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.

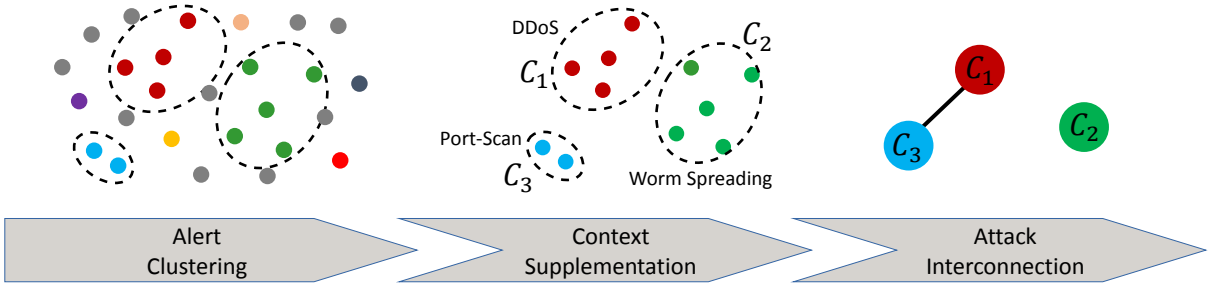


Figure 5.1: General alert correlation process with example for scenario labeling.

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.

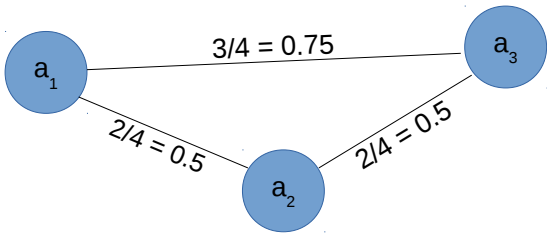
Transforming Alerts into a Graph The set of alerts A is transformed into a weighted *alert similarity graph* $G_{\text{attr}} = (A, E)$ that contains alerts of set A as nodes. Every edge $(a_1, a_2) \in E$ is weighted with the similarity $s = F_{sim}(a_1, a_2) \in [0, 1]$ in between two alerts $a_1,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

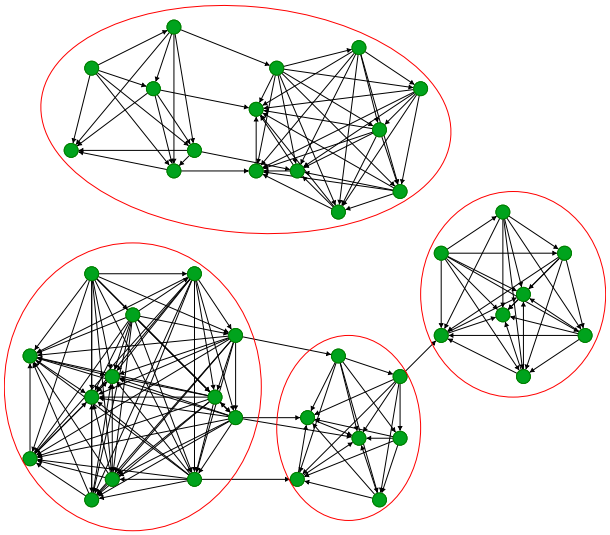


Figure 5.4: Community clustering in an exemplary alert similarity graph.

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:

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

Another attack characteristic is how many attackers and targets are involved in the attack. Especially distributed attacks with many involved system potentially cause many alerts. Thus, the relations among attackers and targets and their numbers, i.e., the *attack topology*, characterizes the attack type. The weak alert model distinguishes four topologies:

- One-To-One `(OtO)`: Single attacker and target
- One-To-Many `(OtM)`: Single attacker and multiple targets
- Many-To-One `(MtO)`: Multiple attackers and single target
- Many-To-Many `(MtM)`: Multiple attackers and targets

Similar to the correlation of IDS alerts to meta alerts (cf. Section 5.1), weak alerts are filtered and correlated to reveal their relations to APT attack steps. The outcome are *weak meta alerts* that have the fields *label* for the attack topology, the *alert direction* of two network zones, and the *alert frequency* in addition to a regular IDS meta alert. The attack topology and alert direction characterize the APT attack regarding lateral movement and the alert frequency is an indication for the stealthiness of the attack. Thus, a weak meta alert is the set of weak alerts triggered by a potentially stealthy and long-lasting attack:

$$w_{ma} := (uid,\ ts,\ label,\ direction,\ alert\_ids,\ attackers,\ victims,\ freq)$$

The problem that arises in practice is that the correlation function $\mathscr{K}$ usually runs on a finite alert set A. Thus, $\mathscr{K}$ is only applied on consecutive alert subsets, i.e., batches, reflecting a certain time span. Especially alerts of temporally dispersed attacks probably spread into several alert batches. However, unfavorable circumstances can also cause the isolation of few alerts from bulk attacks to fall into a separate batch, eventually being classified as weak alerts.

Sensitivity and Specificity of Weakness Weak alerts that result from filtering are not necessarily weak when considering multiple batches. Figure 5.5 illustrates five possibilities how weak alerts might relate to other batches. Batches are marked gray, the colored dots depict alerts. Green alerts are successfully correlated, blue alerts are classified as weak. For lanes 1-3, a small portion of alerts that belong to the same attack fall into another batch. Their weak classification is false positive, as they are actually not weak across multiple batches. Ideally, it is left up to the correlation algorithm $\mathscr{K}$ to compensate such cases or alternatively tolerate them in the generation of weak meta alerts. The more interesting cases for APT detection are the true positive classifications in lanes 4-5. Here, alerts for one attack are very low in volume and spread over multiple batches, or it is a single alert (or very small group) which might either be a rare outlier or belong to a single shot attack, like a malicious one-time download.

For traditional alert correlation with $\mathscr{K}$, false negatives, i.e., relating not every relevant alert to the attack (lanes 1-3), might be accepted as long as the attack is actually detected. However, accepting false negatives for correlating weak alerts (lanes 4-5) might likely result in the APT attack to go unnoticed. Thus, related weak alerts must be correlated, even if they are temporally dispersed. But in contrast to $\mathscr{K}$ that performs on a series of finite alert batches, weak alerts have to be correlated in a steady stream of new alerts, i.e., across batches. Thus, in the context of stealthy and long-standing APT attacks, weak alerts have to be identified and continuously be assembled to attack steps.

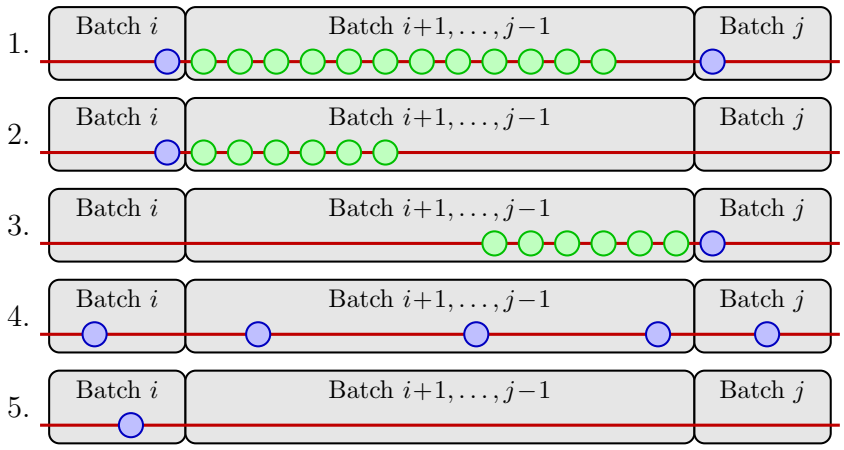


Figure 5.5: Five example cases how the spreading across batches for alerts of an attack results in different classifications regarding weakness.

5.2.2.2 Weak Alert Aggregation

According to the filtering of weak alerts in Section 5.2.2.1, a correlation function $\mathscr{K}$ identifies related alerts in an alert set A that are grouped into clusters. It leaves unclustered alerts as weak alerts W. This correlation function is assumed to run on batches of alerts, which might results in a few weak alerts per batch. However, the challenge is to correlate weak alerts across many batches. For that, the following describes the continuous aggregation of weak alerts.

Characterizing Weak Alerts In the context of APT attacks and the characteristic of lateral movement (cf. Section 2.1.2), alerts that encompass two different network zones are of particular interest. Respective alerts are identified utilizing a *host- & zone-communication graph (HZCG)* as shown in Figure 5.6. The nodes in such a directed graph are source and destination IP addresses of weak alerts, annotated with the respective zone the IP belongs to. The edges indicate the alert direction from one to another zone and summarize the alerts between the two respective IP addresses. In the illustrated example, there are four attackers within `Zone_1` and two targets within `Zone_2`. The targets partly overlap, as the attackers 172.17.0.1 and 172.12.0.2 both attack the two targets, while attackers 172.17.0.3 and 172.17.0.4 each attack a single target only.

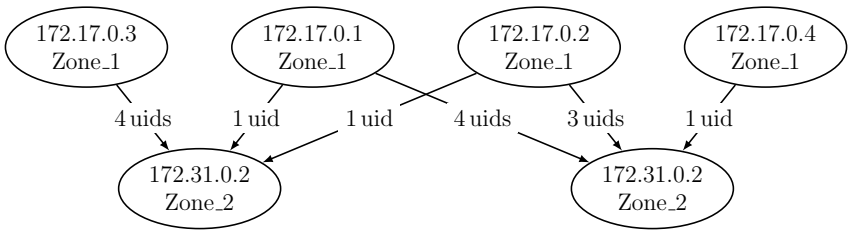


Figure 5.6: Example for weak alerts between two network zones in a HZCG.

Another fundamental characteristic of attacks is whether they are distributed, both in sense of attackers and targets. An attack either encompasses exactly two IP addresses or there are multiple addresses for attackers or targets. However, in the context of APT attacks, noisy attacks with multiple addresses for attackers and targets are likely to cause enough alerts to be reported by traditional approaches. Because of that, the weak alert aggregation focuses on the attack

topologies `OtO`, `OtM`, and `MtO` only (cf. Section 5.2.2.1). The aggregation aims to identify attacks following these patterns among the weak alerts in the HZCG. However, as probably several unrelated attacks are going on from and to different zones as well as false positive weak alerts exist, additional attention has to be paid to the separation between zones and individual attacks.

Aggregating Weak Alerts The aggregation function itself can be thought of as a graph-based *group-by* function on the HZCG. Each node in the graph is checked and its incoming and outgoing edges are iterated separately. Neighbors of each node are grouped by their zone. For example, Figure 5.6 shows that 172.17.0.1 has five alerts to `Zone_2`. The respective neighbor group of node 172.17.0.1 includes the two nodes with addresses 172.31.0.2 and 172.31.0.1 as targets. More specifically, a *neighbor group* consists of objects, each summarizing a neighbor by its IP address and a count of associated alerts:

$$neighbor\_summary := \left\{ \begin{array}{c} \text{IP} : IP\_Address \\ \text{Alerts} : count(alerts) \end{array} \right\}$$

In the example HZCG in Figure 5.6, six *neighbor groups* exist, one for each node. Finally, *neighbor groups* are accumulated per zone to so-called aggregations. An aggregation *agg* carries information about the *time_span* between the first and last alert, the *direction* of the two zones, and topology *label* (`OtO`/`OtM`/`MtO`). Furthermore, an aggregation describes *attackers* and *targets* in the form of neighbor groups but at least one of them consist of a single object only:

$$agg := (time\_span,\ direction,\ label,\ attackers,\ targets)$$

New neighbor groups are calculated whenever a HZCG is constructed, e.g., for each run of the correlation function $\mathscr{K}$ on the next alert batch. Then, they must be merged into existing aggregations. This continuously relates weak alerts across batches to the same attack while it is ongoing. But also neighbor groups within the same batch can be condensed to fewer aggregations.

Inserting Aggregations When a new neighbor group is created, it is directly converted to an aggregation. The collection of aggregations is referred to as database in the following. Any new aggregations have to be inserted into the database of existing aggregations one after another. The pseudocode for the insert function is shown in Algorithm 1. It takes an aggregation as input and returns either the unmodified input, in case no existing and matching aggregation could be found in the database, or returns the merge-result of the input and an existing aggregation object from the database.

Algorithm 1 consists of two outer conditional blocks in Lines 2 and 9. The database is queried depending on the label field of the input aggregation. One-to-one labels must be considered twice. In case of a *one-to-X* label, the database is queried for a single attacker IP (Line 3). Likewise, in case of an *X-to-one* label, the database is queried for a single victim IP (Line 10). A merge is performed if a query result is not empty. The respective field which holds many IP addresses is merged (Lines 4 and 11). The label field is updated accordingly to represent the *many* relation (Lines 6 and 13). The algorithm immediately returns the result whenever a merge is possible. Otherwise, the new aggregation object is inserted into the database without modifications (Line 16).

```
Input: agg: Aggregation, DB: Database with existing Aggregations
Output: Inserted Aggregation
1 begin
2   if agg.label = 'one-to-one' || agg.label = 'one-to-many' then
3     existing_agg ← DB.query(
                                  label = ['one-to-one' OR 'one-to-many']   AND
                          attackers.IPs = agg.attackers.IPs                 AND
                              direction = agg.direction);
      if existing_agg ≠ Null then
4       existing_agg.victims.update(agg.victims);
5       if existing_agg.victims.size ≥ 2 then
6         existing_agg.label ← 'one-to-many';
        end
7       return existing_agg;
      end
8   end
9   if agg.label = 'one-to-one' || agg.label = 'many-to-one' then
10    existing_agg ← DB.query(
                                  label = ['one-to-one' OR 'many-to-one']   AND
                          victims.IPs = agg.victims.IPs                     AND
                            direction = agg.direction);
      if existing_agg ≠ Null then
11      existing_agg.attackers.update(agg.attackers);
12      if existing_agg.attackers.size ≥ 2 then
13        existing_agg.label ← 'many-to-one';
        end
14      return existing_agg;
      end
15  end
16  DB.new_agg(agg);
17 end
18 return agg
```

Algorithm 1: Insert function to form aggregations.

The example in Figure 5.6 results in four aggregation objects after inserting the six neighbor groups according to Algorithm 1. Two `MtO` aggregations exist for the targets 172.31.0.2 and 172.31.0.1, and two `OtM` aggregations exist for the attackers 172.17.0.1 and 172.17.0.2. Event though a target is attacked by multiple attackers or an attacker attacks multiple targets, it does not necessarily mean that all the respective alerts belong to the same attack. Hence, a last step is performed by separating individual attacks among the weak alerts to generate weak meta alerts.

5.2.2.3 Weak Meta Alert Generation

The aggregation procedure groups all those alerts that share the exact same network direction and label. However, the aggregated neighbor groups grow naturally with more weak alerts. The aggregation does not highlight any relations among the weak alerts. Thus, the aggregation process on its own cannot be used for a correlation. The following describes how to leverage the aggregated groups to inspect each aggregation object and to extract smaller clusters of alerts (still in a certain network direction) that are of more interest than others.

The basic idea of clustering weak alerts within an aggregation is to group attackers and targets by their alert frequency. Intuitively, this identifies false positive weak alerts that are only considered weak because of their disadvantageous distribution among alert batches. During aggregation, they eventually stack up and form a large group across batches. However, even true positive weak alerts might falsely be contained in such a stacked-up aggregation group over an extended period of time. They can still be identified and extracted by their lower frequency.

To identify groups of alerts with different frequencies, density-based clustering is leveraged. Such clustering algorithm benefit from how their parameters are used. In contrast to other unsupervised clustering techniques, a density-based algorithm like density-based spatial clustering of applications with noise (DBSCAN) does not require any parameterization regarding cluster sizes or amount of clusters. Instead, DBSCAN is able to extract clusters from an unknown corpus, solely based on the density of the appearance in the search space. The parameters of interest are min_pts and eps. These parameters define how many points are at least required to form one cluster (min_pts) and how big the distance between two points can be at most for them to be grouped to the same cluster (eps). Applied to the weak alert aggregations, the distance between two neighbors is defined by the difference between their alert counts in the respective neighbor summaries. This can be seen as a one-dimensional space in which for each IP address one data point, with the value of the alert count, exists. Intuitively, clusters with the lowest alert frequency might be the most stealthy ongoing attacks. To detect even the extreme stealthy attacks, alerts are not filtered by clustering but should end up in clusters with at least one alert. The pseudocode in Algorithm 2 generates weak meta alerts based on input aggregations.

The generation function starts by iterating all aggregation objects at the beginning of Algorithm 2 in line 2. The algorithm body is divided into three major blocks. DBSCAN has to run with different arguments, depending on the label of the currently iterated aggregation. In case it is a *one-to-many* label (Line 4) the victims form the *many* entity are subject to density-based clustering (Line 6). Similarly, the attackers are subject to density-based clustering in case of *many-to-one* relations (Lines 18 and 20). A new weak meta alert is generated for each cluster that results from DBSCAN. Therefore, the label is re-calculated based on the actually involved IP addresses (Lines 8-11 and 22-25). Similarly, attackers and victims are re-calculated in Lines 12-13 and 26-27. The frequency calculation is defined straightforward. The number of involved IP addresses in each cluster is divided by the total number of IP addresses in the original aggregation object (Lines 14 and 28). Lines 32-37 show the special case that the aggregation was already labeled with *one-to-one*. It is not possible to break down the frequency relations of the involved IP addresses any further.

```
Input: Aggs: List of Aggregations
Output: W_MA: List of Weak Meta Alerts
begin
    for agg in Aggs do
        dir ← agg.direction;
        if agg.label = 'one-to-many' then
            total_alerts ← agg.attackers.Alerts.size();
            clusters ← DBSCAN(agg.victims);
            for cluster in clusters do
                label ← 'one-to-one';
                if cluster.IPs.size() ≥ 2 then
                    label ← 'one-to-many';
                end
                attacker ← agg.attackers[0].IP;
                victims ← cluster.IPs;
                freq ← cluster.Alerts.size()/total_alerts;
                w_ma ← (label, dir, alerts, attacker, victims, freq);
                W_MA.add(w_ma);
            end
        else if agg.label = 'many-to-one' then
            total_alerts ← agg.victims.Alerts.size();
            clusters ← DBSCAN(agg.attackers);
            for cluster in clusters do
                label ← 'one-to-one';
                if cluster.IPs.size() ≥ 2 then
                    label ← 'many-to-one';
                end
                victim ← agg.victims[0].IP;
                attackers ← cluster.IPs;
                freq ← cluster.Alerts.size()/total_alerts;
                w_ma ← (label, dir, alerts, attackers, victim, freq);
                W_MA.add(w_ma);
            end
        else
            label ← 'one-to-one';
            victim ← agg.victims[0].IP;
            attacker ← agg.attackers[0].IP;
            alerts ← agg.attackers[0].Alerts;
            freq ← 1.0;
            w_ma ← (label, dir, alerts, attacker, victim, freq);
            W_MA.add(w_ma);
        end
    end
end
return W_MA
```

Algorithm 2: Generating weak meta alerts.

5.2.3 Summary of Alert Clustering

To reduce the alert volume, two complementing alert clustering algorithms have been presented that isolate alerts from different attacks and summarize respective alerts as meta alerts. These meta alerts represent individual attacks together with all the alerts they caused. The identification of related alerts in clustering leverages that alerts from the same attack are likely to have some equal feature values. This is especially true for the network-wide attacks such as distributed attacks with a lot of similar malicious activity.

For the detection of such bulk attacks, this section has presented a graphed-based algorithm that transforms the alerts into a *alert similarity graph*. Alerts are presented as nodes in this graph, and edges indicate a potential relation between two alerts that is determined by the similarity between the alerts' features. Within this graphs, community clustering is performed to identify well-connected subgraphs. Each subgraph is then supposed to represent the alerts of an individual attack. The accuracy of isolating alerts from different attacks is evaluated in Section 6.5.1.2 and tested on real-world data in Section 6.5.2.2.

In contrast to bulk attacks, where alerts are likely to be clustered by the graph-based algorithm, stealthy APT-like attacks cause infrequent alerts that get lost in the shuffle. While the alerts from bulk attacks form well-interconnected subgraphs that become apparent because of their size, the visibility of relations among the infrequent alerts seems weak, which is why they are denoted as weak alerts. These weak alerts are unlikely to fall in the same alert batch because of the large alert volume from other attacks in between. Consequently, the graph-based algorithm on the basis of alert batches cannot cluster the weak alerts in the first place.

The clustering of weak alerts, therefore, requires a continuous processing of alerts across batches. This section has presented a weak alert correlation algorithm that consumes alerts that remain unclustered by the graph-based alert clustering. The algorithm continuously aggregates these weak alerts until they assemble to a weak meta alert. Thus, the algorithm first transforms the unclustered alerts of every new batch into a *HZCG* that summarizes the weak alerts regarding their IP addresses and network zones. Across alert batches, the algorithm maintains aggregations of weak alerts that are updated with every new graph. Based on the continuous aggregation, the relations among IP addresses from the weak alerts become apparent as more and more related alerts are aggregated during the stealthy and long-lasting APT attack. When this is the case, density clustering is performed to extract the alerts that assemble the pattern of an APT attack step. Section 6.6 investigates to which extent the clustering of weak alerts can supplement the graph-based clustering to identify stealthy attack steps.

5.3 Attack Interconnection with Context

The second and third stage in the alert correlation process (cf. Section 5.1) supplement the clustered network alerts from Section 5.2 and link the alerts from different attacks and their steps, respectively. This linking is required, because an isolated view on the clustered alerts eventually underestimates the network-wide attack. In fact, two clusters might actually should be seen together because of either of the two dimensions: First, an attacker performs multiple attacks against different network areas step-by-step to achieve the overall attack goal. Second, an attacker widely distributes the same attack, e.g., to different network sites with individual IDSes, such that no IDS captures the full attack but only parts of it.

To assemble attack relations despite these challenges, attack interconnection in this section leverages additional context. Context supplementing for the approaches here is based on characterizing the who-targets-whom structure as a representation of the attack scenario. In addition to the concrete feature values among the alerts, interconnection benefits from this value-independent characterization of the attack scenario. This context is derived from the alerts themselves and does not require an external knowledge base.

This section combines revised parts of the two conference papers [HF18] and [HWF19]. The section presents two attack linking approaches for two different purposes. The first approach identifies multi-step attacks, in which the steps build upon each other by exploiting one or multiple targeted hosts to make use of them in the subsequent step. The second approach enables the collaboration of different network sites by identifying that two sites have potentially become a victim of the same attack.

5.3.1 Graph-based Multi-Step Detection

The first approach identifies attack steps that are part of a multi-step attack. The approach first derives additional context about the attack scenario from the alerts of each attack step. The attack scenario is afterwards utilized to assemble multi-step attacks based on equal IP addresses between alerts from different attack steps.

5.3.1.1 Scenario Identification

The supplementing attack scenario is derived from the clusters of alerts that represent individual attack steps. The goal is now to characterize the attack scenario of each cluster. For that, the communication patterns between attackers and victims are determined. Especially distributed attacks such as DDoS, port scans, and worms are of interest here, as they involve many systems at the same time and thus result in a large number of different alerts.

Communication Topology Most alerts in a cluster are usually similar to each other and thus well interconnected in G_{attr} (cf. Section 5.2.1). Hence, such a graph structure is inappropriate to identify attack scenarios. Thus, per identified cluster $C_i \in \hat{C}$, a less dense *alert flow graph* $G_{\text{flow}} = (V, E)$ is established with the nodes V to be the set of all source and destination IP addresses in a cluster C_i. An edge $(u, v) \in E$ in the graph between two nodes $u, v \in V$ exists, if there is an alert $a \in C_i$ with the source node u as attacking IP and the destination node v as target IP. The direction of an edge therefore indicates who attacked whom.

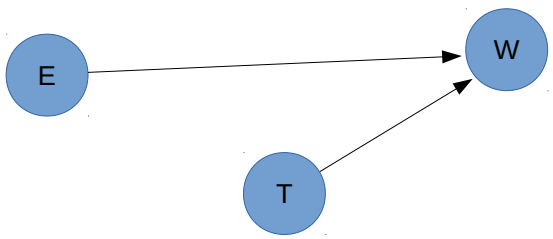


Figure 5.7: An exemplary alert flow graph for three hosts.

Figure 5.7 shows the graph G_{flow} for the alerts in Figure 5.3a. In this example, the three alerts in the graph G_{attr} in Figure 5.3b are assumed to belong to the same cluster. G_{flow} contains the IP addresses E, T and W as nodes and edges from E and T to W, as the node with IP address W is attacked by nodes with the addresses E and T.

To identify attack scenarios, the nodes' degree in $G_{\text{flow}} = (V, E)$ is used in a scheme for distributed attacks involving either multiple sources, destinations, or both. This scheme characterizes nodes $v \in V$ as follows: An *attacker* has an outgoing degree ≥ 1 and a *victim* has an incoming degree ≥ 1. Moreover, in distributed attack scenarios and multi-step attacks, a node can be attacker and victim at the same time. Four attack scenarios are distinguished and further summarized in Table 6.12:

- One-to-One `OtO`: One source is attacking a single destination, which is a special case of the other scenarios.
- One-to-Many `OtM`: One source is attacking multiple destinations, e.g., scanning a subnet. Characteristic for all alerts is the same attacking source IP.
- Many-to-One `MtO`: Multiple sources are attacking a single destination, e.g., in a DDoS attack. When attacking a specific service, all related alerts might have the same destination IP and port.
- Many-to-Many `MtM`: Multiple sources and destinations are involved, e.g., like in a worm spreading. Since such worms spread by targeting a specific application, alerts in this scenario will have at least the same destination ports.

Topology Heuristics To identify the scenario of a given attack, four metrics describe how good a cluster $C_i \in \hat{C}$ matches one of the given attack scenarios. The four metrics $\delta_{\texttt{OtO}}, \delta_{\texttt{OtM}}$, $\delta_{\texttt{MtO}}$, $\delta_{\texttt{MtM}}$ indicate the certainty between $[0,1]$ for a match with the scenarios `OtO`, `OtM`, `MtO`, and `MtM`, respectively. These metrics become 1 if a scenario matches perfectly and they are smaller than 1 if the scenario and C_i do not match exactly. Each metric consists out of three equally weighted summands that describe the three ratios of attacker number $|A|$, of target number $|T|$, and of the difference $|\,|A|-|T|\,|$ all to the expected value in the respective scenario. The formulas are constructed, so that there is a linear relationship between the respective scenario matches completely ($\delta = 1$) or only partially ($0 < \delta < 1$).

A cluster that perfectly matches scenario `OtO` contains two nodes $|V| = 2$, one target $|T| = 1$ and one attacker $|A| = 1$. In that case, all three summands for $\delta_{\texttt{OtO}}$ in Equation 5.2 become one. When more targets or attackers are involved, the metric degrades and converges towards zero. The scenario `OtM` expects one attacker and $|V|-1$ targets, which results in a difference of $|V|-2$. Therefore, the first summand of metric $\delta_{\texttt{OtM}}$ in Equation 5.2 becomes 1 if $|A| = 1$ and the second summand is 1 for $|T| = |V|-1$. This results in a difference between attackers and target of $|V|-2$. $\delta_{\texttt{OtM}}$ becomes closer to 0, if there are more attackers or less targets. Analogous, the scenario `MtO` ($\delta_{\texttt{MtO}}$ in Equation 5.2) expects $|V|-1$ attackers and one target, which results in a difference of $|V|-2$. A perfectly matching `MtM`-cluster ($\delta_{\texttt{MtM}}$ in Equation 5.2) needs to have $|V|$ attackers and $|V|$ targets, so that the difference between them is zero.

$$
\begin{aligned}
\delta_{\texttt{OtO}} &= \frac{1}{3} \cdot \left(\frac{|V|-|A|}{|V|-1} + \frac{|V|-|T|}{|V|-1} + \frac{|V| - |\,|A|-|T|\,|}{|V|} \right) \\
\delta_{\texttt{OtM}} &= \frac{1}{3} \cdot \left(\frac{|V|-|A|}{|V|-1} + \frac{|T|}{|V|-1} + \frac{|\,|A|-|T|\,|}{|V|-2} \right) \\
\delta_{\texttt{MtO}} &= \frac{1}{3} \cdot \left(\frac{|A|}{|V|-1} + \frac{|V|-|T|}{|V|-1} + \frac{|\,|A|-|T|\,|}{|V|-2} \right) \\
\delta_{\texttt{MtM}} &= \frac{1}{3} \cdot \left(\frac{|A|}{|V|} + \frac{|T|}{|V|} + \frac{|V| - |\,|A|-|T|\,|}{|V|} \right)
\end{aligned}
\tag{5.2}
$$

The highest metric determines the scenario and the corresponding label $l \in L = \{$`MtO`, `OtM`, `MtM`, `OtO`$\}$. The certainty of scenario identification and label assignment is:

$$\delta = \max(\delta_{\texttt{OtO}}, \delta_{\texttt{OtM}}, \delta_{\texttt{MtO}}, \delta_{\texttt{MtM}})$$

5.3.1.2 Attack-Step Correlation

For the detection of multi-step attacks, all identified clusters $\hat{C}$ are first transformed into a directed labeled graph that is denoted as *attack similarity graph* $G_{\text{over}} = (\hat{C}, E)$. An edge $(C_i, C_j) \in E$ is included in G_{over} if the two clusters belong to the same multi-step attack.

The linking of attack steps utilizes the labels $l_i, l_j \in L$ of the two clusters $C_i, C_j \in \hat{C}$ to derive a tag `Many` or `One` for the attackers and targets For example, when a cluster is labeled with `OtM`, the set of attacking IPs is tagged with `One` and the set of target IPs is tagged with `Many`.

Then, the approach compares the set of attackers from both clusters (sim_{AA}), the set of targets from both clusters (sim_{TT}), the set of attackers of C_i with the set of targets of C_j (sim_{AT}), and the set of attackers of C_j with the set of targets of C_i (sim_{TA}). For each of the four host comparisons, the host sets X, Y can be attackers or targets. Their tags depend on the tags of the clusters and on the specific comparison. The calculation of the specific host similarity uses one of the following metrics. First, the *Jaccard metric* $J(X,Y) = \frac{X \cap Y}{X \cup Y}$ is used when the two compared sets of hosts X, Y are expected to be equal, i.e., both are tagged equally. Second, the *overlap coefficient* $S(X,Y) = \frac{X \cap Y}{X}$ is used when one set of hosts X is expected to be part of the other set Y, i.e., their tags are different. In this case, X is the set that is tagged with `One` and Y is the one tagged with `Many`.

The comparison of two clusters C_i, C_j with each other computes the four values as mentioned above and then takes the maximum of it $\text{sim} = \max\{\text{sim}_{\text{AA}}, \text{sim}_{\text{TT}}, \text{sim}_{\text{AT}}, \text{sim}_{\text{TA}}\}$. If sim exceeds a predefined threshold σ, an edge (C_i, C_j) with label sim is added to G_{over}. This has to be done for all cluster combinations from set $\hat{C}$. As a result, G_{over} contains the relations among all identified clusters that are above threshold σ. Clusters as part of multi-step attacks then are located on a path connecting them within this graph.

5.3.2 Collaborative Attack Correlation

While the graph-based multi-step detection from Section 5.3.1 requires attacks to be represented by clusters containing all related alerts, the alerts are spatially dispersed in case of a distributed IDS deployment, i.e., a collaborative intrusion detection system (CIDS) [Vas+15b]. Thus, the distributed alert clusters that result from clustering alerts on each IDS locally have to be merged before being processed by the graph-based multi-step detection. However, collection and clustering all alerts centrally does not scale and might be infeasible for large networks. Therefore, the following approach for attack correlation performs a fine-grained characterization of alert clusters regarding their attack scenario and efficiently identifies alert clusters that potentially result from the same attack across different IDSes.

5.3.2.1 Comparing Attack Characteristics

The basic idea of this correlation algorithm is to transform a set of alerts into a much smaller representation that conserves structural characteristics of attacks. For that, network motifs, specifically the so-called motif signatures (cf. the motif-based anomaly detection in Section 3.4.1), are a perfect candidate to summarize the communication structure of the hosts involved in an attack. This allows for a comparison of structural characteristics of attacks, even without prior knowledge of these characteristics and mostly independent from the attack size.

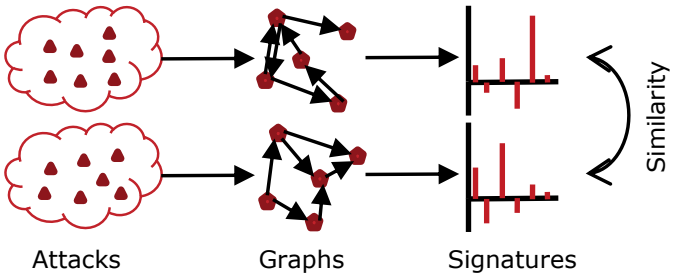


Figure 5.8: Schema to compare the alerts of two attacks by their motif signatures.

Input to this approach for attack interconnection are clustered alerts $C_i \in \hat{C}$ as from the alert clustering with graph-based community clustering and weak alert correlation (cf. Section 5.2) or others (cf. Section 3.4.2). Referred to as attacks, alerts clusters are the input to the collaborative attack correlation, as illustrated by the schematic overview in Figure 5.8. The following paragraphs describe the next steps in detail, which is the transformation of attack data into graphs, the calculation of motifs signatures, and their comparison.

Transformation of Attacks to Graphs The approach assumes that attack characteristics manifest in the structure of the communication graph that contains the malicious communication relations among involved hosts. This graph is derived from all alerts $a_i \in C_j$ of attack $C_j \in \hat{C}$, where an alert a_i has several attributes. From all attributes defined in Section 5.1, the collaborative attack correlation requires only an alert description $a_i = (S{:}T \rightarrow D{:}L)$ with source IP S and source port T and with destination IP D and destination port L. Based on these four attributes of alerts, the approach generates a *communication structure graph* G_{com} for all alerts of an attack C_j. In $G_{com} = (V,E)$, nodes $v \in V$ represent either a host by its IP address or the port on a

specific host. The edges reflect who attacked whom and whether a particular port or several different ports are used in the attack.

To build the graph G_{com} for a specific attack C_j, all alerts $a_i \in C_i$ are added to the graph consecutively. For that, the set of nodes V is extended by nodes representing the hosts, i.e., $\{S,T\}$, and nodes representing their ports, i.e., $\{S{:}T, D{:}L\}$. This notation ensures that ports are always bound to hosts. To reflect who attacked whom, the edges $\{(S,S{:}T),(S{:}T,D{:}L),(D{:}L,D)\}$ are added to E in G_{com}. Intuitively, this describes what is visualized in Figure 5.9, the port and IP used to attack another IP on a respective port.

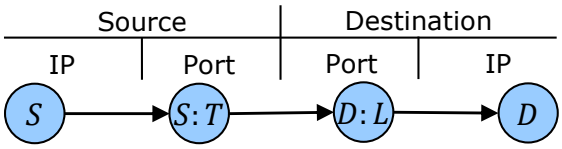


Figure 5.9: Adding an alert ($S{:}T \rightarrow D{:}L$) to graph G_{com}.

Calculation of Scenario Signatures To get from the alerts in C_j via the graph G_{com} to a smaller abstraction, the motif signature of the graph is calculated. For that, all subgraphs $G' = (V',E') \subseteq G_{com}$ of size n, i.e., $|V'| = n$, are numerated, and for each the specific motif pattern m_i with $i \in [0,N-1]$ is determined (cf. the motif-based anomaly detection in Section 3.4.1). Counting the number of occurrences for all motifs m_i results in a *fingerprint* of the graph. The vector that contains the absolute number of occurrences of every m_i is denoted as motif signature F^A.

As F^A is directly dependent on the graph size $|G_{com}|$, it does not allow to compare the structure of two graphs that are of different sizes. To allow such a comparison, the authors in [Mil+02] introduce the so-called Z-Score. It uses the signature F^A of graph G_{com} to calculate for every m_i how much it is over- or underrepresented compared to a random graph of the same size and with the same number of edges as G_{com}. The Z-Score of a specific motif m_i in a graph G is calculated by

$$Z(m_i) = \frac{F^A(i) - F^{rand}(i)}{sd}$$

with $F^A(i)$ being the absolute number of motif m_i in G_{com}, $F^{rand}(i)$ being the average absolute count of motif m_i in respective random graphs, and its standard deviation sd. This is done for every absolute number of motifs in F^A. The resulting motif signature with the Z-Score values is denoted as F^Z. Any motif signature can simply be represented as an array of fixed length, e.g., 16 to cover all possible 3-motifs ($n = 3$).

Comparison of Scenario Signatures Comparing two motif signatures F_1^Z and F_2^Z is supposed to determine how similar they are. Their similarity should be 1, i.e., 100%, if the signatures are equal. However, finding a metric to calculate the similarity in a meaningful manner is not intuitive. The reason is that the values even in the Z-Score signature F^Z are not limited to a fixed range. As the graph size, i.e. attack size, still has an impact on the Z-Score values, the values of every motif in two signatures F_1^Z and F_2^Z cannot directly be compared. Doing so would not achieve high similarity for attacks with similar characteristics but of different size. Therefore, another comparison between two motif signatures is designed in such a way that

similar attack characteristics are identified even if the attacks are of different sizes. For that, a motif that is statistically over- or underrepresented in F_1^Z should also be statistically over- or underrepresented in F_2^Z. Furthermore, the comparison should consider how much a specific motif is over- or underrepresented compared to the other motifs in the signature. The idea is to not have a pairwise comparison of the motif values in F_1^Z and F_2^Z. Instead, the similarity between F_1^Z and F_2^Z reflects how similar the relations among the motif values in F_1^Z are to the relations among the motif values in F_2^Z.

For such a comparison, the Z-Score signatures F_1^Z and F_2^Z are interpreted as vectors $\vec{u}, \vec{v}$, always of fixed length. This make a signature to look like a vector in a multi-dimensional space with the number of dimensions equal to the length of the vectors. The higher the values in a signature, the larger is the vector in the multi-dimensional space. The calculation becomes independent from the vector length when determining the angle between two vectors in the multi-dimensional space. For that, the inner product $< \vec{u}, \vec{v} >$ and the Euclidean norms $||\vec{u}||_2$ and $||\vec{v}||_2$ are calculated for the angle ϕ (Equation 5.3). This finally leads to the similarity $0 \leq sim \leq 1$ (Equation 5.4).

$$cos(\phi) = \frac{< \vec{u}, \vec{v} >}{||\vec{u}||_2 \cdot ||\vec{v}||_2} \tag{5.3}$$

$$sim = \frac{cos^{-1}(\phi)}{\pi} \tag{5.4}$$

Once the alerts of every attack $C_i \in \hat{C}$ are transformed into motif signatures $F_i^Z \in \hat{F}$, the motif signatures of the different attacks, i.e., alert sets, are compared. The goal is to find attacks with the same characteristics. Such a comparison can be calculated efficiently as a motif signature is of a fixed and far smaller size than the respective alert sets. For the attack correlation algorithm, predefined characteristics of attacks can be used to identify specific attack scenarios and to label them accordingly. Alternatively, the attack correlation operates without a knowledge database and learns attack scenarios on its own. It labels attacks according to dynamically derived characteristics of alert sets. Both approaches are described in the remainder of this section. Either way, comparing the Z-Score signatures F^Z to classify attack scenarios requires a threshold τ to require a minimum similarity. Above this threshold, two signatures are assumed to belong to the same attack scenario.

5.3.2.2 Signature-based Classification

The characteristics of already known attack scenarios are defined by reference scenarios $R_x \in \hat{R}$. A reference scenario R_x reflects the characteristics of a specific attack scenario and therefore is representative for all attacks of this attack scenario and their alert sets, respectively. In fact, R_x is just a motif signature F^Z of a typical attack in the attack scenario that should be identified. It is modeled by transforming a representative alert set A into a *communication structure graph* G_{com} and by computing its motif signature F^Z. Hence, the set $\hat{R}$ consists of small motif signatures. The size of this set equals the number of predefined attack scenarios that should be identified.

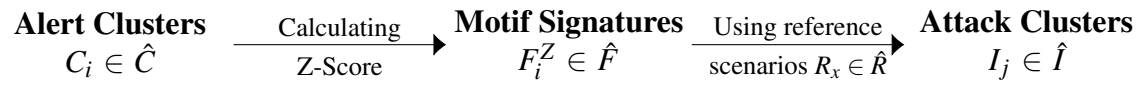


Figure 5.10: Notation for signature-based classification using reference scenarios.

When classifying an attack $C_i \in \hat{C}$, its motif signature F_i^Z is compared to all reference signatures $R_x \in \hat{R}$ using the similarity function described in Section 5.3.2.1. In general, the highest similarity determines the attack scenario that is assigned to attack C_j. However, the minimum similarity threshold τ needs to be respected, because there could be attacks from unknown scenarios that are not included in $\hat{R}$. Hence, they should not be labeled with a known attack scenario. Assigning all attacks $C_i \in \hat{C}$ to reference scenarios results in attack clusters $I_j \in \hat{I}$, as illustrated in Figure 5.10 (cf. Section 5.1). The requirement for attacks of the same scenario $F_i^Z \in I_j$ to have a minimum similarity τ to the respective reference scenario R_x is formalized in Equation 5.5. The requirement of closest match among all reference scenarios $\hat{R}$ is formalized in Equation 5.6.

$$\forall F_i^Z \in I_j : sim(F_i^Z, R_x) \geq \tau \tag{5.5}$$

$$\forall F_i^Z \in I_j \, \forall R_y \in \hat{R} : sim(F_i^Z, R_x) \geq sim(F_i^Z, R_y) \tag{5.6}$$

5.3.2.3 Unsupervised Clustering

Reference signatures are not always available. However, the motif-based comparison of attack characteristics can be used to cluster similar attacks and to dynamically derive reference scenarios $\hat{R}$ for them. This is done by learning new attack scenarios via a hierarchical clustering in two steps. Note that the following two paragraphs for the description of these steps focus on how the motif-based signatures can be used by machine learning to identifying similar alert cluster across different IDSes. In combination with a communication schema, the fundamentals presented here are directly applicable to be used in a CIDS, or they can even be incorporated into network anomaly detection, e.g., to extend the work of Juszczyszyn et al. [JK11].

Hierarchical Clustering The first step to learn attack scenarios clusters all attacks from the same attack scenario. For that, attacks are clustered based on the similarity of their motif signatures F^Z. Generally, the comparison of motif signatures F^Z aims for high similarities among signatures for attacks of the same scenario and low similarities of signatures for attacks from different scenarios. The intention is that clustering the motif signatures of attacks results in attack clusters, one for each detected attack scenario. Hierarchical clustering is most applicable here to cluster attacks into attack scenarios for two reasons. First, it can use the similarity threshold τ (cf. Section 5.3.2.1) as clustering parameter to intuitively control the clustering and its outcome. And second, the visualization as dendrogram allows a manual inspection of the potential clusters depending on τ.

As said, the clustering parameter τ in hierarchical clustering controls the minimum similarity, i.e., maximum distance, for two attacks, so that they are still part of the same attack scenario. Thus, τ must be chosen in a way such that (1) it is low enough to allow attacks of the same scenario to result in one cluster and (2) it is high enough that two attacks of different scenarios do not result in the same cluster. An example for hierarchical clustering of attacks is visualized as dendrogram in Figure 5.11. The x-axis of the figure represents the individual attacks and the y-axis represents $1 - \tau$ as maximum distance for two attacks or scenarios to be merged into the same scenario. Hence, at distance 0, only equal attacks will be merged and at distance 1, all attacks are merged into a single scenario. When stepping from 0 to 1, similar attacks or scenarios are merged once the value on the y-axis reaches their respective distance. The idea in hierarchical clustering is to define a cut-off distance, which determines the final clusters. The

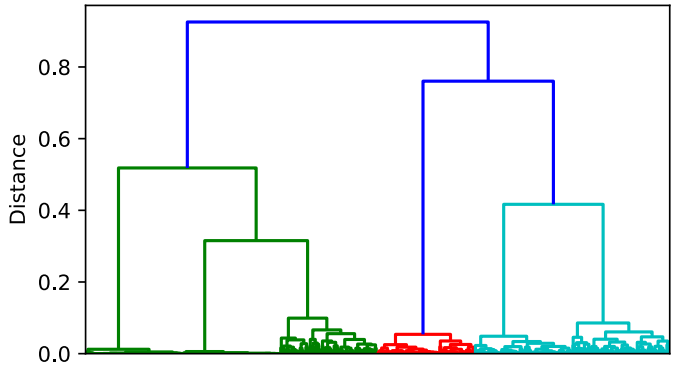


Figure 5.11: Hierarchical clustering for attacks from six example scenarios.

outcome are the clusters, i.e. scenarios, that all the attacks have been merged into at the specific cut-off distance in the dendrogram.

The cut-off distance is $1-\tau$ to form clusters $I_j \in \hat{I}$ of attacks $F_i^Z \in I_J$ with the desired maximum heterogeneity within a cluster. The hierarchical clustering of F^Z for the set of attacks merges clusters with respect to the maximum distance within the resulting cluster, which is known as the `complete` method [Sor48]. This means that attack scenarios are defined by the maximum distance between contained attacks, which can be formally stated as in Equation 5.7.

$$\forall I_j \in \hat{I}: \quad \forall F_1^Z, F_2^Z \in I_j \; sim(F_1^Z, F_2^Z) \geq \tau \tag{5.7}$$

Deriving Reference Scenarios The attacks in a cluster $F_i^Z \in I_j$ formed by hierarchical clustering are supposed to belong to the same attack scenario because of their similar characteristics. To actually extract the characteristics for each attack cluster, a motif signature F^Z is derived per cluster as reference scenario $R_x \in \hat{R}$ to represent the new attack scenario $I_j \in \hat{I}$. Instead of constructing a signature for the attack cluster synthetically, an existing signature from the cluster should be chosen that best represents all other signatures in the cluster is chosen. More specifically, it should be the one with the highest similarity to every other attack on average. This is formalized in Equation 5.8.

$$\forall F_i^Z \in I_j: \quad \sum_{y=0}^{|I_j|} sim(F_i^Z, F_y^Z) \leq \sum_{y=0}^{|I_j|} sim(R_x, F_y^Z) \tag{5.8}$$

After these two steps, a set of attacks, i.e., a set of alert sets, is represented by a number of reference scenarios that is controlled by the similarity threshold τ.

5.3.3 Summary of Attack Interconnection

To interconnect attacks, two attack correlation algorithms have been presented that identify similarities among the attacks based on their alerts. For that, the attack correlation algorithms consume the meta alerts, i.e., the output from alert clustering (cf. Section 5.2). The identification of related attacks for attack interconnection leverages additional context about the attacks. The

two presented algorithms determine the attack scenario and the who-attacked-whom characteristic in particular. They, however, define this characteristic in different ways and use it for two different purposes.

The attack scenario in the first algorithm is defined by a simple model that distinguishes four classes of the who-attacked-whom structure. On the level of hosts, this model labels the number of attackers as well as the number of targets with either *One* or *Many*. Labeling an attack with this scheme results in one of four possible combinations, i.e., the attack scenario. The scenario label is incorporated into the linking of attacks when two of them have the same host involved. The robustness of this label against false positive alerts is evaluated in Section 6.5.1.3. Especially in distributed attacks, quite a number of hosts might be involved in the attack. However, the attacker might follow up on only a small number of these hosts. Comparing not only the hosts' IP addresses among attacks but also the scenario label, assists security operators to identify the important relations among attacks when assembling them to multi-step attacks. The result of this multi-step attack correlation is a graph that concisely summarizes individual attacks as nodes and highlights relations among them as edges labeled with the attack scenario. Section 6.5.2 investigates to which extent the results can help to detect multi-step attacks in the real world.

Apart from detecting multi-step attacks, the second attack correlation algorithm in this section interconnects meta alerts that potentially describe the same attack. This is necessary for large networks that run more than one IDS, or for Internet-wide attacks that hit more than one network site. Either way, each IDS captures the attack only partly, and a common view on this attack among all IDSes is required to be established. Instead of exchanging and comparing all alerts, the attack correlation algorithm identifies candidates for the same attack on the basis of meta alerts. This enables to efficiently exchange small attack summaries and to afterwards compare alerts of candidate attacks only.

The attack summary in this second attack correlation algorithm differs from most algorithms that search for common attributes among the alerts from different attacks. These approaches can reveal if two victims are targeted by the same attacker but they cannot tell if the attacker performs the same kind of attack in both cases. Because of this, the presented attack summary is based on network motifs [Mil+02] to fingerprint attacks. This fingerprint abstracts the characteristic of the attack scenario, considering not only the relations among the involved hosts but also the ports they use. The accuracy of differentiating between attack scenarios based this abstraction is evaluated in Section 6.7.1. The resulting abstraction can be magnitudes smaller than the corresponding alert data and allows for a faster comparison of attacks. With the help of this motif abstraction, the algorithm identifies known attack scenarios and is even able to learn previously unknown scenarios. It can be deployed in a centralized or decentralized manner. To which extent the motif-based abstractions can lower the overhead for exchanging alert information as well as its performance on real-world alerts, is evaluated in Section 6.7.2.

5.4 Summary of Network Alert Correlation

Attack detection by correlating IDS alerts is challenged by the large alert volume, which results in missing the big picture of the attacks and choosing suboptimal mitigation actions. The approaches presented in this chapter assist in assembling alert information for an accurate attack summary. Their deployment is not exclusive but should be combined along the alert correlation process as part of the intrusion detection process (cf. Section 3.2.1) to be most

effective. This way, several measures can be implemented that work in two different dimensions. These dimensions are derived from the insight that some alerts describe the same malicious root cause, while others describe different kind of malicious activity and stem from consecutive attack steps.

Thus, the first stage in the alert correlation process achieves a volume reduction by identifying alerts that describe the same attack and clustering them to meta alerts. Apart from one-shot attacks with a single alert that results in a single meta alert each, the reduction in alert volume becomes apparent especially for network-wide attacks such as distributed attacks. Instead of having an IDS alert for every malicious network flow or every involved host of the distributed attack, these related alerts are abstracted by a single meta alert, i.e., a cluster of alerts that represents a single attack. After the transition to meta alerts, the alert correlation process supplements them with addition context before linking the supplemented attack representations for the identification of attack interconnections.

An end-to-end implementation of the full alert correlation process has been presented by *graph-based alert correlation* (GAC) that consists out of a graph-based community clustering for the detection of distributed attacks and the graph-based attack interconnection for the detection of multi-step attacks. GAC achieves alert clustering by identifying alerts that share similar features among each other. GAC then assigns one of four labels to each identified attack to determine the class of distributed scenario. These labeled attacks are used by GAC afterwards when linking attacks steps that involve the same hosts. The resulting multi-step attack graph summarizes individual attacks and highlights relations among the attack steps with additional context about the attack scenario. The scenario context enables to precisely describe the relations among the involved hosts across different network- and non-network-wide attack steps. GAC and its steps are extensively evaluated in Section 6.5.

Apart from clustering of alerts via GAC, this chapter has presented an alternative correlation algorithm for slow and stealthy APT attacks. These attacks cause temporally dispersed alerts that are denoted as weak alerts because a relation among these alerts is weakly visible, if visible at all. More likely, weak alerts get lost in the shuffle of daily alerts and are filtered by GAC as they look like unrelated or false positive alerts. As regular alert clustering overlooks these weak alerts because of their infrequent occurrence, the algorithm for *weak alert correlation* continuously aggregates them over a long time until they assemble to an APT attack step. For that, the correlation algorithm leverage some APT characteristics that become visible in the alert data, including the expansion to another network zone during lateral movement and a steadily stealthy behavior even when involving multiple hosts. Once the aggregation of weak alerts evolves to an attack step, the respective weak meta alert can be fed back to GAC for the purpose of detecting multi-step attacks. To which extent the weak alert correlation can complement GAC, is evaluation in Section 6.6.

Apart from linking of attacks via GAC, this chapter has presented another correlation algorithm for the identification of similar attacks across network monitors. An attack that is not captured by a single IDS but by multiple IDSes in parts, causes spatially dispersed alerts. Without establishing a global picture of the attack, its consequences are tried to be mitigated independently for every monitor instead of collaboratively working on a mitigation for the whole network. To efficiently identify similar attacks without exchanging each and every alert, the motif-based *collaborative attack correlation* exchanges and compares small fingerprints of the meta alerts. Instead of summarizing the alert features, this fingerprint abstracts the attack scenarios regarding the who-attacked-whom structure. This allows to identify similar attacks in a privacy-friendly

manner, even when different sources are used per targeted network site. For a fine-grained characterization of the attack scenario, however, not only the relations among the hosts but also the usage pattern of ports is incorporated into the fingerprint. The accuracy of this collaborative attack correlation is evaluated in Section 6.7.

In combination, the different alert correlation algorithms transform raw IDS alerts into a concise attack representation. In particular, the correlation overcomes the challenges of temporally and spatially dispersed alerts. As a result, the correlation algorithms assemble alerts from distributed and stealthy APT attacks. Furthermore, the algorithms link alerts from related actions to reconstruct multi-step attacks.