An Attack Description and Response Architecture Based on Multi-level Rule Expression Language

Samih Souissi, Layth Sliman, Benoit Charroux

An Attack Description and Response Architecture Based on Multi-level Rule Expression Language

Samih Souissi
Télécom ParisTech
Paris, France
souissi@telecom-paristech.fr

Layth Sliman, Benoit Charroux
EFREI Engineering College
Villejuif, France
{sliman, charroux@efrei.fr}

***Abstract*—In the recent years, cyber-attacks have increased rapidly and have become more diverse and unpredictable. Having devastating impacts, the selection of appropriate countermeasures has become a major challenge. We present an attack description and response system based on multi-level rule expression language. It provides a framework to evaluate, identify, classify and defend against sophisticated attacks. Our approach helps simplify complex rules' expression and event handling, thanks to a modular architecture and intuitive rules along with a powerful expression language. The proposed system is flexible and takes into consideration several attack properties in order to simplify attack handling and aggregate defense mechanisms.**

***Index Terms*— Attack Description, Attack Classification, Fuzzy Matching, Security Architecture, Intrusion Detection, Prevention Systems, Detection Rules**

I. Introduction

Cyber-attacks have become more numerous and sophisticated. Thus, Security has become a major concern and has gained more interest for enterprises and corporations. Security aims at protecting firm resources from undesired access by users and applications. Improving security in enterprise information system relies on analyzing threats, risks and vulnerabilities to specify suitable countermeasures. This imposes several challenges to tackle with security issues. One of these challenges is attack detection and response.

To deal with the growing complexity of new attacks, several solutions such as intrusion detection and prevention systems (IDS/IPS) and web application firewalls (WAF) have been proposed. These solutions can be based either on signature or on behavior detection. They play an important role in countering security threats. Signature-based system tend to use static rules and to detect only specific attacks or anomalous behaviors that are already known. In anomaly-based case, they need learning process and detection is more complex. In addition, attack detection techniques are far from being satisfactory [1]. In fact, solutions like IDSs provide unmanageable amount of "false positives" alarms which are hard to inspect. Furthermore, many detection systems do not offer an appropriate compromise between acceptable performance and detection language simplicity. In order to have a good detection rate, the more complex the attack the more complex the defined rule sets. How can a detection system cope with challenges while providing an easy to use and to interact platform?

In attacks detection system the choice of the detection system architecture, implemented rules and parameters, as well as attack modeling are crucial issues. However, the current paper focuses only on the architectural aspects such as modularity, flexibility, extendibility, expressiveness, and simplicity of use in heterogeneous environments. We have already dealt with modeling issues in a previous work [2]. The objective of this work is to bring a level of abstraction that makes the detection of complex attacks more feasible and the detection rules and security policy definition simpler. To this end, hereafter we introduce a novel evaluative classification-based attack detection and response architecture while providing a simple, user-oriented detection rules and integration language. We focus in this paper on the use of our system in a heterogeneous environment requiring complex events correlation and aggregation.

The remainder of this paper is organized as follows. Section II details the related work concerning existing attack detection solution. In section III, we present our proposition describing the architecture, the language, and their interaction. The feasibility of our solution is illustrated in section IV through a use case and a solution technical description. Finally, section V presents the conclusion and perspectives for future work.

II. Related Work

In this section we consider research works in both detection and response architectures and Security languages.

A. Detection and Response Solutions

Over the last decade, on an architectural level, many solutions and mechanisms have been proposed to detect computer and network attacks. Most of them are intrusion

detection systems that enable to write basic vulnerability signatures.

Snort [3], one of the most prevalent IDS, uses a signatures' ruleset. Packets are captured, decoded and diagnosed within a preprocessor. Then detection occurs according to the predefined rules to generate events and report by various means. Snort deployment is easy and it has already existing rich rules database. However, it may not be adapted to detect complex attack or to allow mitigation scenarios defining. Unlike Snort, Bro [4] implements a scripting environment. This IDS is highly customizable, with a powerful scripting language. However, it does not provide a well-documented ruleset. Besides, these solutions are better in detecting attack on a packet level.

For deeper applicative level detection WAF are often used. ModSecurity [5] is a signature-based attack detection solution and has relatively good performances. Though, this system is strongly related to some types of web servers and it only analyses POST queries to avoid performance deterioration. In addition, the rules' defining is very complex, needing a high expertise in HTTP protocol and regular expressions. Naxsi [6] uses a heuristic approach for the detection of XSS and SQL injection attacks. Its performances are acceptable but require a learning process to define white-lists. Defined rules are static and limited to the context of injection attacks using a cumulative scoring system. These systems do not offer a compromise between acceptable performance and simplicity.

Simmons et al. [8] present a cyber-attack taxonomy called AVOIDIT used to identify and characterize attack. Using attack components, a set of metrics is defined and used by an attack defense performance taxonomy (ADAPT system [9]). This system is game model-based. ADAPT allows classifying and detecting blended attacks. It helps make an intelligent decision when defending against attacks. However, the taxonomy lacks defense strategies and it relies on a game decision system that the user is not necessarily able to modify or to define. In [10], Wu et al. propose an attack classification for automatic response systems. Based on this 3 dimensions response-oriented classification (Source: attack origin, Technique: method used by the attacker, Result: outcome of the attack), a correspondence matrix for every attack technique is defined taking into account different sources and results as matrix parameters to define automatic defense techniques. This approach is interesting as the classification helps describe the attack and allows defense mechanisms aggregation. However, types of target are not taken into account. Besides, blended and complex attacks are difficult to classify and thus to counter.

In [7], Dasgupta & Gonzalez describe a decision support for IDS system that uses multi-level parameter monitoring. The system observes user, system and process information levels using them in a Genetic classifier-based IDS. It is an adaptive learning system that evolves ruleset to cope to the environment. Rules are generated from a general knowledge base. Genetic algorithms are used following natural evolution metaphor. It follows the principle of survival of the fittest to provide appropriate rules. This system is interesting as it can perform real-time monitoring, analyzing and providing appropriate response. However, modifying parameters to fit defined security policies is not an obvious task. Golling et al. [11] propose multi-layered detection system. This system uses a manager that communicates with different types of IDS/IPS: flow-based, protocol-based, statistical-based and DPI-based ones. Each IDS is used based on the data stream to monitor. The manager has an important role within the system as it helps find indications, rate them, investigate them in more details, evaluate result and eventually react to malicious traffic. The architecture is built in such a hierarchical manner that allows reducing costs by being deployable on commodity hardware. It is also adapted to high speed networks as the most appropriate detection system is used, thus attack detection is faster. However, policy definition in such hierarchical system is not obvious to set up.

B. Attack and Security Languages

As cyber-attacks have become widespread, a need to represent them has emerged. Attack languages are needed to recognize an attack given a manifestation, to react to it and to analyze relationships between attacks in order to identify scenarios and provide the appropriate response. In [14], Vigna et al. classify attack languages into six different classes: exploit, event, detection, correlation, reporting and response. Exploit languages describe the steps of an intrusion. Event languages define the format of the event used. Detection languages express the manifestation of an attack. Correlation languages analyse alerts from different sources to find a relationship between them. Reporting languages describe the format of alerts raised by security devices. Finally, response languages express defense mechanisms used to counter the attack after its detection.

There exist many security tools using diverse languages to describe attacks and security policies. If we take into consideration the different security languages used in existing solutions, three major language categories come up: Misuse detection, Anomaly detection and Policy Specification Languages.

Most of existing languages are Misuse detection based. These languages look for pattern or predefined sequences of events defining a known attack. The language allows describing computer penetrations as sequences of actions that an attacker performs to compromise a computer system. STATL [14] and IDIOT [15] are examples of such a language. The first one considers an attack scenario as series of states and transitions using State Transition Diagrams and the second one uses Colored Petri-Nets to model attacks. Other languages in this category that describe attacks from different perspectives are Lambda [16] and Adele [17]. Lambda intends to describe all aspects of a cyber-attack. It is at the same time an exploit, detection and alert correlation language. It takes into account

attack precondition, post-conditions, scenario, detection and verification. Unlike Lambda, which uses a declarative approach, Adele provides similar functionalities with an imperative approach using XML language.

Another language category is Anomaly detection that detects deviations from normal behavior i.e. specifies normal and abnormal behaviors of a process as logical assertions about an application program's sequence of system calls and their argument values. One good candidate is ASL [18] and S language [19].

The last category contains Policy Specification Languages. Such language describes the intended behavior of programs using arbitrary events. Usually the policy is specified in term of Patten- Action or Condition - Pattern - Action combinations. One good example is BMSL [20]. Several works have been done to propose different languages to describe attack from different points of view (manifestation, impact, correlation, scenario…). They were able to provide a good background to define an attack in order to detect and describe it. But, they have different level and no language covers the different level from solution integration to attack/misuse detection and response to policy description.

Researchers have done promising works in the field of attack detection and automated intrusion response. Nevertheless, no model that covers attack detection and response issue from integration to policy description is entirely practicable and widely accepted. As mentioned above, many challenges need to be faced to have a complete, expressive, easy-to-use and manage detection system able to detect complex attacks.

III. CONTRIBUTION

The challenge is how to guarantee a good detection of attacks while providing architecture modularity, rule writing simplicity in order to be able to detect complex attacks and respond automatically according to a user defined security policy. To overcome these problems, we present in this section AIDD (Attack Identification Description and Defense) system. This solution should satisfy a set of criteria that will be mentioned at first. Then, we describe our proposal that is composed of two complementary parts: a functional part and a communication part. We present the functional part of our architecture, its different modules and how it works. Then, we introduce the communication part with our new composed language to write detection rules and describe attack scenarios. After that, we explain the interaction sequence between them.

A. AIDD Criteria

In our architecture, a module is an element of the system that performs a predefined function and is able to communicate with other modules. These modules are reusable and interconnected to create a system global function. Our modules and solution should satisfy different criteria:

- Flexibility and Reusability: Our system is independent of the runtime environment, topology and security devices and probes used. It can be reused in different network architectures and contexts, though a period of adaptation is needed.

- Expressiveness: the used language guarantees a high power of expression for describing attacks, writing commands or detection rules to help non security experts.

- Availability: Working also as security monitor, in case of a denial of service attacks, certain links may be no longer available. Nevertheless, our system is still available for monitoring and attack visualization purposes. Our system is proactive as it helps the other areas of the network be aware of what is happening globally.

- Extensibility: User can define its own module to upgrade the system services and extend the architecture. He can also update detection rules, attack scenarios and security policy without modifying what already exists.

- Multi-criteria: Our proposal is adapted to different devices. Specification of input from each device is needed. It can handle security tools from different constructors, open source or not.

Taking into consideration these different characteristics, we define the AIDD architecture modules and language in addition to their interaction.

B. AIDD Architecture

As exposed in [26], the attack detection and response system, shown in Fig. 1, is responsible of flow analysis, attack detection and response. It is composed of the following modules:

- **Dissection Module:** Input (logs/session/event/alert) is transformed, normalized and dissected according to a user defined configuration. A hook system (a hook is an event that will trigger a rule) is closely related to the dissection mechanism. Indeed, hooks are placed and appropriate rules (rule schemes) are associated to evaluate security rules for each dissected field.

- **Analysis Module:** Input can be a dissected network traffic, system/applicative logs or alert. The attack signature or the malicious behavior is described within the detection rules. Seen from another angle, these rules can be considered as a signature database. The

detection engine that is used is IDS/IPS/WAF-like system. The analysis can be based on one or many events coming from one or many probes. The analysis can be either offline (log file) or continuous (events, traffic, etc.). This analysis raises an alert or reacts to eventual attack detection.

- **Classification Module:** The originality of our work consists on adding classification to detection. Detection is no longer Attack-centric but based on attack categories having generic patterns or behavior for each class. This classification will help detect attacks whose signatures are not available but whose behavior or related collected data allows classifying it into a certain category of attack. Information needed to classify the attack are: source, target, vector and result of the attack. This approach allows to aggregate defense mechanisms. If given events or alerts from the same or different sources, it will match them with predefined attack scenarios so that the system is able to respond to complex attacks.

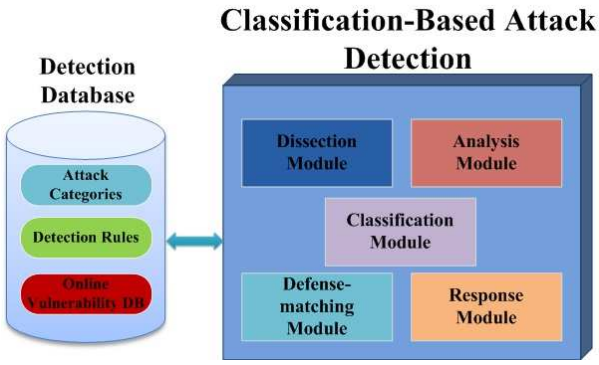

Fig. 1. AIDD Architecture

- **Defense Matching Module:** This module matches each attack category with the appropriate classification and hence to the appropriate defense mechanism(s). Defense mechanisms are classified into different categories (detection, prevention, response (mitigation, remediation), tolerance, etc.). To tackle with altered attack signature, this module uses approximate matching (often referred to as Fuzzy Matching [21]).

- **Response Module:** According to the defense matching module, different reactions to attacks can be defined. The reaction can be responsive (mitigation/remediation) or passive (tolerance) or informative (alert/log/awareness). After response, data (events/alerts) can be resent to analysis module for further review.

- **Detection Database:** It contains all the information needed by our system: attack classification scheme and detection rules. In fact, we propose a generic approach to define Attack categories based on our attack classification [2]. These categories will be the base of our detection process. Detection rules (basic and orchestrated) and known complex attack scenarios are also stored. They can be updated by the user. Orchestration rules are predefined and assigned to specific queries. Our system is able to get updated information by accessing online vulnerability databases such as Open Source Vulnerability Database (OSVDB) [12], MITRE Corporation's Common Vulnerabilities and Exposures (CVE) list [13], etc.

This architecture focuses on the concept of detecting attacks predefined classes and proposing the appropriate defense mechanisms. Our solution provides security by operating in the following way: (1) evaluation of the queries (events), (2) attack identification, (3) extraction of the scenario and the category that are relevant to the identified attack, (4) assessment of candidate defense mechanisms and (5) relevant ones execution. Our solution accepts different types of input. Data come from logs generated by operating systems and applications, information from the network and even alerts generated by IDS or WAF (traffic analysis systems in general). As shown in Fig. 2, the system interacts with sensors and actuators. These sensors can be system, network, application, firewall, IDS or WAF. The actuators can be a firewall or a reverse-proxy based WAF, able to alert, accept, drop or log. The sensors feed the information to the decision system which identifies the attack in question. The knowledge system is composed by the basic rule database and the orchestration rules that describe the policy defined by the user. It also includes attack schemes that need to be detected. When detected, the attack information is sent to AIDD to assess the attack and provide the attack class in order to select the optimal defense mechanism(s).

C. AIDD Language

Given the complexity of the existing formalisms, our original idea, as mentioned in [26], is to define a formalism based on three languages:

- **Atomic Rules Language:** Contains single action rules. Different rule types can be found: Action, Alert, Comparison, Detection, Log, Transformation and Normalization rules.

- **Composition Rules Language:** Composes the basic rules defining the scheme of rules to follow at the detection engine. Different operators can be used to compose these rules: Algebraic, Logic, Correlation and Synchronization operators.

- **Orchestration Language:** In our detection architecture, the communication between the different modules and within each one is handled by a composed language. This language helps define a simpler formalism, give it a high power of expression and bring modularity to security controls.

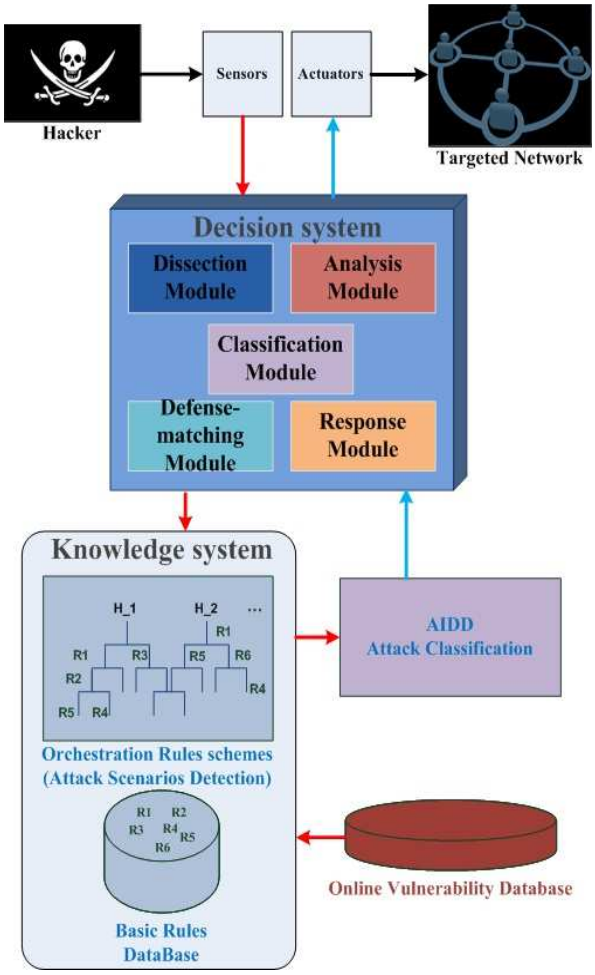

Fig. 2. AIDD Architecture In Context

To this end, in our system we use Compose Language. The use of DSL Compose, a new DSL introduced in [22] allows a clear division and separation of concerns regarding the different aspects of the aforementioned system. Furthermore, it allows a separation of roles between the different actors involved in the system. For instance, a security specialist defines rules for actions to be taken in case of attacks, while a system architect integrates the various modules (analysis, classification ...) In fact, compose can be used for two purposes: Orchestration and coarse grain executable security policy i.e. to express and trigger the actions to be conducted in case of complex attacks (usually actual attacks are composed of a series of fine gained attacks). Compose is based on Spring Expression Language of Spring Framework [23]. Hence, many expressions can be used to handle the description and the countermeasures of complex attacks such as Literal Expressions, Boolean and Relational Operators, Regular Expressions, Class Expressions, Calling Constructors, Relational Operators and User Defined Functions. The architect of the system that integrates the various modules (dissection, analysis, classification...) uses the DSL Compose for its ability to integrate heterogeneous applications. The architect compose the different modules via the DSL Compose, while the exchange of messages between the different modules and their integration in the system is supported by the integration framework underlying Compose. This framework provides the following features:

- Transformer to convert in a message from one format to another
- Filter to transmit messages to modules under certain conditions
- Router that sends a message to multiple modules
- Splitter that divides a message into multiple messages to multiple modules
- Aggregator that combines several message between them
- Adapter that connects the system to the outside (files, database, message broker, protocols (ftp, http...)

Furthermore, Compose integrates natively with any Remote Code Deposit which supports its APIs. This helps in the automatic deployment of new countermeasure codes and provide a continuous integration server that performs regression testing for each deployment of a new version of the application (in the case where the security is provided as a service SEcaaS).

IV. Feasibility

In this section, we illustrate how our model architecture works in a heterogeneous environment. We highlight how AIDD architecture can bring a higher level of abstraction to better aggregate attacks. Then, we provide an implementation example of our proposition.

A. Use case

For more clarity, we present a use case that explains how our security system architecture guarantees security, availability, flexibility, easy interaction and adaptation to heterogeneous environment. To achieve automated security control and management, we will apply our proposed vision to classical network architecture. In this scenario, as shown is Fig. 3, users are connected via internet to certain heterogeneous servers that provide different services: web applications, mailing, database, etc. These servers are protected by different security devices: WAF, IDS firewall.

These devices are from different constructors and have different behaviors and detection mechanisms.

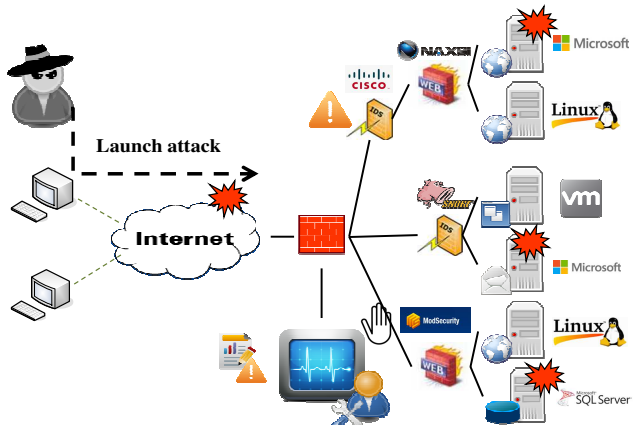

Fig. 3. AIDD Architecture In Context

An attacker intercepts internet traffic, bypasses the firewall to attack the servers as shown in the Fig. 3. He executes different attacks several times: SQL injection, XSS, Malware injection, DoS. The nature of the attack is not the aim of this example, but the attacks diversity and number is to take into account. Some devices detect few attacks and either alert or block them. Alerts and reports are sent back to the system administrator. Other attacks reach their final goals.

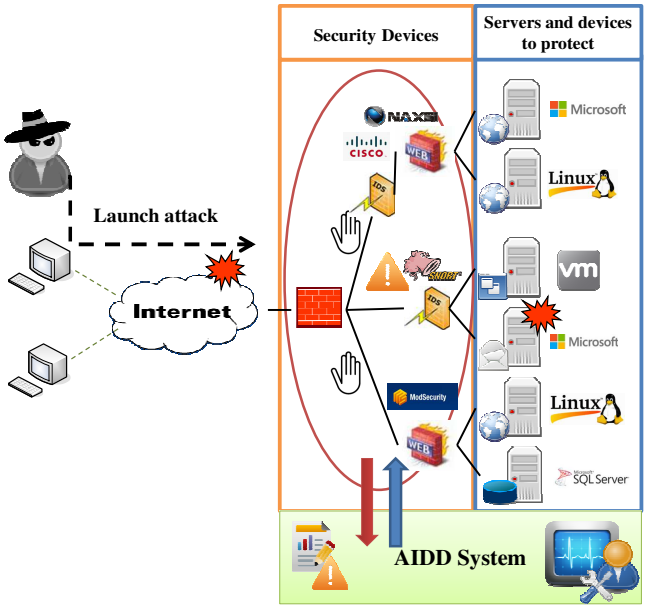

Fig. 4. Architecture Components Interaction

Security architecture in such a topology has several drawbacks. Before attacks occur, maintenance problem can be faced. As used tools are different, system administrator must have high knowledge of different security solutions from different constructors. Detection rules for certain devices like mod security are complicated and need expertise. During attacks, security issues are faced as no overview of the attack scenario, intentions are provided and the attacker can exploit the fact that different deployed security devices do not communicate. After attacks occur, as security devices are different, each one has its own syntax for alert (complicated or not). At the end, security or system administrator will end up with hundreds lines alert log at least for a complex attack that is not easy to handle or to understand.

In order to avoid these drawbacks and to offer an easy to use interface for the system administrator, we introduce our AIDD system as show in Fig. 4. It considers the security devices as a whole and it is connected to each one of them to get its input. It receives the different outputs from the different security component (IDS, WAF, probe, Firewall…) and considers them as a unique interacting system.

Our system analyses their outputs, correlates them, aggregates attacks according to a predefined classification, assigns appropriate defense mechanisms to the detected attack category and communicates it to the actioners (firewall WAF) to either block the attack or update rules. By doing so, an overview of the attacks is possible and even anticipating the final attack strike is conceivable.

B. Workflow diagram

Fig. 5 shows how our solution handles events to be able to figure out attack categories and provides appropriate response.

Our solution takes as an input a raw event. This event can come from different types of security devices as mentioned above. This event is then transformed and normalized (1) to a standard form (IDMEF-like event) so that our systems can handle several events with different types and syntaxes. Then, this normalized event is analyzed (2) to check for patterns able to help classify the attack. Events may be aggregated according to similar patterns at this level. When these parameters are retrieved, the classification (3) is done. The first case is when the attack is known. There will be a check if this attack is a part of an already defined attack scenario (4). Threat Scenarios are predefined by an expert after a risk analysis. These scenarios are modeled using enhanced attack trees [24]. The second case is when the attack is not known. There will be a similarity check test (4'), to verify if the event reported is close enough to an attack behavior. Then, it goes through the scenario check entity (4). After that, we verify if one or several rules related to this attack is available. If yes, appropriate rule is retrieved (5) and then response is executed (6). In fact, the rule retrieving module and the policy enforcement module are part of the same entity which is the Policy Manager. This policy manager handles defense matching and response choice. If the rules are not available, new rules are generated (5'). The rules are then executed by the policy enforcement module. These new rules can be used to update rule database within rule retrieving module (7'). As attacks are evolving, the classification should

evolve. Thus, a classification update can be done by the policy enforcement module (8').

Working as described above, our proposal offers answers to the challenges mentioned before. In fact, it is flexible as it adapts to the changes of topology. Hence, if a new probe or a detection system is added, AIDD can easily be updated to handle this. It is also independent from the type of security devices deployed. Besides, the system stays available to inform the administrator of security state even during a DoS attack to inform user about the state of the secured perimeter. In addition, the predefined rules are simple to define and alerts are aggregated. Thus, when having a lot of attacks, they are classified and the end user is not overwhelmed with alerts.

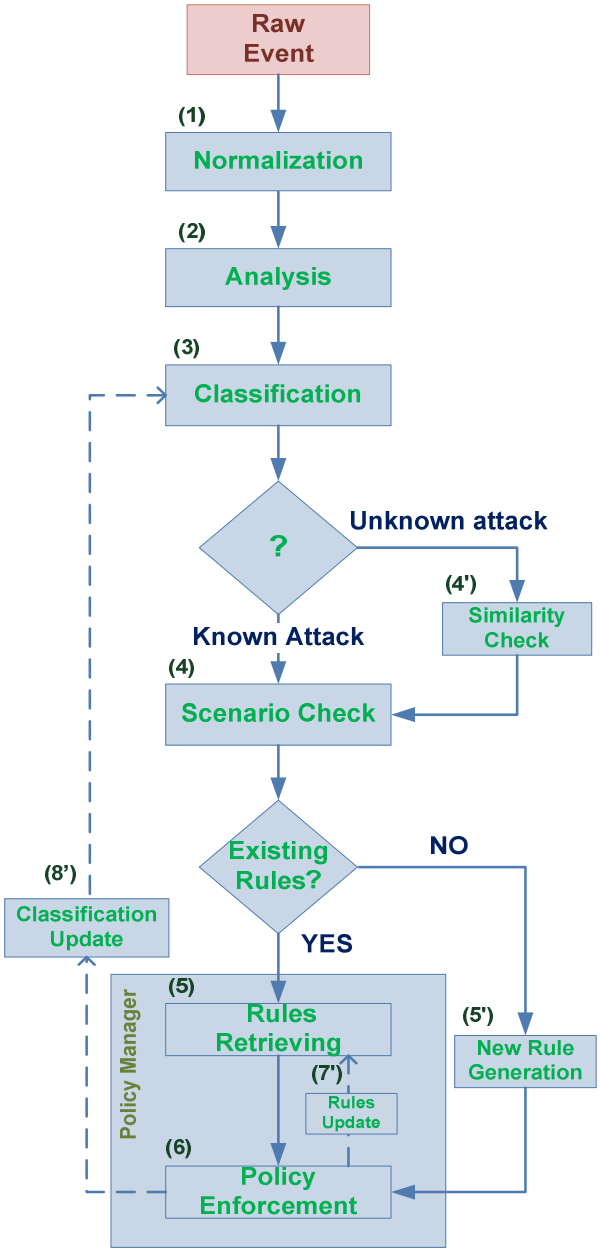

Fig. 5. Architecture Workflow

The advantage of our work is that defense mechanisms are related to the class of the detected attack which allows defense mechanisms aggregation. Furthermore, the modular architecture can be used in several contexts and can use other detection device output as an input. Finally, with its composed language, writing security policies has become easier especially for non-security experts.

C. Language implementation

Our system is build using our Domain Specific Language named Compose [22] (used as orchestration language in our context). Basically Compose is a generic and a multipurpose integration language. It hides enterprise integration patterns behind simple instructions. Thus it addresses any domain, providing modularity and modelling abilities like Business Process Management Notation or UML does. Nevertheless, unlike BPMN, Compose is also a programming language. The architecture of our system (Fig. 5) has been designed and also implemented with Compose. In addition, Compose can be easily customized, adapted and extended to be getting used to another domain. One of these domains is security. The next section introduces the use of Compose as the design language of our system while the following presents its adaptation to security.

1) System building with Compose

Our system can be connected and used by many applications and devices. It provides a number of Channel Adapters out of the box to support various transports, such as JMS, File, HTTP, Web Services, Mail… These adapters are used at the first step to deal with the normalization of input data (see step 1 in Fig. 5). So the first instruction written in Compose for our system is:

Compute normalization with data imported from http://datasecurity.com/..., ftp://...

Normalization allows our system to support many inputs each having its own format, so it has to accommodate and resolve the differences between the varying formats. Message Translator patterns are included into each input data flow. The right translator is picked automatically according to the data flow payload. To aggregate data from many sources and devices, normalization uses the integration pattern Aggregator. Thus, normalization defined by the instruction above is composed of many instructions, each one also coded with Compose:

Compute normalize as code1 with data imported from http://datasecurity.com/...
Compute normalize as code2 with data imported from ftp://...
When code1 terminates with [expression] and code2 terminates with [another expression] Then compute Analysis

Normalize is integrated after each translator associated to each data source. Normalize is coded with any languages supported by message brokers like ActiveMQ (C, Python, PHP…). Exchanges between Compose and external programs like normalize use Java Messaging Service.

Compose offers the clause When expressing events on a code computation (like terminates with). The clause Then allows to indicate which action should be done when the event occurs. As mentioned above in the given example, clauses can use expressions. Expressions are written in several languages (Javascript, Ruby or Groovy). They describe conditions on computed data. The next example adds a condition on the terminates event:

When code1 terminates with "code1.result.value < 10"

Expressions are evaluated at run time. Besides, scatter-gather pattern implementation [22] is used to allow compare algorithm and select the best one:

Compute code2 with data1
Compute code3 with data1
When code2 terminates and code3 terminates Then Store code2.result.value<code3.result.value ? code2.result : code3.result into file

2) Domain Specific Language adaptation for security

In this subsection, we focus on rules language that will be used by our system and handled by compose language. On a more integration level, the proposed language is used to events transformation, attack classification, attack scenarios detection and defense mechanisms definition. It deals with events, event sequences (order, repetition, non-occurrence, time constraints), constraints (contextual) and defense mechanisms. The two different rules' types are defined as following:

- Atomic Rules are functions: Atomic_Rule_Type (Param1, Param2, Param3…)
- Composition Rules are lists: Composition_Rule {Hook1, Rules_composition1, Hook2, Rules_composition2… Options}

Attack components are retrieved from the received events. These components are parameters indicating some aspect of the system, malfunction or failure. They are composed of various anomalies which are observed by sensors.

a) Atomic rules

The different rules that are taken into consideration by our language are:

- Transformation Rules: Trans_Rule (Event, trans1). One transformation per rule. Transformations are used to unify the structure of an event or the pattern of an attack (alert)
- Control Rules: Control_Rule (Event, normal behaviour, Score): check for anomalous flow. Control_Rule (Parameter1, verification): for contextual information
- Match Rules: Match_Rule (event, attack signature) / Match_Rule (parameter1, parameter2) : match 2 parameters: IP address, for example
- Log Rules: Log_Rule (Event)
- Alert Rules: Alert_Rule (Event, Alert type)
- Action Rules: Action_Rule (Parameter, Action), parameter variable is not mandatory, i.e: Action_Rule (Reporting)/Action_Rule (IP,Filtering)

b) Composition Rules

They define the way atomic rules are executed. For example C_Rule (Hook, R1 & R2), where a hook is the area where the control is done. This is meant to optimize rules memory calls. Several operators may be used to combine atomic rules:

- R1 and R2
- R1 or R2
- R1 || R2: 2 Rules executed in parallel
- R1 oand R2: Ordered and
- If R1 then R2 else R3: Condition
- While R1 do R2: Loop

These operators are used also to define attack scenarios and to create a more complete composition rule.

Both Atomic Language and Composition Language are our system's knowledge base components. User composes his rules according to his policies. These rules are handled by compose language to offer and easy-to-use framework for a non-experts in security

V. CONCLUSION

So far, few rule based attack detection systems have taken into account the extensibility of the architecture, the simplicity of rules writing and a Fuzzy Matching attack response. In this paper, we have proposed a novel rule-based attack detection system that is easy to configure. It offers modular and flexible architecture which is able to learn from previous detected attacks. The system can handle altered attack signature using Fuzzy Matching mechanism. It can also handle complex attacks thanks the incremental rules expression languages. It can be adapted to new topologies and can include new attack techniques.

Our can be used by system and network administrator and people that are necessarily security experts to provide information about attacks and how to respond to them. Our system can be ameliorated by including encrypted information handling and defining metrics to enhance the attack-defense matching process.

In this paper, we focused on the architectural aspect of the solution. The next step is to specify the attack classification mechanisms and to study the performance of the system in heterogeneous environments such as multiservice providers and Cloud Computing.

REFERENCES

[1] Dhanakoti Vennila, R.Nedunchezhian "Correlated Alerts and Non-Intrusive Alerts", Department of Computer Science, Anna University of Technology/Sri Ramakrishna Engineering College, INDIA, International Journal of Soft Computing, 7: 302-309, 2012.

[2] Samih Souissi, Ahmed Serhrouchni « AIDD: A novel generic attack modeling approach », Télécom ParisTech, Proceedings of HSPC Conference, Bologne-Italy, 2014.

[3] Snort IDS, available at http://www.snort.org

[4] Vern Paxson, "Bro: A System for Detecting Network Intruders in Real-Time", Lawrence Berkeley National Laboratory, Berkeley, CA, in the Proceedings of the 7th USENIX Security Symposium San Antonio, Texas, January 26-29, 1998.

[5] Ivan Ristic: ModSecurity Handbook: The Complete Guide to the Popular Open Source Web Application Firewall, 2010.

[6] Naxsi (Nginx Anti Xss & Sql Injection) Available at: https://www.owasp.org/index.php/OWASP_NAXSI_Project

[7] D. Dasgupta, F. A. Gonzalez, "An Intelligent Decision Support System for Intrusion Detection and Response", the International Workshop on Information Assurance in Computer Networks: Methods, Models, and Architectures for Network Security, Springer, Vol. 2052, Jan. 2001.

[8] Chris Simmons, Charles Ellis, Sajjan Shiva, Dipankar Dasgupta, Qishi Wu, "AVOIDIT: A Cyber Attack Taxonomy", University of Memphis, 9th Annual Symposium on Information Assurance (Asia'14), Albany, Ny, 2014.

[9] Chris B. Simmons, Sajjan G. Shiva, Harkeerat Bedi, Vivek Shandilya "ADAPT: A Game Inspired Attack-Defense And Performance Metric Taxonomy", University of Memphis, Proceedings of 28th IFIP 11th International Conference SEC 2013, Auckland, New Zealand, 2013.

[10] Zheng Wu, Yang Ou, Yujun Liu, "A Taxonomy of Network and Computer Attacks Based on Responses", Proceedings of International Conference on Information Technology, Computer Engineering and Management Sciences (ICM), 2011.

[11] Mario Golling, Robert Koch, Rick Hofstede "Towards Multi-layered Intrusion Detection in High-Speed Networks", Universität der Bundeswehr München Neubiberg, Germany, University of Twente Enschede, Netherlands, Proceedings of 6th International conference on cyber conflict, 2014.

[12] Open Source Vulnerability Database OSVBD, available at: http://www.osvdb.org

[13] Common Vulnerabilities and Exposures CVE, available at: http://www.cve.mitre.org

[14] S. Eckmann, G. Vigna and R. Kemmerer, "STATL: An Attack Language for State-based Intrusion Detection", University of California Santa Barbara, 2000.

[15] S. Kumar and E. H. Spafford, "A pattern-matching model for misuse intrusion detection", Proceedings of the national computer security conference, 1994.

[16] F. Cuppens and R. Ortalo, "LAMBDA: A Language to Model a Database for Detection of Attacks", ONERA / NEURECOM, France, Recent Advances in Intrusion Detection, 2000.

[17] C. Michel and L. Mé, "Adele: an attack description language for knowledge-based intrusion detection", Proceedings of 16th International Conference on Information Security (IFIP/SEC), 2001.

[18] Ravi Shankar Vankamamidi "ASL: A specification language for intrusion detection and network monitoring", Master's Thesis, Iowa State University, 1998.

[19] Khaled Labib and V. Rao Vemuri, "Anomaly Detection Using S Language Framework: Clustering and Visualization of Intrusive Attacks on Computer Systems", University of California, Proceedings of Fourth Conference on Security and Network Architectures, 2005.

[20] R. Sekar V. N. Venkatakrishnan Samik Basu Sandeep Bhatkar Daniel C. DuVarney, "Model-Carrying Code: A Practical Approach for Safe Execution of Untrusted Applications", Stony Brook University, Proceedings of SOSP Conference, 2003.

[21] N. Bashah , I. B. Shanmugam, "Novel Attack Detection Using Fuzzy Logic and Data Mining", Proceedings of the 2006 International Conference on Security & Management, SAM 2006, Las Vegas, Nevada, USA, June 26-29, 2006. CSREA Press 2006.

[22] B. Charroux, L. Sliman and Y. Stroppa, "Compose: a Domain Specific Language for Scientific Code Computation". Proceedings of CFIP-NOTERE, IEEE, Paris, 2015.

[23] K. Srinivasan, Introduction to Spring Expression Language, Spring Framework, 2011. Available at: http://www.javabeat.net/introduction-to-spring-expression-language-spel/

[24] S. A. Camtepe, B. Yener, "Modeling and detection of complex attacks", SecureComm, 2007.

[25] Scatter-gather pattern implantation. Available at: http://www.enterpriseintegrationpatterns.com/patterns/messaging/BroadcastAggregate.html

[26] S. Souissi, L. Sliman, B. Charroux, "A Novel Security Architecture Based on Multi-level Rule Expression Language", Proceedings of Hybrid Intelligent systems Conference, Nov. 2015.