Conceptual Analysis of Intrusion Alarms

Benjamin Morin and Hervé Debar

France Télécom R&D, Caen, France
{benjamin.morin, herve.debar}@rd.francetelecom.com

Abstract. Security information about information systems provided by current intrusion detection systems (IDS) is spread over numerous similar and fine-grained alerts. Security operators are consequently overwhelmed by alerts whose content is too poor. Alarm correlation techniques are used to reduce the number of alerts and enhance their content. In this paper, we tackle the alert correlation problem as an information retrieval problem in order to make the handling of alert groups easier.

1 Introduction

Security information about information systems provided by current intrusion detection tools is spread over numerous similar and fine-grained alerts. Security operators are consequently overwhelmed by alerts whose content is poor. As suggested by Jakobson and Weissman [7], alert correlation must both reduce the number of alerts and improve their semantics. In intrusion detection, alert correlation techniques can basically be split in two kinds of approaches.

The first kind of approach is a scenario recognition problem. Alarm sequences are matched on-line against the steps of attack scenarios written by experts in a dedicated language ; a high level alert is raised by the correlation system if a scenario is recognized. The approaches proposed by Cuppens and Ortalo [9] as well as Michel and Mé [6] fall into this category.

The second type of approach is a data analysis and clustering problem. Alarms are grouped together according to similarity measures in order to provide the security operator with a synthetic point of view of the security of the information system. The techniques proposed by Valdes and Skinner [11] as well as Cuppens [10] fall into this category. The main critique of this kind of approach is that the alert groups have no description. The number of alerts displayed to the operator is indeed reduced, but the operator still has to investigate the content of the alert groups.

In [8], Julisch proposes a correlation technique based on a variant of a data mining technique called *attribute oriented induction* (AOI). AOI consists in repeatedly replacing alert attributes by more abstract values, according to attribute taxonomies. This approach is interesting because the alert groups are qualified by the abstract values of the taxonomies. However, Julisch's approach has three drawbacks. Firstly, the *hierarchical* structure of the alert groups is restrictive because alerts only belong to one cluster. Secondly, the attribute taxonomies are trees, whose expressive power is limited (a value can only be generalized to another value). Lastly, the alert groups are not constructed incrementally.

In this article, we tackle the alert correlation problem as an information retrieval problem, in order to provide operators with an alert investigation tool for offline use. Information retrieval includes representation, storage, organization and access to information. More specifically, we apply the concept analysis paradigm [5], which is an information retrieval scheme.

In the remainder, we first briefly sketch concept analysis paradigm. Then we define the logic of alerts used to describe the alerts and also to query them. In section 4 we describe a prototype implementation based on a logical file system. Before concluding, we evoke experiences in handling alerts with our approach.

2 Concept Analysis

Existing alert management systems like ACID[1] display alerts in a web-based interface connected to a relational database, where alerts are stored. The security operator submits SQL queries *via* the web interface and the alert management system returns alert subsets which satisfy the query. The number of alerts returned can be huge, so the user has to refine the query in order to select a smaller set of alerts. Unfortunately, there is no simple relation between a variation in the query and the corresponding variation in the answer. The user has to infer an *intensional* modification (*i.e.* a modification of the query) from an *extensional* answer (a set of alerts). Instead of simply displaying the alerts to the operator, the alert management system should also suggest relevant modifications of a query which can be used to reduce the number of answers.

From an information retrieval point of view, this functionality is known as navigation. Information retrieval can be divided in two paradigms : navigation and querying. Querying consists in submitting a request to the information system, which in turn returns the objects that satisfy the query. Navigation consists in storing the information in a classification structure like a tree or a graph. Searching is done by navigating in a classification structure that is often built and maintained manually. We wish to take advantage of the navigation *and* the querying functionality. As shown by Godin *et al.* [4], Formal Concept Analysis (FCA) is a information retrieval paradigm that combines navigation and querying.

Given a *formal context* $K = (\mathcal{O}, \mathcal{A}, \mathcal{I})$ where $\mathcal{O}$ is a finite set of objects, $\mathcal{A}$ is a finite set of attributes and $\mathcal{I}$ is a relation between objects and attributes (*i.e* a subset of $\mathcal{O} \times \mathcal{A}$), formal concept analysis builds a *lattice of concepts* [5]. A *concept* (O, A) is a maximal set of objects $O \subseteq \mathcal{O}$ that share the same properties $A \subseteq \mathcal{A}$. The O part of a concept is called it's *extent* and the A part is called its *intent.* The fundamental theorem of formal concept analysis is that all concepts that can be built on a given formal context forms a complete lattice when it is ordered by set-inclusion of concept extensions.

[1] `http://www.andrew.cmu.edu/~rdanyliw/snort/snortacid.html`

$$\begin{array}{lll} P ::= & a \text{ is } v & a \in \mathcal{A}, v \in \mathcal{V} \\ \mid & q & q \in \mathcal{Q} \\ \mid & P \wedge P & \\ \mid & P \vee P & \\ \mid & \neg P & \end{array}$$

(a) Alert logic syntax

$$\begin{array}{lll} \mu \models a \text{ is } v & \iff & v \in \mu[\{a\}] \\ \mu \models q & \iff & \mu \models \nu(q) \\ \mu \models P \wedge Q & \iff & \mu \models P \text{ and } \mu \models Q \\ \mu \models P \vee Q & \iff & \mu \models P \text{ or } \mu \models Q \\ \mu \models \neg P & \iff & \mu \not\models P \end{array}$$

(b) Alert logic semantics

Fig. 1. Alert logic

The various application domains of FCA bring the need for more sophisticated formal contexts than the mere presence/absence of attributes. Indeed, in the intrusion detection context, we need to handle valued attributes (*e.g.* IP addresses) organized in a taxonomy in order to describe alert groups.

In [1], Ferré proposes a generalization of FCA, called Logical Concept Analysis (LCA). In LCA, sets of attributes are replaced by expressions of an almost arbitrary logic. This property satisfies our requirements.

3 Alert Description Logic

In order to apply LCA to intrusion alerts, we first need to define a description logic for alerts. The logic used to describe alerts is a multi-valued propositional logic, *i.e* a object-attribute-value logic, called $\mathcal{L}_A$. Multi-valued logic have been proved to fulfill the requirements of LCA [2].

The syntax of $\mathcal{L}_A$ is given in Figure 1(a). $\mathcal{A}$ is a set of attributes and $\mathcal{V}$ is a set of values, which are described thereafter. $T = (Q, \nu)$ is a terminology where Q is a set of qualifiers and ν is a mapping between the qualifiers and the formulas of $\mathcal{L}_A$.

Alarm qualifiers can be conceived as aliases for $\mathcal{L}_A$ formula. For example, any formula mapped with the `false_positive` qualifier defines which alert patterns should be considered as false positives.

$\mathcal{L}_A$ is both used as a query language *and* as a description language for alerts. However, alerts are only described by conjunction of attributes. Disjunction, negation and qualifiers are only used for queries.

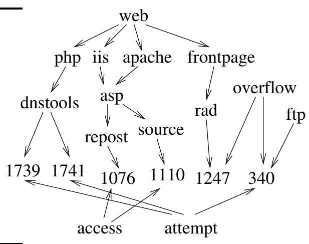
The semantics of $\mathcal{L}_A$ is given in Figure 1(b). In order to give the semantics of $\mathcal{L}_A$, an interpretation is first defined as a relation μ from the attributes $\mathcal{A}$ to the values $\mathcal{V}$ ($\mathcal{M}$ is the set of interpretations). Notation $\mu \models P$ means that interpretation μ satisfies formula. The entailment relation is defined as follows :

$$P \models Q \iff \forall \mu \in \mathcal{M} : \mu \models P \Rightarrow \mu \models Q$$

The taxonomic relations of the attributes domains are considered as axioms of the alert logic. If an attribute value a_1 is more specific than a_0, and an object x has property a_1, then it must also have property a_0. In logical terms this means that $d(x) \models a_1$ must entail $d(x) \models a_0$. This is achieved by considering the taxonomic relationships as axioms of the logic of alerts, *e.g.* $a_1 \models a_0$. For example,

1739	WEB-PHP DNSTools administrator authentication bypass attempt
1741	WEB-PHP DNSTools access
1076	WEB-IIS repost.asp access
1110	WEB-MISC apache source.asp file access
340	FTP EXPLOIT overflow
1247	WEB-FRONTPAGE rad overflow attempt

(a) Signatures documentation



(b) Attack taxonomy

Fig. 2. Attack taxonomy

the axiom "VICTIM is `www` $\models$ VICTIM is `web_server`" means that the host whose hostname is `www` is a `web_server`, so the requests which involve `web_server` should include `www` in the answer.

An alert has four attributes types : the attack, the attacker, the victim and a timestamp. Thus, $\mathcal{A} = \{\text{ATTACK}, \text{ATTACKER}, \text{VICTIM}, \text{TIME}\}$. The domains of the attributes are organized in taxonomies. Attribute taxonomies represent background knowledge about the attacks and the monitored environment which is used to enhance the content of the alerts and also to describe the alert groups with abstract values. In this approach, the taxonomies are modeled as directed acyclic graphs (DAGs).

Attack taxonomy. The attack taxonomy is made of attack identifiers and documentation keywords. Attack identifiers are the most specific values of the taxonomy (*i.e.* they are the leaves of the taxonomy). Documentation keywords provide information about the attack properties, such as the type of vulnerable target and the consequences of successful attacks. Relevant keywords are extracted from the plain-text documentation of the attacks by an expert. In our implementation, we used the Snort signature database to build this taxonomy.

Table 2(a) is an excerpt of Snort attack identifiers with their associated documentation ; Figure 2(b) illustrates how they are translated into the taxonomy. In this taxonomy, `cgi`, `php`, `iis`, `apache` and `frontpage` are all kinds of web-like attacks. `dnstools` and `hyperseek` are vulnerable cgi and php scripts, respectively. Numbers in the taxonomy are the signature identifiers; `access` or `attempt` keywords denote the severity of the alerts; `overflow` denotes an attack type (which is independent of the target type).

Victim taxonomy. The victim taxonomy is made of IP addresses, hostnames, host functions and domain names. Victims are referred to with their IP address in IDS alerts, so they are the most specific identifiers of victims in the taxonomy.

Hostnames represent an abstraction of IP addresses. This abstraction is necessary to handle cases where a host is related to more than one IP address. This is the case when a host has several network interfaces, or when the network address translation mechanism (NAT) is used in the monitored environment. For example, a web server can be referenced by its public IP address in an alert triggered by a sensor located in front of a firewall and by its private IP address in an alert triggered by a sensor inside the DMZ (*DeMilitarized Zone*) of an information system. Hostnames are also more suggestive than IP addresses from a security operator's point of view.

Internal network domain names are an abstraction of hostnames. Hosts (*e.g.* gateways) may belong to several domains. This abstraction level allows the security operator to know if an attack only targets a single host, or if it is a larger-scale phenomenon which involves several hosts of the same subnet.

Hosts functions are also an abstraction of hostnames ; they denote the role of a host in the network (*e.g.* FTP or HTTP server).

Attacker taxonomy. The top of the attacker taxonomy is divided into `external` and `internal` attackers. The internal attackers taxonomy is the same as the victims taxonomy.

Since little information is available about external attackers we use the IANA's[2] database netname field as an abstraction of external IP addresses. The IANA is responsible for the assignments of IP addresses ranges to organisms (ISPs, universities, corporations, etc.). The netname is basically a string that identifies the owner of a IP range. For example, the IP address `80.11.2.166` belongs to the IP range `80.11.2.0-80.11.2.255`, whose netname is `IP2000-ADSL-BAS`. A netname is generally more suggestive for a human operator than an IP address, and it identifies the source as an organism. As such, it offers natural grouping for alerts and differentiates global phenomenons from targeted attacks.

Time taxonomy. We are not interested here in the total order relationship of the time attribute. We want to be able to identify tendencies from an alert set. For instance, we want to know if a given phenomenon only occurs during off-work hours, or during the week-ends. Thus, timestamps are generalized to the hour of the day, and the day of the week. In turn, hours are generalized to `work-hour` or `off-hour` and day of the week are generalized to `week-end` or `week-day`.

4 Implementation

4.1 Prototype

Our implementation consists in storing the alerts in the LISFS logical file system proposed by Padioleau and Ridoux [3]. LISFS provides a generic framework that serves as a basis for any application that requires information retrieval.

In a hierarchical file system, files are located in a single place. Users access the file with their absolute or relative path, whose directories are separated with the `/` character. In LISFS, files are objects and paths are formulas of the logic used to describe the files. The `/` character denotes conjunction, so `propA/propB`

[2] Internet Assigned Numbers Authority, `http://www.iana.org/`

and `propB/propA` contain the same files, *i.e.* those whose description satisfies the `propA` $\wedge$ `propB` logical formula. The `|` and `!` characters respectively denote disjunction and negation in LISFS paths.

Conventional UNIX-like commands are used to query and navigate in the file system. `mkdir` creates a directory, *i.e.* registers a new attribute value. Taxonomic relations like $a_1 \models a_0$ are created using the `mkdir a1/a0` command.

The working directory, denoted by `pwd`, is the current query. `pwd` evolves incrementally, depending on the refinements requested by the user with the `cd` command : `cd p` adds the property `p` to `pwd`, which becomes `pwd` $\wedge$ `p`.

`ls` is used to consult the current place, *i.e* the list of files whose description satisfies the description of the current location. In fact, LISFS does not list *all* the files whose description satisfies the query ; only the files whose description cannot be further refined by any other property are displayed, *plus* the list of relevant properties that can be used to refine the current query. These properties are the navigation links. Moreover, the relevant properties are only the most abstract ones with regard to the taxonomies defined by the user.

StoreAlert	CreateProp
for each d_i **do**	**if** $\neg exists(d_i)$ **do**
CreateProp(d_i)	$D := \{d'_i \mid d_i \models d'_i\}$
done	**for each** $d'_i \in D$ **do**
`cp` m_a d_1/.../d_n/a	CreateProp(d'_i)
	done
	`mkdir` d'_1/.../d'_n
	done

Fig. 3. Alert storage procedures

In our implementation, each alert produced by an IDS is stored as a file in LISFS. Given an alert identifier a, its description $d(a) = d_1 \wedge \cdots \wedge d_n$ and its content m_a, procedure StoreAlert stores a in the logic file system (see Figure 3). The content of an alert is the raw message produced by an IDS, as well as the traces left by the attack, such as a network packet payload. The identifier of a file is a couple formed with the name of the sensor and a serial number. Command `cp` m_a d_1/.../d_n/a creates a file named a, with description d_1/.../d_n and content m_a. Alarm content can be consulted using the `cat` a command.

The CreateProp procedure adds the missing taxonomic properties entailed by the alert attributes in the file system. Predicate $exists(d_i)$ holds if a property d_i exists in the file system.

For example, if an IP address `192.168.10.80` appears as a victim in an alert, then properties "victim is `192.168.10.80`", "victim is `www`", "victim is `web-server`" and "victim is `dmz`" are added if `192.168.10.80` is a web server located in the DMZ and if these properties have not previously been added.

4.2 Experience with the Approach

The approach has been validated with a corpus containing about 10000 alerts produced by Snort sensors in an operational environment. Alerts are real in the sense that no attack has been simulated.

Submitting `ls -l` at the root of the file system returns 36 navigation links, and no alerts. The following list is an excerpt of the answer :

2182	`acker is int`	8702	`acker is ext`	7033	`vic is dmz`
298	`att is scan`	1	`att is smtp`	6	`att is overflow`
799	`att is snmp`	7039	`att is web`	526	`vic is web-proxy`

The numbers are the number of alerts whose description is subsumed by the associated formula (*e.g.* `att is scan`), *i.e.* the size of the concept extension. `acker`, `att` and `vic` respectively denote the attacker, attack and victim attributes.

Let us suppose that the operator is interested in the alerts whose attack is a web-like attack, and whose victim is a web proxy. The corresponding request is `cd "/att is web/vic is web-proxy"`. When consulting the current place with the `ls` command, the operator sees that among the links proposed by the system, the only one concerning the attacker is `acker is int`, which means that attackers only come from the inside.

Alerts description is subsumed by "ATTACK is `web` $\wedge$ VICTIM is `web-proxy` $\wedge$ ATTACKER is `internal`" are likely to be false positives. Indeed, web proxies forward HTTP requests of the internal users to the outside. However, from the point of view of an IDS monitoring the internal traffic, HTTP requests are addressed to the web proxy. Thus, a suspicious HTTP request would trigger an alert although it is addressed to the outside world, because the IDS thinks the web proxy is the victim of an attack. Since several IDSes signatures are based on the mere presence of known vulnerable web applications in urls (not actual attacks), many false positives are triggered. The operator can map the above alert pattern with the `false_positive` qualifier with the command `mv "/att is web/vic is web-proxy/acker is int" "/false_positive"`. This way, the operator may remove all false positives (`rm false_positive`) or select only alerts which are not known false positives (`cd /!false_positive`).

5 Conclusion

In this paper, we proposed an implicit alert correlation approach based on logical concept analysis. This paradigm allows the operator to handle alert groups in a flexible way, navigate through them and query the system. The alert groups are naturally described with the intent of their corresponding concept. The lattice structure allows intersecting alert groups. Moreover, the lattice construction in incremental.

In order to instantiate the LCA paradigm to the alerts, we proposed a logic of alerts. The vocabulary of the language allows abstract descriptions of the groups because to the alert attributes are structured in taxonomies. Our implementation consists in storing the alerts in the LISFS file system, which is an existing framework for information retrieval-demanding applications.

Experiences with the approach show that the handling of alerts is easier. The current implementation shows limitations in the time required to process alerts as the number of previously stored alerts grows, but newer versions of the logical file systems show very encouraging enhancements in terms of performances.

References

1. S. Ferré and O. Ridoux, A Logical Generalization of Formal Concept Analysis, *International Conference on Conceptual Structures (ICCS 2000)*, 2000.
2. S. Ferré and O. Ridoux, Introduction to Logical Information Systems, *IRISA Research Report RR-4540*, 2002.
3. Y. Padioleau and O. Ridoux, A Logic File System, *Usenix Annual Technical Conference*, 2003.
4. R. Godin, R. Missaoui and A. April, Experimental comparison of navigation in a Galois Lattice with conventional information retrieval methods. *International Journal of Man-Machine Studies*, 38 (5), pp. 747–767, 1993.
5. B. Ganter and R. Wille, Formal Concept Analysis - Mathematical Fundations, *Springer*, 1999.
6. C. Michel and L. Mé, Adele: An Attack Description Language For Knowledge-Based Intrusion Detection, *Proceedings of the 16th International Conference on Information Security (IFIP/SEC 2001)*, 2001.
7. G. Jakobson and M. D. Weissman, “Alarm correlation”, *IEEE Network Magazine*, 7(6), pages 52–60, 1993.
8. K. Julisch, “Mining Alarm Clusters to Improve Alarm Handling Efficiency”, *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC 01)*, December 2001.
9. F. Cuppens, R. Ortalo, “LAMBDA: A Language to Model a Database for Detection of Attacks”, *Third International Workshop on the Recent Advances in Intrusion Detection (RAID'00)*, H. Debar, L. Mé and F. Wu editors, LNCS 1907, 2000.
10. F. Cuppens, “Managing Alerts in Multi-Intrusion Detection Environment”, *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC 01)*, 2001.
11. A. Valdes and K. Skinner, “Probabilistic Alert Correlation”, *Proceedings of the 4th Recent Advances in Intrusion Detection (RAID2001)*, W. Lee, L. Mé and A. Wespi editors, LNCS 2212, pages 85–103, 2001.