5 Correlation of Network Alerts

The outcome of security monitoring reflects malicious activity that an IDS reports as alerts. An isolated view on these low-level IDS alerts, however, misses the context of the attack they belong to. The resulting incomplete view on attacks often renders an effective mitigation of the attack impossible. Thus, as part of the intrusion detection process as described in Section 3.2.1, alerts need to be summarized and correlated to obtain the bigger picture of an attack.

However, there are two particular challenges for reconstructing network-wide attacks from their alerts. First, the large alert volume overwhelms security operators when tying to identify relations among the alerts. Especially the big picture of distributed attacks can rarely get assembled when the relations of alerts from coordinated sources stay hidden from the security operators. Second, stealthy and comprehensive attacks, i.e., advanced persistent threats (APTs), additionally impede their reconstruction because of spatially and temporally distributed alerts. As a result, such stealthy attacks result in only a few alerts, while at the same time a large number of alert for conventional attacks are reported. Failing to link these alerts can result in the attacker to succeed without being noticed for a long time.

The alert correlation – the automated assembling of alerts to a descriptive summary of attacks – therefore, has to overcome the challenges of temporally and spatially dispersed alerts. Thus, the alert correlation algorithms presented in this chapter leverage different kinds of similarity among alerts and attacks, respectively. The correlation outcome provides the security operators with the attacks to their whole extend without analyzing every single alert themselves.

In particular, this chapter first defines and summarizes three stages of the alert correlation process in Section 5.1 for the processing of IDS alerts into attack representations. The subsequent two sections of this chapter present concrete correlation algorithms that implement these process stages. Algorithms of the first stage in Section 5.2 group all alerts that directly belong to the same attack or attack step. Section 5.3 presents algorithms that combine the second and third stage to link attacks with each other based on additional information about the attack scenario. More precisely, this chapter presents an aligned solution denoted as *graph-based alert correlation (GAC)* for clustering alerts from distributed attacks (cf. Section 5.2.1) and linking alert clusters from multi-step attacks (cf. Section 5.3.1). In addition, the *weak alert correlation* complements the clustering particularly with respect to APT alerts (cf. Section 5.2.2) and *collaborative attack correlation* assists in linking attacks detected in a distributed IDS deployment (cf. Section 5.3.2).

5.1 Alert Correlation Process for Attack Detection

Alert correlation algorithms ease the task of analysis alerts by correlating alerts with each other to obtain the bigger picture of an attack. A common approach is to group alerts based on similar attributes [ZLK09; Jul03; Vas+15a; Loc+05] like source and destination IP, and to report common attribute patterns (cf. Section 3.4.2). Other approaches correlate alerts of multi-step

attacks [NCR02; Sun+16] to identify and link the individual steps of an attacker, e.g., a port scan that is followed by an exploit of a specific vulnerability on the target host (cf. Section 3.6). Thus, alert correlation is the overall process to make relations among alerts visible, which can serve as indication of them belonging to a larger attack.

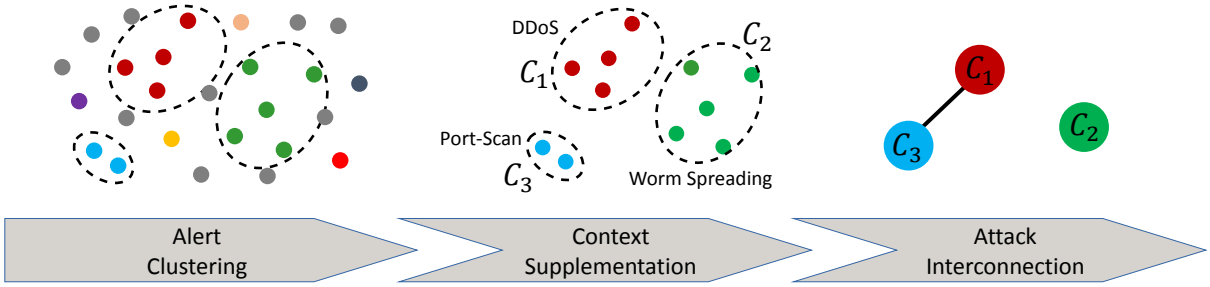


Figure 5.1: General alert correlation process with example for scenario labeling.

The alert correlation process converts a set of alerts into a representation of attacks. This correlation is an essential part of intrusion detection. However, alert correlation does not include intrusion detection and the reporting of alerts. Instead, alert correlation algorithms rely on sensors to classify malicious events as alerts (cf. the intrusion detection process in Section 3.2.1).

This section gives a unified description of the alert correlation process for transparent analysis and comparison of correlation algorithms. It is a revised version of the alert correlation process in the journal article [HF19]. The description characterizes alert correlation algorithms by breaking them down into their core building blocks (cf. Figure 5.1): *alert clustering*, *context supplementation*, and *attack interconnection*. Depending on the goal of specific alert correlation algorithms, individual blocks are less important or not necessary at all. If a step is not addressed, the input of a block is also its output. The following first introduces a formal model for the alert correlation process and then describes each building block in more detail.

IDS Network Alert An alert $a \in A$ can come from arbitrary sources like a network- or host-based IDS and indicates a potential security breach or generic malicious activity in the network. It can be either a true positive or a false positive, depending on the performance of the underlying intrusion detection. Every alert $a \in A$ consists of a fixed vector of attributes $a = (a^1, a^2, \ldots a^n)$, e.g., source and destination addresses, as well as source and destination ports. In case of network intrusion detection, the classification of malicious events as alerts is based on the network communication between hosts (cf. Chapter 4). Any network alert has a randomly assigned unique identifier *(UID)*, the timestamp *ts* of the network flow, the source and destination *IP addresses* and *ports* as well as the transport *protocol*. In addition to the event's attributes, the alert contains the *alert_type*:

$$a := (uid,\ ts,\ src\_ip,\ src\_prt,\ dst\_ip,\ dst\_prt,\ proto,\ alert\_type)$$

IDS Meta Alert An attack, or more precisely all alerts that are caused by the same malicious action, i.e., an attack i, will result in a set of true positive alerts S_i. The set of all alert sets from different steps is given by $\hat{S} = \{S_0, S_1, \cdots, S_{n-1}\}$. Applying alert correlation to this set reduces the alert volume and results in *meta alerts* that abstract and reference a set of at least one IDS alert. For that, a correlation function $\mathscr{K}$ clusters an alert set A into a set of clusters $\hat{C}$, with

each cluster $C_i \in \hat{C}$ being supposed to consist out of all alerts that represent a particular attack step. In contrast to an IDS network alert, a meta alert has its own assigned *UID*, the *time span* between first to last alert, the set of all *alert ids*, the set of attacker and victim *IP addresses* among all alerts, and a *message* that describes the attack. Thus, an IDS meta alert is the set of alerts triggered by the same attack action. It is the outcome of an alert correlation algorithm $\mathscr{K}$ that transforms an alert set A into clusters $C_i \in \hat{C}$:

$$m := (uid,\ ts,\ alert\_ids,\ attackers,\ victims,\ message)$$

A multi-step attack $M_j \subseteq \hat{S}$ contains several single-step attacks. The set of all multi-step attacks is $\hat{M} = \{M_0, M_1, \cdots, M_{m-1}\}$. The following paragraphs describe the tasks in the intrusion detection process as building blocks to achieve a clustering of alerts into clusters $\hat{C}$ and summarizing of these into multi-step attacks $\hat{M}$. Table 5.2 summarizes the notation that is used throughout this chapter to refer alerts when processed by the alert correlation process.

Correlation Stage				
Alert Clustering		**Attack Interconnection**		
$a_k \in A$	$S_i \in \hat{S}$	$l_i \in L$	$M_j \in \hat{M}$	Ground Truth
	$C_i \in \hat{C}$		$I_j \in \hat{I}$	Correlation Result
Alerts	**Alert Clusters**	**Context Labels**	**Attack Clusters**	
Symbol Name				

Figure 5.2: Notation used throughout the alert correlation process.

Alert Clustering The alert clustering will partition alerts into respective clusters of alerts $\hat{C} = \{C_0, C_1, \cdots C_{n-1}\}$. Each cluster $C_i \in \hat{C}$ is supposed to represent an attack and the alerts that belong to it. The alert clusters are supposed to model the actual attacks $S_i \in \hat{S}$, so in best case this leads to $\hat{S} = \hat{C}$. For that, two tasks will be carried out here: *Alert filtering* identifies duplicates as well as false positives among alerts and *attack isolation* clusters alerts that belong to the same attack.

Alert filtering takes care of filtering false positives, so that in the best case it holds true that:

$$\forall a \in A: \quad \exists S_i \in \hat{S} \wedge a \in S_i \iff \exists C_i \in \hat{C} \wedge a \in C_i$$

Please remember that the definition of *false positive* can depend on the context and the attacks to be detected, respectively. Anyway, this does not require a mapping from alerts to clusters with respect to attack steps. Instead, this task requires alerts to be assigned to a cluster and thus be included as input into the next building block if and only if they are induced by an attack in $\hat{S}$.

Attack isolation requires clustering to assign each alert out of set A to one cluster $C_i \in \hat{C}$. The correlated clusters $\hat{C}$ should reflect the original attack steps $S_i \subseteq A, S_i \in \hat{S}$. This task may vary from traditional clustering that aims for high homogeneity within clusters and high heterogeneity among the clusters. Hence, the challenge of clustering in the field of alert correlation is to find an assignment that reflects the reality, which might not be the optimal solution from a data mining point of view.

Context Supplementation After alerts have been filtered and clustered in the previous step, each alert cluster is supplemented with additional context. Such context could be information from knowledge databases, which provide information on vulnerabilities that might be exploited in a particular attack.

Another type of information could be a description of the attack. Thus, giving the clusters a meaningful label eases human analysis. A popular technique is to summarize alert features by finding common attributes in alerts, e.g., the source IP of the attack that might be present in all alerts related to a particular attack, and to suppress alert attributes that have high variance. Description of clusters such as counting the most frequent attribute values or value combinations is highly valuable for analysts. This might be the reason why most approaches for alert clustering tend to search for alert subsets that result in labels that are easy to understand by humans. Although using the most frequent attribute values works well in practice most of the time, this has some limitations as it imposes constraints on the attacks that can be detected.

For context supplementation, each cluster in $\hat{C}$ is labeled with a label $l_i \in L$ that gives additional information, e.g., environmental context or cluster description.

Attack Interconnection The last building block in the alert correlation process attempts to find relations among the alert clusters in $\hat{C}$. The resulting attack clusters $I_i \in \hat{I}$ reflect the assembling of single attack steps $\hat{S}$ into the multi-step attacks $\hat{M}$. An example is a scan for vulnerable web services in a subnet, followed by an exploit against a server in this subnet. To connect such attack steps, usually a combination of sequential- and causal-based correlation mechanisms is applied.

In worst case, each alert results in an own group during alert clustering. The second step enriches each alert with additional information, e.g., information on the vulnerability that is exploited. Then, most effort of the correlation algorithm lies in this third block of attack interconnection, as the definition of attacks is implemented in the dependencies between attacks.

5.2 Alert Clustering

The first stage in the alert correlation process (cf. Section 5.1) identifies alerts that are all caused by the activities or effects of the same attack step. In particular, two challenges exist in this process stage (cf. Section 1.1). First, there are distributed attack scenarios such as distributed denial-of-service (DDoS), port scans, and worm spreadings. These attacks cause a bulk of similar – almost redundant – alerts for the several affected hosts. Second, the occurrence of alerts from slow and stealthy APT-like attacks is a good deal more infrequent compared to alerts from bulk attacks. Consequently, the alerts' weak relations is likely to get lost in the shuffle. Alert clustering in this section analyses the alerts regarding similar features that indicate relations among the alerts and overcomes the two challenges in particular.

This section combines revised parts of the conference paper [HF18] and of the supervised master thesis [Ort19]. The section presents two alert clustering approaches for two different purposes. The first approach processes chunks of consecutive alerts, i.e., batches, and identifies related alerts within each alert batch. The second approach is an addition to the first one and clusters alerts of stealthy attacks that are likely to be filtered by the first approach.

5.2.1 Graph-based Community Clustering

The first approach for alert clustering represents alerts as nodes in a graph and adds edges between similar alerts. This enables the search for alerts sharing similar attributes and the clustering of alerts within these graphs afterwards. The following first describes the used graph model, then the clustering method.

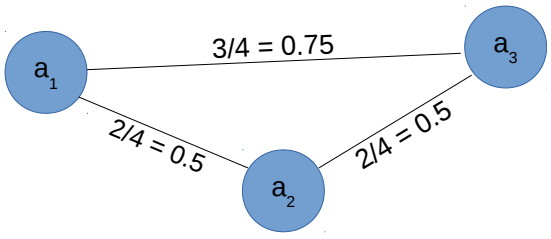
Transforming Alerts into a Graph The set of alerts A is transformed into a weighted *alert similarity graph* $G_{\text{attr}} = (A, E)$ that contains alerts of set A as nodes. Every edge $(a_1, a_2) \in E$ is weighted with the similarity $s = F_{sim}(a_1, a_2) \in [0, 1]$ in between two alerts $a_1, a_2 \in A$. Function F_{sim} compares all n attributes $(a^0, \ldots, a^{n-1})$ of the alerts, respectively, as follows:

$$F_{sim}(a_1, a_2) = \sum_{j=0}^{n-1} c^j \cdot h^j(a_1^j, a_2^j) \tag{5.1}$$

Per attribute, an attribute-specific comparison function h^j delivers a similarity value in $[0, 1]$. All attribute comparisons are weighted according to a vector $c = (c^0, \ldots, c^{n-1})$ such that $\sum c^j = 1$. The edge weight s determines whether a particular edge is present in the graph. The similarity between two alerts is required to be equal or higher than a minimum similarity threshold τ, for an edge (a_1, a_2) to be included in E^τ and G^τ_{attr}, respectively. Thus, τ controls the number of edges $|E|$, by removing edges between alerts that are most probably unrelated. The weight of edges depends on the implementation of F_{sim}. To choose a suitable threshold τ, it is beneficial to know the expected alert similarity among alerts of the attacks that should be detected.

	Source		Destination	
ID	**IP**	**Port**	**IP**	**Port**
a_1	E	X	W	P
a_2	T	Y	W	P
a_3	E	Z	W	P

(a) Example alert set.



(b) The resulting G_{attr} graph.

Figure 5.3: Transformation of an alert set into an alert similarity graph.

With the focus on network alerts, the most important and common attributes that should be supported by any network intrusion detection system (NIDS) and detection method are the four attributes: IP and port of both source and destination. Furthermore, attributes are tested for equal values to keep the correlation algorithm free from additionally required knowledge, such as subnets and the application type related to a port, e.g., Hypertext Transfer Protocol (HTTP) for ports 80 and 443. Thus, according to Equation 5.1, all h^j return 1 for equal attribute values and 0 otherwise. Attribute comparison is weighted equally with $c^j = 1/n$ for any $c^j \in c$. Other examples for attribute selection and h^j can be found in [VS01]. Another method to assign weights is described by so-called log-graph in [Pei+16]. Figure 5.3b shows the *alert similarity graph* with three nodes for the alerts in Figure 5.3a. E.g., $F_{sim}(a_1, a_2) = 0.5$ because two out of four attributes among the alerts a_1 and a_2 are equal.

Clustering Alerts within the Graph It is important to consider that usually several attackers try to break into an IT system at the same time. Therefore, it is very likely that individual attacks target the same host in the network without a connection between the attacks. One attack might aim to scan for SQL injection on a specific webserver, whereas a second unrelated attack scans for webservers in the same subnet. As both attacks hit the webport on the same server, they will probably be linked with a non-zero similarity even in the filtered graph G^{τ}_{attr}. Therefore, alerts of individual attack steps are further separated by clustering them, i.e., identifying subgraphs in G^{τ}_{attr}. The motivation for clustering is illustrated for an example graph G_{attr} in Figure 5.4. In the bottom one of the two isolated components, one can intuitively identify three loosely coupled subgraphs. These clusters (marked with red circles) are no isolated components, because a few of their alerts are similar to alerts of other clusters. These clusters are supposed to reflect individual attack steps. That some similarity exists between the clusters eventually indicates that they are related. Relations among the clusters, however, is out of scope at this processing stage.

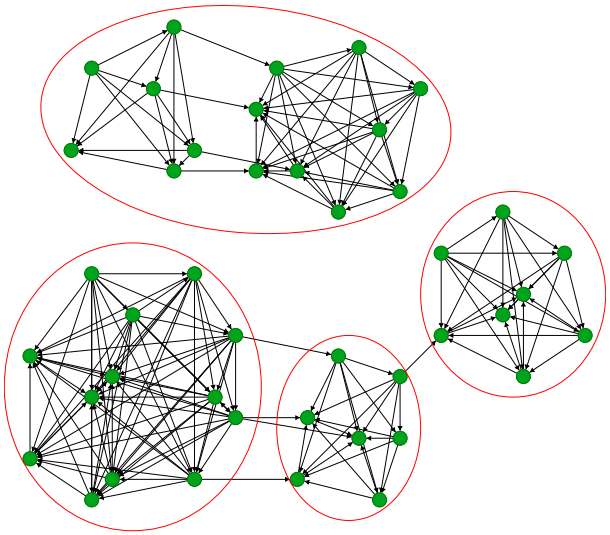


Figure 5.4: Community clustering in an exemplary alert similarity graph.

The alert clustering leverages the graph structure to identify and isolate subgraphs of loosely coupled attacks. For that, community clustering is used, especially the *clique percolation method* (CPM) [Pal+05], to cut the connection between loosely coupled clusters. CPM detects communities by searching for k-cliques, i.e., fully connected subgraphs of size k, that share $k-1$ nodes. This correlates well with the homogeneous inter-connections among alerts in the G_{attr}. These result, e.g., from distributed attacks, which cause several alerts with similar attribute patterns. Furthermore, k-clique communities can also reflect uncertainty in clustering as it allows to assign a node to several communities, i.e. clusters. Although clusters then potentially contain alerts of unrelated attacks as well, they will more likely include all true positive alerts.

This clustering performs well, especially on bulk attacks that cause many alerts roughly at the same time. Infrequent alerts from stealthy attacks, however, are likely to fall into separate alert batches, for which this graph-based alert clustering fails. Such alerts are incorporated into clustering by the following approach.

5.2.2 Weak Alert Correlation

In addition to the bulk attacks detected by the graph-based community clustering of alerts from Section 5.2.1, the following introduces the notion of *weak alerts* that are likely to result from stealthy attacks such as APT attacks. Thus, instead of filtering those alerts as irrelevant that do not assemble to a cluster immediately, the *weak alert correlation* approach can cluster these alerts over long time instead. For that, the approach first groups weak alerts to intermediate aggregations. This data structure enables runtime efficient updates. Furthermore, the aggregation algorithm runs continuously and is not restricted to individual alert batches. Once an aggregation becomes stable, the aggregation result is turned into a *weak meta alert*, similar to an alert cluster.

5.2.2.1 Weak Alert Model

Compared to the number of alerts from bulk attacks, alerts from stealthy attacks are rare. Because of their infrequent occurrence, they seem unrelated to any other alerts, although related alerts in historical data exist. This definition of *weak alerts* is detailed in the following model.

Weak Alert Definition After clustering alerts in an alert set A with a correlation function $\mathscr{K}$, some alerts remain unclustered, i.e., they are not included in any meta alert m_i or belong to any cluster C_i, respectively. Thus, an alert is considered weak, if it cannot be correlated with other alerts and therefore cannot be related to a specific attack. Thus, weakness is a relative descriptor for single alerts, denoted as binary function $\omega : a \in A \mapsto \{0,1\}$:

$$\omega(a) = \begin{cases} 0, & \text{if } \exists C_i \in \mathscr{K}(A) \wedge a \in C_i \\ 1, & \text{otherwise} \end{cases}$$

This results straight into the definition of what the set of weak alerts W is. It is with respect to an alert set A that has the correlated alerts $CA = \cup_{\hat{C}}$, i.e., the union of all clustered alerts among $C_i \in \hat{C}$. The set of all *weak alerts* is defined as the set difference of all IDS alerts minus the correlated alerts:

$$W = A \setminus CA, \text{ with } |W| = |A| - |CA|$$

Not all weak alerts in W necessarily belong to the same stealthy attack. Thus, this model further introduces two characteristics that are used to separate weak alerts from different attacks.

Large IP networks are usually organized in subnets. Reasons for that include to limit broadcast domains, but also to secure subnets of different trust levels, i.e., zones, by measures like firewalls. Especially in APT attacks, lateral movement in the network of an organization is often seen to access data or services that are not directly accessible from the Internet. To reflect this characteristic of respective attacks, the network is assumed to be organized in zones, e.g., the `Internet`, `Intranet`, or `data center`. The direction between two network zones `A`, `B` is denoted with an arrow symbol as $\texttt{A} \rightarrow \texttt{B}$. Consequently, the *alert direction* of an alert a is given by the network zones of its source and destination IPs:

$$Dir(a) := \texttt{A} \rightarrow \texttt{B} \text{ iff. } a.src\_ip \in \texttt{A} \wedge a.dst\_ip \in \texttt{B}$$