An orchestration approach for unwanted Internet traffic identification

Eduardo Feitosa\textsuperscript{a,b}, Eduardo Souto\textsuperscript{b}, Djamel H. Sadok\textsuperscript{a,*}

\textsuperscript{a}Informatics Center, Federal University of Pernambuco, 50740-540 Recife Pernambuco, Brazil
\textsuperscript{b}Institute of Computing, Federal University of Amazonas, 69077-000 Manaus Amazonas, Brazil

Abstract

A simple examination of Internet traffic shows a wide mix of relevant and unwanted traffic. The latter is becoming increasingly harmful to network performance and service availability, while often consuming precious network and processing resources. Coordinated attacks, such as distributed denial-of-services (DDoS), large-scale scans, and worm outbreaks, occur in multiple networks simultaneously and become extremely difficult to detect using an individual detection engine. This paper presents the specification of a new orchestration-based approach to detect, and, as far as possible, to limit the actions of these coordinated attacks. Core to the proposal is a framework that coordinates the receiving of a multitude of alerts and events from detectors, evaluates this input to detect or prove the existence of anomalies, and consequently chooses the best action course. This framework is named Orchestration-oriented Anomaly Detection System (OADS). We also describe an OADS prototype implementation of the proposed infrastructure and analyze initial results obtained through experimentation with this prototype.

1. Introduction

An analysis of contemporary Internet traffic shows a wide array of known and unknown services, new and legacy applications, legitimate and illegitimate traffic, solicited and unsolicited data, highly relevant and unwanted traffic. Of these various types of data, unwanted Internet traffic is increasingly becoming harmful to network performance and service availability, often consuming precious network and processing resources. Typically, unwanted traffic is generated by backscatter\textsuperscript{1} from spoofing activities, unsolicited electronic messages (spam), phishing attempts, denial of service attacks (DoS and its distributed form – DDoS), viruses and worms spreading, or device or software misconfiguration.

Unwanted Internet traffic can be considered a plague, the consequences of which are reflected in financial losses around the world. Statistics provided by the Computer Security Institute (CSI) \cite{1} and public security agencies such as the FBI in the United States indicate that the financial losses caused by network attacks, intrusions and anomalies reached approximately US $252 million during the years 2005–2007 \cite{2–4}. The Gartner Inc. \cite{5} revealed that financial fraud hit 7.5% of Americans in 2008. The Radicati Group \cite{6} estimated global losses equivalent to US $198 billion related to spam messages in 2007 only. In addition, they predicted that the number of spam messages would reach 79% of the volume of global e-mail messages in 2010. Recent studies prove that the proliferation of this traffic is so vast that 3G networks \cite{7} are beginning to feel its negative effects.

Despite both isolated and coordinated efforts to stem this tide, unwanted traffic continues to grow. Firstly, it represents a wide range of user applications, network data types and harmful information with different objectives for its existence. Some of it may be pure nuisance such as spam messages, other unwanted traffic can consist of bulky multimedia content resulting from technological trends in applications, networks and users’ habits, including P2P file sharing (e.g., using Emule, Bit torrents), video

\textsuperscript{*}Corresponding author.
E-mail addresses: elf@cin.ufpe.br, efeitosa@icom.ufam.edu.br (E. Feitosa), esouto@icom.ufam.edu.br (E. Souto), jamel@cin.ufpe.br (D.H. Sadok).

\textsuperscript{1} Backscatter is the traffic received from victims that are responding to denial of service attacks.
shaming (e.g., Justin.tv, Joost, YouTube), and recreational traffic (e.g., MP3 downloads, instant messaging, Skype, MSN). Finally, one finds specifically designed intrusive traffic such as worms, viruses, and denial of service attacks that target networking resources and service availability.

A second reason for the existence of unwanted traffic is the notorious inefficiency of current solutions for identifying and preventing such traffic as described in recent works [8–10]. The increase in Internet link bandwidth and service mix makes the timely detection of unwanted traffic a demanding task that does not scale easily as such links increase in capacity. Further, typical existing solutions trigger high numbers of alarms often after damage has already been perpetrated. More active strategies need to be put in place to speed up detection and response.

Lastly, a clear definition of unwanted traffic is needed. One party’s unnecessary or harmful traffic may be seen as someone else’s normal service. Recreational applications such as online games, instant messages, P2P applications, VoIP and video services, and emerging social networks can be considered normal activities by given ISPs, while inappropriate in most enterprise networks.

In order to address the problems involving unwanted traffic, many efforts [11–14] were made to correlate suspicious traffic (evidences) between different security elements (sensors, detectors, IDSs and ADSs, for example) to improve detection efficiency. Collaborative solutions have the potential to detect intrusions and anomalies that occur across the network (including the Internet) simultaneously by correlating events and alerts among different sub networks. In addition, unlike isolated and individual solutions, where a huge number of false alarms can be produced, collaborative solutions have the potential to reduce computational costs by sharing detection resources between networks and making a high level overview of the security state of the whole network available.

This work shares the view that combining pieces of evidence (on suspicious/anomalous network activity) from multiple networks simultaneously makes it possible to detect attacks (coordinated or not), intrusions and anomalies at an earlier stage, hopefully before they cause significant damage. This paper presents a new approach based on controlled collaboration among different techniques to provide an anomaly detection system. It includes understanding how the different elements (sensors, alerts, techniques, etc.) collaborate and developing new methods to orchestrate such collaboration. We referred to this approach as Orchestration oriented Anomaly Detection Systems (OADS).

We summarize the contributions of this paper as follows:

- We propose a new architecture designed to automatically manage and adapt the execution of various detection instruments (IDS, ADS, remediation systems, firewalls, walled gardens, traffic analysis appliances, and so on) to achieve the desired effect of making the network as secure as possible.
- We implement a proof-of-concept prototype for evaluating OADS architectural and performance characteristics. We use frequency based analysis (FEA) to build rules and identify correlations between observations made by different security and traffic analysis sensors. Moreover, we employ a data fusion strategy to help the detection of unwanted traffic. The initial results indicate that our idea for the composition of collaborative security services is practical and effective.

The remainder of this paper is organized as follows. Section 2 presents an overview of unwanted Internet traffic, including definitions, context problems and classification. In Section 3, the proposal of the Orchestration oriented Anomaly Detection System (OADS) is presented and discussed. The idea behind this approach is to harmonize a range of components (anomaly detectors, information bases, alert handlers, Analyzers, and Decision Service) via an orchestration engine, emulating the interaction among different and distinct elements and increasing the accuracy of a diagnosis. Section 4 describes the implementation of the OADS prototype (all components), including an explanation of the orchestration heuristic. Section 5 presents the evaluation of our approach, where various experiments, representing and emulating different, real world scenarios, are conducted in order to test the OADS prototype. Section 6 presents related works. Lastly, in Section 7, we present our conclusions and make suggestions for future work.

2. Unwanted Internet traffic

Introduced as early as the 1980’s, the expression “unwanted traffic” was always associated with incidents linked to viruses, worms, intrusions and attacks. Some of the early examples of such traffic achieved global prominence, i.e. the Internet worm [15] and, more recently, Amazon and CNN.com DDoS attacks [16].

Lately, the term “unwanted traffic” has been used to define any unsolicited, non-productive, undesirable, and illegitimate traffic. Xu et al. [17] characterize unwanted traffic as malicious or unproductive information that attempts to compromise vulnerable hosts, propagate malware, spread spam or deny the use of valuable services. Other existing nomenclatures are: “junk” traffic [18], background traffic [8], and abnormal traffic [19].

The above definitions fail to acknowledge a recent and important factor: financial gains. We define unwanted traffic as being: any unrequested and unexpected network traffic, which has a unique purpose or outcome of consuming network and computing resources, wasting communication, processing, storage time, or money of users or resource owners while sometimes generating profitability for a party in some form [20].

2.1. Context of problem

For the duration of this paper, we use the terms anomaly and unwanted traffic interchangeably. Novel unwanted traffic classes may partially be detected through matching their generated traffic with the patterns of traffic considered to be a result of normal behavior (or acceptable profile). For example in [8], an anomaly detection framework is presented where the random forests algorithm builds profiles for network services using traffic data.
Determining outliers related to the identified patterns can detect intrusions. Traffic falling outside such normal behavior is seen at least as suspicious. In addition to the detection of new attacks, the definition of behavior patterns or profiles can also be useful in identifying anomalies related to hardware physical problems, application-specification errors and miss-configurations, wrongful policy usage, and so on.

Despite these definitions, there is no consensus yet on what unwanted traffic precisely represents. For example, China treats the traffic generated by the SkypeOut application, a service that allows Skype voice over IP users to access the worldwide public switched telephone (PSTN) lines, as illegal because it affects the profit of the government owned Telecommunication Company.

Similarly, many ISPs, Telecoms, public and private enterprises began limiting the use of Peer-to-Peer (P2P) file sharing and video sharing (YouTube, for example). They argue that the traffic generated by these applications is potentially a means for distributing malicious code such as viruses, worms, spyware, or bots. In addition, it breaches copyright protection, consumes unnecessary bandwidth, and wastes work-time of employees.

Similar views have been attributed to recreational applications (online games) and activities such as emerging social networks (Orkut, MySpace, Facebook, for example) and relationship applications such as IRC (Internet Relay Chat), and instant messaging services (MSN and Google Talk, for example). A recent case involved the Comcast Company, which, in October 2007, was discovered to secretly degrade several popular peer-to-peer applications, including BitTorrent [21].

2.2. Classification

Typically, the strategy for dealing with unwanted traffic is to, first, learn about it and, then, establish effective ways for detecting and extracting it from the traffic mix. But first, let us categorize and classify unwanted traffic in terms of its nature (root causes, common types, targets and effects).

The formal unwanted traffic taxonomy was specified at an IAB (Internet Architecture Board) [8], which suggested the classification of deliberately created unwanted traffic in enterprise networks into three categories: Nuisance, Malicious, and Unknown.

- **Nuisance** refers to background traffic that clogs bandwidth and resources like computing power and storage. Typical examples include Spam and P2P file sharing, since this kind of traffic normally carries malware or lures the user to access unreliable links.
- **Malicious** traffic is responsible for spreading malware, including viruses, worms, spyware, and others. Due to its potential high level of losses, this class of unwanted traffic requires a fast and efficient response and generally needs costly constant software updates. In addition, it is normally specific to targeting operating systems, router and other software vulnerabilities. To complicate matters, there are even kits available in the Internet to teach any user how to create new versions of known viruses and worms.
- **Unknown** involves traffic that could not be identified. It includes possibly malicious, encrypted or simply unclassified traffic. Quiet worms like Storm [22] are another example. They open backdoors on hosts and stay dormant for a long time. Generally, they result in high financial losses.

The classifications of unwanted traffic seen so far do not consider one aspect: legitimate traffic. As previously mentioned, the discussion about what is actually unwanted traffic depends on where the traffic occurs and the structure of the business and networks being used. Potentially, any traffic, even legitimate traffic, that infringes one’s business model is not welcome and is labeled as unwanted.

3. An orchestration oriented anomaly detection system

For a long time, Internet services relied on the informal agreements and goodwill of member sites in protecting and correctly forwarding each other’s traffic. Although lacking centralized control or ownership, the Internet is one of the few, if not the only, self-governing infrastructures that manages to operate reasonably well under such a paradigm.

Today, this trust model is coming under intense attack as a result of the diversified communities that comprise the modern Internet. Can this way of life be maintained? At what price? And what can be done to make a network administrator’s work easier in handling problems? One cannot simply blacklist the domains where unwanted traffic comes from. Moreover, this kind of solution would only benefit those who are exploiting existing Internet weaknesses.

In an attempt to address these concerns, this section presents the design of a new orchestration-based approach to detect, and as much as possible, to limit anomalous (unwanted) traffic. Core to the proposal is a framework named Orchestration-oriented Anomaly Detection System (OADS) that coordinates the reception of a multitude of alerts and events from detectors, evaluates this input to detect or prove the existence of anomalies, and chooses the best course of action.

Firstly, we provide an overview of the system to explain the orchestration concept and how it is applied in this proposal. Next, some essential design requirements are presented that differentiate OADS from other systems. These include features such as the collaboration between distinct network domains to exchange vulnerabilities, attacks, and security incidents. Then, all OADS architectural components will be individually detailed. Lastly, the benefits and limitations of the proposed approach will be explained, detailing how it will deal with special services.

3.1. OADS overview

The research in the field of collaborative anomaly and intrusion detection systems is extensive, but yet very
few integrated systems have been proposed. In addition, due to the technological trends and evolution of strategies and mechanisms for generating unwanted Internet traffic (anomalous traffic), the typical collaboration solutions can be considered deficient in dealing with the current level of Internet traffic, especially for high-speed links.

Orchestration is nothing but a modern metaphor that describes an already well-known security network administrator activity. Analogously to a concert maestro responsible for keeping the rhythm and cueing the different players, security managers organize the harmony and rhythm of various detection instruments (IDS, ADS, remediation systems, firewall, walled gardens, traffic analysis appliances, and so on) to make the network as secure as possible.

The idea behind the OADS approach is to automatically manage and adapt the execution of different anomaly detectors that are traditionally unaware of each other’s presence. In other words, the proposed approach permits and explores the added benefits obtained from the collaboration and harmonization among different techniques against malicious activities. Collaboration enables two or more processes to work together towards a common goal without the need for a pre-established leadership. In the music world, this occurs when musicians work on the same musical album or song. In the information security context, this work sees collaboration as a facilitator of relationships between different anomaly detectors. For instance, two or more detectors can share the same traffic base (traces) to perform analysis or the result of one detector can be used as input for another to help reach a better decision. Harmony, also seen as an interesting concept, means that two or more different sounds or notes fit well together. We show that patterns, in the form of decision tables, emerge among the different security modules. Their use gives greater assurance to network managers in diagnosing unwanted traffic.

3.2. OADS architecture

To better clarify the orchestration approach, Fig. 1 illustrates its organization and shows its components.

The OADS architecture consists of four basic elements: the Alert Pre-Processor, OADS Analyzer, Decision Service, and an OADS Miner. In addition, OADS uses anomaly detectors as external input elements. This section explains in details each one of them.

3.2.1. Anomaly detectors

Anomaly detectors may implement the intelligence necessary to analyze traffic information and to identify potential attacks or abnormal events. They extract data related to suspicious and anomalous traffic and supply the orchestration core of OADS approach with summaries or alerts.

In practical terms, anomaly detectors are logical devices that may be implemented in hardware or software. If in hardware, they are commonly aggregated with sensors that implement a variety of techniques such as sampling and filtering, packet level capture, flow aggregation, and Deep Packet Inspection (DPI). Currently, there are a number of hardware-based products to capture and inspect network traffic in real time. P-Series (Force 10 networks) [23], Orca-flow (Cetacea networks) [24], and Cloudshield technologies [25] are some examples. Regarding software-based detectors, a number of tools, techniques, and systems to process traffic may be used, including IDSs (Snort [26], BRo [27] and Prelude IDS [28]), honeypots (Honeyd [29] and Nepenthes [30]), and many other academic open software prototypes.

3.2.2. Alert Pre-Processor

The Alert Pre-Processor component can be seen as the front door of the OADS approach. Its role consists in receiving information (raw alerts) from anomaly detectors and preparing it for analysis. Basically, it performs two activities.
The first role of the Pre-Processor is determining the adequacy of alerts. Although the OADS approach adopts the Intrusion Detection Message Format (IDMEF) standard, there is still need for content adequacy towards IDMEF output. A good example involves two known free IDS software tools: Snort [26] and Prelude IDS [28]. Basically, they have different nomenclatures for alert identification. While Snort employs the standard nomenclature (Alert messageid), Prelude names this field as Alert ident for example.

The second is aggregation. Since attacks and anomalies might consist of one or more steps, and the anomaly detectors are capable of creating alerts for each of these steps, aggregation makes it easy to build hypotheses about the anomalies and possible defense strategies, and therefore reduces the volume of data. The Alert Pre-Processor component may employ aggregation schemes based on similarity and clustering. These explore the distance in time between similar alerts and between determined alert fields. The idea is to fuse alerts if they are both close in time and typical attributes such as Source IP, Source Port, Destination IP, Destination Port, Attack Class, and source detector.

3.2.3. OADS Analyzer

The Analyzer is the brain of the proposed architecture. It is a component that correlates incoming reports, trying to confirm if there are or are not attacks and anomalies. Moreover, it is also capable of predicting future threats and targets. In general terms, this component receives aggregated alerts built by the Alert Pre-Processor, correlates them to increase their accuracy and consequently validates the assumptions contained in each one of them, while possibly even predicting their occurrence in the future with some level of confidence.

The idea behind this correlation is to build an anomaly traffic patterns base that can be used in the form of a decision table. In other words, all confirmed positive diagnosis (true abnormal traffic) would generate rules that can be stored, for future consultation. Take, for example, a network scenario where a host is infected by a botnet inject- ing a low-rate TCP SYN attack on the network. This attack occurs constantly at 10 min intervals in a network where there are two detectors. The first detector (ADS1) was deployed to evaluate the TCP message exchange behavior pattern while the second one (ADS2) is specialized in detecting massive traffic anomalies. When both are put together to evaluate this anomalous traffic, only ADS1 detects the abnormality because it recognizes a progressive growth in TCP connections as illustrated in Fig. 2. As a result, a generic rule will be created by the OADS Analyzer to identify this anomalous behavior clearly perceived on the picture. Such a rule must not only reflect the state of ADS1 responsible for the detection, but also factor in the equally important state of ADS2, which failed to do so.

3.2.4. Decision service

The Decision Service is responsible for the decision process related to analyzed network traffic. For this, it uses a set of alerts sent by the OADS Alert Pre-Processor or a set of rules sent by an OADS Analyzer as input, evaluates them and then defines the next steps. The evaluation process (decision-making) follows finite state machines such as the finite state automata, Markov chains, or stochastic regular grammars. In our prototype, we used a finite state machine to correlate the information received in possible states that could be used in enforcement actions. In Section 4.4, this process will be explained in detail.

In addition, the Decision Service can perform attack and anomaly prediction. In the network security context, prediction is a technique that seeks to predict divergent behavior of networks, systems, devices, and so on, comparing the features observed at the time with previously observed and defined patterns. The basic idea of prediction is that, after suffering a certain type of attack or anomaly, the sequence of events that generated the anomaly is stored (Decision service has a history base). If this pattern is repeated, measures can be taken to avoid it. This way, the damage caused by these attacks and anomalies could be reduced. Typically, prediction solutions are based in probabilistic or machine learning techniques. For instance, the works of Ye et al. [31], Pikoulas et al. [32] and Hu and Heywood [33] use, respectively, Markov chains, naïve Bayes, and SVM (Support Vector Machine) and SOM (Self-Organizing Map).

3.2.5. OADS miner

The Miner receives queries and responds with summarized and specific content based on information and anomaly news obtained from the Internet and stored locally in an anomaly and vulnerability base.

This Miner is divided in two distinct modules. The first one named OADS Crawler constantly gathers from the Internet new information about traffic anomalies and vulnerabilities, attacks and their origins. This module acts like a crawler collecting and concentrating the maximum possible of information available on the Internet (technical and alert reports, summary traffic, black and white lists, and vulnerabilities’ databases) and stores it in a unique repository. All operation control like activation, deactivation, and parameter change (number of pages searched, initial URLs, specific content, for example) of the OADS Crawler is made by the Decision Service component. The configuration results are stored in a configuration file using an XML-like description language.

The second module provides a differentiated search engine, which is designed to support the decision-making. It is capable of receiving either general or specific queries elaborated from end-users (typically network and IT
managers) or other systems and tools, and returning specific summarized information. This functionality is better explained by the following example. Assuming that many anomaly detectors are reporting suspicious behavior in a determined set of IP addresses and that our Alert Pre-Processor summarizes them in a same cluster. The Decision Service then decides to search for possible causes. For this, it consults the OADS Miner about information related with this case. The OADS Miner executes the queries in its anomaly and vulnerability base looking for any data similar or peculiar to the behavior. Hence, all available information matched with the original query is retrieved, summarized to eliminate duplications or inconsistencies, and forwarded to the Decision Service to help conclude whether or not the suspicious behavior is anomalous.

4. Implementation

An initial AODS prototype has been developed. It consists of the four main modules, namely, anomaly detectors, Alert Pre-Processor, OADS Analyzers, and OADS Decision Service.

4.1. Anomaly detectors

As explained earlier, the core of this work is based on making use the known concept of orchestration to explore the collaboration and harmonization among different anomaly detectors. In other words, the essence of the OADS framework lies in the power gained from the clever combination and coordinated orchestration of different attack detection modules. As a proof of that, the OADS prototype combines modules from existing tested IDS tools such as Snort [26], as well as new strategies we implemented such as Profiling [31] and the ChkModel [12], which will also be described.

4.2. Alert Pre-Processor

In order to deal with the huge number of alerts triggered by numerous security and traffic analysis tools, the Alert Pre-Processor employs the clustering technique proposed by Xu et al. [34] for profiling Internet backbone traffic and discovering significant behavior patterns of interest. Entropy, an information-theoretic approach, is taken to classify traffic into meaningful clusters and consequently to measure the amount of Relative Uncertainty (RU) contained in each data. For more details on how to calculate the RU see [31].

Unlike the original work, we have refocused our work on extracting clusters needed to identify attacks and anomalies. Significant clusters are also used to reduce the number of alerts and to avoid the cognitive overloading of managers. Moreover, the Xu et al. work uses the four-feature space (source IP address, destination IP address, source Port, and destination Port) to determine the communication patterns of the end hosts and services, our Alert Pre-Processor adopts a feature-space composed of the three elements, source IP address (srcIP), destination IP address (dstIP) and class of attack (class).² The extracted srcIP and dstIP clusters represent a set of “interesting” host behaviors (communication patterns), while the class cluster yields a set of “interesting” class/impact information of the attack alerts. No use is made of source Port and destination Port information in cluster identification, due to the fact that source port can be easily changed to hide an attack and destination port is normally related with the attack class field.

Fig. 3 depicts a functional diagram of the architectural components in accordance with their roles. The main entities that compose Alert Pre-Processor are:

1. **Handler Module** – plays a role in receiving information (raw alerts) from different detectors and preparing these for analysis. More specifically, it performs two activities:
   - **Translation** – converts alert messages from different formats to the IDMEF standard format.
   - **Ordering** – performs the temporal ordering and synchronization of alerts according to their timestamps. It is important to emphasize that this work considers that all detectors employ some kind of time synchronization like the NTP (Network Time Protocol) or a GPS device.

2. **Aggregation Module** – is the core of the Alert Pre-Processor. It has the main task of aggregating alerts from multiples sources (detectors) that have some common feature values. It receives ordered alerts and executes a cluster-based algorithm in order to extract only significant alerts. The results (summarized alert information) are then forwarded for possible correlation analysis or examination of the likelihood of attack or anomaly.

From an implementation view, the Alert Handler module employs a client/server approach, where the clients and server are called Handler Agents (HA) and Handler Server (HS) respectively. The HAs are developed using Java (Version 1.6) and deployed together with anomaly detectors in the form of daemon processes. Their activities include: (i) verification of the existence of alert files generated by detectors. The typical interval of checking is set to 1 min; (ii) translation (conversion) of the original alerts to the IDMEF format; (iii) sending the alerts, via socket communication, to the HS. This receives the alerts from HAs, ordered according to their timestamps, and then forwards these to the Aggregation Module.

The Aggregation Module receives ordered alerts from the Handler module and then processes them to extract the most significant clusters. As a final result, a list composed of the following attributes: source IP address, source

![Fig. 3. Functional diagram of OADS Alert Pre-Processor.](image-url)
port, destination IP address, destination port, class of attack, timestamp and alert severity is produced.

4.3. OADS Analyzer

To support the OADS Analyzer functionalities, we developed two distinct modules: one to deal with alert correlation problems and the other for anomaly detection uncertainty.

4.3.1. FER Analyzer

The first solution to address alert correlation issues uses the concept of Frequent Episodes Rules (FER) to perform sequence analysis and, consequently, detect anomalies, including unknown attack patterns [35]. Proposed originally by Mannila et al. [36] for monitoring alarms in telecommunication networks and finding relationships among them, FER is based on the fact that the data subject to analysis consists of a sequence of events. FER is employed for observing and developing a specific knowledge, in the form of probabilistic rules, of the relationships among events (alerts) that anticipate and make up a given attack or anomaly. Not only is it capable of building adaptive event basis signatures, but it also can be used to predict the buildup and preparation towards a possible attack before it is actually carried out, hence giving networks managers a kind of early warning system. This module was entirely developed in Java language. Fig. 4 depicts a functional diagram of the architectural components in accordance with their roles.

4.3.1.1. Alert handler module. The alert handler module performs alert translation into an internal format (values from a finite alphabet starting with A) supported for analysis.

All received alerts are correlated (translated) to an event type in preparation for establishing frequent episode discovery. The selected and extracted attributes of each alert are: class or type of attack, IP address and port number of source system or host, and IP address and port number of target system or host. A correspondence table between events and alerts attributes is built as shown in Table 1.

Another attribute selected and extracted from each alert is the Timestamp. It is used in association with Table 1 to build a list of formatted events time ordered (in seconds), as shown in Fig. 5.

Once completed, this list may then be sent to frequent episodes analysis.

4.3.1.2. Frequent episode analysis module. The present Frequent Episode Analysis (FEA) module is used to generate rules needed to indicate the eminent occurrence of attacks as a result of observing some known sequences of events.

The FER parameters window analysis size and decision threshold are important for the precision of the results as shown in the algorithm description. Note that the processing overhead is not a concern, as this processing may be performed offline and only requires periodic occurrence in order to retrain the Frequent Episode Analysis (FEA) module. The analysis made by this module can be divided into four activities: event collector, candidate generator, generator of frequent episodes and rules generator.

4.3.1.2.1. Event collector. The Event collector scans the list with the event types (Fig. 5) and identifies those that occur most frequently. This process is an algorithm described in [36]. Basically, it receives a list of episodes of similar size and determines of these episodes occur most frequently. It verifies if each episode is contained in the global event sequence. As a result, all frequent episodes are returned.

In the algorithm, an episode is represented as a lexicographically sorted array of event types.

4.3.1.2.2. Candidate generator. The candidate generator receives a list of frequent episodes of size X and generates a new list with possible frequent episodes of size X + 1. This process describes the candidate’s generation for serial episodes. This calculation demands a careful design, as it is crucial make the search for frequent episodes as efficient as possible, considering that the number of possible frequent episodes grows exponentially with the increase of the window size.

In order to provide a better understanding of how the candidate generator works, a simple example is explained as follows. Given a frequent episodes list composed of four elements {AA, AB, AC, AD}, the possible candidates of the

3 The events (alerts) presented in Table 1 are extracted from DARPA 2000 dataset [37], LLDOs 1.0 inside scenario.
episode AB will be (ABA, ABB, ABC, ABD, …, ABZ) and each one of them will be tested to generate possible candidates. Evaluating the first candidate, ABA, the subsets originated from it are AB, BA and AA. So, to prove that the candidate ABA is frequent, its subsets (AB, BA and AA) are matched with the frequent episodes list. As result, ABA is a possible candidate since AB, BA and AA are represented in the frequent episodes list by the episodes AA and AB. On the other hand, the candidate ABZ is not since the subsets AZ and BZ do not have any representation in the frequent episodes list.

4.3.1.2.3. Generator of frequent episodes. A generator of frequent episodes can be seen as the driver for the previous two modules. It calculates all frequent episodes of all sizes using the output from the candidate generator and event collector. Algorithm 1 [36] describes how to calculate a collection of frequent episodes from an event sequence E of episodes.

Algorithm 1. Calculating frequent episodes

\textbf{Input:} event sequence \( E \), window size \( \text{win} \) and frequency \( \text{fr} \)

\begin{verbatim}
01: \( C_1 = \{ \text{all elements in } E \text{ with size equal to } 1 \} \)
02: \( \text{candidateSize} = 1 \)
03: \( \text{candidateVector} = \{ \text{all element } \in C_1 \text{ and \ element} \equiv 1 \} \)
04: \textbf{for candidateSize to win do}
05: \( \text{/* Checking for frequent episodes (Event Collector) */} \)
06: \( \text{FrequentEpisodes} = \text{checkFrequentEpisodesInEventList} \) \( (\text{candidateVector}, \text{win}, \text{fr}) \)
07: \( \text{AllFrequentEpisodes} += \text{FrequentEpisodes} \)
08: \( \text{candidateSize++} \)
09: \( \text{/* Candidate generation (Candidate Generator) */} \)
10: \( \text{candidateVector} = \text{generateCandidates} \) \( (\text{FrequentEpisodes}) \)
11: \textbf{end for}
12: \textbf{return} \( \text{AllFrequentEpisodes} \)
\end{verbatim}

It starts with the definition of a set \( C_1 \) containing all elements from event sequence \( E \) with size equal to 1 (line 1), a control variable \( \text{candidateSize} \) to permit computing frequent episodes according to the window size \( \text{win} \) (line 2) and a structure \( \text{candidateVector} \) to receive all generated candidates of frequent episodes (line 3). To calculate the frequent episodes, the algorithm keeps running until it achieves the window size limit (line 4). On each iteration, the algorithm first verifies the frequency of the candidate episodes from the event sequence (line 5) calling the event collector module. As a result, the returned frequent episodes are stored in a general list \( \text{AllFrequentEpisodes} \) (line 7) and the \( \text{candidateSize} \) variable is incremented by one. Next, it calls the candidate generator module to generate the candidates for frequent episodes, returning all possible candidate episodes with size incremented by one (line 10). The algorithm finishes when returning all frequent episodes.

4.3.1.2.4. Rule generator. A rule generator extracts the rules identified by the frequency analysis phase. It notifies an application, such as an IDS, as to how likely an attack or an anomaly is to commence. Three main advantages are important to mention here: considerable reduction of alert messages; higher precision and confidence in alerts; and the prediction of attacks and anomalies. The pseudo-code presented in Algorithm 2 [36] is responsible for rule calculation.

Algorithm 2. Rule calculation

\textbf{Input:} event sequence \( E \), window size \( \text{win} \), frequency \( \text{fr} \) and confidence \( \text{conf} \)

\begin{verbatim}
01: \( \text{rules} = [] \)
02: \textbf{// Find frequent episodes (Algorithm 1) */}
03: \( \text{FrequentEpisodes} = \text{calculateFrequentEpisodes} \) \( (E, \text{win}, \text{fr}) \)
04: \textbf{// Generate rules */}
05: \textbf{for all} \( \text{episode in} \) \( \text{FrequentEpisodes} \) \textbf{do}
06: \textbf{for all} \( \text{subepisode in} \) \( \text{episode} \) \textbf{do}
07: \( \text{if frequency (episode)/frequency (subepisode)} \geq \text{conf} \) \textbf{then}
08: \( \text{rules} += \{ \text{episode, subepisode, confidence (episode/subepisode)} \} \)
09: \textbf{end if}
10: \textbf{end for}
11: \textbf{end for}
12: \textbf{return} \( \text{rules} \)
\end{verbatim}

This simple algorithm starts by calling the Algorithm 1 (\textit{calculateFrequentEpisodes}) to calculate frequent episodes (line 3) for a given event sequence \( E \), when using a window of size \( \text{win} \) and a frequency threshold \( \text{fr} \). As a result, a list of all frequent episodes is returned (\textit{FrequentEpisodes}). Next, the algorithm performs the rule calculation process through a series of iterations. The first iteration extracts episodes that compose a list of frequent episodes (line 5). The second extracts sub-episodes relative to the parent episode (line 6). The extraction of sub-episodes says that if an episode is frequent in an event sequence, then all its sub-episodes are also frequent. Next, with the episode and its sub-episodes at hand, the algorithm tests if the relation (proportion) between an episode and its sub-episode is greater than or equal to a defined confidence threshold (line 7). If the result is true, then a new rule is generated as shown at line 8. The algorithm terminates by returning all generated rules.
4.3.1.2.5. Rule reduction. Although functional and essential to frequent episodes analysis, the calculation and generation of rules typically results in a huge number of FER and, consequently, a high number of redundant or repeated rules. In order to solve this inefficiency, Qin and Hwang [38] propose an algorithm to reduce the rule space and to provide a simplified view of data patterns (see Algorithm 3). The idea is to establish if an FER is effective (more frequently used) or ineffective (rarely used).

Algorithm 3. Simplified algorithm for rule calculation

```
Input: rules r
01: reducedRules = []; newRules = [];
02: for all rule r do
03: /* Application of Transposition Law */
04: newRules += TranspositionReduction (rule);
05: end for
06: for all rule r in newRules do
07: /* Application of Elimination of Redundant Law */
08: reducedRules += EliminationRedundant (rule);
09: end for
10: return reducedRules
```

The first law, transposition, asserts that given these two FERs (A → AAA and A → AAAA), which describe behaviors for event A, the first is seen as being more effective than the second. This is because of its satisfaction of the transposition law, that is, the first instance can induce the second. Therefore, the first rule is kept and the second one is removed. In general terms, the goal is to make the left hand side (LHS) as short, or as general, as possible due, to the fact that shorter rules are often easier to apply or to compare.

The elimination of redundant laws also assumes that rules with shorter LHSSs are more effective than rules with longer LHSSs. This way, if there are two FERs (A → B and B → C) in the rule set and there is a very frequent rule (A → BC), it is correct to assume that the rule (A → BC) is redundant, since it can be reconstructed from the two previous ones. Therefore, the two rules are kept and the last one is removed. The result of the algorithm is a set of rules without redundant elements.

4.3.2. ADS-fusion

Another solution used in the OADS Analyzer component is based on a data fusion technique. Such a technique deals with uncertainty or imprecision of anomaly detection results and consequently increases the degree of confidence of intrusive or malicious activities, allowing for more accurate decisions to be made.

ADS-Fusion [39] is based on Dempster–Shafer’s Theory of Evidence (DST) [40–42]. DST is a well-known mathematical model that represents uncertainty in knowledge-based systems. It focuses on solving problems and modeling uncertainty when using purely probabilistic methods. Unlike other Bayesian probabilistic theory, DST does not need prior knowledge of the probabilistic distributions of the studied elements. This allows attributing belief values – Basic Probability Assignment or simply bpa in DST – for a subset of possibilities and not only for simple events.

Architecturally speaking, ADS-Fusion is a module that receives the outputs generated by anomaly detectors as input, makes data fusion of these inputs, and produces an inference with a greater degree of certainty than the uncertainty generated by anomaly detectors individually. It is composed of three elements: the collector, sensors, and data fusion engine. The Collector is responsible for capturing network events. Sensor components are responsible for analyzing data generated by the collector, detecting possible anomalies, and assigning a belief for each generated inference. The Data Fusion Engine is responsible for making decisions. It uses DST combination rules to associate and correlate the different analysis and results of distinct sensors to generate more accurate inferences with a greater degree of accuracy.

ADS-Fusion was originally implemented by Lins et al. [39] in C++, but currently it is deployed in Java using JDS [43]. In addition, there was no need for the collector and sensor components in our architecture. Instead, the ADS-Fusion receives such information from the Alert Pre-Processor. It is important to explain the ADS-Fusion module’s important role as a hypotheses generator about the possible real network state. To create a valid working prototype, we also developed other traffic sensors, namely, Profiling [31] and ChkModel [12] in addition to the numerous Snort detectors. It was necessary to adjust their alerts by assigning a well-calibrated bpa value for each alert type. Details of how this is made are explained below.

4.3.2.1. ChkModel. This is a protocol model checking module that identifies protocol violations or unfinished protocol conversations. Rather than establishing the legitimacy of individual packets, ChkModel observes connection and socket behavior and classifies them as being legitimate or attacks.

The premise of detection implemented in ChkModel was proposed in [12]. However, to analyze the ratio between incoming and outgoing TCP packets, an algorithm called Adaptive Threshold was implemented. When the ratio exceeds a certain threshold considered normal, an alarm is triggered.

The bpa generation in ChkModel is based on the distance among obtained values by applying an adaptive threshold function. This way, the greater the distance between the values of obtained and established thresholds, the greater is the belief in the existence of an attack.

Considering the example with an output that contains the detection of an anomalous connection between 58.33.126.229 and 192.168.0.163 IP addresses, where the threshold of packets exchange was calculated as 6. Supposing that for this network, the threshold for “Normal” state is equal to 5, any connection that is above this limit will be considered anomalous. By fixing the belief of the normal state at 0.5, it is possible to determine the belief of the attack as being the sum between belief and the normal rate of increase (6/5 = 1.2, which represents a percentage increase of 20%), obtaining thus a bpa = 0.6.
4.3.2.2. Profiling. Traffic profiling is a strategy based on the use of entropy for the classification of traffic and the identification of patterns. The $bpa$ generation in profiling is offered by evaluating the behavior classification (BC), the frequency of repetition between them and the quantity of flows associated with this classification. For example, considering dstIP as a cluster key, at the first interaction (default time slot) the IP destination 10.108.40.X (150 flows) is classified as BC $= 24$ (DDoS attack for key group). On the second interaction, the BC for this IP remained the same and the number of flows increased to 450, therefore it is possible to increase the belief on this inference. Fixing the belief of the normal state at 0.5, it is possible to determine the belief of the attack as the sum between belief and the normal rate of increase (450/150 = 3, which represents a percentage increase of 200%), obtaining thus a $bpa = 1.0$, i.e., the maximum possible value.

4.3.2.3. Snort. The $bpa$ generation in Snort is based on the severity field present in its alerts. For this, initial beliefs for each possible value of this parameter are established as follows: Low corresponds to 0.5, Medium corresponds to 0.65 and High corresponds to 0.8. In addition, the frequency of repetition is also considered to increase these values.

For example, in a sequence of alerts, the first five represent the same attack and have the same features, including severity level equal to low. This way, the first alert to be evaluated will have an attribute $bpa$ equal to 0.5. The second will increase the $bpa$ in 0.01. Now the $bpa$ is 0.51. The next three alerts also will be increasing the $bpa$. After the five alerts have been evaluated, their combined $bpa$ is 0.55. For alerts with severity level equal to medium and high, the degree of increase is 0.05 and 0.1, respectively.

4.4. OADS Decision Service

The Decision Service receives two types of input. The first is a set of reduced alerts sent by the Alert Pre-Processor, whereas the second consists of rules sent by the OADS Analyzer (made up of the FER Analyzer and ADS-Fusion modules). Although, ultimately, both inputs have the same structure composed of a source IP address, source port, destination IP address, destination port, class of attack and its severity level, the treatment given to each is different. This is because, unlike with the rules, the reduced alerts are typically received first and contain more information, due to the process of significant cluster extraction.

For this reason, the Decision Service adopts two different strategies: one to deal with reduced alerts and a second for rule processing. The former translates alerts to basic firewall rules for enforcement at other devices. Recall that enforcement actions are not the focus of this work. The rules feed a simple finite state machine. It correlates the information processed by different analyzers in possible states that the Decision Service could take. Formal tools for state deterministic finite state machine analysis (DFA), verification and compression may be used.

Table 2 shows the machine states that can be assigned according to the used detectors.

Note that the information relative to the classification (good, bad, transient, low, medium and high) is transported by IDMEF messages into its severity field. Fig. 6 depicts the mapping between this information and a possible enforcement of actions.

Possible values to the enforcement actions are: Limitation, during 60 s; Extreme Limitation, during 300 s; and Blocking of given traffic. Note that such mapping is usually subject to established network policies.

Regarding prediction, we are currently working with a technique that uses the Active LeZi algorithm [45] so that Decision Service realizes predictions using rules previously generated by FER Analyzer.

The idea is to discover patterns in the sequence of generated rules and to consequently improve the decision-making process. For this, a set of stored rules (history base) is evaluated by Active LeZi algorithm to calculate what rules have the highest probability of occurring. The rules that are most likely to occur are presented for the security team for subsequent containment measures.
to shed some light on the way the concept of orchestration is used in the context of this work.

Formally, orchestration refers to an executable business process that may interact with both internal and external complex computer systems, middleware, and services. Currently, orchestration is mainly related to connecting Web services in a collaborative fashion. It establishes the sequence of steps within a process, including conditions and exceptions, and creates a central controller to implement the sequence.

Though OADS may adopt Web Services as its underlying coordination engine to coordinate its actions within a Web environment, it may also operate on a standalone manner by relying on its built-in orchestration service. Typically, the use of emerging Internet standard effort as the Business Process Execution Language for Web Services (BPEL4WS) [44] and the Service Oriented Architecture (SOA) [45] has the clear advantage of opening access to an unlimited number of services and security applications that adopts such technologies. At the time of this work, there is still limited adoption of Web Services and, as such, one does not see the need for the added complexity toward OADS design.

A simple, parameterized and effective heuristic (represented as an algorithm) mimicking orchestration is used instead. Such a heuristic is based on common rules and acquired knowledge that governs the way the received information (alerts and rules) should be treated by the OADS core. Despite its simplicity, this intelligent module obtains good results as shown in latter sessions of this article.

Next, the orchestration heuristic is given in Algorithm 4.

Algorithm 4. Simplified algorithm for orchestration

\[
\text{Step 0: Initialization} \\
\text{Read alerts every x time} \\
\text{While TRUE} \\
\text{Step 1: Alert Pre-Processor receive all multi-source alerts} \\
\text{All received alert are prepared for possible extraction of significant cluster process.} \\
\text{Step 2: Significant cluster extraction} \\
\text{Execute significant cluster extraction process.} \\
\text{All alerts classified as significant are send to the Decision Service (Step 6).} \\
\text{Step 3: If there are alerts to evaluate, go to Step 6} \\
\text{Step 3a, otherwise go to Step 6} \\
\text{Step 3a: If number of remain alerts} > \text{fer.threshold, go to Step 4; Otherwise go to Step 5} \\
\text{Step 5: ADS-Fusion examines received alerts} \\
\text{If alerts have two or more sources, go to Step 5a, otherwise go to Step 6} \\
\text{Step 5a: Execute Dempster–Shafer analysis in alerts.} \\
\text{Send all inferences (rules) to Decision Service (Step 6)} \\
\text{Step 6: Decision Service receives alerts or analysis results} \\
\text{Evaluate the received information} \\
\text{If necessary use OADS Miner to discover extra information} \\
\text{Make decisions} \\
\text{End}
\]

Before proceeding, we will explain the above six steps in the algorithm. All received multi-source alerts are handled by the Alert Pre-Processor component, which extracts all the required alert attributes for analysis (Step 1). Next, the extraction of significant clusters is started (Step 2). As previously described in Section 4.2, the idea is to extract significant information from clusters of interest (srcIP, dstIP and class). The output of this process (in the form of classified alerts) is directly sent to the Decision Service component. It decides what to do next (Step 6).

Although our study demonstrates that the automatic identification of relevant information is successful, it is also possible that some alerts, or even all of these in some cases, are not considered relevant enough to take any decision. However, instead of simply ignoring these, the adopted orchestration heuristic follows a series of selective
procedures, in an attempt to make use of these and possibly improve the current analysis. Such steps are:

1. First, the heuristic verifies if there are any alerts classified as not relevant subsequent to the significant cluster extraction process. If none are encountered, meaning that all alerts are significant, the algorithm proceeds directly to Step 6, where the Decision Service component must process all such information in order to reach one or more decisions. Otherwise, the current amount of alerts is compared with a pre-defined threshold (fer_threshold), used to evaluate the viability of performing a FER analysis over these alerts (Step 3). The studies about FER analysis show that the smaller the number of events, the lower the probability of detecting any frequent episodes. Consequently, some important considerations are needed with regard to configuring FER analysis as shown in Table 3.

   In a case where the number of alerts is greater than the threshold fer_threshold, the FER Analyzer module receives these alerts (Step 4). Otherwise, the remaining alerts are sent to the ADS-Fusion module (Step 5).

2. Under FER Analyzer (in Step 4), the alerts are processed to discover the existence of frequent episodes. These are calculated and episode rules can be generated. Lastly, the episode rules are sent to Decision Service component according to Step 6.

3. During the ADS-Fusion analysis (shown as Step 5), the alerts are evaluated using the known Dempster–Shafer evidence theory. This is used to reduce the uncertainty of these alerts and increase their degree of confidence. However, before undertaking this analysis, the received alerts are verified to determine if they are descendant of two or more distinct detectors, a requirement of DST analysis. If this is not case, the alerts are forwarded to the Decision Service. Otherwise, they are processed and the obtained inferences (or rules) are then sent to the Decision Service.

The Decision Service performs the last step in this important heuristic. To put it simply, it is fed with a diversified set of inputs, including alerts, episode rules and inferences, which it must process before reaching any decision.

It is important to emphasize that OADS’s approach allows for a number of different types of reactions. Under the present implementation, two main decisions are supported. The first one consists of limiting or mitigating something considered malicious by blocking its traffic. This decision is the most common and usually is taken by the Decision Service component. The second option consists of simply not taking any action. This lack of decision may be the case when there is insufficient certainty (evidence) to act upon.

5. Evaluations

OADS testing was purposely confined to an isolated testbed of real machines within our laboratory referred to as GPRT (Networking and Telecommunication Research Group). The idea was to create a controllable environment that resembles a realistic network topology that can be subjected for example to DDoS attacks. As depicted by Fig. 8, this testbed contains around 60 PCs, 3 Cisco switches with 24 and 48 10/100/1000 Mbps interfaces. The PCs are used as edge nodes running different user applications, software routers, and application level traffic generators. Different operational systems including Windows family

<table>
<thead>
<tr>
<th>Number of alerts (fer_threshold)</th>
<th>Window size</th>
<th>Frequency threshold</th>
<th>Confidence threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;50</td>
<td>10</td>
<td>0.01</td>
<td>0.80</td>
</tr>
<tr>
<td>&gt;500</td>
<td>10</td>
<td>0.02</td>
<td>0.80</td>
</tr>
<tr>
<td>&gt;5000</td>
<td>20</td>
<td>0.05</td>
<td>0.80</td>
</tr>
</tbody>
</table>

Fig. 8. Full OADS testbed topology.
Analysis and Injection Tool or simply Packit, is a network and a range of python scripts. The former, called Packet, additional tools were deployed: a tool for packet injection and one was set up to operate over shorter time intervals of one minute. Obviously, this difference will reflect on the number of generated alerts during the analysis.

5.1. Malicious traffic generation

In order to test different attacks and anomalies, two additional tools were deployed: a tool for packet injection and a range of python scripts. The former, called Packet Analysis and Injection Tool or simply Packit, is a network tool designed to customize, inject, monitor, and manipulate IP traffic. It allows the spoofing of nearly all TCP, UDP, ICMP, IP, ARP, RARP, and Ethernet header options. Packit is useful for testing firewalls, intrusion detection/prevention systems, port scanning, simulating network traffic, and general TCP/IP auditing. Packit was used to create customizable DoS and DDoS attack scripts.

The latter is a set of python scripts using Scapy, a powerful interactive packet manipulation python library. Scapy is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more. It can easily handle most classical tasks like scanning, trace routing, probing, unit testing, attacks or network discovery. In addition to these solutions, an Internet script to perform Slowloris HTTP DoS attacks was also used.

5.2. Orchestrating analysis

Recall that the essence of the OADS approach lies in the power gained from the clever combination and coordinated orchestration of different attack detection modules. In order to cover a variety of attacks and to fairly evaluate the robustness of this work, different attack scenarios were planned. Those scenarios are described in the following section.

5.2.1. Scan UPnP

The first experiment for analysis is, in fact, an unplanned event. It was comprised of residual traffic collected by a Snort detector, located on a Firewall/Gateway computer, during three initial minutes of monitoring while preparing for a DNS cache poisoning attack (second experiment).

Snort (Firewall/Gateway) sent 61 alerts reporting “SCAN UPnP” service discovery from a laboratory’s computer (150.161.192.X) to the Internet (239.255.255.250), during thirty minutes. Fig. 9 illustrates the time line of these alerts.

As established in Algorithm 4, for each time interval, the orchestration algorithm must evaluate the received alerts and make decisions about what course of action to take. At 15:01, 7 alerts are received by the Alert Pre-Processor component for handling alerts (Step 1) and were then forwarded for significant cluster extraction (Step 2). The process begins with the calculation of the RU (Relative Uncertainty) value, considering all elements inside the set (in this case, 7 alerts). The results are RU (A)=0.502345126754 for class, srcIP and dstIP cluster. Consequently, these alerts were considered significant, since the RU value is less than our threshold $\beta$ (0.9).

Thus, according to Algorithm 4, the next step consists of sending the alerts to the Decision Service (Step 6). This service receives all alerts and employs its state machine in order to evaluate them. As explained in Table 2, the states correspond to the combination of the alert’s source (XP, Vista and 7) and Ubuntu Linux were used. The malicious PCs also ran similar operating systems.

The OADS server is an Intel Core 2 Quad CPU, with 4 processor Q6600 (2.40 GHz), 4 Gb of RAM, 500 GB of HDD and one network interface 10/100/1000. Although all OADS components (Alert preprocessor, FER Analysis, DST Analyzer and Decision Service) were designed and implemented to work in a distributed setup, they are collocated in the testbed server.

A range of detectors were used, including: various Snort modules [26] version 2.8.6, Profiling [31] and ChkModel [12]. As depicted by Fig. 8, these detectors were spread across specific interest points of the network. All servers ran Linux distributions, including Ubuntu and Debian. Our firewall/router server understandably ran the FreeBSD operational system. Table 4 describes the localization and type of employed detectors in the testbed.

It is important to emphasize several aspects of these detectors. Most Snorts were set up to execute only with default configuration, provided by the Snort distribution. Two different Snorts were set up to detect specific attacks. The first one, located on a DNS server, had a configuration tailored for detecting DNS attacks and anomalies provided by Emergent Threats [46]. The second one, located on a Web server 2, had a configuration set for the detection of Web attacks as well as anomalies also provided by Emergent Threats.

The ChkModel was designed and used to evaluate only TCP packets and cannot be used to analyze connectionless (memoryless) UDP attacks. Additionally, two different Profiling configurations were used. Though both followed the original specifications [31], the first one was set up to perform evaluations over a time interval of five minutes (according to the original proposal) whereas the second one was set up to operate over shorter time intervals of one minute. Obviously, this difference will reflect on the number of generated alerts during the analysis.

| Table 4 |
| Distribution of detectors in OADS testbed. |
| Server | Detector |
| Firewall/gateway | Snort (default configuration) + Profiling + ChkModel |
| DNS server | Snort (DNS configuration) + Profiling (1 min configuration) |
| SMTP server | Snort (default configuration) + Profiling (1 min configuration) |
| Web server 1 | Snort (default configuration) + ChkModel |
| Web server 2 | Snort (emergent configuration) + ChkModel |
| Firewall/router | Snort (default configuration) + Profiling + ChkModel |

Fig. 9. UPnP alerts time line analysis.
5.2.2. DNS cache poisoning

DNS cache poisoning is an attack that consists of changing or adding records to the resolver caches, either on the client or the server. The attack is designed so that a DNS query for a domain returns an IP address for an attacker’s domain instead of the intended domain. According to Hyatt [50], DNS cache poisoning results in pharming, which allows the attackers to perform identity theft, dissemination of false information, and man-in-the-middle attacks.

5.2.2.1. The experiment. The current DNS cache poisoning experiment aims to add a new domain named feitosa.tnt into the authoritative DNS server of the GPRT laboratory. In order to achieve this goal, two computers are used (both running Linux distributions). They both execute a python script (called DNsCachePoisoning.py) that exploits the vulnerability demonstrated by Dan Kaminsky [51]. This script sends fake recursive queries to insert a dummy record in the vulnerable DNS server by guessing the transaction ID. It also inserts an Authority record for a valid record of the targeted domain. The script then uses a random source IP address, a source port number equal to 32,883 (the vulnerable DNS port for recursive queries) and the transaction ID starting with 1024 which then increases by 1 for each interaction.

This attack experiment targeted GPRT’s DNS server. Fig. 10 clearly shows the increase of packet numbers seen before and after the attack is started. The line plotted in the graphic depicts the amount of UDP DNS packets during a time interval between 15:00 and 15:50 of June 02 2010, without the presence of any type of defense. Before the attack, there was a mean of 167 packets per second, whereas this suddenly increased to 586 packets per second once the attack was launched.

The first thirty minutes corresponded to normal user traffic while all detectors were running and without any attack traffic. At the thirtieth minute, attacks are injected into the local area network. Hence, from now on, alert classification, evaluation and decision processes all start taking place. Ten minutes into the experiment, the attack traffic is halted.

5.2.2.2. Analysis. The DNS cache poisoning attack took place at 15:30. As shown in Fig. 10, the average of received packets per second suddenly increased from 167 to 586 packets per second once the attack was launched.

During the first minute of the attack (15:31), our Alert Pre-Processor received two alert files, in the IDMEF format. These represent alerts from the Snort (Firewall/Gateway) and Snort (DNS server), containing 6864 and 30 alerts respectively. The Pre-Processor has to therefore perform significant cluster extraction. On the other hand, our ChkModel did not generate any alerts because it only supports TCP inspection and not UDP packet monitoring. Profiling only generated alerts after two time slots (in this case, two minutes). The Snort Detector (Firewall/Gateway) generated a huge number of alerts of type “DNS response for RFC1918,” via to the original DNS rule. In addition, the other Snort Detector (DNS server) generated alerts classified as “ET CURRENT EVENTS DNS Query Responses with 3 RR’s set (50+ in 2 s) – possible A RR Cache Poisoning Attempt”, via Emergent threat’s [46] DNS rule file. The difference of the number of alerts between both Snort detectors is due to the fact that the first one is located in the local LAN segment of the GPRT network (that uses NAT IP addresses) and is therefore directly hit by the attack. The second Snort detector running at the DNS server, however, is located on a Demilitarized Zone (DMZ) segment and only receives the attack once it passes though the two hosts (Firewall/Gateway linking the LAN segment and the Gateway of DMZ segment). Therefore, this reduces the number of received packets by the second Snort detector. This results in a scenario in which, when the second Snort generates one alert every 2 s, the first one generates between 5 and 8 alerts per second. Such a discrepancy reflects the different rules employed in these detectors.

With regard to the orchestration analysis (Algorithm 6), all Snort alerts were considered significant according to the three keys: srcIP, dstIP and class (all of them had RU (A)=0.303334312), and were consequently forwarded to the Decision Service. After receiving these alerts, the Decision Service performs a simple validation and eliminates those that are duplicated. Consequently, only two alerts were analyzed and the following decisions were made:

- To block any packet sent by IP address 150.161.192.253, source port 53, destined to DNS server with destination port 32,883 and;
- To block any packet sent by IP address 192.168.0.7 with source port 53, destined to 150.161.192.2:32883.
At the third minute of the attack (15:02), the Alert Pre-Processor received 1 alert file from the Profiling (DNS server) and 44 alerts from Snort (Firewall/Gateway). The Profiling alert (signaling entropy change) represents the increase of the number of flows from IP address 150.161.192.253 (gateway) to IP address 150.161.192.2 (DNS server), as previously detected by the Snort (DNS server). It is important to emphasize that the Profiling detector (DNS server) needs at least two minutes to start generating alerts and its output is naturally summarized, thus explaining having only a single alert in our scenario.

The Snort alerts (Firewall/Gateway) are of the same classification as those from the previous (DNS response for RFC1918). However, the amount of generated alerts is smaller. This fact is directly related to the enforcement of the decisions by the firewall. These decisions block all packets from 150.161.192.253:53 to 150.161.192.2:32883 and all packets from 192.168.0.7:53 to 150.161.192.2:32883 from passing.

Considering the orchestration analysis, all Snort alerts were taken to be significant and sent to the Decision Service. Since we only had a single Profiling alert this was well below the minimal FER threshold condition imposed by Step 3a of the orchestration algorithm, as a result it was not forwarded to the FER module. Ultimately, the Decision Service suggested blocking any packet sent by IP address 150.161.192.253 with source port 53 destined to the DNS server on port 32883.

Fig. 11 gives a closer view of the entire attack. The mean number of packets before this attack was 167 per minute. Within the first minute of the attack, the mean number of packets increased suddenly to 1456 packets per minute. With the evaluation and decisions taken by the OADS heuristic, the attack’s effects were felt only during the first 120 s. This time represents the average time that the architecture requires to detect an anomaly and take action to mitigate its effects. From then on, only the “normal” packets are seen in the network.

Note that the decisions taken by OADS (via Decision Service) are applied at the two firewalls of our test-bed topology. Consequently, Profiling and Snort, located on the DNS server, stop generating alerts through remaining duration of the attack, as the internal traffic with source port 32,883 was now blocked from getting into the DMZ zone. The same also happens with Snort and Profiling detectors located on the Firewall/Gateway computer.

5.2.3. SPAM

The next attack scenario is that of an e-mail spam flooding towards an external SMTP Server. This experiment represents a hypothetic scenario where computers corrupted by the Storm worm [22,52] try to infect others via e-mail.

5.2.3.1. The experiment. In order to achieve this setup, four (4) computers (running Linux distributions) ran a simple shell script (called spamflood.sh) that uses the Packet tool [47] to initiate TCP communication (TCP SYN) with the GPRT SMTP server. For five minutes, the script uses forged IP addresses (and subnets) with randomly allocated client port numbers in the originator’s addressing fields.

Fig. 12 illustrates the number of packets destined to GPRT’s SMTP server during 15 min, between 09:15 and 09:30 (the time of highest server activity) of June 09 2010. There is an increase in the amount of packets once the attack is started. Before this, the mean number of SMTP packets was around 305 per minute, whereas it reached approximately 18,000 packets per minute once the attack was launched. It is important to explain that this specific time interval was chosen because it represents the period with increased SMTP activity by GPRT users.

5.2.3.2. Analysis. Before the attack, none of the four detectors acting in this experiment (see Table 3) reported any alert. This changed quickly after the first minutes of the attack.

Approximately 5 s into the second minute (09:21), the Alert Pre-Processor received a single alert file from the ChkModel (TCP Model Checker that monitors connection asymmetric traffic behavior) summarizing events that took place during the first minute. This file contained 1160 alerts and indicated SUSPICIOUS activities coming from different hosts towards GPRT’s SMTP server (150.161.192.192). Since the ChkModel detection is based on the ratio of sent and received packets, these are marked as suspicious because there is a rate of 1 to 0 observed (no packets sent back).

Regarding the orchestration analysis, the significant cluster extraction process confirmed that all alerts were significant (using the destination address field as a key) and, for this reason, they were forwarded to the Decision Service. Here, the state machine was used to evaluate them. Considering that the lack of Profiling and Snort alerts is represented by good state, the result of the state machine analysis is the state B1 (represented by the
combination: Profiling = good, ChkModel = suspicious and Snort = good). Consequently, this decision was translated to the following actions: limit during 60 s any packet destined to IP address 150.161.192.192 with destination port 25.

In the next minute, (09:22) a change of roles occurs. The ChkModel detector, which had classified the initial e-mail traffic as suspicious, adjusts its thresholds, and, from now on, it considers all traffic as legitimate since the TCP handshake mechanism for connection establishment is used correctly. For this reason, it did not generate any new alerts. On the other hand, Profiling generates 1 alert file (containing 1240 source IP addresses) about this “massive” anomaly and sends it to the Alert Pre-Processor.

As the Profiling alert is composed by a unique IDMEF alert, it was not considered significant by the cluster extraction and also could not be applied in the FER analysis as it did not attend the minimal count requirement (number of alerts > fer_threshold). Thus, it was sent to the DST analysis. However, it was also refused because it had a unique source (recall that at least two sources are required to initiate DST analysis). Lastly, the only possible step was sending it to the Decision Service.

After evaluating the Profiling alert, the Decision Service, using the state machine analysis, attributes the state C1 (combination of Profiling = bad, ChkModel = good and Snort = good). Consequently, this decision was translated to the following actions: limit during 300 s any packet destined to IP address 150.161.192.192 with destination port 25.

Observe that since the Profiling was developed to detect massive anomalies, its alerts have more weight in the state machine analysis.

For the next minutes (09:23, 09:24 and 09:25), until the end of the attack, only the Profiling detector continues to generate alerts (containing 1055, 890 and 335 source IP addresses, respectively) and sends them to be analyzed. The evaluations using our state machine are the same (state C1) and the decisions taken are also similar, namely, to limit during 300 s any packet destined to IP address 150.161.192.192 at port 25.

Fig. 13 shows a complete view of this spam attack and its detection process. See that only the Profiling strategy was capable of detecting the attack. This can be explained by the fact that, for the ChkModel, spam e-mail is legitimate traffic, as it looks just like normal e-mail when making standard use of the TCP handshake mechanism for connection establishment. In other words, spam e-mail does not violate the TCP model of connection establishment. It is its content that is harmful and wasteful of user time, not the format. On the other hand, the Profiling method detects the attack as it senses a sudden increase of the number of flows targeting a single SMTP server or IP address. However, the response time for the Profiling detector remains relatively high as it borders the two minutes.

Fig. 13 is derived using flow information as parameters because both detectors (ChkModel and Profiling) employ this type of aggregation to perform their analysis and evaluations.

In summary, the decisions taken by OADS (via its own Decision Service) in this experiment can be considered correct and useful. However, it is undeniable that they are also somewhat inefficient. The actions of limiting any and all packets targeting our SMTP server (150.161.192.192) with destination port 25, although proven to be effective, also stops legitimate and well-behaved connections. A solution for this issue could involve the use of Trusted IP Lists (TIL) [12]. The main idea of TIL consists in keeping a table with the description of the history of “good” connections already established with the network, so that, during attack situations such as these, only known well-behaving partners are favored with most of the bandwidth available to the detriment of unknown connections and/or possible aggressors who will be limited by filters. Traffic shaping may therefore be used to differentiate both types of TCP connections.

5.2.4. Slowloris

Slowloris [49] is a low rate service denial attack (though it really not is a DoS attack). It operates by sending legitimate but incomplete HTTP requests, very similar to SYN flood packets, but at the application layer. This results in fewer packets needed and more granularity to collapse a Web server.

A Slowloris attack takes advantage of Web server design, typically protected from massive attacks (mainly DDOS), occupying all available sockets, causing that the server to continue “waiting for the rest of requests”. Fig. 14 illustrates a Slowloris HTTP request.

What differentiates this example from a functional HTTP request is the final line, where it should be finished with an additional \r\n clause causes some web servers to wait for completion, as maybe the missing carriage

![Fig. 13. SPAM attack with defense.](image_url)

![Fig. 14. Typical Slowloris HTTP request.](image_url)
return/line feed (CRLF) is still on its way. Waiting is also one strategy for protecting the server against a brute force attack such as DDoS. The problem is that, by default, some Web servers will wait five minutes. This results in one resource being occupied for five minutes, unnecessarily in this case of Slowloris. However, it is important that each resource is kept busy, so every so often a new header is sent with the missing CRLF. If the exact form of this header is changed in each iteration, writing intrusion detection signatures becomes significantly harder.

5.2.4.1. The experiment. In order to execute this experiment, two computers, one located in the Informatics Center of the Federal University of Pernambuco (UFPE/Brazil) and the other located in the Institute of Computing (IComp/Brazil) of the Federal University of Amazonas (UFAM) were used to attack GPRT’s Web servers as depicted in Fig. 8. These attackers were set up to shoot simultaneous attacks at the targets. In addition, in order to observe the attacks and their effects, a vulnerable distribution of the Apache Web server on the targets was installed. Understandably, it was decided to only enforce the decisions to stop the attack on Web server 1 (150.161.192.192), as it is our production server.

It is important to explain that, although the Slowloris script uses random source IP addresses, the attackers were located at networks behind NAT servers. For this reason, only two distinct source IP addresses will be perceived in this attack.

The experiment was initiated at 15:00 PM on June 17, 2010, and lasted 20 minutes. Fig. 15 illustrates the increase of the number of established connections to the Web server 2 (150.161.192.51), where clearly it is possible to see after the attack taking place, this Web server reaches its maximum capacity of connections, configured to 250.

5.2.4.2. Analysis. The Slowloris attack took place at 15:00. Two minutes into this attack (15:01), among the seven (7) detectors used in this scenario, Profiling and Snort set up to use emergent configuration (Emergent threats) were unable to generate alerts. The explanation for this is simple. As this attack has the same behavior of a low rate SYN flood, it generates a low number of flows and hence slips through the control of the Profiling technique.

Nonetheless, both Snort’s and ChkModel detectors saw a number of isolated “TCP SYN” packets going towards the same destination servers and therefore should be capable of detecting the attack. In fact, the Alert Pre-Processor received 7 IDMEF alerts (4 for Web server 1 and 3 for Web server 2). Snort classified the attack as “SPECIFIC-THREATS Slowloris http DoS tool” with severity equal to 2, and ChkModel classified it as BAD.

Recall that according to Algorithm 4, the execution of significant cluster extraction is the first step. However, at the end of such process, all five alerts were considered not significant since there were only a few of them and were relatively similar to each other. Hence, the next step is to verify if the FER Analyzer can evaluate these alerts. Again, due to their reduced number, these alerts also failed to pass this verification. Consequently, they are then sent to the DST Analysis (Step 5). Unlike other attacks, this experiment generated alerts from distinct detectors, making these alerts good candidates for evaluation using the Dempster–Shafer Theory.

In the DST analysis, ADS_Fusion begins with the synchronization of alerts to establish connections among them. So, the received alerts are aggregated according to their affinities. In our case, the alerts are split according to their targeting of Web server 1 or Web server 2. After that, they are combined to generate bpa values. For this, their severity parameter, the number of equal alerts and thresholds are used. The two first are extracted from the Snort alert and the latter from the ChkModel alerts. Then, the bpa for each alert is calculated. Snort alerts (Firewall/Router and Web server 1) have only a single alert each with severity as medium. This way, the calculated bpa for each one is 0.65 (corresponding to medium severity as explained in Section 4.4, Table 2). For the ChkModel alerts (Firewall/Router and Web server 1), the calculated bpa is based on threshold values. So, the calculated bpa for each one is 0.8. Tables 5 and 6 describe the bpa values of all alerts.

The last step is inference generation. This step requires the definition of a frame of discernment, an element that contains the possible states of the network, and an evaluation of the hypothesis. In this work, all generated frames of discernment have only two possible elements to represent the network state: Normal or Anomalous (θ = {Normal, Anomalous}). In addition, the hypothesis to be questioned

### Table 5

<table>
<thead>
<tr>
<th>Detector</th>
<th>Severity</th>
<th>Threshold</th>
<th>Bpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Snort (firewall/router)</td>
<td>Medium</td>
<td>0.65</td>
<td>0.65</td>
</tr>
<tr>
<td>ChkModel (firewall/router)</td>
<td>Bad</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>Snort (Web server 1)</td>
<td>Medium</td>
<td>0.65</td>
<td>0.65</td>
</tr>
<tr>
<td>ChkModel (Web server 1)</td>
<td>Bad</td>
<td>0.8</td>
<td>0.8</td>
</tr>
</tbody>
</table>

### Table 6

<table>
<thead>
<tr>
<th>Detector</th>
<th>Severity</th>
<th>Threshold</th>
<th>Bpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Snort (firewall/router)</td>
<td>Medium</td>
<td>0.65</td>
<td>0.65</td>
</tr>
<tr>
<td>ChkModel (firewall/router)</td>
<td>Bad</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>ChkModel (Web server 2)</td>
<td>Bad</td>
<td>0.8</td>
<td>0.8</td>
</tr>
</tbody>
</table>
always is if the network state is Anomalous, that is, the network is under attack or \(H = \{\text{Anomalous}\}\).

Next, the belief, \(B(\cdot)\), and plausibility, \(P(\cdot)\), functions are calculated, considering the hypothesis \(H\) and, as a consequence, a range of belief, \(I(H)\), which expresses the range of values in which it is possible to believe in the hypothesis \(H\).

Overall, DST analysis for Web server 1 assumes:

- \(m_1\) as the mass function of the Snort attack evidence from Firewall/Router, \(m_2\) as the mass function of ChkModel attack evidence from Firewall/Router, \(m_3\) as the mass function of Snort attack evidence from Web server 1 and \(m_4\) as the mass function of ChkModel attack evidence from Web server 1.

- **Frame of discernment is** \(\Theta = \{\text{Normal}, \text{Anomalous}\}\), and:
  - \(m_1(\text{Anomalous}) = 0.65\) and \(m_1(\text{Normal}) = 0.35\)
  - \(m_2(\text{Anomalous}) = 0.8\) and \(m_2(\text{Normal}) = 0.20\)
  - \(m_3(\text{Anomalous}) = 0.65\) and \(m_3(\text{Normal}) = 0.35\)
  - \(m_4(\text{Anomalous}) = 0.8\) and \(m_4(\text{Normal}) = 0.20\)

- The belief and plausibility for all mass functions are 1, i.e., \(B(\{\text{Normal, Anomalous}\}) = 1\) and \(P(\{\text{Normal, Anomalous}\}) = 1\).

As result, the Dempster combination obtained the following values: \(m_1 + m_2 + m_3 + m_4(\text{Anomalous}) = 0.881355\) and \(m_1 + m_2 + m_3 + m_4(\text{Normal}) = 0.118645\). Similarly, for Web server 2, DST analysis assumes:

- \(m_1\) as the mass function of Snort attack evidence from Firewall/Router, \(m_2\) as the mass function of ChkModel attack evidence from Firewall/Router, and \(m_3\) as the mass function of ChkModel attack evidence from Web server 2.

- **Frame of discernment is** \(\Theta = \{\text{Normal}, \text{Anomalous}\}\), and:
  - \(m_1(\text{Anomalous}) = 0.65\) and \(m_1(\text{Normal}) = 0.35\)
  - \(m_2(\text{Anomalous}) = 0.8\) and \(m_2(\text{Normal}) = 0.20\)
  - \(m_3(\text{Anomalous}) = 0.8\) and \(m_3(\text{Normal}) = 0.20\)

- The belief and plausibility for all mass functions are 1, i.e., \(B(\{\text{Normal, Anomalous}\}) = 1\) and \(P(\{\text{Normal, Anomalous}\}) = 1\).

As result of the Dempster combination, the same values for Web server 1, i.e., \(m_1 + m_2 + m_3 + m_4(\text{Anomalous}) = 0.881355\) and \(m_1 + m_2 + m_3(\text{Normal}) = 0.118645\), were obtained.

Since the inferences were generated, according to **Algorithm 4**, the next step is to send the results to the OADS Decision Service. Note that they include the inferences and original alerts. After a validation, the Decision Service makes the decision to block all packets sent by IP addresses 150.161.2.53 (CIn/UFPE) and 200.17.49.5 (IComp/UFAM), destined to Web server (150.161.192.192) on port 80.

Ultimately, the Slowloris attack on Web server 1 is blocked and the detectors no longer generate any more alerts. **Fig. 16** demonstrates the effects of the attack on Web server 1. Note that after the first minute of the attack, the enforcement actions are taken and the attack is blocked.

For the purposes of evaluation, it was decided not to apply the enforcement actions destined to Web server 2. In other words, the decisions to block the packets of the attackers were not enforced, hence allowing the monitoring of the subsequent alerts and their analysis by DST. **Table 7** represents the individual belief of the detectors and the Dempster combination for the hypothesis: “the network is under attack”.

Note that the belief of the Snort detector increases with the number of repeated alerts received. For example, during the initial time period (15:00), a unique alert has its belief equal to 0.65. At the second minute (15:01), one more equal alert was received. For this reason, its belief was 0.65 (its severity) + 0.05 (second apparition of the same alert), totaling 0.7. As a result, the Dempster combination for the hypothesis that the network is under attack also increases. At 15:07, after receiving 8 identical alerts from Snort, the belief achieved the maximum level (1.0). From this point on, DST analysis achieved almost 100% of belief (confidence) regarding the existence of an attack.

### 5.2.5. Multi-step attack

The last experiment (but not the least important one) was composed of a set of attack actions; it was a multi-step attack.
attack. According to Robiah et al. [53], a multi-step attack is a sequence of attack steps that an attacker has performed, where each step of the attack is dependent on the successful completion of the previous one.

The interesting and relevant aspect of multi-step attacks is that they can and must be observed by different detectors. However, it is necessary to gather all pieces so that an attack scenario can be seen as a multi-step attack.

For the experiment at hand, a multi-step attack scenario caused by the Blaster worm [54] spreading mechanism was emulated. The Blaster worm scans the local class C subnet, or other random subnets on port 135, in an attempt to discover vulnerable systems and thus use them as targets. Then the exploit code opens a backdoor on TCP port 4444 and instructs them to download and execute the file MSBLAST.EXE from a remote system via the Trivial File Transfer Protocol (TFTP), running over UDP port 69 to the %WinDir\n system32 directory of the infected system [53].

5.2.5.1. The experiment. In order to implement a Blaster worm experiment, a different testbed, shown in Fig. 8, was built. Nine (9) computers running Linux distribution, were “prepared” to makeup this experiment. The scenario is that of an attack lasting 30 min and affecting as many as 60 computers in the GPRT laboratory. One of these was selected to act as the attacker machine, whereas the others emulated Windows machines.

Based on the Blaster worm operational steps described above, the experiment followed these steps. First, the attacker was activated and it began a scanning process in the network, looking for open port 135 TCP to explore DCOM RPC vulnerability in Microsoft Windows. For this, a port scan script (pscan.py) with 192.168.0.0/24 as target and 192.168.0.96 as source IP address was used.

The second step consisted of exploring the vulnerability on TCP port 135. In order to emulate this step, the attacker executed a script (blaster.py) where each one of the 8 “vulnerable” computers executed another script (blaster_client.py) to communicate with the attacker. Basically, the attacker sent a message instructing the vulnerable computers to open a backdoor on port 4444 TCP. As proof of concept, the attacker script tried next to connect to the vulnerable computers on port 4444 TCP and to access an image file, called BLASTER.jpg. A specific Snort rule to detect this communication, as shown in Fig. 17, was developed.

Third, the vulnerable computer makes a TFTP connection to the attacker to get the file MSBLAST.EXE. In order to make this step viable, the TFTPy API [55] was used to implement all TFTP communication between the attacker and the vulnerable computers. After this step, the attacker terminates its activities.

To end the attack, all vulnerable computers try to establish connections with Web sites where this Storm worm [22,52] can be found, in order to create a new infection. For this, each one of them has a list containing 10 Web site addresses recognizably related with this worm, and chooses only 2 to try a connection. After this step, as the attacker, the vulnerable computers also close their activities.

Fig. 18 illustrates our Blaster worm testbed, indicating the attacker, the vulnerable computers, and the location of the detectors.

It is interesting to notice that multi-step attack scenarios must be observed by different detectors, like signature-based network IDS, ADS and file integrity checker. However, in this scenario only Snort detectors were used due to the fact that they are more prepared, due to their use

```
alert tcp $HOME_NET any -> $HOME_NET 4444 (msg:“Blaster Worm Simulation 4444”; flow:established; uricontent:“BLASTER.jpg”; nocase; sid:1000001; rev:1);
```

Fig. 17. Specific Snort rule to detect Blaster worm simulation.
of rules. In addition, both ChkModel and Profiling are not adequate in this case, since this experiment does not generate differences between ingress and egress TCP packets (which discards the use of ChkModel), nor does it generate huge amounts of traffic which, in turn, discards the use of Profiling to detect it. For such reasons, only Snort detector versions 2.8.3.2 and 2.8.6 were orchestrated.

5.2.5.2. Analysis. The Blaster worm experiment was initiated at 10:00 of July 26 2010. After the first minute, the Alert Pre-Processor received three alert files from the Snort Firewall/Gateway (version 2.8.6) and Snort’s PC 1 and PC 25 (version 2.8.3.2), containing 240, 193, and 194 alerts, respectively.

These alerts represented all attack steps and were classified by Snort’s as “PSNG_TCP_Portsweep” (551 alerts), “Blaster worm simulation 4444” (24 alerts), “TFTP Get” (24 alerts), and “Storm worm phone address” (48 alerts). The first represents the port scan activity. The second one represents the TCP connection opening to port 4444 in vulnerable computers. The third one represents the TFTP connection to get the MBLASTER.exe file. The last alerts show the attempt to connect to Websites related to the Storm worm.

With regard to the orchestration analysis (according to Algorithm 6), the alerts were evaluated using the significant cluster extraction process. As a result, all alerts of the “PSNG_TCP_Portsweep” type are considered relevant (RU (A) = 0.5074532071 for class and RU (A) = 0.8054653980 for srcIP). Next, they were forwarded to the Decision Service, which decided to block any packet sent by the IP address 192.168.0.96, destined to 192.168.0.0/24, as shown in this simple IPTables example (IPTABLES –A INPUT –s 192.168.0.96 –d 192.168.0.0/24 –j DROP).

One must emphasize that although the Decision Service took an enforcement action, it had no effect. The reason for this is simple. In this testbed, the attack was totally inside the network (internal to the network) making the first point of enforcement – namely, the Firewall/Gateway computer – useless in this case. This way, although an explicit order was issued to block all packets from this source, their presence continues in the network.

Nonetheless, there are still other alerts to be analyzed. Those alerts that were considered to be irrelevant in Step 2 of the orchestration algorithm may still be used. The next step is to verify if the FER Analyzer can evaluate these alerts. As the number of alerts is greater than fer_threshold (96 > 50), the discovery of frequent episodes is triggered.

Consequently, all alerts are translated into events. Table 8 exemplifies some alerts and event types in this scenario.

Next, using the established parameters from Table 3 (window size 10, frequency threshold 0.01, and confidence threshold 0.8), the computation of frequent episodes was made and the following values (Table 9) were discovered.

Note that the presence of a low number of alerts generated a more focused number of frequent episodes. Such affirmation was proven by the final number of frequent episodes (32) found for a maximum value window size.

The next step of FER analysis has to do with episode rules generation. FER generated 4334 normal rules and 137 reduced rules, respectively, using a frequency threshold of 0.01 and a confidence level of 0.8 (Table 3). Among the reduced rules, it is possible to find representations of the multi-step attack. For instance, the rules A → AI with confidence 1.00 and I → IQR with confidence 1.00 allow us to deduce that in 100% of the cases when event A (Blaster worm simulation on port 4444, from 192.168.0.96:34521 to 192.168.0.50:4444) occurs, the event sequence AIQR (Blaster worm, TFTP Get, Storm worm and Storm worm) also occurs.

The processing time – including frequent episodes calculation and episodes rule generation – reached 67 s.

Once the episode rules have been generated, the next step is sending these rules (and event tables) to the Decision Service. After validation, a series of decisions was taken to block the communication between the attacker (192.168.0.96) and the vulnerable computers.

<table>
<thead>
<tr>
<th>Event type</th>
<th>Event name</th>
<th>Source IP:Port</th>
<th>Destination IP:Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Blaster worm simulation 4444</td>
<td>192.168.0.96:34521</td>
<td>192.168.0.50:4444</td>
</tr>
<tr>
<td>B</td>
<td>Blaster worm simulation 4444</td>
<td>192.168.0.96:50674</td>
<td>192.168.0.0:31:4444</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>Blaster worm simulation 4444</td>
<td>192.168.0.96:12543</td>
<td>192.168.0.57:4444</td>
</tr>
<tr>
<td>I</td>
<td>TFTP Get</td>
<td>192.168.0.50: 5643</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>TFTP Get</td>
<td>192.168.0.51: 3027</td>
<td>192.168.0.96:69</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>TFTP Get</td>
<td>192.168.0.57: 3027</td>
<td>192.168.0.96:69</td>
</tr>
<tr>
<td>Q</td>
<td>Storm worm phone address</td>
<td>192.168.0.50:64267</td>
<td>222.252.232.184:22861</td>
</tr>
<tr>
<td>R</td>
<td>Storm worm phone address</td>
<td>192.168.0.50:4530</td>
<td>216.139.142.17:10789</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F1</td>
<td>Storm worm phone address</td>
<td>192.168.0.57: 1155</td>
<td>217.77.54:253:12358</td>
</tr>
<tr>
<td>G1</td>
<td>Storm worm phone address</td>
<td>192.168.0.57: 7987</td>
<td>222.33.177.224:1255</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Window size</th>
<th>Candidates</th>
<th>Frequent episodes</th>
<th>Level of participation (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32</td>
<td>32</td>
<td>100.00</td>
</tr>
<tr>
<td>2</td>
<td>289</td>
<td>153</td>
<td>52.94</td>
</tr>
<tr>
<td>3</td>
<td>1376</td>
<td>612</td>
<td>44.47</td>
</tr>
<tr>
<td>4</td>
<td>3468</td>
<td>1428</td>
<td>41.17</td>
</tr>
<tr>
<td>5</td>
<td>5712</td>
<td>2142</td>
<td>37.50</td>
</tr>
<tr>
<td>6</td>
<td>6426</td>
<td>2142</td>
<td>33.33</td>
</tr>
<tr>
<td>7</td>
<td>4998</td>
<td>1428</td>
<td>28.57</td>
</tr>
<tr>
<td>8</td>
<td>2632</td>
<td>612</td>
<td>23.07</td>
</tr>
<tr>
<td>9</td>
<td>918</td>
<td>153</td>
<td>16.66</td>
</tr>
<tr>
<td>10</td>
<td>187</td>
<td>32</td>
<td>17.11</td>
</tr>
</tbody>
</table>
(192.168.0.50–57). Note that these decisions had no effect. A solution would be reached through the use of an automatic access control mechanism as proposed in [56], where the authors employ 802.1x [57] to implement access control based on physical access device ports. This way, specific user traffic may be filtered out at layer two access switches.

5.3. Experimenting with real traces

This section presents an evaluation of the OADS approach when submitted to real malicious traffic traces.

5.3.1. SERPRO dataset

The SERPRO dataset was obtained from the Brazilian Federal Service of Data Processing (SERPRO) [58]. This dataset consists of alerts produced by 10 Snort IDS employed to detect anomalies and intrusions in the SERPRO public network, which includes Web Services, sites and domains of the Brazilian government. It contains about 1.5 million alerts collected during 30 min. The main characteristics of this dataset are summarized in Table 10. It is important to explain that due to security and confidentiality questions, information like dates, IP addresses and ports were anonymized.

The interesting and relevant aspect of this dataset is the diversity of the anomalies presented. Due to being collected in 10 IDS servers in different network locations, the generated alerts range from simple ICMP ping to SQL Injection and malicious code (cross-site scripting). Table 11 describes the Top Ten classes of alerts presented in the SERPRO dataset.

5.3.1.1. The experiment. In order to perform this experiment, we set up 10 computers, running Linux distribution and Snort 2.8.3.2 version, to represent the IDS servers in the SERPRO environment.

Due to hardware differences between the PCs in the experiment, we decided to distribute the traces as follows: Traces 1 and 2 to PC1; Traces 3 and 4 to PC2, Traces 5 and 6 to PC3; Trace 7 to PC4; Trace 8 to PC5; Trace 9 to PC6; Trace 10 to PC7; Trace 11 to PC8; Trace 12 to PC9; and Traces 13 and 14 to PC10. In addition, we configured all HA components (section 4.2.1) to generate alerts every 1 min.

5.3.1.2. Analysis. The experiment using the SERPRO dataset was initiated at 10:00 on December 12, 2011. After the first minute, the Alert Pre-Processor received 10 alert files containing a total of 80,194 alerts.

Among these alerts, the most representative are: “shellcode x86 inc. ebx noop”, a buffer overflow attack (77,910 alerts), “shellcode x86 noop” (171 alerts), “et exploit zilab chat and instant messaging heap overflow vulnerability” (127 alerts), “et policy suspicious inbound to oracle sql port 1521” (109 alerts), “et policy external mysql server connection” (123,931 alerts), “et policy suspicious inbound to mysql port 3306” (107 alerts), and “et worm potential mysql bot scanning for sql server” (106 alerts).

With regard to the orchestration analysis (Algorithm 4), the alerts were first evaluated using the significant cluster extraction process. As a final result, from 80,194 alerts, 80,152 were considered relevant and forwarded to the Decision Service. Nonetheless, there were still other alerts to be analyzed. The next step is to verify if these alerts can be evaluated by the FER Analyzer. As the number of alerts...
is less than \( \text{fer}_\text{threshold} \) (42 < 50), the discovery of frequent episodes is not triggered.

As the remaining alerts are generated by the same source (Snort detectors), these are not submitted to the ADS-Fusion technique. This situation could be resolved easily if the trace contained information about the actual server that generated the alert instead of using single name “Snort”.

In the next time interval (10:01), another 10 alert files are received, containing 81,071 alerts with a similar composition as the previous: “shellcode x86 Inc. ebx noop” (64,887 alerts), “et user_agents suspicious user agent” (1073 alerts), “icmp destination unreachable host unreachable” (822 alerts), “et exploit zilab chat and instant messaging heap overflow vulnerability” (391 alerts), “shellcode x86 noop” (244) and “et policy external mysql server connection” (201 alerts).

The significant cluster extraction step considered 81,053 alerts as relevant and forwarded them to the Decision Service. Similarly to the previous time interval, neither the FER Analyzer nor the ADS-Fusion could be used.

Only the significant cluster extraction step is applied to the SERPRO alerts (Table 12 describes the alerts and duration of all the SERPRO traces).

In the time interval 16 (10:15), only 1 alert file, from trace 14, is received, containing 22,777 alerts. Using significant cluster extraction (Step 2), 21,899 alerts are considered relevant and forwarded to the Decision Service. According to Algorithm 4, the remaining alerts are then tested to determine if they can be evaluated by the FER Analyzer. With the number of remaining alerts being greater than the \( \text{fer}_\text{threshold} \) (878 > 50), frequent episode discovery is triggered.

Using the same established parameters from Table 3 (window size 10, frequency threshold 0.01, and confidence threshold 0.8), the computation of frequent episodes is made and the values listed in Table 13 are discovered.

The result of FER analysis was almost 900 normal rules, 108 of which were listed as confidence 1.00 and 37 were reduced.

Among these reduced rules, the most interesting one is A8B8 \( \rightarrow \) ABB8C8D8F8G8B8K8L8M8, with confidence 1.00. All of its related events deal with Suspicious User-Agent, commonly seen with attacks targeted at servers through standard Internet ports. It is responsible for Web application attacks, attacks involving usage of the PUT and DELETE requests, and attempts to bypass authentication routines for certain Web applications.

Once the episode rules were generated, the next step is the sending of these rules (and event tables) to the Decision Service. The Decision Service suggested blocking any packet sent by network 10.200.201/24 destined to HTTP server 10.12.1.59 on port 80. It is important to clarify that the entirety of this process is performed in 37 s.

The results of the remaining time intervals are similar. Here, the significant cluster extraction process discovered a huge number of significant alerts and forwarded them to the Decision Service and, similarly, the FER Analyzer evaluated the remaining alerts.

To sum up, in all traces analyzed in the SERPRO dataset, the OADS approach was able to make a decision about more than 95% of alerts. These results are encouraging and may be enhanced by further fine-tuning of the software.

### 6. Related work

Collaboration between distinct anomaly detectors in an attempt to identify traffic anomalies quickly and accurately is by no means new. Yet, very few experimental works have been conducted to this point.

A seminal implementation was EMERALD [59], a distributed detection system capable of using both knowledge-based and anomaly-based detection techniques. DWARD [60], a network-based and autonomous detection system, is a source-end DDoS defense system that prevents outgoing attacks from deploying further into the targeted networks. Although employed in a distributed system, DWARD agents do not share information, limiting detection accuracy. COSSACK [61], a DDoS detection and response system, is a highly distributed architecture that combines multicast communications, traditional IDS systems, network topology, vulnerability information, and novel blind detection techniques in a powerful combination that should prove to be effective against a wide variety of DDoS attacks. It uses dynamic probability assignment, which makes detection decisions more flexible.

Other works such as DefCOM (Defensive Cooperative Overlay Mesh) [62] and SCOLD (Secure Collective Defense) [63] are cooperative by nature. DefCOM, a distributed framework for DDoS defense, consists of heterogeneous defense nodes organized into a peer-to-peer network, communicating to achieve a dynamic cooperation defense. It combines source-end, victim-end, and core-network defense and is capable of detecting an ongoing attack and responding by rate limiting the traffic while still allowing legitimate traffic to pass through the system. SCOLD implements a secure collective Internet defense system that utilizes collective resources from participating organizations, tightened coordination and new cyber security defense techniques. It is built to tolerate DDoS attacks with alternate routes via a set of proxy servers with intrusion detection and secure DNS updates. However, both solutions require several infrastructure changes including specialized hardware (commonly routers) and new service config-

### Table 13
Performance for SERPRO dataset (16 min).

<table>
<thead>
<tr>
<th>Window size</th>
<th>Candidates</th>
<th>Frequent episodes</th>
<th>Level of participation (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>348</td>
<td>11</td>
<td>3.10</td>
</tr>
<tr>
<td>2</td>
<td>121</td>
<td>4</td>
<td>3.30</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>5</td>
<td>62.50</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>6</td>
<td>87.50</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>6</td>
<td>75.00</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>7</td>
<td>87.50</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>6</td>
<td>66.6</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>4</td>
<td>50.00</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>2</td>
<td>50.00</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>1</td>
<td>50.00</td>
</tr>
</tbody>
</table>
A recent decentralized, multi-dimensional alert correlation algorithm for CIDSs is proposed by Zhou et al. [64]. A multi-dimensional alert clustering algorithm is used to extract the significant intrusion patterns from raw intrusion alerts. First, the single-feature correlation is used to locally correlate the alerts on the basis of a single traffic feature (e.g. source IP address of the suspicious traffic), then a multi-dimensional alert correlation algorithm is employed to report patterns of alerts that are found based on multiple traffic features.

DSOC (Distributed Security Operation Center) [65] is a hierarchical architecture for collaborative intrusion detection. DSOC is organized into three levels: in the first level, the data is collected by several data collection boxes (e.g. local and remote firewall/IDS). In the second level, local detection engines are used to analyze the data collected and generate alerts accordingly. Finally, all the generated alerts are processed by a global intrusion detection engine to find more complex intrusions and give a global view of the network security.

In the same line as DSOC, Servin and Kudenko [66] propose a hierarchical architecture based on Multi-Agent Reinforcement Learning (MARL). The proposed approach uses an architecture of distributed sensor and decision agents. Sensor agents extract network-state information and send signals to Decision agents, which learn to interpret those signals to categorize different network states.

Certainly, the hierarchical architectures scale better than the centralized alternatives. However, nodes at the higher levels in the hierarchy still limit the scalability of the CIDS and their failure can stop the operation of their whole sub-tree. MADIDF (Mobile Agents based Peer-to-Peer Distributed Intrusion Detection Framework) [67] attempts to overcome this limitation. In this framework, each component consists of four agents: a monitor agent, analysis agent, executive agent, and manager agent. In addition, a mobile agent migration strategy is applied in the framework to allow agents not only to collect information from direct-linked “neighbours” but also from other hosts in the network. While benefiting from agent and Peer-to-Peer techniques, this framework seems to be more scalable than DSOC and MARL but may lack real-time support due to agent mobility.

Rehak et al. [68] propose a strategy in the same line as OADS. Based on multi-agents, the idea is to detect malicious traffic in high-speed networks and alert the operators efficiently. Basically, this architecture employs three layers to detect unwanted traffic aggregated in flows. The first layer, Traffic Acquisition and Preprocessing, is responsible for capturing and preprocessing network traffic, and providing traffic features to upper layers. The second layer, Cooperative Threat Detection, is responsible for detecting traffic anomalies by using trusting agents. For this, they are based on the principles of trust modeling. The last layer, Operator and Analysis Interface, coordinates the detection layer.

Architecturally speaking, both proposals seem similar, but the OADS approach contains some features that we believe give it an advantage over others. Firstly, the traffic acquisition and preprocessing layer has the same function as the OADS anomaly detectors. In addition, this architecture is limited to processing captured flows to detect anomalies. Some types of anomalies, such as low-rate attacks, could pass undetected. Furthermore, OADS embraces a variety of techniques (detectors) to effectively discover traffic anomalies. A look at the detection process offered by Rehak’s architecture shows that all trusting agents receive the same network flows and preprocessed statistics in order to try to define anomalies. The result (list of anomalies) is then forwarded to other agents. With this method, all trusting agents need to implement the same detection techniques. The amount of traffic exchanged flows may overload the network. In addition, it is not clear if collaboration in Rehak’s architecture is mandatory. Lastly, updates or insertions of new detection techniques are not mentioned in the Rehak’s architecture. OADS, due to its distributed and collaborative design, permits the support of updates and the continuous insertion of new components covering new threats.

In summary, the orchestration provided by OADS consists of capturing rules and sequences of how and when the different techniques will collaborate with each other. Their orchestration consists of creating an executable process model, in the form of a workflow, which implements a new service by harmonizing preexisting services.

OADS can be effectively deployed within corporate networks at a large scale. Its deployment in the public Internet requires prior agreement among different domains. Such agreements may be manually established or accomplished through bilateral agreements such as SLAs.

We believe that the collaborative communication style between the security services represents a significant step toward the development of self-defending networks.

7. Lessons learned

7.1. OADS flexibility

Although it is intuitive that a collaborative approach should yield better results than isolated techniques, the way these approaches interact in a workflow-like structure determines the quality and efficacy of the security diagnosis. OADS design is flexible in that it allows for different ordering of and combinations of its components.

This is demonstrated by evaluating a database with real network trace of Brazilian DNS traffic, courtesy from OARC (Operations, Analysis, and Research Center) DITL 2008, 2009 and 2010 project [69]. Using Snort IDS, configured with DNS rules, and Profiling as detectors, we found that a combination using Alert Pre-Processor and FER Analyzer was NOT efficient in identifying DNS attacks anomalies such as spam bots. The number of significant alerts was very low and consequently no rules were generated by FER Analyzer.

Nonetheless, as the alerts were generated mostly by the Profiling detector, which needs more time to find traffic anomalies, we decide to change the sequence of OADS analysis components. This way, we use Alert Pre-Processor and ADS-Fusion, instead of Alert Pre-Processor and FER

utation (DNS, for example) and it is unclear if they scale to high-speed links.
Analyzer. As result, OADS was able to identify a larger number of security DNS attacks such as spam bots or fast-flux domain.

Another aspect of OADS’ flexibility lies in the fact that it is entirely possible to add or change the techniques that comprise the OASD component implementation. Recently, we developed a new Alert Pre-Processor that uses K-medoids clustering techniques to aggregate multi-source alerts [70]. Our tests show that the effectiveness of aggregation and prioritization of alerts is very similar to those implemented in OADS using entropy and relative uncertainty. However, a test combining these two techniques, applying, first, the relative uncertainty in a complete set of alerts of DARPA 2000 and, in sequence, applying K-medoids in remain alerts, showed that almost 95% of all alerts in all DARPA scenarios were aggregated and prioritized.

7.2. Detectors and IDMEF

There are, unquestionably, a large number of efficient solutions for dealing with unwanted traffic. Their actual integration and testing within an implementation setup represents many challenges often due to a lack of technical details. When requested, the authors of many important strategies could not make their code available due to privacy issues within their projects. Additionally, a Java parser was written to convert the output of the detectors used in this work into IDMEF format.

Regarding commercial and public detectors, we tested three tools: Snort [26], BrO [27] and Prelude IDS [28]. Although all seemed to offer similar services, we choose Snort (version 2.8.3.2) due to its ease of installation, simple rules update mechanism and the existence of the Snort-IDMEF plugin [74]. However, each time we upgraded to a new Snort version with more recent rules, the Snort-IDMEF plugin stopped working. As result, we ended up developing our own translator of Snort logs to the IDMEF format.

7.3. Know your traffic traces

Despite the OADS evaluation taking place using attack scenarios (originated from five experiments) and one real traffic trace, we strongly believe that further evaluations are needed to better demonstrate the effectiveness of our approach. However, despite the increased interest in security, partially due to the widespread use of social networks, it is now practically impossible to find traces for comprehensive testing.

We initially used three (3) well known real traces to evaluate OADS approach. The first one was CAIDA’s 2007 DDoS Attack Dataset [71]. This trace had approximately two hour of anonymized traffic traces from a DDoS attack, totaling 21 GB, where only attack traffic to the victim and responses to the attack from the victim are included. Although this trace contains bidirectional traffic, none of the detectors used in our testbed were able to identify any attack. The reason was simple. Since the payload was removed from all packets in this trace, Snort detectors, for example, were unable to match any of their content-based signatures. Our own detectors ChkModel and Profiling could not detect attacks due to the trace anonymization process.

The second and third traces are UMass Gateway Link 3 Trace [72] and MAWI 2006 sample-point B [73]. UMass trace is listed as containing anomalous traffic, but no alerts were generated. MAWI trace does not indicate if there were any attacks or anomalies in its data.

8. Future directions

8.1. Distributed support and cooperation

Collaboration between detection systems is necessary for improving, in both precision and scope, the process of detecting anomalies, suspicious events, and security incidents. This collaboration must be based on information exchange (data and control) between local and remote detection systems. Standard message formats and protocols such as IDMEF [74], IODEF [75] and IDXP [76] can be used for this purpose. Recently, the use of description languages as WSDL (Web Service Description Language) [77] and OWL (Web Ontology Language) [78] have also been adopted in the context of IDS collaboration.

8.2. Trusting IDMEF

To address the current unwillingness, for different reasons, to share security alerts across network providers and domains, some works have been proposed. Lincoln et al. [79] proposes a set of sanitization techniques to obscure sensitive fields including both IP addresses and data. Sensitive associations such as the configuration and defense coverage of a network site are also hidden. Xu and Ning [80] proposed the use of concept hierarchies to balance privacy requirements and the need for intrusion analysis. Already Gross et al. [81] proposed a privacy-preserving mechanism using Bloom filters for use in a CAIDS. A second concern raises the use of authentication and data integrity to prevent wrong or forged information to be injected as part of the generated messages by the CAIDS elements. The works described in [82,83] use certificates to authenticate the messages and thus to guarantee the security of alerts and participants. In addition, these approaches use a central certificate authority unit that can cause scalability problems. The proposal of Brandão et al. [84] describes a framework to integrate IDS, called IDS Composition, based on web service technology, where a security service is established using WS-Security standard [85], XML-Encryption [86], and XML-Signature [87] to deal with authentication and access control of elements and to exchange IDMEF messages.

8.3. Studies on distribution of the OADS orchestration

Further studies should be conducted on the feasibility of distributing the orchestration functionality. Since the collected traffic can easily exceed the processing capacity of a server, especially in broadband networks, OADS orchestration can be quickly overwhelmed by the hun-
dreds or thousands of alerts and diagnoses within a short time.

Intuitively, we can point out two solutions to solve the problem. The first is to use more than one OADS orchestration engine to share the responsibility for the analysis and decisions. The second possible solution (more interesting, but more complex) exploits the concept of parallelism to harmonize multiple OADS orchestration. This harmonization requires cooperation based on the use of fixed time intervals for making the analysis and would employ the same dictionary of decisions. A clever division of OADS orchestration into parallel distributed and possibly independent orchestrations would be a solution to evaluate in the future.

8.4. Improving the orchestration algorithm

The study of other attack scenarios and the identification of new orchestration sequences are needed to enrich the current proof of concept.

8.5. Design and implementation of an inter-domain advertisement service

The authors are in the process of adding a controlled advertisement service that could be shared across domains. All cooperating domains may then publically advertise the detection scope and types of defenses that they offer, creating opportunities for cooperation with other domains.

To initiate this activity, our information bases, including alert (received and pre-processed), analysis of OADS components (FER Analyzer and ADS-Fusion, for example), history of decisions and vulnerabilities reports and Internet traffic statistics (gathered by the OADS Miner) will be made available in a shared storage space.

8.6. Information bases

To assist the anomaly detection process, one may leverage the existence of many information bases to store available and useful data about alerts and traffic summaries (anomaly detectors output), vulnerabilities and Internet anomalies, and history data. Orchestration tools may then use network state information to help in the decision process.

In the OADS approach, two data structures (information bases) that could improve orchestration are:

- **Alert base** contains the outputs of the analysis or traffic summaries performed by anomaly detectors. The idea is to keep processed network traffic while respecting its specificities. This information can be used by OADS as important feedback for improving collaborative activities, taking new decisions, or looking for new anomaly patterns based on these observed results. For instance, the OADS approach can determine that a detector specialized in spam uses the generated alerts by the Profiling [31] detector to further evaluate those results considered suspicious and not yet confirmed involving TCP port 25 (SMTP protocol). Such a feedback procedure increases the chance of discovering previously unperceived anomalies and consequently enhancing the network security level.

- **History base** contains all previous historical decisions taken by the OADS approach for possible future processing. This way, when similar network situations occur, including the same network behavior and anomaly detector activity, the OADS approach can be compared or even immediately take the same old decision. A strategy such as case based reasoning may be used to carry out this processing.

9. Conclusions

A comprehensive definition of Unwanted Internet traffic and ways to minimize its effects are still topics for further discussion and study. The existence of an "evil industry", coupled with the emergence of new services and applications, the constant technological evolution, and the population boom of new (and often unskilled) users imposes new challenges in the activity of detection and limitation of unwanted traffic.

This work has made the case for studying unwanted Internet traffic and proposed an orchestration oriented anomaly detection system (OADS) approach to identifying unwanted Internet traffic. It has also demonstrated the effectiveness of the OADS approach in the identification and mitigation of such traffic through different scenarios and experiments.

In this work, we generate rules and infer with a greater degree of certainty than the uncertainty generated by existing isolated anomaly detectors. We may need to continue our research in calibrating the beliefs for the used sensors and their levels to reflect the fact that, in some contexts, the diagnostic of a particular sensor can be more important more accurate than others. Our decision table needs to grow to embrace new sensors and heuristics.

The problem of unwanted Internet traffic identification is still far from being solved. We expect that this work contributes by providing background on the issue and introducing new research for the design of new effective approaches for unwanted traffic identification. The hope is to see other more multi-step consolidated advanced approaches emerging in the near future.

We imagine a future where such modules may be offered by different providers and one may choose (or compose) selectively among these using emerging Web Services technologies to build more complete and effective unwanted traffic detection models. The emerging Service Oriented Architecture (SOA) may facilitate cooperation over the Internet and information sharing, a model that is missing from today's solutions.

With the emergence of OpenFlow [74] as an open interface for remotely controlling the forwarding tables for switches, routers and access points, high levels of security controls may be established at these devices running at wire-speed. Our proposal may therefore be extended to make use of OpenFlow to enlist the help of routers and switches in the identification and removal of unwanted traffic.


Eduardo Feitosa is a Professor of Computer Science at the Federal University of Amazonas in the North of Brazil. He obtained his Ph.D. recently from the Federal University of Pernambuco while working on an orchestrated architecture for secure networking systems. His interests include work in the areas of security and intrusion detection systems.

Eduardo J. Souto is currently of a Professor of Computer Science at the Federal University of Amazonas in the North of Brazil. His interests include work in the areas of sensor networks, wireless communications, security systems and is actively involved in the development of ad hoc and emergency networks.

Djamal F.H. Sadok is currently Professor of Computer Science at the Federal University of Pernambuco, Brazil. He obtained his Ph.D. from the University of Kent at Canterbury – UK in 1990. He leads a research group working in close cooperation with industrial partners in Europe in the areas of Traffic Measurement, Policy based management, security systems, next Internet architectures and the design of emergency networks.