Toward a novel rule-based attack description and response language

Samih Souissi

Toward a Novel Rule-based Attack Description and Response Language

Samih Souissi
INFRES Department
Telecom ParisTech
Paris, France
samih.souissi@telecom-paristech.fr

***Abstract*—In recent years, attacks have become more diverse and complex, their detection has emerged as a major issue and a primary security challenge. There is a need to represent and share information about these attacks. This paper presents a new language for attack detection and response. The objective is to simplify complex rules' expression, thanks to a modular and intuitive syntax that gives a high power of expression. The originality of our approach is that rules' syntax can be deduced from a certain behavior or automatically generated from a valid behavioral scenario. The paper presents the main concepts behind the proposed approach that deals with the growing complexity of information systems, applications and attacks.**

***Index Terms*—attack description, detection rules, attack language, composition, security event**

I. Introduction

Nowadays, computer and network attacks are becoming more complex, diverse and sophisticated. The main concern of a security officer, facing a security issue, is to respond rapidly and efficiently to these attacks. Thus, many solutions have been proposed to deal with threats and avoid devastating outcomes. We can find, for example, intrusion detection and prevention systems (IDS/IPS) or web application firewalls (WAF). They play an important role in countering security threats. Ideally, the security officer is seeking to have a simple language allowing him to write security rules and deploy them quickly. The language must be expressive enough to detect sophisticated attacks and complex attacks.

However, existing solutions do not offer a compromise between language simplicity and good detection performance. In fact, detection languages should be complex enough to adapt to complexity of attacks. For a matter of formalism simplicity and better performances, some approaches offer adapted solutions for a kind or two of attacks or a certain type of flow. However, unlike attacks that evolve very quickly, they are neither extensible nor scalable and they are far from satisfactory [1]. Moreover, actually, many security devices are from different constructors, open source or not, offering different formalisms. Thus, security officer should be proficient in the different languages of the devices deployed in order to modify security rules or at least understand raised alert. Besides, among the different existing languages, few take into consideration the response aspect proposing only simple mitigation scenarios. How can a language guarantee high expression power, modularity and ease of use while being response oriented?

In our context, our proposal is based on an attack classification from the defender point of view [2] that helps describe the manifestation of the attack in a high level manner. Our proposed language describes an attack and associates the appropriate response according to the context and the defined security policy. Thanks to the classification, aggregating defense mechanism is possible. We define a rule-based attack description language that can be used to model various alerts and events raised by one or several probes (IDS/WAF…) in order to provide suitable response. The language allows to define a generic description of an attack operation independently of detection systems used within the protected perimeter. This generic description is completed by the possibility of verifying the feasibility of action in a certain system and predefining not only detection scenarios for complex attacks but also appropriate defense mechanisms.

The objective of our approach is to provide system administrator with a simple language, easy-to-use by non-security experts, modular and independent from the runtime environment. In a heterogeneous environment, this approach should allow to take into account specifications of detection engines that raise events or alerts. Then, a relationship between an abstract representation of a detection scenario included in the attack process and the event/alert raised by a security device is established. Our language is intended to be used to control security within a defined perimeter, to deploy security controls, or to investigate on specific attack scenarios and provide the appropriate response.

The remainder of this paper is organized as follows. Section II details the related work. In section III, we present our language proposal showing its requirement and its syntax. We expose in section IV a use case showing how our language helps to define security rules to help detect attacks in an easy-to-use manner. Finally, section V presents the conclusion and perspectives for future work.

II. Related work

As attacks are becoming more common, there is a need to represent and share information about them. 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 [3], 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, analyses 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 numerous security tools providing a language to write security attack description and in our case security rules. Most of them are intrusion detection systems that allow to write basic signatures. We studied several tools and languages to find out how to combine simplicity of use with expressiveness and better complex attack description.

Snort [4] offers a simple and easy to use language where a rule is written in a single line and can be combined with other ones. Referencing separated files and rules and using substitution variables is permitted. The rule header is as follows: “Alert tcp any any -> 192.168.1.0/24 any”. Action done by the rule can be: Alert, Log, Pass, Activate and Dynamics; and protocol can cover IP, TCP, UDP and ICMP. The advantage of such language is that it provides large amount of easy to use and simple to write rules and does not generate traffic delays. However, it does not detect all attacks, needs a constant rule database update, reporting is not efficient and it does not allow vulnerabilities description. Unlike Snort, Bro [5] implements a scripting environment. After reducing incoming packet stream into high level events that reflect underlying network activity, a Policy script interpreter executes a set of event handlers to detect attacks. Bro provides also a signature engine to match patterns in packet streams. This IDS is highly customizable, with a powerful scripting language simple configuration. However, it does not provide a well-documented ruleset. Besides, these solutions are better in detecting attack on a packet level.

For applicative level detection, ModSecurity [6] is a widely used firewall thanks to its broad core rule set and its relatively good performances. It offers a large signature database and easily "pluggable" set of generic attack detection rules that provide web application base level protection. Modsecurity offers a high expressiveness defining several variables (request, response, information, user, performance, environment, etc.), operators (char, normalization, transformation, validation, evaluation, behavior, specific, etc.) and actions (altering and non-altering on data or stream). A modSecurity rule has the following form: “SecRule VARIABLES OPERATEURS [TRANSFORMATION, ACTIONS]”. This allows to describe patterns and to cope with the complexity of a protocol like HTTP. However rules’ defining is very complex and needs high expertise in HTTP protocol and regular expressions. Naxsi [7] which uses a heuristic approach offers an easier syntax. This WAF provides a simple way to detect attacks using a whitelisting of accepted queries (needing a learning process) and a scoring system related to malicious characters. It offers an easy syntax but rules are static and limited to injection attacks. These systems do not offer a compromise between acceptable performance and simplicity. Thus, defining good security rules that guaranties a suitable overall security level is not obvious.

Other languages that describe attacks from different perspectives are Lambda [8] and Adele [9]. 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. This allows describing the attack from a point of view of both attacker and defender and deducing the alert sequence automatically from pre and post conditions. It helps have better attack description and explicit and implicit alert correlation. It share many elements with language such as STATL [10] that focuses of state transition and uses finite state machine graphs; or IDIOT [11] language that detect attacks through pattern matching a signature against audit log and which uses Colored Petri-Nets. However, Lambda signatures are independent from detection algorithm; it provides description of the attack, the detection process and its verification. These components are separated which brings more freedom and modularity to the language. Unlike Lambda, which uses a declarative approach, Adele provides similar functionalities with an imperative approach using XML language. Adele provides also syntax to define events and attack responses. Both can be viewed as a programming language that offers developers a rich set of statement and libraries. However, defining attack scenarios can be tedious and need high security and language expertise.

Another language that is a user-oriented solution to write security rules is Haka [12]. This network and security-oriented language allows to check not only security policy, through security rules defining, but also protocols’ conformity. It uses Lua [13] as a framework language because of its expressivity, readability and popularity. Lua is a relatively simple language, compact and earns notoriety judging by many security tools that use it (Suricata, Wireshark). In addition, it has a JIT Version (luajit) that allows to improve performance. API has been defined to allow easily manipulating the contents of packets/flows, and responding actively or passively to an eventual detected anomaly. However, it stays a prototype of a language useful especially to validate protocol conformity. Even though it offers the user the possibility to define detection rules, these rules are more for the packet level and the user needs a lot of expertise on attack patterns to define good security rules which can turn out to be a tedious task.

Several works have been done to propose different languages to describe attack from different points of view (manifestation, impact, correlation…). They were able to provide a good background to define an attack in order to detect and describe it. Nevertheless, these languages are supposed to be used by security experts and a random user with no network, programming or security expertise is faced with non-human-like languages and needs training before mastering the language. As mentioned above, many challenges need to be faced to have a complete, expressive, easy-to-use language able to detect complex attacks.

III. Contribution: Composed Language

In this section, we describe our proposed language. The faced challenge is how to propose a language easy-to-use and high level to describe attack scenario and help detect complex attacks while having an intuitive syntax and guaranteeing extendibility of the language. The language is rule based, as in IDS Snort or WAF ModSecurity. Although rule-based systems involve expert knowledge to operate, they are easier to implement and maintain. Rules can be presented in a human comprehensive text format, and therefore, operator can understand and add new personalized rules to the system. Writing rules in a human-like language is obviously easier than writing scripts filled with regular expressions and details about the events. Therefore, a rule-based approach is selected.

As our language is response oriented, its syntax allows specifying defense mechanisms to prevent attack. This feature is interesting when the protected area faces a complex attack, our approach helps prevent the final strike of the attack. This language creates a knowledge base that will be working using inference engine and a memory to constitute an Expert System.

A. Requirements

The defined language should have the following requirements:

- **High power of expression:** The language is able to describe: action detection, scenario detection and attack effects verification. The occurrence of attack actions is deduced from detection actions.
- **Modularity:** In both Rule construction and scenario definition. In Rule construction, a combined set of rulesis used, thus the creation of a complex rule is just the matter of combining rules allowing to hide the complexity from the end user. On a higher level, a scenario description should use action corresponding to low level attacks.
- **Ease of use:** The complexity of detection is hidden from the administrator (pattern matching rules, signatures…). Defined rules describe events that should be detected to confirm the occurrence of an attack. Administrators just combine atomic rules as bricks to define composition rules which describe a certain security policy. This approach follows Lego metaphor.
- **Deduction:** If several malicious activities are detected by one or several probes, the language can take into account that these actions are part of the same attack scenario. Modeling scenarios appropriate to the protected area context is primordial.

B. Conceptual model

Before defining the language, we model the different entities that appear in our language. Fig.1. represent a high level overview of the elements appearing in our language framework and the way they interact with each other. A security tool generates events detecting attacks. These attacks need defense mechanisms that are either proposed to the security officer which responds, or automated.

We define an attack as a combination of actions rising one or many alerts from a source toward a target. The description of attack is done by a set of component: Localization (Source/target), Vector, Impact (Consequence of the attack) and Scenario (from a detection point of view).

- Localization: Source which indicates the origin of the attack and target which consists of final destination of the attack.
- Attack Vector (technique): Is inspired by what have been proposed by the previous attack taxonomies. Categories and sub categories should be redefined to bring more precision. We consider the "vulnerability" exploited to execute the attack.
- Effect (Impact or Result): Contains High level information of the impact of the attack.
- Scenario: Indicates the different combination of actions done by the attacker to perform the described attack. The actions are corresponding to attack steps.

We differentiate between an attack and a suspicious action. In fact, as defined previously an attack is an action that violates a security policy, whereas a suspicious action can be used in a scenario of an attack. We focus on the events raised by detection process and we take into account the fact that actions performed by the attacker may be different form detected actions. While predefining scenarios, instead of defining attack scenarios we define detection scenarios corresponding to the eventual attacks that may occur within a protected perimeter. To verify attack impact, audit programs (System checks/audit scripts) are used and event associated to these programs are used as input to our language.

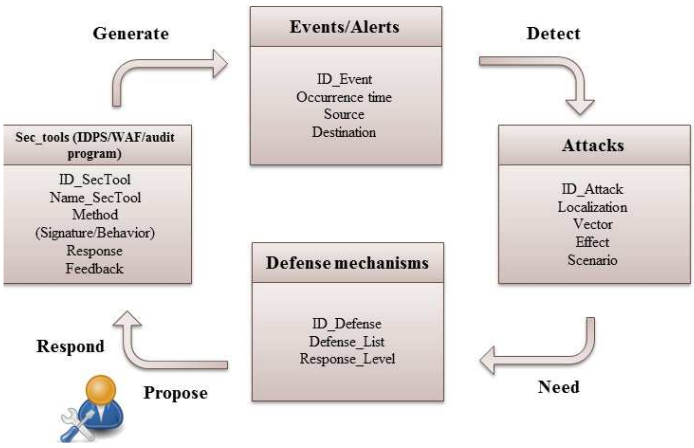

Fig. 1. Language conceptual model

All these parameters are described within our proposal, a combined language that allow system administrator to define easily his security policy.

C. Language Specification

The objective behind this work is to define a simpler formalism, to give it a high power of expression, to bring modularity to security controls and to be able to describe scenarios to detect complex attacks. The original idea is to define a two languages based formalism:

- Atomic Rule Language: contains single action rules. They can be transformation/normalization rules, log, alert, matching, control or active action rules.
- Composition Language: composes the atomic rules to define the scheme of rules to follow at the detection engine. They use logic, algebraic, correlation or synchronization operators.

The proposed language is used to treated events and alerts transformation/normalization, high level attack classification, eventually correlation when dealing with attack scenarios and defense mechanisms definition. It deals with events, event Sequences (order, repetition, non-occurrence, time constraints), constraints (contextual) and defense mechanisms.

Lua [13] will be used as a language framework to build to build detection and description rules. The 2 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… *Option*}.

The identified attack is resolved into “attack components”. These attack components are parameters indicating some aspect of the system, eventual malfunction or failure, affected by the attack. They are composed of various anomalies which are observed by sensors such as Firewalls, IDS.

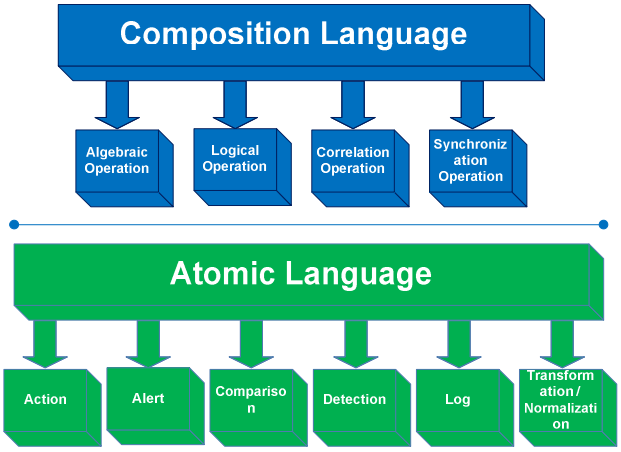

Fig. 2. Atomic and Composition languages components

1) Atomic Rules

We specify below the different types of rules that we define as atomic rules.

- 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)
- Normalization Rules: Norm_Rule (Event, Norm1), normalization according to predefined templates.
- Control Rules: Control_Rule (Event, normal behaviour, Score): controlling that the flow is anomalous. Control_Rule (Parameter1, verification): for contextual information
- Match Rules: Match_Rule (event, attack signature) /Match_Rule (parameter1, parameter2) : match 2 parameters: IP address,
- 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)

2) Composition rules

They define the way atomic rules are executed. For example O_Rule (Hook, R1 & R2) where a hook is the area where the control is done, it is a way 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 operations are useful when defining scenarios and can be used at the same time to create a more complete composition rule.

D. Potential applications

This language is used in our AIDD (Attack Identification, Description and Defense) architecture defined in [14]. This language may have several applications. In fact, when an alert occurs, it is not always possible to decide if the attack failed of succeeded. Our language helps check the success of the attack by observing the occurrence of the impact, as in IDS or WAF impacts are not reliable or complete (attacker hiding effects, or target not available). Besides, our language can be used to create high level IDS/WAF signature. Given a suitable correlation module various elements maybe used to check potential intruders. In addition, it allows end user to customize his own security with a defined perimeter with a set of intuitive language rules. Our proposal may be used to automate response within intrusion response systems.

In [10], it is said that by abstracting away from the details of a particular attack, it is possible to detect previously unknown variations of an attack or attacks that exploit similar mechanisms. Thus creating rule abstracting attacks may help detect new attacks. Fig. 3 shows the event format that we have for each attack. It is an IDMEF-like data format with appropriate fields related to our language.

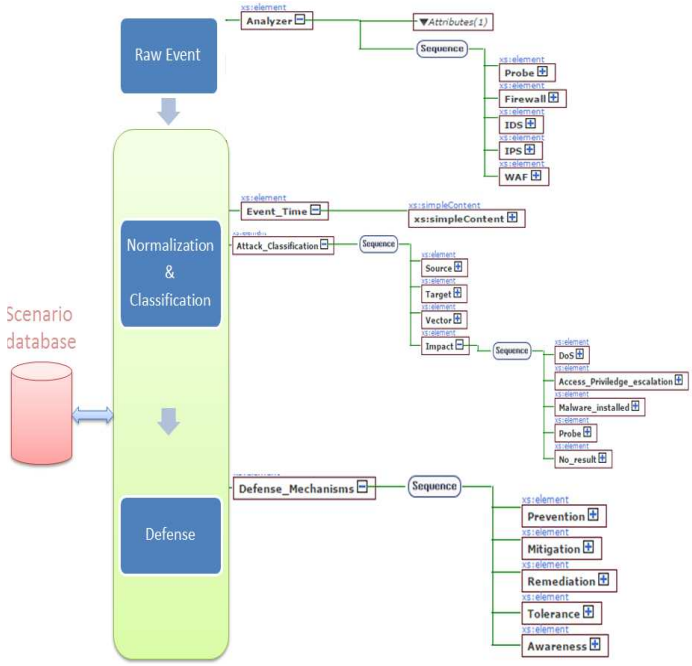

Fig. 3. Normalized Event Format

IV. USE CASE

In this section, in order to prove the feasibility of our proposition, we illustrate how our language helps to define rules in order to identify and classify attacks and offer responses. We highlight how it can bring a higher level of abstraction in order to better classify attacks. We take into consideration 2 types of attacks: a simple one with SQL injection and complex attack scenario. We use in these cases the AIDD system we defined in [14] for attack detection. This system uses our Composed Language to define rules.

A. Simple use case

In this use case, we consider a SQL injection attack and more specifically a tautology that leads to a user access and data information leakage. It is an attack on a web application. As the query is dynamically generated from user input, this takes place when the attackers maliciously craft this user input with SQL keywords and operator to execute a command on the database server. The structure of such a query is different from what is defined by the application code.

Let us consider in Fig.4 that the application contains two parameters: *login* and *password*. It uses these parameters for authentication. For example, if a malicious user enters the following input: “Admin” as a login and “toto' or 'x'='x” as a password. To make avoid eventual detection, as SQLi evasion mechanism, the attacker encodes the login in hexadecimal (0x41646d696e0d0a), and instead of putting “ ' ”, he uses char(39) encoding. The query string will be evaluated as true and admin access will be guaranteed for the attacker.

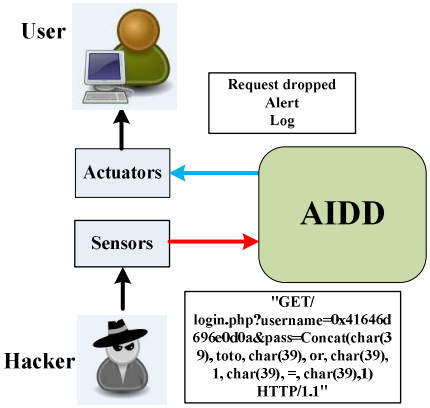

Fig. 4. Simple use case: SQL injection Attack scenario

This is a simple matching case. We use this case to show the basic detection and handling of the attack by our architecture and language. The steps followed by our system:

To describe the policy which is set up we use our combined rule sets. The atomic rules used are the following:

- R1 = Match_Rules (password, SQLi_chars,3), where SQLi_chars are regular expression for SQL injection pattern matching and 3 is the score assigned for the query
- R2 = Action_Rule (Log, SQLi attempt)
- R3 = Action_Rule (drop)
- R4 = Control_Rule (Version, ueq, 1)

An example of composition rule used is: O_Rule {H_URI, R1 AND R2, H_Version, R3 AND R4}, where H_URI is a hook at the URI level and H_Version the Version hook. This composition rule can be different as it expresses the policy defined by the user. Besides, this is a simple rule where we combine the different atomic rules to detect the attack. It does not show necessarily the added value of our language but it shows at least the mechanisms that simplify the user’s rule definition process.

These rules are not that static, as patterns and “signatures” can get updated through and online vulnerability database such as CVE [15], OSVBD [16], etc.

B. Complex use case

Our systems is able to classify complex blended attacks by subdividing each step, considering each step as an attack on its own and providing for each step defense mechanisms. This is done thanks to the AIDD model that helps define detected attack classes and composition rule schemes that helps define attack scenarios. Thus our system anticipates, stopping the attack before the occurrence of the final impact.

In this attack scenario (Fig.5), the attacker compromises a host to attack a target. From this host, he performs a scan to gather information about the targets. Then he exploits a misconfiguration to install a worm. This worm will help the attacker launch a buffer overflow attack leading to a resource misuse.

This use case is different from the previous one as the attack is more complex and has more than one phase to reach its final goal. Sensors can be in a network level or a host level. In Table I, this attack is split into 3 different phases according to our AIDD attack classification.

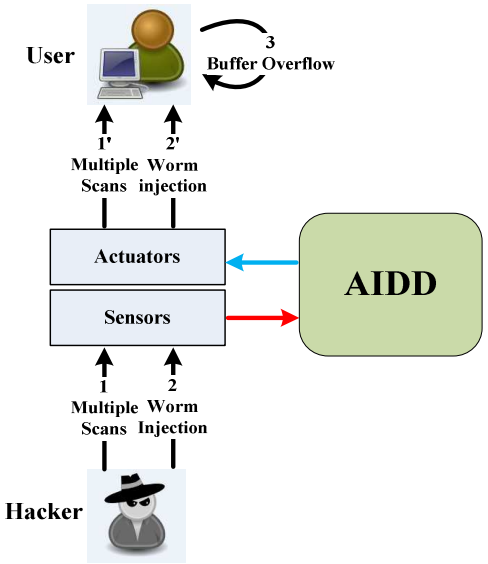

Fig. 5. Complex use case: Blended attack scenario

TABLE I. COMPLEX ATTACK USE CASE

Attack	Attack classes			
Phase 1	***Source***	***Target***	***Vector***	***Impact***
	Distant network	Network	Design flaw	Probe
Phase 2	***Source***	***Target***	***Vector***	***Impact***
	Distant network	Application	Misconfiguration	Malware installation - Worm
Phase 3	***Source***	***Target***	***Vector***	***Impact***
	Local	Application	BoF	Resource misuse

This scenario’s phases are treated separately and sequentially by our system and the answer is provided in an automatic intrusion response like manner. The different steps are as follow:

TABLE II. DETECTION / RESPONSE MATCHING

Detected attack attempt result	Response deployed
No	Log
Probe (information disclosure)	Log, warn, filter
Worm installation	Patch (fix bug), Reporting, Referencing (CVE- 2002-0649)
Resources misuse	Quarantine, Patch, Reporting

To detect the attack scenario we describe the set up policy using our combined rule sets. The atomic rules that can be used by the system administrator are the following:

- R1 = Norm_Rule (Events), normalizes the input flow, where the flow can be IP packets or IDS output.
- R2 = Control_Rule (Events, Scan), controls if a scan attempt is detected.
- R3 = Control_Rule ("Events"), detect if the flow contains anomalous behaviors to anticipate attack.
- R4 = Match_Rule (Event, "Worm signature/Behaviour"), detects a signature worm if signature or behaviours have not been changed by the attacker.
- R5 = Control_Rule (Host_IP, Buffer overflow), controls if a buffer overflow attack attempt has been launched within the target.
- R6 = Action_Rule (IP_Source, Tolerance), tolerates the actions done by the IP source.
- R7 = Action_Rule (IP-Source, Filtering), filters the flow coming from the IP source.
- R8 = Action_Rule (IP_Log), logs events happening at a defined host.
- R9 = Action_Rule ("Warning"), warns administrator (by email for example) of attack attempt.
- R10 = Action_Rule ("Reporting"), report and reference the vulnerability that was exploited to launch the attack (CVE ID for example).
- R11 = Action_Rule ("Vul_Patch"), helps do a patch and fix the bug at the origin of the attack.
- R12 = Action_Rule ("IP, Quarentine), puts the host with the according IP in quarantine (disconnect from the network).

We use these single simple rules to build the attack detection policy. An example of composition rules that can be used: O_Rule {H_Event1, R1 and R2 and R7 and R8 and R9, H-Event2, R1 and (R3 or R4) and R8 and R9 and R10 AND R11, H_Event3, R1 and R5 and R9 and R10 and R11 and R12}.

In this policy, the attack was subdivided into three phases with three different flows corresponding to each phase. We applied a combination of rule to match the appropriate defense mechanism with the detected attack. 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 use other detection device output as an input. Finally, with its composed language, writing security policies has become easier especially for non-security experts.

V. CONCLUSION

Up to now, few attack languages have focused on attack description from several angles (Adele, lambda), and according to our researches on the related work, no rule based languages provided a balance between easiness of use and high power of expression. In this paper, we have proposed a novel rule based language that combines these two criteria allowing extensibility and deduction. Our language helps users (system administrators) to combine several predefined blocks to build security rules. The originality is that it allows also generating rules from a certain behavior. The language is interesting to use when facing complex attacks and Advanced Persistent Threats as scenario predefinition has a great importance in deducing final attack strikes.

We have shown that our language allows to define security rules in a simple way allowing composition and complex attack description while offering appropriate defense mechanism pools. It is conceived in such a high level manner, it can be adapted to different contexts and devices. Our language is being implemented and can be ameliorated to handle encrypted information and metrics should be defined to enhance the attack-defense matching. The next step is to interface this language with other tools actively and reactively and to study its performances.

REFERENCES

[1] D. Vennila, .R.Nedunchezhian "Correlated Alerts and Non-Intrusive Alerts", International Journal of Soft Computing, 2012

[2] Samih Souissi, Ahmed Serhrouchni "AIDD: A novel generic attack modeling approach", HSPC conference, 2014

[3] G. Vigna, S. T. Eckmann, and R. A. Kemmerer "Attack languages", IEEE Information Survivability Workshop, 2000

[4] Snort IDS, http://www.snort.org

[5] Vern Paxson, "Bro: A System for Detecting Network Intruders in Real-Time", 7th USENIX Security Symposium, 1998

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

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

[8] F. Cuppens et R. Ortalo, "Lambda: A language to model a database for detection of attacks", the Third International Workshop on the Recent Advances in Intrusion Detection, 2000

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

[10] S.T. Eckmann, G. Vigna and R. Kemmerer, "Statl : An attack language for state-based intrusion detection", ACM Workshop on Intrusion Detection, 2000

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

[12] HAKA security project, http://www.haka-security.org/

[13] The programming language Lua, http://www.lua.org

[14] Samih Souissi, "Toward a Novel Classification-based Attack Detection and Response Architecture", NoF conference 2015

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

[16] Open Source Vulnerability Database, http://www.osvdb.org