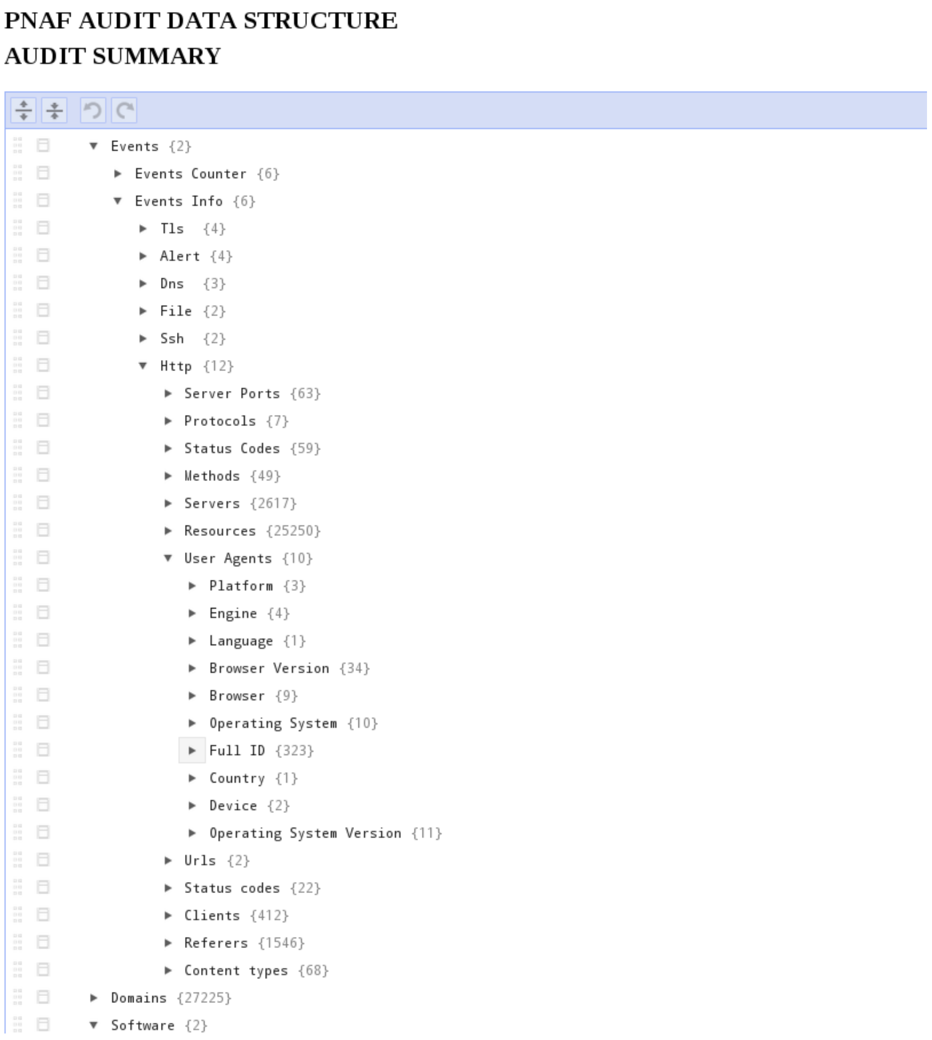
BLACKLIST IP REPUTATION

```
▼ object {2}
    ▶ 192.168.48.164 {2}
    ▶ 192.168.55.182 {2}
```

BLACKLIST DOMAIN REPUTATION

```
▼ object {5}
    ▶ 192.168.181.46 {2}
    ▶ 192.168.86.141 {2}
    ▶ 192.168.181.59 {2}
    ▶ 192.168.133.88 {2}
    ▶ 192.168.120.219 {2}
```

Figure A.1: PNAF Audit Actual Output (Dashboard).

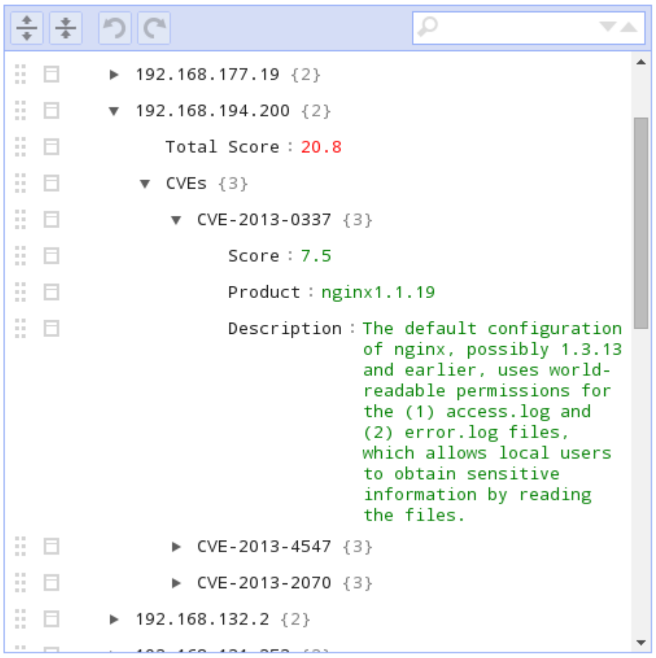


PNAF AUDIT DATA STRUCTURE
AUDIT SUMMARY

```
▼ Events {2}
    ▶ Events Counter {6}
    ▼ Events Info {6}
        ▶ Tls  {4}
        ▶ Alert {4}
        ▶ Dns  {3}
        ▶ File {2}
        ▶ Ssh  {2}
        ▼ Http {12}
            ▶ Server Ports {63}
            ▶ Protocols {7}
            ▶ Status Codes {59}
            ▶ Methods {49}
            ▶ Servers {2617}
            ▶ Resources {25250}
            ▼ User Agents {10}
                ▶ Platform {3}
                ▶ Engine {4}
                ▶ Language {1}
                ▶ Browser Version {34}
                ▶ Browser {9}
                ▶ Operating System {10}
                ▶ Full ID {323}
                ▶ Country {1}
                ▶ Device {2}
                ▶ Operating System Version {11}
            ▶ Urls {2}
            ▶ Status codes {22}
            ▶ Clients {412}
            ▶ Referers {1546}
            ▶ Content types {68}
▶ Domains {27225}
▼ Software {2}
```

Figure A.2: PNAF Audit Data structure visualization.

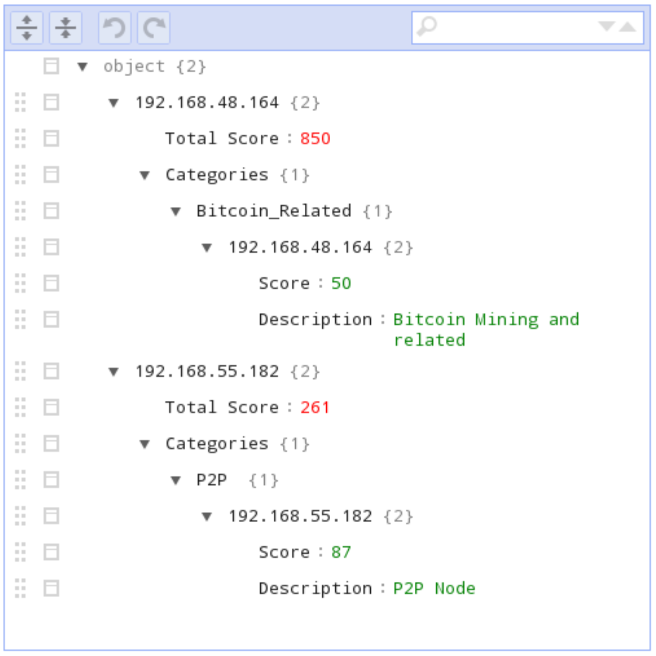
DICTIONARY NVD NIST (CVE-MITRE)



```
▶ 192.168.177.19 {2}
▼ 192.168.194.200 {2}
    Total Score : 20.8
  ▼ CVEs {3}
    ▼ CVE-2013-0337 {3}
        Score : 7.5
        Product : nginx1.1.19
        Description : The default configuration
                      of nginx, possibly 1.3.13
                      and earlier, uses world-
                      readable permissions for
                      the (1) access.log and
                      (2) error.log files,
                      which allows local users
                      to obtain sensitive
                      information by reading
                      the files.
    ▶ CVE-2013-4547 {3}
    ▶ CVE-2013-2070 {3}
▶ 192.168.132.2 {2}
```

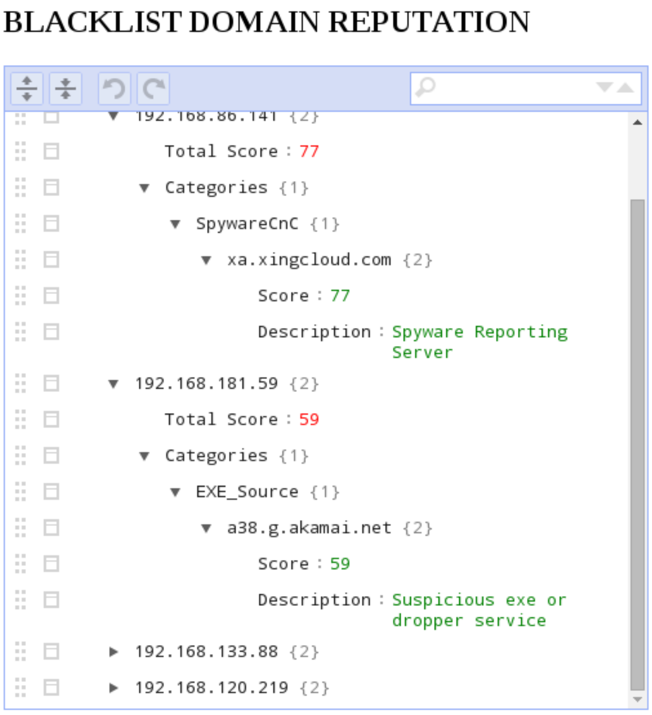
Figure A.3: PNAF Audit Vulnerability Assessment.

BLACKLIST IP REPUTATION



```
▼ object {2}
  ▼ 192.168.48.164 {2}
      Total Score : 850
    ▼ Categories {1}
      ▼ Bitcoin_Related {1}
        ▼ 192.168.48.164 {2}
            Score : 50
            Description : Bitcoin Mining and
                          related
  ▼ 192.168.55.182 {2}
      Total Score : 261
    ▼ Categories {1}
      ▼ P2P {1}
        ▼ 192.168.55.182 {2}
            Score : 87
            Description : P2P Node
```

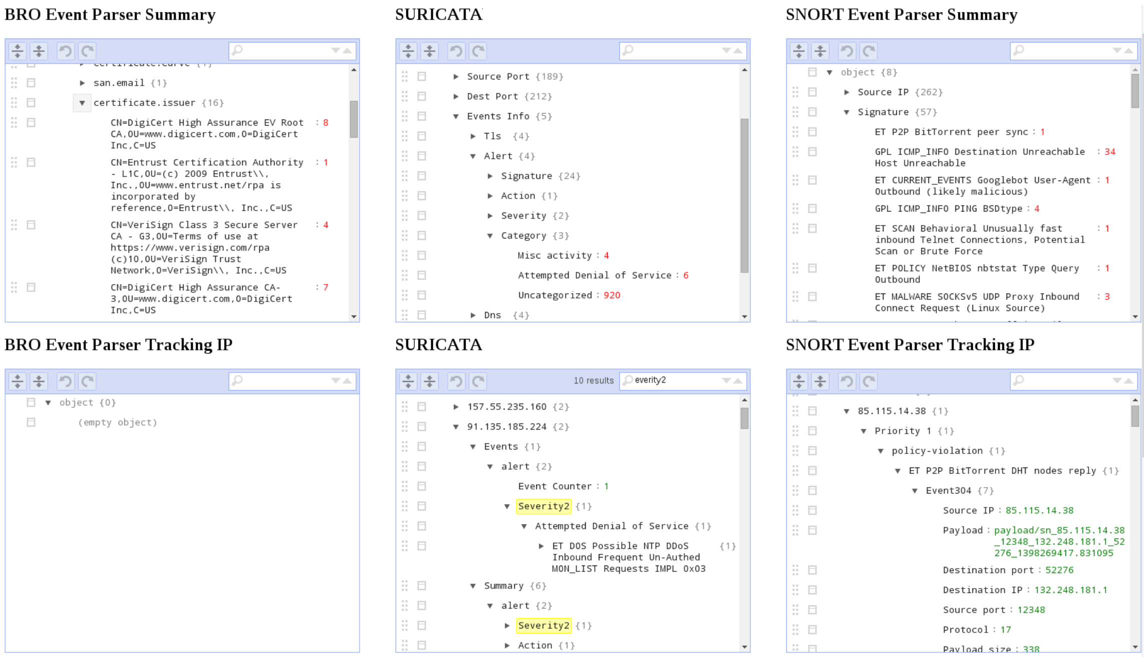
Figure A.4: PNAF Audit Data. Blacklist results. (IP reputation).



BLACKLIST DOMAIN REPUTATION

```
▼ 192.168.86.141 {2}
    Total Score : 77
  ▼ Categories {1}
    ▼ SpywareCnC {1}
      ▼ xa.xingcloud.com {2}
          Score : 77
          Description : Spyware Reporting
                        Server
▼ 192.168.181.59 {2}
    Total Score : 59
  ▼ Categories {1}
    ▼ EXE_Source {1}
      ▼ a38.g.akamai.net {2}
          Score : 59
          Description : Suspicious exe or
                        dropper service
▶ 192.168.133.88 {2}
▶ 192.168.120.219 {2}
```

Figure A.5: PNAF Audit Data. Blacklist results. (Domain reputation).



BRO Event Parser Summary

```
  ▶ certificate.curve {1}
  ▶ san.email {1}
  ▼ certificate.issuer {16}
      CN=DigiCert High Assurance EV Root      : 8
      CA,OU=www.digicert.com,O=DigiCert
      Inc,C=US
      CN=Entrust Certification Authority      : 1
      - L1C,OU=(c) 2009 Entrust\\,
      Inc.,OU=www.entrust.net/rpa is
      incorporated by
      reference,O=Entrust\\, Inc.,C=US
      CN=VeriSign Class 3 Secure Server       : 4
      CA - G3,OU=Terms of use at
      https://www.verisign.com/rpa
      (c)10,OU=VeriSign Trust
      Network,O=VeriSign\\, Inc.,C=US
      CN=DigiCert High Assurance CA-          : 7
      3,OU=www.digicert.com,O=DigiCert
      Inc,C=US
```

SURICATA

```
▶ Source Port {189}
▶ Dest Port {212}
▼ Events Info {5}
  ▶ Tls  {4}
  ▼ Alert {4}
    ▶ Signature {24}
    ▶ Action {1}
    ▶ Severity {2}
    ▼ Category {3}
        Misc activity : 4
        Attempted Denial of Service : 6
        Uncategorized : 920
  ▶ Dns  {4}
```

SNORT Event Parser Summary

```
▼ object {8}
  ▶ Source IP {262}
  ▼ Signature {57}
      ET P2P BitTorrent peer sync : 1
      GPL ICMP_INFO Destination Unreachable   : 34
      Host Unreachable
      ET CURRENT_EVENTS Googlebot User-Agent  : 1
      Outbound (likely malicious)
      GPL ICMP_INFO PING BSDtype : 4
      ET SCAN Behavioral Unusually fast       : 1
      inbound Telnet Connections, Potential
      Scan or Brute Force
      ET POLICY NetBIOS nbtstat Type Query    : 1
      Outbound
      ET MALWARE SOCKSv5 UDP Proxy Inbound    : 3
      Connect Request (Linux Source)
```

BRO Event Parser Tracking IP

```
▼ object {0}
    (empty object)
```

SURICATA

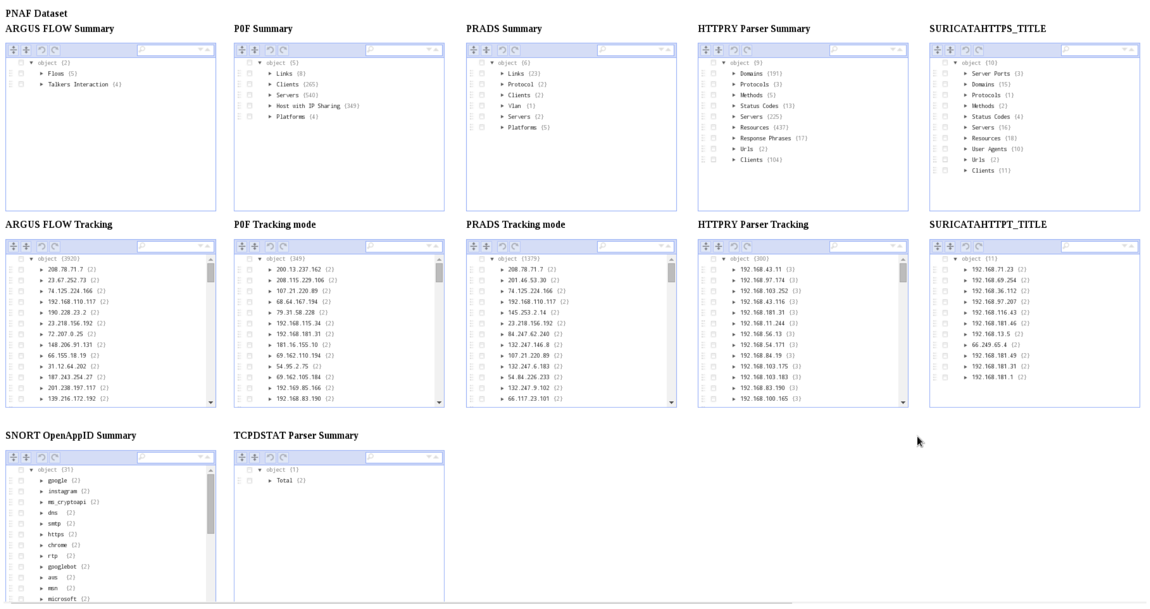
```
10 results   everity2

▶ 157.55.235.160 {2}
▼ 91.135.185.224 {2}
  ▼ Events {1}
    ▼ alert {2}
        Event Counter : 1
      ▼ Severity2 {1}
        ▼ Attempted Denial of Service {1}
          ▶ ET DOS Possible NTP DDoS          {1}
            Inbound Frequent Un-Authed
            MON_LIST Requests IMPL 0x03
  ▼ Summary {6}
    ▼ alert {2}
      ▶ Severity2 {1}
      ▶ Action {1}
```

SNORT Event Parser Tracking IP

```
▼ 85.115.14.38 {1}
  ▼ Priority 1 {1}
    ▼ policy-violation {1}
      ▼ ET P2P BitTorrent DHT nodes reply {1}
        ▼ Event304 {7}
            Source IP : 85.115.14.38
            Payload : payload/sn_85.115.14.38
                      _12348_132.248.181.1_52
                      276_1398269417.831095
            Destination port : 52276
            Destination IP : 132.248.181.1
            Source port : 12348
            Protocol : 17
            Payload size : 338
```

Figure A.6: PNAF Audit Data. IDS Events Dataset.



PNAF Dataset

ARGUS FLOW Summary

```
▾ object {2}
    ▸ Flows {5}
    ▸ Talkers Interaction {4}
```

P0F Summary

```
▾ object {5}
    ▸ Links {8}
    ▸ Clients {265}
    ▸ Servers {540}
    ▸ Host with IP Sharing {349}
    ▸ Platforms {4}
```

PRADS Summary

```
▾ object {6}
    ▸ Links {23}
    ▸ Protocol {2}
    ▸ Clients {2}
    ▸ Vlan {1}
    ▸ Servers {2}
    ▸ Platforms {5}
```

HTTPRY Parser Summary

```
▾ object {9}
    ▸ Domains {191}
    ▸ Protocols {3}
    ▸ Methods {5}
    ▸ Status Codes {13}
    ▸ Servers {225}
    ▸ Resources {437}
    ▸ Response Phrases {17}
    ▸ Urls {2}
    ▸ Clients {104}
```

SURICATAHTTPS_TITLE

```
▾ object {10}
    ▸ Server Ports {3}
    ▸ Domains {15}
    ▸ Protocols {1}
    ▸ Methods {2}
    ▸ Status Codes {4}
    ▸ Servers {16}
    ▸ Resources {18}
    ▸ User Agents {10}
    ▸ Urls {2}
    ▸ Clients {11}
```

ARGUS FLOW Tracking

```
▾ object {3920}
    ▸ 208.78.71.7 {2}
    ▸ 23.67.252.73 {2}
    ▸ 74.125.224.166 {2}
    ▸ 192.168.110.117 {2}
    ▸ 190.228.23.2 {2}
    ▸ 23.218.156.192 {2}
    ▸ 72.207.0.25 {2}
    ▸ 148.206.91.131 {2}
    ▸ 66.155.18.19 {2}
    ▸ 31.12.64.202 {2}
    ▸ 187.243.254.27 {2}
    ▸ 201.238.197.117 {2}
    ▸ 139.216.172.192 {2}
```

P0F Tracking mode

```
▾ object {349}
    ▸ 200.13.237.162 {2}
    ▸ 208.115.229.106 {2}
    ▸ 107.21.220.89 {2}
    ▸ 68.64.167.194 {2}
    ▸ 79.31.58.228 {2}
    ▸ 192.168.115.34 {2}
    ▸ 192.168.181.31 {2}
    ▸ 181.16.155.10 {2}
    ▸ 69.162.110.194 {2}
    ▸ 54.95.2.75 {2}
    ▸ 69.162.105.184 {2}
    ▸ 192.169.85.166 {2}
    ▸ 192.168.83.190 {2}
```

PRADS Tracking mode

```
▾ object {1379}
    ▸ 208.78.71.7 {2}
    ▸ 201.46.53.30 {2}
    ▸ 74.125.224.166 {2}
    ▸ 192.168.110.117 {2}
    ▸ 145.253.2.14 {2}
    ▸ 23.218.156.192 {2}
    ▸ 84.247.62.240 {2}
    ▸ 132.247.146.8 {2}
    ▸ 107.21.220.89 {2}
    ▸ 132.247.6.183 {2}
    ▸ 54.84.226.233 {2}
    ▸ 132.247.9.102 {2}
    ▸ 66.117.23.101 {2}
```

HTTPRY Parser Tracking

```
▾ object {300}
    ▸ 192.168.43.11 {3}
    ▸ 192.168.97.174 {3}
    ▸ 192.168.103.252 {3}
    ▸ 192.168.43.116 {3}
    ▸ 192.168.181.31 {3}
    ▸ 192.168.11.244 {3}
    ▸ 192.168.56.13 {3}
    ▸ 192.168.54.171 {3}
    ▸ 192.168.84.19 {3}
    ▸ 192.168.103.175 {3}
    ▸ 192.168.103.183 {3}
    ▸ 192.168.83.190 {3}
    ▸ 192.168.100.165 {3}
```

SURICATAHTTPT_TITLE

```
▾ object {11}
    ▸ 192.168.71.23 {2}
    ▸ 192.168.69.254 {2}
    ▸ 192.168.36.112 {2}
    ▸ 192.168.97.207 {2}
    ▸ 192.168.116.43 {2}
    ▸ 192.168.181.46 {2}
    ▸ 192.168.13.5 {2}
    ▸ 66.249.65.4 {2}
    ▸ 192.168.181.49 {2}
    ▸ 192.168.181.31 {2}
    ▸ 192.168.181.1 {2}
```

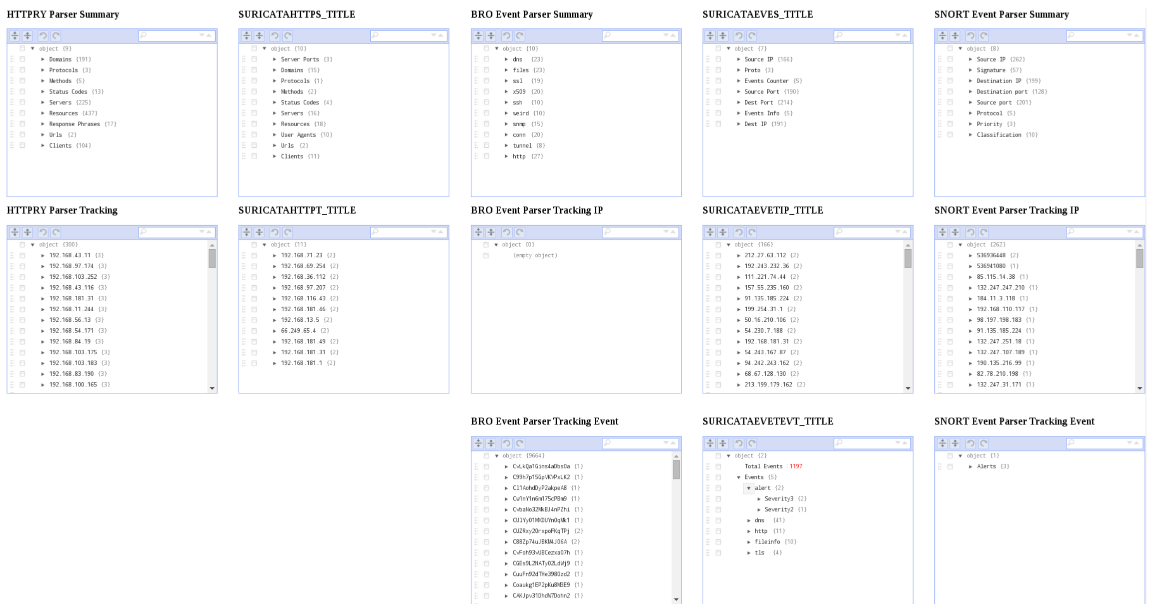
SNORT OpenAppID Summary

```
▾ object {31}
    ▸ google {2}
    ▸ instagram {2}
    ▸ ms_cryptoapi {2}
    ▸ dns {2}
    ▸ smtp {2}
    ▸ https {2}
    ▸ chrome {2}
    ▸ rtp {2}
    ▸ googlebot {2}
    ▸ aws {2}
    ▸ msn {2}
    ▸ microsoft {2}
```

TCPDSTAT Parser Summary

```
▾ object {1}
    ▸ Total {2}
```

Figure A.7: PNAF Tool dataset (group 1).



HTTPRY Parser Summary

```
▾ object {9}
    ▸ Domains {191}
    ▸ Protocols {3}
    ▸ Methods {5}
    ▸ Status Codes {13}
    ▸ Servers {225}
    ▸ Resources {437}
    ▸ Response Phrases {17}
    ▸ Urls {2}
    ▸ Clients {104}
```

SURICATAHTTPS_TITLE

```
▾ object {10}
    ▸ Server Ports {3}
    ▸ Domains {15}
    ▸ Protocols {1}
    ▸ Methods {2}
    ▸ Status Codes {4}
    ▸ Servers {16}
    ▸ Resources {18}
    ▸ User Agents {10}
    ▸ Urls {2}
    ▸ Clients {11}
```

BRO Event Parser Summary

```
▾ object {10}
    ▸ dns {23}
    ▸ files {23}
    ▸ ssl {19}
    ▸ x509 {20}
    ▸ ssh {10}
    ▸ weird {10}
    ▸ snmp {15}
    ▸ conn {20}
    ▸ tunnel {8}
    ▸ http {27}
```

SURICATAEVES_TITLE

```
▾ object {7}
    ▸ Source IP {166}
    ▸ Proto {3}
    ▸ Events Counter {5}
    ▸ Source Port {190}
    ▸ Dest Port {214}
    ▸ Events Info {5}
    ▸ Dest IP {191}
```

SNORT Event Parser Summary

```
▾ object {8}
    ▸ Source IP {262}
    ▸ Signature {57}
    ▸ Destination IP {199}
    ▸ Destination port {128}
    ▸ Source port {201}
    ▸ Protocol {5}
    ▸ Priority {3}
    ▸ Classification {10}
```

HTTPRY Parser Tracking

```
▾ object {300}
    ▸ 192.168.43.11 {3}
    ▸ 192.168.97.174 {3}
    ▸ 192.168.103.252 {3}
    ▸ 192.168.43.116 {3}
    ▸ 192.168.181.31 {3}
    ▸ 192.168.11.244 {3}
    ▸ 192.168.56.13 {3}
    ▸ 192.168.54.171 {3}
    ▸ 192.168.84.19 {3}
    ▸ 192.168.103.175 {3}
    ▸ 192.168.103.183 {3}
    ▸ 192.168.83.190 {3}
    ▸ 192.168.100.165 {3}
```

SURICATAHTTPT_TITLE

```
▾ object {11}
    ▸ 192.168.71.23 {2}
    ▸ 192.168.69.254 {2}
    ▸ 192.168.36.112 {2}
    ▸ 192.168.97.207 {2}
    ▸ 192.168.116.43 {2}
    ▸ 192.168.181.46 {2}
    ▸ 192.168.13.5 {2}
    ▸ 66.249.65.4 {2}
    ▸ 192.168.181.49 {2}
    ▸ 192.168.181.31 {2}
    ▸ 192.168.181.1 {2}
```

BRO Event Parser Tracking IP

```
▾ object {0}
    (empty object)
```

SURICATAEVETIP_TITLE

```
▾ object {166}
    ▸ 212.27.63.112 {2}
    ▸ 192.243.232.36 {2}
    ▸ 111.221.74.44 {2}
    ▸ 157.55.235.160 {2}
    ▸ 91.135.185.224 {2}
    ▸ 199.254.31.1 {2}
    ▸ 50.16.210.106 {2}
    ▸ 54.230.7.188 {2}
    ▸ 192.168.181.31 {2}
    ▸ 54.243.167.87 {2}
    ▸ 94.242.243.162 {2}
    ▸ 68.67.128.130 {2}
    ▸ 213.199.179.162 {2}
```

SNORT Event Parser Tracking IP

```
▾ object {262}
    ▸ 536936448 {2}
    ▸ 536941080 {1}
    ▸ 85.115.14.38 {1}
    ▸ 132.247.247.210 {1}
    ▸ 184.11.3.118 {1}
    ▸ 192.168.110.117 {1}
    ▸ 98.197.198.183 {1}
    ▸ 91.135.185.224 {1}
    ▸ 132.247.251.18 {1}
    ▸ 132.247.107.189 {1}
    ▸ 190.135.216.99 {1}
    ▸ 82.78.210.198 {1}
    ▸ 132.247.31.171 {1}
```

BRO Event Parser Tracking Event

```
▾ object {9664}
    ▸ CvLkQa1Gins4aDbs0a {1}
    ▸ C99h7p15GpVKVPxLK2 {1}
    ▸ C11AohdDyP2akpeA8 {1}
    ▸ Cw1nY1n6m175cPBm9 {1}
    ▸ CvbaNo32MkBJ4nPZhi {1}
    ▸ CU1Yy01MXDUYn0qMk1 {1}
    ▸ CUZRxy20rxpoFKqTPj {2}
    ▸ C88Zp74uJBKM4J06A {2}
    ▸ CvFoh93vUBCezxa07h {1}
    ▸ CGEs9L2NATy02LdWj9 {1}
    ▸ CuuFn92dTHe3980zd2 {1}
    ▸ Coaukg1EP2pKu8M3E9 {1}
    ▸ CAKJpv31DhdW7Dohn2 {1}
```

SURICATAEVETEVT_TITLE

```
▾ object {2}
    Total Events : 1197
    ▾ Events {5}
        ▾ alert {2}
            ▸ Severity3 {2}
            ▸ Severity2 {1}
        ▸ dns {41}
        ▸ http {11}
        ▸ fileinfo {10}
        ▸ tls {4}
```

SNORT Event Parser Tracking Event

```
▾ object {1}
    ▸ Alerts {3}
```

Figure A.8: PNAF Tool dataset (group 2).

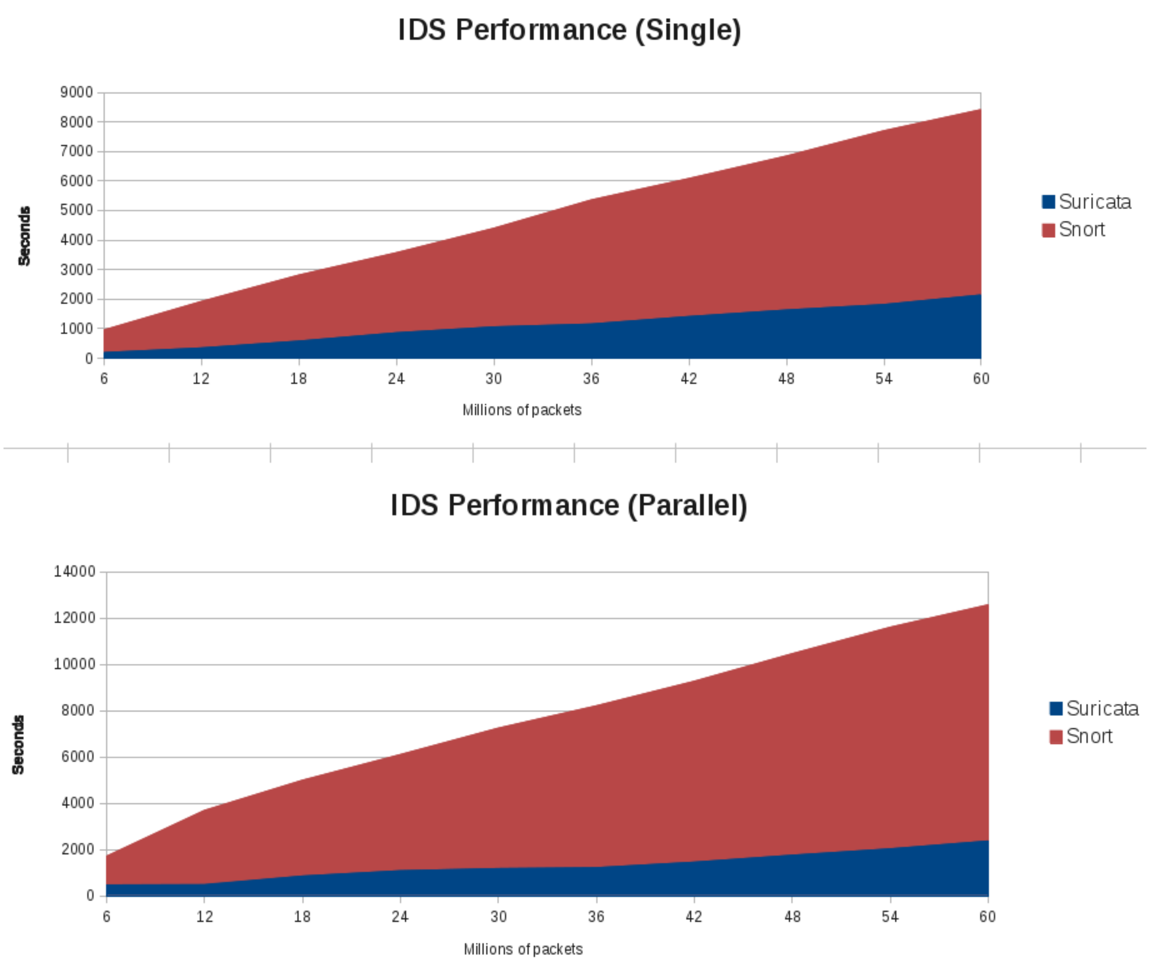
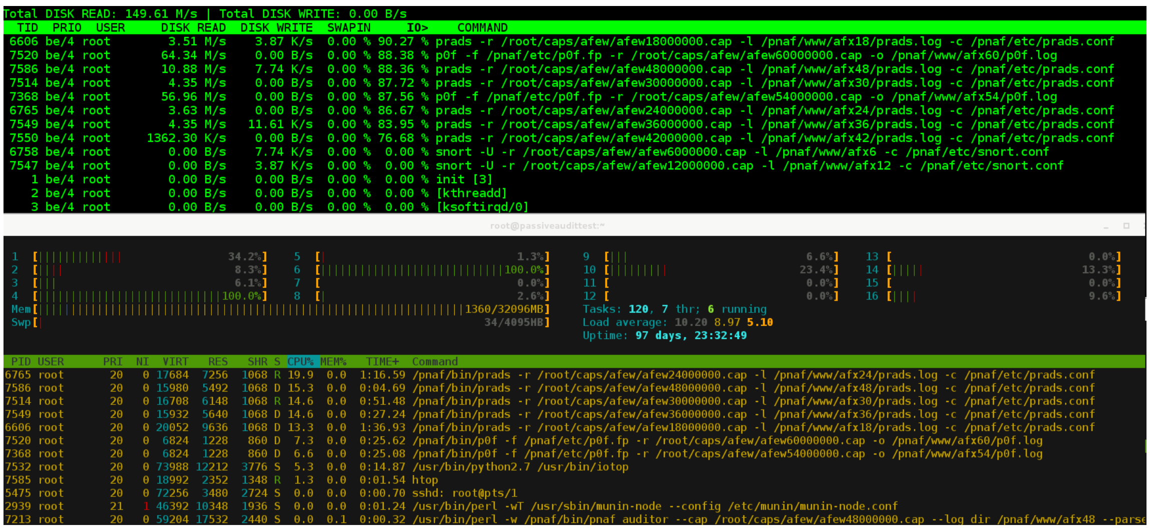


Figure A.9: IDS Performance comparison

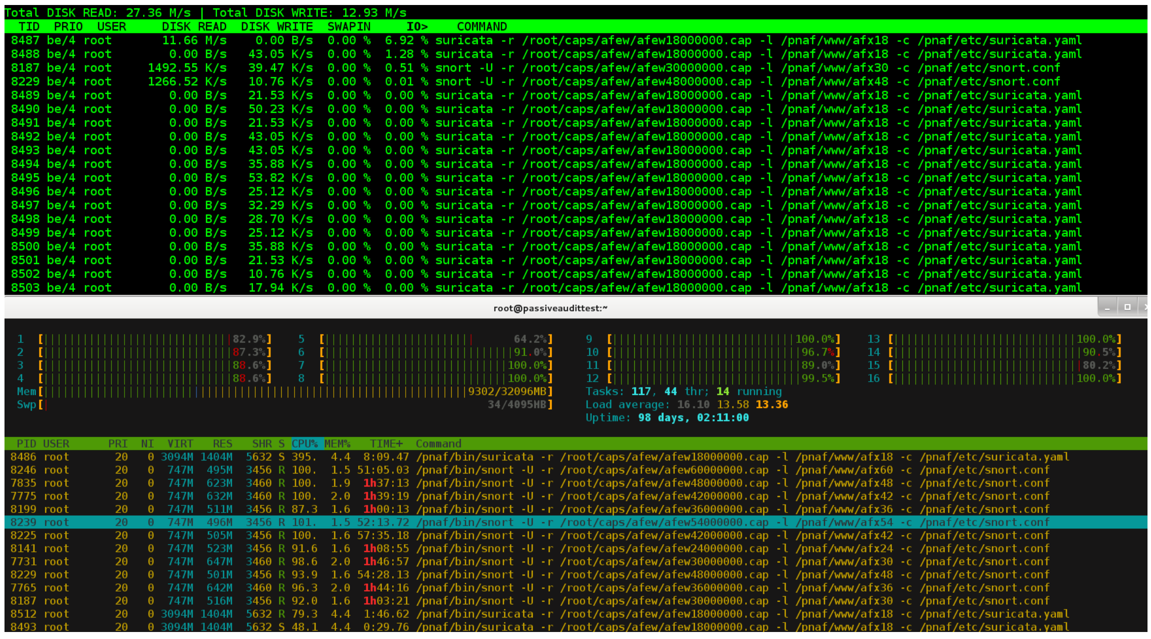


```
Total DISK READ: 149.61 M/s | Total DISK WRITE: 0.00 B/s
  TID  PRIO  USER      DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
 6606 be/4 root        3.51 M/s    3.87 K/s  0.00 % 90.27 % prads -r /root/caps/afew/afew18000000.cap -l /pnaf/www/afx18/prads.log -c /pnaf/etc/prads.conf
 7520 be/4 root       64.34 M/s    0.00 B/s  0.00 % 88.38 % p0f -f /pnaf/etc/p0f.fp -r /root/caps/afew/afew60000000.cap -o /pnaf/www/afx60/p0f.log
 7586 be/4 root       10.88 M/s    7.74 K/s  0.00 % 88.36 % prads -r /root/caps/afew/afew48000000.cap -l /pnaf/www/afx48/prads.log -c /pnaf/etc/prads.conf
 7514 be/4 root        4.35 M/s    0.00 B/s  0.00 % 87.72 % prads -r /root/caps/afew/afew30000000.cap -l /pnaf/www/afx30/prads.log -c /pnaf/etc/prads.conf
 7368 be/4 root       56.96 M/s    0.00 B/s  0.00 % 87.56 % p0f -f /pnaf/etc/p0f.fp -r /root/caps/afew/afew54000000.cap -o /pnaf/www/afx54/p0f.log
 6765 be/4 root        3.63 M/s    0.00 B/s  0.00 % 86.67 % prads -r /root/caps/afew/afew24000000.cap -l /pnaf/www/afx24/prads.log -c /pnaf/etc/prads.conf
 7549 be/4 root        4.35 M/s   11.61 K/s  0.00 % 83.95 % prads -r /root/caps/afew/afew36000000.cap -l /pnaf/www/afx36/prads.log -c /pnaf/etc/prads.conf
 7550 be/4 root     1362.30 K/s    0.00 B/s  0.00 % 76.68 % prads -r /root/caps/afew/afew42000000.cap -l /pnaf/www/afx42/prads.log -c /pnaf/etc/prads.conf
 6758 be/4 root        0.00 B/s    7.74 K/s  0.00 %  0.00 % snort -U -r /root/caps/afew/afew6000000.cap -l /pnaf/www/afx6 -c /pnaf/etc/snort.conf
 7547 be/4 root        0.00 B/s    3.87 K/s  0.00 %  0.00 % snort -U -r /root/caps/afew/afew12000000.cap -l /pnaf/www/afx12 -c /pnaf/etc/snort.conf
    1 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % init [3]
    2 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [kthreadd]
    3 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % [ksoftirqd/0]
                                    root@passiveaudittest:~

  1  [|||||||||||||              34.2%]   5  [|                           1.3%]   9  [|||                         6.6%]   13 [                            0.0%]
  2  [||||                        8.3%]   6  [||||||||||||||||||||||||||100.0%]   10 [|||||||||                  23.4%]   14 [|||||                      13.3%]
  3  [|||                         6.1%]   7  [                            0.0%]   11 [                            0.0%]   15 [                            0.0%]
  4  [|||||||||||||||||||||||||||100.0%]  8  [|                           2.6%]   12 [                            0.0%]   16 [||||                        9.6%]
  Mem[|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||1360/32096MB]   Tasks: 120, 7 thr; 6 running
  Swp[|                                                               34/4095MB]   Load average: 10.20 8.97 5.10
                                                                                   Uptime: 97 days, 23:32:49

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
6765 root       20   0 17684  7256  1068 R 19.9  0.0  1:16.59 /pnaf/bin/prads -r /root/caps/afew/afew24000000.cap -l /pnaf/www/afx24/prads.log -c /pnaf/etc/prads.conf
7586 root       20   0 15980  5492  1068 D 15.3  0.0  0:04.69 /pnaf/bin/prads -r /root/caps/afew/afew48000000.cap -l /pnaf/www/afx48/prads.log -c /pnaf/etc/prads.conf
7514 root       20   0 16708  6148  1068 R 14.6  0.0  0:51.48 /pnaf/bin/prads -r /root/caps/afew/afew30000000.cap -l /pnaf/www/afx30/prads.log -c /pnaf/etc/prads.conf
7549 root       20   0 15932  5640  1068 D 14.6  0.0  0:27.24 /pnaf/bin/prads -r /root/caps/afew/afew36000000.cap -l /pnaf/www/afx36/prads.log -c /pnaf/etc/prads.conf
6606 root       20   0 20052  9636  1068 D 13.3  0.0  1:36.93 /pnaf/bin/prads -r /root/caps/afew/afew18000000.cap -l /pnaf/www/afx18/prads.log -c /pnaf/etc/prads.conf
7520 root       20   0  6824  1228   860 D  7.3  0.0  0:25.62 /pnaf/bin/p0f -f /pnaf/etc/p0f.fp -r /root/caps/afew/afew60000000.cap -o /pnaf/www/afx60/p0f.log
7368 root       20   0  6824  1228   860 D  6.6  0.0  0:25.08 /pnaf/bin/p0f -f /pnaf/etc/p0f.fp -r /root/caps/afew/afew54000000.cap -o /pnaf/www/afx54/p0f.log
7532 root       20   0 73988 12212  3776 S  5.3  0.0  0:14.87 /usr/bin/python2.7 /usr/bin/iotop
7585 root       20   0 18992  2352  1348 R  1.3  0.0  0:01.54 htop
5475 root       20   0 72256  3480  2724 S  0.0  0.0  0:00.70 sshd: root@pts/1
2939 root       21   1 46392 10348  1936 S  0.0  0.0  0:01.24 /usr/bin/perl -wT /usr/sbin/munin-node --config /etc/munin/munin-node.conf
7213 root       20   0 59204 17532  2440 S  0.0  0.1  0:00.32 /usr/bin/perl -w /pnaf/bin/pnaf_auditor --cap /root/caps/afew/afew48000000.cap --log_dir /pnaf/www/afx48 --parse
```

Figure A.10: Hard disk read/write transactions, and CPU core usage during reading phase



```
Total DISK READ: 27.36 M/s | Total DISK WRITE: 12.93 M/s
  TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
 8487 be/4 root       11.66 M/s    0.00 B/s  0.00 %  6.92 % suricata -r /root/caps/afew/afew18000000.cap -l /pnaf/www/afx18 -c /pnaf/etc/suricata.yaml
 8488 be/4 root        0.00 B/s   43.05 K/s  0.00 %  1.28 % suricata -r /root/caps/afew/afew18000000.cap -l /pnaf/www/afx18 -c /pnaf/etc/suricata.yaml
 8187 be/4 root     1492.55 K/s   39.47 K/s  0.00 %  0.51 % snort -U -r /root/caps/afew/afew30000000.cap -l /pnaf/www/afx30 -c /pnaf/etc/snort.conf
 8229 be/4 root     1266.52 K/s   10.76 K/s  0.00 %  0.01 % snort -U -r /root/caps/afew/afew48000000.cap -l /pnaf/www/afx48 -c /pnaf/etc/snort.conf
 8489 be/4 root        0.00 B/s   21.53 K/s  0.00 %  0.00 % suricata -r /root/caps/afew/afew18000000.cap -l /pnaf/www/afx18 -c /pnaf/etc/suricata.yaml
 8490 be/4 root        0.00 B/s   50.23 K/s  0.00 %  0.00 % suricata -r /root/caps/afew/afew18000000.cap -l /pnaf/www/afx18 -c /pnaf/etc/suricata.yaml
 8491 be/4 root        0.00 B/s   21.53 K/s  0.00 %  0.00 % suricata -r /root/caps/afew/afew18000000.cap -l /pnaf/www/afx18 -c /pnaf/etc/suricata.yaml
 8492 be/4 root        0.00 B/s   43.05 K/s  0.00 %  0.00 % suricata -r /root/caps/afew/afew18000000.cap -l /pnaf/www/afx18 -c /pnaf/etc/suricata.yaml
 8493 be/4 root        0.00 B/s   43.05 K/s  0.00 %  0.00 % suricata -r /root/caps/afew/afew18000000.cap -l /pnaf/www/afx18 -c /pnaf/etc/suricata.yaml
 8494 be/4 root        0.00 B/s   35.88 K/s  0.00 %  0.00 % suricata -r /root/caps/afew/afew18000000.cap -l /pnaf/www/afx18 -c /pnaf/etc/suricata.yaml
 8495 be/4 root        0.00 B/s   53.82 K/s  0.00 %  0.00 % suricata -r /root/caps/afew/afew18000000.cap -l /pnaf/www/afx18 -c /pnaf/etc/suricata.yaml
 8496 be/4 root        0.00 B/s   25.12 K/s  0.00 %  0.00 % suricata -r /root/caps/afew/afew18000000.cap -l /pnaf/www/afx18 -c /pnaf/etc/suricata.yaml
 8497 be/4 root        0.00 B/s   32.29 K/s  0.00 %  0.00 % suricata -r /root/caps/afew/afew18000000.cap -l /pnaf/www/afx18 -c /pnaf/etc/suricata.yaml
 8498 be/4 root        0.00 B/s   28.70 K/s  0.00 %  0.00 % suricata -r /root/caps/afew/afew18000000.cap -l /pnaf/www/afx18 -c /pnaf/etc/suricata.yaml
 8499 be/4 root        0.00 B/s   25.12 K/s  0.00 %  0.00 % suricata -r /root/caps/afew/afew18000000.cap -l /pnaf/www/afx18 -c /pnaf/etc/suricata.yaml
 8500 be/4 root        0.00 B/s   35.88 K/s  0.00 %  0.00 % suricata -r /root/caps/afew/afew18000000.cap -l /pnaf/www/afx18 -c /pnaf/etc/suricata.yaml
 8501 be/4 root        0.00 B/s   21.53 K/s  0.00 %  0.00 % suricata -r /root/caps/afew/afew18000000.cap -l /pnaf/www/afx18 -c /pnaf/etc/suricata.yaml
 8502 be/4 root        0.00 B/s   10.76 K/s  0.00 %  0.00 % suricata -r /root/caps/afew/afew18000000.cap -l /pnaf/www/afx18 -c /pnaf/etc/suricata.yaml
 8503 be/4 root        0.00 B/s   17.94 K/s  0.00 %  0.00 % suricata -r /root/caps/afew/afew18000000.cap -l /pnaf/www/afx18 -c /pnaf/etc/suricata.yaml
```

root@passiveaudittest:~

```
  1  [|||||||||||||||||||||||||||82.9%]    5  [|||||||||||||||||||||||        64.2%]     9  [|||||||||||||||||||||||||||100.0%]    13 [|||||||||||||||||||||||||||100.0%]
  2  [|||||||||||||||||||||||||||87.3%]    6  [|||||||||||||||||||||||||||||||91.0%]     10 [|||||||||||||||||||||||||||96.7%]     14 [|||||||||||||||||||||||||||90.5%]
  3  [|||||||||||||||||||||||||||88.6%]    7  [||||||||||||||||||||||||||||||100.0%]     11 [|||||||||||||||||||||||||||89.0%]     15 [|||||||||||||||||||||||||||80.2%]
  4  [|||||||||||||||||||||||||||88.6%]    8  [||||||||||||||||||||||||||||||100.0%]     12 [|||||||||||||||||||||||||||99.5%]     16 [|||||||||||||||||||||||||||100.0%]
  Mem[|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||9302/32096MB]     Tasks: 117, 44 thr; 14 running
  Swp[|                                                                  34/4095MB]     Load average: 16.10 13.58 13.36
                                                                                        Uptime: 98 days, 02:11:00

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 8486 root       20   0 3094M 1404M  5632 S 395.  4.4  8:09.47 /pnaf/bin/suricata -r /root/caps/afew/afew18000000.cap -l /pnaf/www/afx18 -c /pnaf/etc/suricata.yaml
 8246 root       20   0  747M  495M  3456 R 100.  1.5 51:05.03 /pnaf/bin/snort -U -r /root/caps/afew/afew60000000.cap -l /pnaf/www/afx60 -c /pnaf/etc/snort.conf
 7835 root       20   0  747M  623M  3460 R 100.  1.9  1h37:13 /pnaf/bin/snort -U -r /root/caps/afew/afew48000000.cap -l /pnaf/www/afx48 -c /pnaf/etc/snort.conf
 7775 root       20   0  747M  632M  3460 R 100.  2.0  1h39:19 /pnaf/bin/snort -U -r /root/caps/afew/afew42000000.cap -l /pnaf/www/afx42 -c /pnaf/etc/snort.conf
 8199 root       20   0  747M  511M  3456 R 87.3  1.6  1h00:13 /pnaf/bin/snort -U -r /root/caps/afew/afew36000000.cap -l /pnaf/www/afx36 -c /pnaf/etc/snort.conf
 8239 root       20   0  747M  496M  3456 R 101.  1.5 52:13.72 /pnaf/bin/snort -U -r /root/caps/afew/afew54000000.cap -l /pnaf/www/afx54 -c /pnaf/etc/snort.conf
 8225 root       20   0  747M  505M  3456 R 100.  1.6 57:35.18 /pnaf/bin/snort -U -r /root/caps/afew/afew42000000.cap -l /pnaf/www/afx42 -c /pnaf/etc/snort.conf
 8141 root       20   0  747M  523M  3456 R 91.6  1.6  1h08:55 /pnaf/bin/snort -U -r /root/caps/afew/afew24000000.cap -l /pnaf/www/afx24 -c /pnaf/etc/snort.conf
 7731 root       20   0  747M  647M  3460 R 98.6  2.0  1h46:57 /pnaf/bin/snort -U -r /root/caps/afew/afew30000000.cap -l /pnaf/www/afx30 -c /pnaf/etc/snort.conf
 8229 root       20   0  747M  501M  3456 R 93.9  1.6 54:28.13 /pnaf/bin/snort -U -r /root/caps/afew/afew48000000.cap -l /pnaf/www/afx48 -c /pnaf/etc/snort.conf
 7765 root       20   0  747M  642M  3460 R 96.3  2.0  1h44:16 /pnaf/bin/snort -U -r /root/caps/afew/afew36000000.cap -l /pnaf/www/afx36 -c /pnaf/etc/snort.conf
 8187 root       20   0  747M  516M  3456 R 92.0  1.6  1h03:21 /pnaf/bin/snort -U -r /root/caps/afew/afew30000000.cap -l /pnaf/www/afx30 -c /pnaf/etc/snort.conf
 8512 root       20   0 3094M 1404M  5632 R 79.3  4.4  1:46.62 /pnaf/bin/suricata -r /root/caps/afew/afew18000000.cap -l /pnaf/www/afx18 -c /pnaf/etc/suricata.yaml
 8493 root       20   0 3094M 1404M  5632 S 48.1  4.4  0:29.76 /pnaf/bin/suricata -r /root/caps/afew/afew18000000.cap -l /pnaf/www/afx18 -c /pnaf/etc/suricata.yaml
```

Figure A.11: Hard disk read/write transactions, and CPU core usage during processing phase