On the Alert Correlation Process for the Detection of Multi-step Attacks and a Graph-based Realization

Steffen Haas
University Hamburg
Mittelweg 177
Hamburg, Germany
steffen.haas@informatik.uni-hamburg.de

Mathias Fischer
University Hamburg
Mittelweg 177
Hamburg, Germany
mathias.fischer@informatik.uni-hamburg.de

ABSTRACT

Monitoring tools like Intrusion Detection Systems (IDS), Firewalls, or Honeypots are a second line of defense in the face of an increasing number of distributed, increasingly sophisticated, and targeted attacks. A huge amount of security alerts needs to be analyzed and correlated to gather the complete picture of an attack. However, most conventional IDS fall short in correlating alerts that have different sources, so that many distributed attacks remain completely unnoticed. In this paper, we define alert correlation as a process and describe the consecutive steps along with their properties and goals. Following this process, we propose *Graph-based Alert Correlation* (*GAC*), a novel correlation algorithm that isolates attacks, identifies attack scenarios, and assembles multi-stage attacks from huge alert sets. Our evaluation results on artificial and real-world data indicates that *GAC* is robust against false positives, can detect distributed attacks, and scales with an increasing number of alerts.

CCS Concepts

•**Security and privacy** → **Intrusion detection systems;** *Denial-of-service attacks;* •**Applied computing** → *Network forensics;* •**Computing methodologies** → Machine learning;

Keywords

alert correlation; intrusion detection; distributed attacks

1. INTRODUCTION

As the number of Internet-enabled devices grows, a huge amount of hosts can be used for coordinated attacks, e.g., by utilizing botnets [5]. Thus, when protecting a network with an IDS [17, 15] or with Honeypots [8], these systems will report increasingly more alerts as a result of distributed attacks in which either several attackers and/or victims are involved. Additionally, a single attack can cause several alerts, either because the malicious communication happens very frequently or because it triggers alerts on different sensors at the same time, e.g., in case of Collaborative Intrusion Detection Systems [24]. As a result, security operators are increasingly overwhelmed with alerts and have a hard job in splitting the relevant ones from the ones that can be ignored.

Alert correlation algorithms ease this task by correlating alerts with each other to obtain the bigger picture of an attack. A very common approach is to group alerts based on similar attributes [26, 6, 23, 10], e.g., source and destination IP, and to report common attribute patterns. Other approaches correlate alerts of multi-step attacks [12, 21] 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. Thus, alert correlation is the overall process to make relations among alerts visible, which can serve as indication of their belonging to a larger attack.

The contribution of this paper is twofold and based on former work of us that appeared as a conference paper [4]. First, and by extending our earlier work, we highlight and describe alert correlation as a three-step process. Second, along with [4] we propose *GAC*, a novel *Graph-based Alert Correlation (GAC)* algorithm that realizes the alert correlation process. Compared to [4] we provide a more detailed evaluation and discussion in this paper. *GAC* can be used for the detection of generic (distributed) attacks, e.g., DDoS [25, 9], port scans [7, 20], or worm spreading [1], and operates in three steps:

1. Alerts are clustered based on the similarity in between particular alert attributes. Thus, each resulting cluster contains all alerts that might be part of the same attack or single attack step.

2. The communication patterns among the hosts within each cluster are identified, which can be either *One-to-One*, *One-to-Many*, *Many-to-One*, or *Many-to-Many*.

3. Finally, when a multi-step attack is present in the input alert set, clusters of single attack steps get interconnected with each other based on the communication patterns within clusters.

Our *GAC* algorithm requires alerts to contain only source and destination IP addresses as well as port numbers. This reduces the complexity in alert correlation and increases the

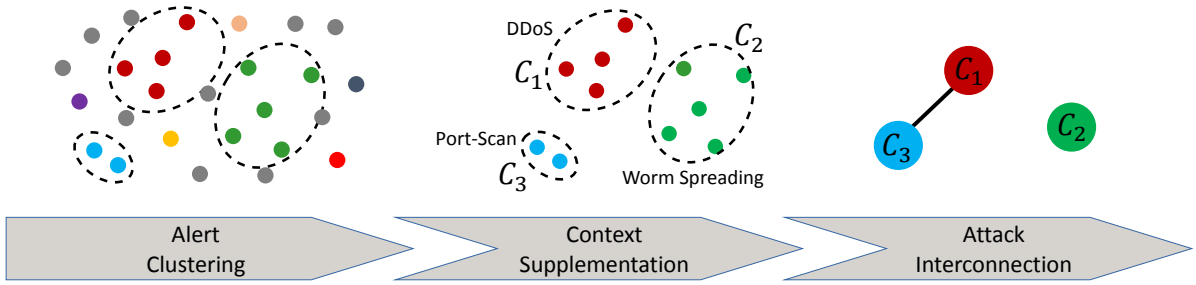


Figure 1: General alert correlation process with example for scenario labeling

compatibility to different sensors. Additional knowledge is mined from the alerts themselves, which is an advantage compared to many other alert correlation algorithms that presume some kind of knowledge database, especially when they attempt to detect multi-step attacks.

Our experimental results indicate that *GAC* achieves good accuracy for separating alerts from different attacks, not only for fine-tuned initialization of the clustering method but for a wider range of parameter values. *GAC* considers the uncertainty when identifying the attack scenarios of alert clusters and calculates a certainty value that reflects the trust in the identified scenario. Our experiments indicate that even in worst case when clusters have been inaccurately identified, a low false positive rate can be expected for the scenario identification. In the evaluation with real-world alerts, for $\approx$ 75% of the clusters, *GAC* is able to identify their respective scenario with high certainty of $\geq$ 90%. The high certainty in scenario identification and the huge reduction of an alert set into the multi-step attack representation in *GAC* indicate that it can reduce the burden on security operators.

The remainder of this paper is structured as follows: The alert correlation process in described in Section 2. Section 3 summarizes related work. Our graph-based correlation algorithm is introduced in Section 4 and Section 5 summarizes our evaluation results. Section 6 concludes our work and outlines future work.

2. ALERT CORRELATION PROCESS

The overall purpose of alert correlation process is to convert a set of alerts into a representation of attacks. Thus, alert correlation is an essential part of alert analysis and is still an active field of research with related work in different research areas and different application scenarios. We want to emphasize that alert correlation in the scope of this paper does not include the detection of individual alerts. Instead, alert correlation algorithms have to rely on the sensors underneath to classify malicious events as alerts.

We propose a unified description of the alert correlation process for transparent analysis and comparison of correlation algorithms. The novel process description characterizes alert correlation algorithms by breaking them down into their core building blocks (cf. Figure 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.

In the remainder of this section, we introduce a formal model for the alert correlation process and then describe each building block in more detail.

2.1 Formal Model for Alerts and Attacks

An alert $a \in A$ can come from arbitrary sources, e.g., 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 alert detection. Every alert $a \in A$ consists of a fixed vector of attributes $a = (a^1, a^2, \dots a^n)$, e.g., source and destination addresses as well as source and destination ports.

An attack, or more precisely every step of an attack i, will induce a set of true positive alerts S_i. The set of all induced alert sets from different and multi-step attacks is given by $\mathcal{S} = \{S_0, S_1, \cdots, S_{n-1}\}$. A multi-step attack j that comprises several single-step attacks will induce a subset $M_j \subseteq \mathcal{S}$. The set of all multi-step attacks is $\mathcal{M} = \{M_0, M_1, \cdots, M_{m-1}\}$.

We note that there are alert correlation algorithms for the detection of specific attack scenarios or attack types, e.g., distributed attacks. In a specific context like this, the definition of $\mathcal{S}$ and $\mathcal{C}$ needs to be adapted. Therefore, only true positive alerts with respect to the specific context contribute to $\mathcal{S}$ and $\mathcal{C}$.

2.2 Alert Clustering

The alert clustering will cluster alerts into respective clusters of alerts $\mathcal{C} = \{C_0, C_1, \cdots C_{n-1}\}$. Each cluster $C_i \in \mathcal{C}$ is supposed to represent an attack and the alerts that belong to it. The alert clusters are desired to model the actual attacks $S_i \in \mathcal{S}$, so in best case this leads to $\mathcal{S} = \mathcal{C}$. 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 $\forall a \in A : \exists S_i \in \mathcal{S} : a \in S_i \iff \exists C_i \in \mathcal{C} : a \in C_i$ holds. Please remember that the definition of *false positive* can depend on the context and the respective attacks to be detected. 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

Table 1: Dissecting selected related work according to the process building blocks

Algorithm	Alert Clustering	Context Supplementation	Attack Interconnection	Attributes
AoI [6]	Attribute combinations with attribute generalization	Implicit by rule pattern		Any
PAC [22]	Pairwise similarity of (meta-)alerts	Comparison of attack classes as reported by sensors	Meta-alert Similarity	srcIP, srcPort, dstIP, dstPort, class, time
Lattice [26]	Feature combinations with same source IP in fixed patterns	Pattern index		srcIP, srcPort, dstPort, protocol

and thus be included as input into the next building block if and only if they are induced by an attack in $\mathcal{S}$.

Attack isolation requires clustering to assign each alert out of set A to one cluster $C_i \in \mathcal{C}$. The correlated clusters $\mathcal{C}$ should reflect the original attack steps $S_i \subseteq A, S_i \in \mathcal{S}$. 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.

2.3 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 might ease 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 human 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 in practice it works well most of the time, we find this to be dangerous, as it imposes constraints on the attacks that can be detected.

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

2.4 Attack Interconnection

The last building block in the alert correlation process attempts to find relations among alert clusters in $\mathcal{C}$ to assemble single attack steps in $\mathcal{S}$ into multi-step attacks $\mathcal{M}$. 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 in the first process block. 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.

In best case, attack detection is implemented during alert clustering. Then, attacks have already been identified when processing comes to this last interconnection block and relations among attacks can be searched based on the identified attacks.

3. RELATED WORK

Approaches for alert correlations are usually classified according to their correlation method [19], e.g., similarity-based, sequential-based, or case-based. However, such a classification does not give any information on what an algorithm is actually used for. Furthermore, related work often leverages from combinations of such methods. For this reason, we describe related work with respect to our novel description of the alert correlation process as introduced in Section 2. Any correlation algorithm is a concrete realization of the alert correlation process and its three building blocks or steps. Table 1 dissects three algorithm from related work into these blocks, summarizes their contributions and the alert attributes that they use. Depending on the goal of the alert correlation algorithm either all or only a subset of the building blocks from the process are actually realized. Thus, we classify related work and their alert correlation algorithms either as *clustering-centric*, *interconnection-centric*, or *process-centric*.

Clustering-centric: Algorithms in this class primarily attempt to aggregate alerts that are related, i.e., that belong to the same attack. Clustering alerts of individual attack steps is the prerequisite for assembling multi-step attacks.

Zhou et al. in [26] search for predefined alert attribute patterns utilizing a lattice structure. A pattern is a combination of one to eight attributes. Instances of attribute combinations are reported such that they are balanced in their detail and aggregation level. The authors assume the algorithm runs on a local site and thus the destination IP is not significant for patterns. The pattern itself can be used for the description of clusters. The index of the possible attribute combinations, i.e., patterns, in the lattice additionally reflects the attack scenario. A heuristic alert-clustering

method that is called Attribute-oriented Induction (AoI) supports root cause analysis and is proposed by Julisch [6]. The algorithm attempts to find dominant instances of alert attribute combinations. A resulting pattern is an arbitrary combination of attributes, where each one is either a concrete value or a generalized class, i.e., a predefined set of concrete values. The pattern itself represents a compact description of the entire cluster. A domain-specific algorithm is proposed by Staniford et al. [20] for graph-based portscan detection. The authors define different functions to calculate the similarity between two alerts, e.g., to check if the port differs by one. In the graph of alerts, only similar alerts are connected to separate alerts from different portscans and also to detect stealthy attackers.

Interconnection-centric: Algorithms in this class primarily attempt to detect multi-step attacks. The focus of the correlation is on the interconnection between alerts of different attack steps. Only few approaches identify related alerts within each attack step.

Ning et al. describe in [12] a model to supplement alerts with prerequisites and consequences that are required for a respective attack to be successful. For connecting attacks, the resulting hyper-alerts are then added to a so called correlation graph to detect multi-step attacks by the match among prerequisites and consequences. Similar hyper-alerts are then aggregated over prerequisites and consequences. Zhu et al. [27] use supervised machine learning to mine if two alerts belong to the same attack strategy. The probabilities for same attack strategy among the alerts are written to a so called correlation strength matrix. It is used to extract attack strategies, i.e., attack paths, by assembling alerts that are strongly related. Multi-step attacks can also be detected by supplementing the network environment with context. Roschke et al. [18] identify all alerts that match against a given attack graph. Matched alerts are summarized in a so called dependency graph according to the host relations in the attack graph. This allows to determine the most interesting attack paths through the network. Sun et al. in [21] require another kind of environmental context. They create an object instance graph, e.g., with files and processes as nodes, from collected system calls on network hosts. Evidence from intrusion detection, i.e., alerts, are then leveraged to track infection propagation in the object instance graph. Derived probabilities allow to identify attack paths even though intermediate steps are missed.

Process-centric: All three process blocks are used for the alert correlation. Algorithms comprise sophisticated techniques for single and multi-step detection.

Ning et al. [13] combine alert clustering and multi-step detection. They create alert clusters with any preferred clustering algorithm and a correlation graph [12] independently. Both are combined to reconstruct attacks even if not all steps have been detected. Valdes et al. [22] propose a multi-stage approach that merges and aggregates alerts based on the similarity between different alert attributes. Specific attributes are used in each stage to further merge alerts. Merging alerts results in a so called meta-alert, where attributes are not values but lists of values from all encompassing alerts. Additional knowledge is required to compare the similarity between attack classes. The authors afterwards compare the meta-alerts to reveal attack interconnections.

To the best of our knowledge, we have seen no process-centric related work, that mines a characterization from clusters without additional knowledge that is used for the reconstruction of multi-step attacks. In the next section, we describe our graph-based implementation for this purpose.

4. GRAPH-BASED ALERT CORRELATION

In this section, we present a novel graph-based approach for alert correlation. Section 4.1 introduces the basic approach of our Graph-based Alert Correlation algorithm *GAC* and points out its individual steps, which are explained in more detail in Sections 4.2 - 4.4.

4.1 Overview on Approach

The input to our Graph-based Alert Correlation algorithm *GAC* is an alert set A consisting of individual alerts $a \in A$ with attributed $a = (a^1, a^2, \dots a^n)$ (cf. Section 2.1). Such an alert set usually contains true positives and false positives, depending on the performance of the underlying alert detection. Following the alert correlation process, the task of our correlation algorithm is to analyze the alert set A and to group its alerts according to the single and multi-step attacks $\mathcal{S}$ and $\mathcal{M}$ without knowing the ground truth. Furthermore, the output of the correlation should include a description or representation of detected attacks.

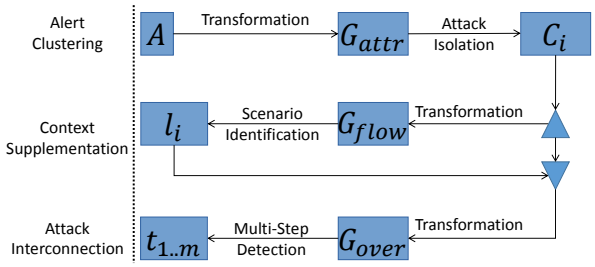


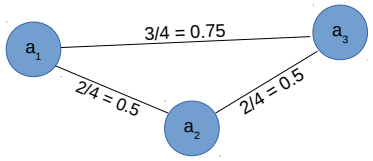
Figure 2: Blocks in Graph-based Alert Correlation

We propose *GAC*, a correlation algorithm to detect, annotate and link distributed attacks. It can also generally be used as a method to reveal the relation among alerts and to identify attacks. *GAC* transforms alerts into different graph representations, which are processed successively during the correlation process. We realize the three basic building blocks of the alert correlation process (cf. Section 2): *Alert clustering*, *context supplementation*, and *attack interconnection*. We summarize their general purpose along with their graph-based implementation in *GAC* as illustrated in Figure 2 in the following:

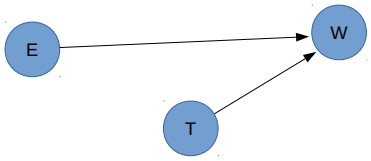
1. **Alert clustering** clusters alerts into respective clusters of alerts $\mathcal{C} = \{C_0, C_1, \cdots C_{n-1}\}$ that in best case leads to $\mathcal{S} = \mathcal{C}$. *GAC* starts by transforming the alert set A into an *Alert Similarity Graph* G_{attr} that reflects attribute similarities among alerts. A graph-based clustering algorithm is applied afterwards to cluster all alerts into subsets $\mathcal{C}$ as candidates for attack

	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



(c) The resulting G_{flow} graph

Figure 3: Graph models for example alert set

steps. This is described in more detail in the next Section 4.2.

2. **Context is supplemented** for each attack step by assigning a label $l_i \in \mathcal{L}$ to each cluster $C_i \in C$. These labels enrich the description of attacks both for a better understanding of them as well as for the support of consecutive steps in alert correlation. In *GAC* we mine characteristics directly from alert information and alert clusters. For that, each cluster C_i is transformed into an *Alert Flow Graph* G_{flow}. We describe the relation of malicious communication among attackers and victims within each cluster to identify attack scenarios. Section 4.3 describes our approach to supplement additional context in more detail.

3. To **interconnect clusters**, the cluster labels that encode additional context are leveraged in the next step of our correlation process. Thus, we interconnect clusters in $\mathcal{C}$ that represent single-attack steps with each other to multi-step attacks $\mathcal{M}$. Compared to other approaches based on causality, *GAC* does not require external knowledge on vulnerabilities. *GAC* transforms all the clusters C_i with their corresponding labels l_i into a single *Attack Similarity Graph* G_{over} to search for overlapping IP addresses among clusters. For any two clusters $C_i, C_j \in \mathcal{C}$ that share IP addresses, the labels $l_i, l_j \in \mathcal{L}$ are used to derive a tag $t_{i,j}$ describing the type of multi-step attack. Section 4.4 describes in more detail our approach to interconnect different attacks.

The example for graph transformation in Figure 3 is based on the three alerts $a_1, a_2, a_3 \in A$ in Figure 3a, where two attackers E and T target destination W on port P.

4.2 Alert Clustering

We propose to represent alerts as nodes in a graph and to add edges between similar alerts. This enables us to search for alerts that share similar attributes and to cluster alerts within these graphs afterwards. For this, we first describe the used graph model and then the clustering method.

4.2.1 Transforming Alerts into a Graph

We transform the set of alerts A into a weighted *Alert Similarity Graph* $G_{\text{attr}} = (A, E)$ that contains alerts 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 $(a^0 ... a^{n-1})$ of the alerts, respectively, as follows:

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

Per attribute, an attribute-specific comparison function h^j delivers a similarity value in $[0, 1]$. All attribute comparisons are weighted according a vector $c = (c^0 \dots c^{n-1})$ such that $\sum c^j = 1$. We require the similarity between two alerts to be equal or higher than a minimum similarity threshold τ to include an edge $(a_1, a_2) \in E$ to G^{τ}_{attr}. Thus, τ controls the number of edges $|E|$, by removing edges 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.

As we focus on network-related alerts, we consider the attributes IP and port of source and destination only, as these are the most important and common attributes that should be supported by any sensor type and method used for alert detection. 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., http for ports 80 and 443. Thus, according to Equation 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 [22], for other methods to assign weights, we refer to the generation of a so called log-graph in [16]. Figure 3b shows the *Alert Similarity Graph* with three nodes for the alerts in Figure 3a. E.g., $F_{sim}(a_1, a_2) = 0.5$ as two out of four attributes are equal between the alerts a_1 and a_2.

4.2.2 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, we further separate alerts of individual attack steps by clustering them, i.e., identify subgraphs in G^{τ}_{attr}. The motivation for clustering is illustrated for an example graph G_{attr} in Figure 4. In the bottom one of the two isolated components, one can intuitively identify three loosely coupled subgraphs. These clusters (marked with red circles)

Table 2: Similarity values among alerts of different attack scenarios

Scenario	SrcIP	SrcPort	DstIP	DstPort	Example	Similarity
One-to-One	X		X		Vertical Scan (V-Scan)	2/4
One-to-Many	X			X	Horizontal Scan (H-Scan)	2/4
Many-to-One			X	X	Service DDoS	2/4
			X		Distributed V-Scan	1/4
Many-to-Many				X	Worm	1/4

are no isolated components, because a few of their alerts are similar to alerts of other clusters. These clusters reflect individual attack steps that can be related intentionally.

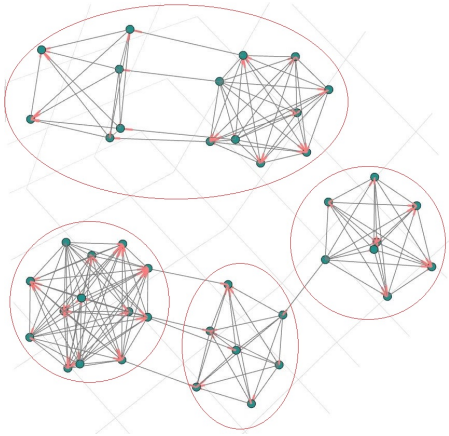


Figure 4: Community clustering in the graph G_{attr}

GAC leverages the graph structure to identify and isolate subgraphs of loosely coupled attacks. We use community clustering, especially the *Clique Percolation Method (CPM)* [14], 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.

4.3 Context Supplementation

Input to this step are clusters of alerts that represent individual attack steps. The goal is now to characterize the attack scenario of each cluster. For that, we look at the communication patterns between attackers and victims. 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.

Most alerts in a cluster are usually similar to each other and thus well interconnected in G_{attr}. Hence, such a graph structure is inappropriate to identify attack scenarios. Thus, per identified cluster $C_i \in \mathcal{C}$, we establish a less dense *Alert Flow Graph* $G_{\text{flow}} = (V, E)$ 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 3c shows the graph G_{flow} for the alerts in Figure 3a. In this example, we assume the three alerts in the graph G_{attr} in Figure 3b to be assigned 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 node degrees in $G_{\text{flow}} = (V, E)$ are used. We give a scheme for distributed attacks involving either multiple sources, destinations, or both in G_{flow}. We can characterize 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. We distinguish four attack scenarios as follows and as summarized in Table 2:

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.

We identify these scenarios by metrics that describe how good a cluster $C_i \in \mathcal{C}$ matches one of the scenarios given above. 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. We developed a concept for these metrics to be 1 if a scenario matches perfectly and that they are lower if the scenario and C_i do not match completely. 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 2 become one. When more targets or attackers are involved the metric degrades and converges towards zero. For the scenario `OtM`, we expect 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 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, for the scenario `MtO` ($\delta_{\texttt{MtO}}$ in Equation 2) we expect $|V|-1$ attackers and one target, which results in a difference of $|V| - 2$. A perfectly matching `MtM`-cluster ($\delta_{\texttt{MtM}}$ in Equation 2) needs to have $|V|$ attackers and $|V|$ targets, so that the difference between them is zero.

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

The highest metric determines the scenario and the corresponding label $l \in \mathcal{L} = \{\texttt{MtO}, \texttt{OtM}, \texttt{MtM}, \texttt{OtO}\}$. The certainty of scenario identification and label assignment is $\delta = \max(\delta_{\texttt{OtO}}, \delta_{\texttt{OtM}}, \delta_{\texttt{MtO}}, \delta_{\texttt{MtM}})$.

4.4 Attack Interconnection

In the third and last step of our alert correlation process, we are interested in the relations between individual attack steps. For two attack steps $S_i, S_j \in \mathcal{S}$ that are part of the same multi-step attack $M_i \in \mathcal{M}$, *GAC* potentially derived two alert clusters $C_i, C_j \in \mathcal{C}$ from the *Alert Similarity Graph* (cf. Section 4.2) that we need to assemble as multi-step attack. First, all identified clusters $\mathcal{C}$ are transformed into a directed labeled graph that we call *Attack Similarity Graph* $G_{\text{over}} = (\mathcal{C}, E)$. An edge $(C_i, C_j) \in E$ is included in G_{over} if the two clusters belong to the same multi-step attack.

GAC utilizes the labels $l_i, l_j \in \mathcal{L}$ of the two clusters $C_i, C_j \in \mathcal{C}$ to derive a tag `Many` or `One` for the attackers and targets (cf. Table 2). 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, we compare 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. We propose to use one of the following metrics to calculate the specific host similarity. 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`.

When comparing two clusters C_i, C_j with each other, we compute the four values as mentioned above and then take 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 pre-defined 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 $\mathcal{C}$. 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. EVALUATION

To evaluate our graph-based correlation algorithm *GAC* that is presented in the last section, we split our evaluation into two parts. The first part (Section 5.1) performs an in-depth analysis on the inner working and behavior of *GAC*. The second part (Section 5.2) presents results from applying *GAC* to real-world data. Both parts are performed with an implementation of *GAC* in python using *igraph*[1] and *networkx*[2] for graph representation and clustering.

5.1 Analysis of Individual Steps

In this section, we present evaluation results on the individual steps of *GAC* (cf. Section 4). For that, we use artificially created data to have both the control and the ground truth about the data that is used for the analysis of *GAC*. Only by doing so, we can systematically analyze the behavior of every step of our alert correlation algorithm. Section 5.2 will then apply *GAC* on real-world data, for which we, however, have no ground truth. In Section 5.1.1, we describe the data and how we generated it as input for the experiments. Section 5.1.2 evaluates the *GAC* algorithm with respect to the performance of alert clustering, both regarding filtering unrelated alerts and separating alerts from different attacks. Section 5.1.3 evaluates the attack scenario identification of *GAC* for distributed attacks. We conduct independent experiments for clustering and scenario identification to systematically analyze their robustness in general.

5.1.1 Artificial Alert Data for Analysis

Although we are aware of criticism of using artificial data [11], we emphasize the difference that the scope of this paper is not alert detection but alert correlation. Moreover, it is not about generating a data set representative for the real-world, but about generating input to analyze the effectiveness of what the algorithm was designed for.

Input for the analysis of *GAC* consists out of alerts from one or more artificial attacks. Depending on the process step that is analyzed, the alert data is modified to generate input variations for the experiment. As *GAC* is designed to detect distributed attacks, we have to generate them. We use the patterns of attacks according to Table 2 for attack generation. This table defines which IPs and ports are equal

[1] https://igraph.org/python/
[2] http://networkx.github.io/

for all alerts of the same attack and which are probably different and can be determined randomly. Since *GAC* requires no knowledge about subnets and type of port, we use the full IP (0.0.0.0 to 255.255.255.255) and port range (0 to 65535). We then generate different kinds of attacks: In a horizontal *Scan* (`OtM`), one source targets several hosts on the same destination port. In a *DDoS* attack (`MtO`), several sources target the same IP and port combination from arbitrary source ports. In a *Worm* (`MtM`) spreading, each host targets a subset of other hosts on a fixed destination port.

We assume that several attacks happen at the same time in the alert data used in the evaluation. Thus, we generate alerts for several attacks independently and partially merge them by mapping two systems from different sets to a single system, i.e., replacing IP X from the first and IP Y from the second alert set by a third IP Z. The *overlapping parameter* determines the fraction of IPs that overlap. This input challenges alert clustering to isolate alerts into groups of different attacks. In addition, in some experiments we blur alert data via an *blurring parameter* β to create additional alerts that represent isolated single alerts or false positives, e.g., as a result of an inaccurate sensor. Blurring adds no alerts if $\beta = 0$ and adds more random alerts until $\beta = 1$. We not only add additional alerts for the n existing hosts in the alert data, but we also create alerts for new hosts. We prepare two different sets of hosts that we want to generate alerts for: one set of attacking and one set of target hosts. Each of them is assembled from up to $\beta \cdot n$ existing hosts and up to $\beta \cdot n$ new hosts. Pairwise for the next attacker and target in both sets, an alert is added until an alert is generated for every host in both sets. As a result, on average we add $\beta \cdot n$ new hosts and $1.23 \cdot \beta \cdot n$ new alerts.

To evaluate the process step of scenario identification we create separate data sets. Each of them contains a single attack plus additional alerts as false positives (via blurring parameter) in form of a cluster to simulate an inaccurate clustering or an inaccurate sensor. The specific parameterization to generate the alert data used in our evaluation, is described in more detail together with the results.

5.1.2 Alert Clustering in Alert Similarity Graph

First, we take a closer look at the graph G_{attr} (cf. Section 4.2) and the clustering of alerts to attack steps. The minimum similarity threshold τ is used to filter edges in G_{attr}. We choose $\tau = 0.25$ as this is the minimal value at which related alerts stay connected for attacks in Table 2. On the filtered graph, community clustering is applied to potentially result in one cluster per attack (step). For that, we investigate the clique size parameter k that determines the minimum community size. In particular, we are interested in how to choose k to achieve best clustering performance. Technically, the clustering method CPM requires clique size $k \geq 2$. And obviously, it should be $k \leq |G_{\text{attr}}|$, i.e., not larger than the number alerts in the graph, to allow mining of any cluster. More specific, k should not be larger than the number alerts from the smallest attack, i.e., $min(|S_i|) \in \mathcal{S}$. The following experiments investigate the performance between the lower and upper bound of the range $k \in [2, |G_{\text{attr}}|]$.

Apart from showing how graph-based clustering performs, we also compare ourselves to a similar approach that we refer to as Attribute-oriented Induction (AoI) [6] (cf. Section 3). AoI is an efficient algorithm to find attribute patterns in a set of alerts to establish clusters of minimum size k, similar to *GAC* that employs community clustering and forms cliques of at least size k. In the following experiments, we report the average results across 16 runs.

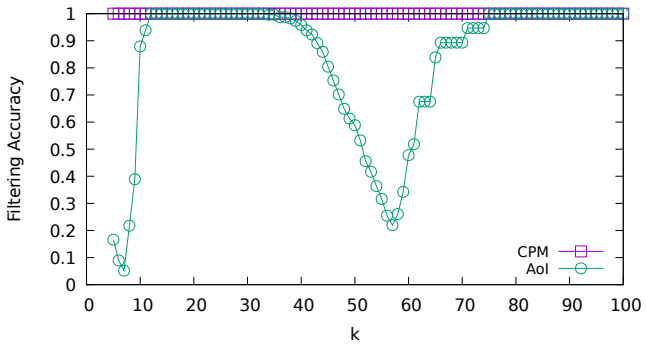
Metrics: For *alert filtering* and *attack isolation* (cf. Section 4.2), we did not find any metric that reflects the clustering performance to both challenges appropriately. Therefore, we propose two novel methods for evaluating both challenges individually. We explain how to incorporate the standard metric accuracy $ACC = \frac{TP+TN}{P+N}$ that compares the ground truth to one identified cluster.

Alert filtering accuracy should only consider the union of alerts in all identified clusters as detected attacks $DA = \bigcup_{C_i \in \mathcal{C}} \bigcup_{c \in C_i} c$ and the union of alerts in original attacks as ground truth $GT = \bigcup_{S_i \in \mathcal{S}} \bigcup_{s \in S_i} s$. The accuracy is used to evaluate the performance of DA modeling GT by comparing these two sets of alerts.

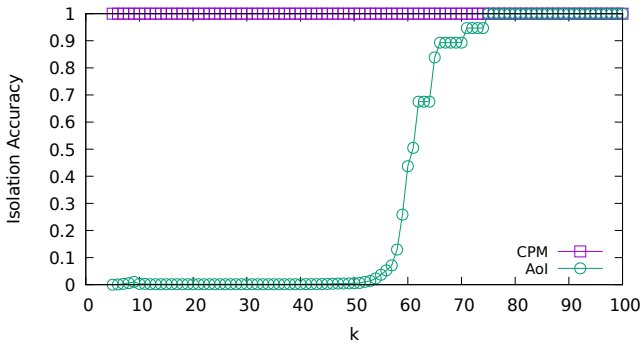
Attack isolation accuracy should evaluate the performance of $\mathcal{C}$ modeling $\mathcal{S}$. It describes a correct grouping with respect to all original attacks and clusters of alerts. As the accuracy ACC can only compare one attack and one cluster, we break the comparison down to individual pairs of $\mathcal{C}$ and $\mathcal{S}$. Our methods aims to find which original attack $\mathcal{C}_i \in \mathcal{C}$ models which alert cluster $\mathcal{S}_j \in \mathcal{S}$ best. The accuracy ACC is calculated for every possible combination and a mapping between $\mathcal{C}_i \in \mathcal{C}$ and $\mathcal{S}_j \in \mathcal{S}$ is derived under the constraint that each attack and cluster can only be mapped once. A greedy algorithm creates a mapping such that either only single attacks or single clusters are left unmapped. A accuracy of 0 is defined for any unmapped cluster or attack, respectively. The total accuracy is then the average accuracy for all mapped and unmapped attacks or clusters, respectively.

Clustering alerts from single attacks: Figure 5 illustrates the clustering results for alerts from an unmodified DDoS attack, in which 10 hosts attack one target. The alert set consists out of 500 alerts, so 50 alerts are generated per source on average. Therefore, the accuracy for alert filtering (Figure 5a) and the attack isolation (Figure 5b) drops to zero for $k > 500$ for both our community clustering (CPM) in *GAC* and AoI. For lower values, AoI tries to mine clusters while generalizing as few attributes as possible. If an AoI cluster, i.e., attribute pattern, itself consists of many smaller patterns, AoI will report these smaller clusters if k allows. This behavior is a problem when using AoI on alerts from distributed attacks, e.g., when each source in a DDoS attack causes multiple alerts. This is the reason why the attack isolation accuracy of AoI is below 0.3 for $k \leq 60$ and why the alert filtering accuracy of AoI drops for $60 \leq k \leq 75$. Community clustering, instead, can combine these smaller clusters, i.e., k-cliques, to communities and therefore achieves an accuracy of 100% for both filtering and isolation.

The advantage of community clustering, therefore, is that it potentially works even though k is much lower than the smallest attack. However, k has to be high enough to filter noise and to split alerts of loosely coupled attacks. We will investigate this in the next experiment.

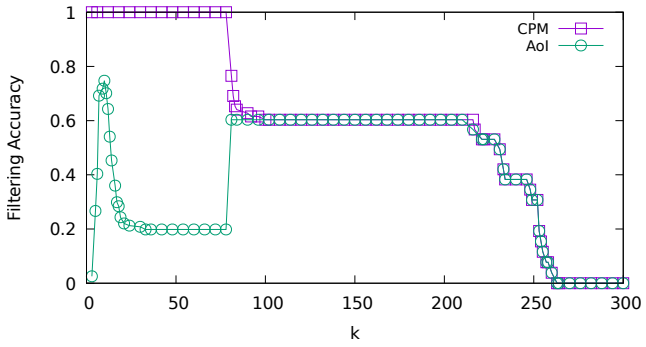


(a) Accuracy for alert filtering

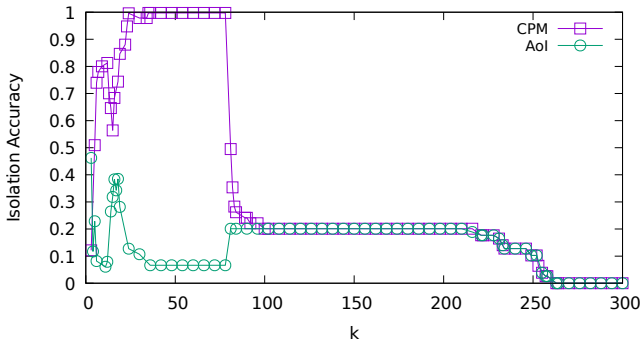


(b) Accuracy for attack isolation

Figure 5: Accuracy for clustering alerts from a DDoS attack depending on clustering parameter k



(a) Accuracy for alert filtering



(b) Accuracy for attack isolation

Figure 6: Accuracy for clustering alerts from overlapping attacks depending on clustering parameter k

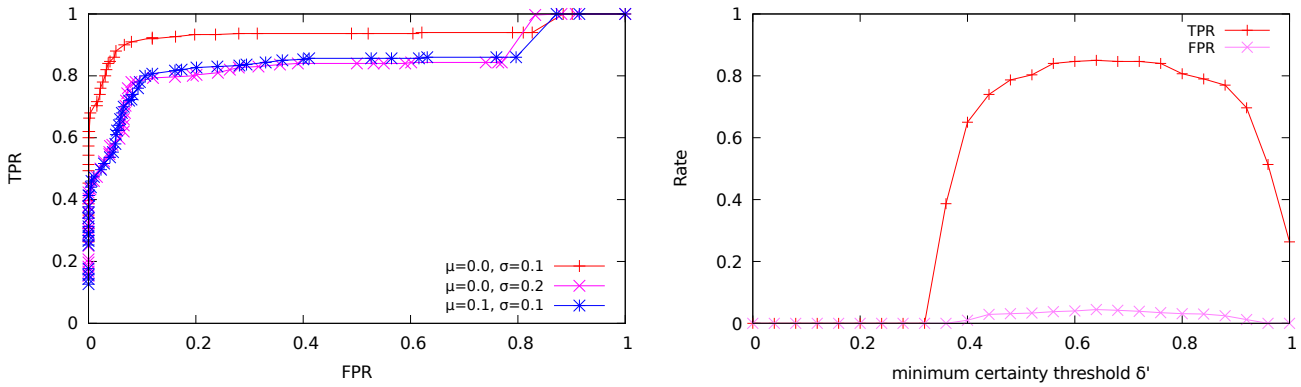
Clustering alerts from overlapping attacks: The clustering results for three attacks overlapping 30% of their IP addresses are plotted in Figure 6. There is one DDoS with 80 alerts, one Scan with 80 alerts and one worm for 20 hosts with a spread factor of 0.7 (266 alerts on average). Other experiments showed a linear dependency of the clustering parameter k on the presence of false positive alerts. Because of space constraints, we omit a detailed analysis on the influence of additional alerts on requiring k to be increased to filter false positives. Thus, experiments here are performed without blurring. The accuracy of alert filtering (Figure 6a) and attack isolation (Figure 6b) are plotted depending on the parameter k. Community clustering (CPM) has an alert filtering accuracy of 100% for $3 \leq k \leq 80$, i.e., for any k that is less or equal than the smallest attack size. The range of k to achieve an accuracy of attack isolation of (almost) 100% is different for the lower bound and requires $24 \leq k \leq 80$. Values for $k \leq 24$ make the CPM to merge alerts of different attack steps into one cluster because of the overlapping IP addresses. We achieve at least equal accuracy like AoI in the first place and can even cluster all attacks correctly. This is because we achieve 100% accuracy for alert filtering and attack isolation for the same values of k. AoI instead has its maximum accuracy of 73% for alert filtering at $k = 8$ and 38% for attack isolation at $k = 18$ Thus, choosing a value for k in AoI is always a balance between good alert filtering or attack isolation.

Our results for alert clustering of overlapping distributed attacks indicates that (1) *GAC* achieves higher accuracy than AoI and (2) the clustering parameter k allows to achieve these high accuracy for a wide range of the parameter.

5.1.3 Scenario Identification on Alert Flow Graphs

GAC classifies attacks represented in the form of clustered alerts as `OtO`, `OtM`, `MtO`, or `MtM` (cf. Section 4.3). Such an attack characterization must be error tolerant, because there might be false positives or false negatives when clustering alerts. For that, we designed scenario metrics $\delta_{\texttt{OtO}}, \delta_{\texttt{OtM}}, \delta_{\texttt{MtO}}, \delta_{\texttt{MtM}}$ that represent the confidence in the respective scenarios to identify a clustered alert set. As defined in our classification algorithm (cf. Section 4.3), the metric with maximum value determines the scenario label.

In the following experiments, we evaluate the robustness of attack scenario identification on the basis of artificially created attacks on which we apply blurring (cf. Section 5.1.1) to simulate false positives in clustering. We try to identify the scenarios of a DDoS attack with 300 attackers, a scan attack with 300 targets, and a worm attack with 75 hosts and spreading factor 0.7. For one repetition of an experiment, the three resultant attack clusters are each blurred with a specific parameter β. We present two different types of evaluations for the scenario identification of the three blurred attack variants. In the first one, we calculate and evaluate the scenario metrics for `OtO`, `OtM`, `MtO`, and `MtM` individually.



(a) Matches for individual attack scenarios **(b) Unambiguous matches among all scenarios**

Figure 7: Attack classification with individual scenario metrics depending on minimum certainty threshold δ'

In the second one, the attack variants are classified by *GAC* and the highest scenario metric among `OtO`, `OtM`, `MtO`, and `MtM` labels the attack.

Individual Scenario Labels: The attack variations are generated with a blurring factor drawn from a normal distribution with its parameters location μ and scale σ. Figure 7 shows two plots to evaluate the performance of the scenario identification based on individual scenario labels. Instead of using the highest scenario metric among `OtO`, `OtM`, `MtO`, and `MtM` to label the attack, we define a minimum certainty threshold δ'. We calculate all four scenario metrics $\delta_{\texttt{OtO}}$, $\delta_{\texttt{OtM}}$, $\delta_{\texttt{MtO}}$, $\delta_{\texttt{MtM}}$ and label an attack with the all scenarios that have a respective metric above δ'. Thus, an attack can be labeled with between zero and up to four labels.

In the first experiment, we define a true positive classification as an attack that was assigned the correct label, even if the attack has additional labels assigned. Any incorrect labeling of an attack counts as a false positive. These definitions are used to calculate the True Positive Rate (TPR) and False Positive Rate (FPR). For high values of δ', we require a high confidence in the identified scenario. Hence, we can expect less wrong classifications and therefore a low FPR. However, some attacks might not perfectly fit our scenario definitions and are not labeled at all. Thus, a lower TPR is expected. When we assume low values of δ', we expect both a high TPR and FPR.

The concrete relation between TPR and FPR is illustrated in Figure 7a, which plots a Receiver Operating Characteristic (ROC) curve. It is used for the analysis of the minimum similarity threshold δ' to analyze its effect on the true positives and false positives. The curve plots the TPR depending on the FPR for blurring factors drawn from three different normal distributions. A diagonal curve would indicate a random classification. The result plot, however, clearly shows that using δ' as classification parameter indicates good classification performance. The point nearest to the upper left corner is yield for δ' of value 0.50, 0.54, and 0.50 for the specific blurring variants. From the plot we can see that lower values for δ' massively increase the FPR of the individual metrics while the TPR already is ≈ 0.9.

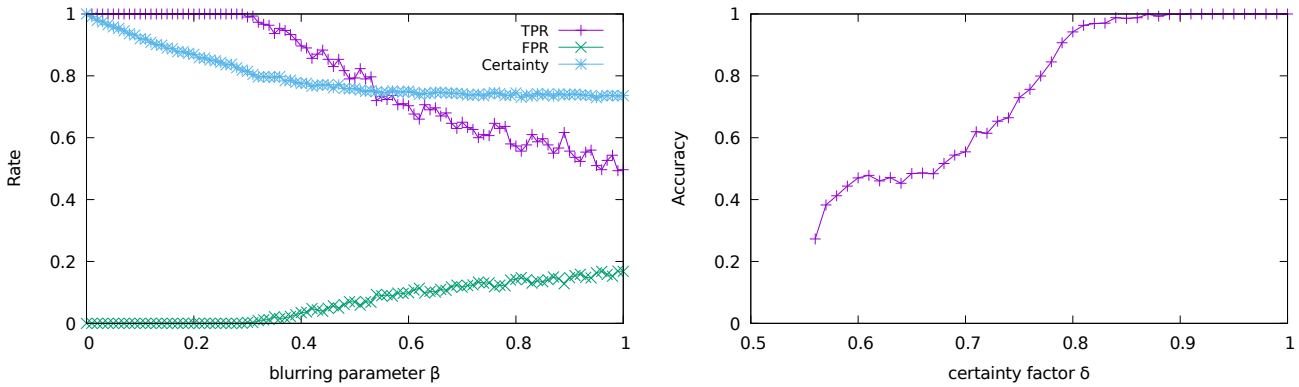
In the second experiment for individual scenario labels, we change the definition of true and false positives. For the calculation of the True Positive Rate (TPR) and False Positive Rate (FPR), we define a true positive classification as an attack that was assigned the correct but no additional label. Thus, we require an unambiguous identification with the correct scenario. If more than one scenario metric is higher than the minimum required δ' at the same time, there is no true positive classification but also no false positive classifications for the respective attack.

The evaluation with the new definitions are illustrated in Figure 7b, which plots the TPR and FPR depending on the minimum certainty threshold δ' used for classification. The curve is calculated for blurring with parameters $\mu = 0.0$ and $\sigma = 0.1$. On this data set, we achieve a low FPR (≤ 0.045) and an acceptable TPR (≤ 0.85).

The experiment shows that the certainty factor δ in the attack scenario identification helps to tolerate errors in the previous clustering block without a significant increase of the FPR. Furthermore, good performance requires $\delta \geq 0.7$.

Most Certain Scenario Label: In the following experiments, we use the exact definition of scenario classification as defined by *GAC*, which is that the highest scenario metric among $\delta_{\texttt{OtO}}$, $\delta_{\texttt{OtM}}$, $\delta_{\texttt{MtO}}$, $\delta_{\texttt{MtM}}$ determines the respective scenario of an attack.

The first experiment investigates the performance of scenario identification in presence of inaccurate clustering. To simulate this, we vary the blurring parameter β in between zero and one in steps of 0.01. The complete experiment is executed 100 times, such that for every value of β, we generate 100 blurred variants of the three attacks. This is necessary because blurring introduces randomness when generating false positive alerts for a specific value of the blurring parameter. The True Positive Rate (TPR) and the False Positive Rate (FPR) depending on β are plotted in Figure 8a as well as the average certainty δ. The figure shows that scenario identification in *GAC* can completely tolerate false positive alerts, i.e., still achieve 100% accuracy, to some extend. It also shows that it can be TPR = 1 and the FPR = 0, even though the certainty $\delta < 1$. The TPR starts



(a) Detection rate depending on blurring

(b) Accuracy depending on certainty

Figure 8: Classification of attacks with highest scenario metric depending on false positive alerts

to decrease once the blurring parameter exceeds $\beta \geq 0.3$. During the experiment, we have observed that classifying DDoS and scan attacks is more sensitive to blurring than classifying worm attacks. When $\delta = 1$, the metric $\delta_{\texttt{MtO}}$ for a DDoS attack is 0.52, while the metric $\delta_{\texttt{MtM}}$ for a worm attack is 0.79. As a consequence of the bias towards $\delta_{\texttt{MtM}}$, blurred worm attacks are always successfully identified as `MtM`. On overage, the certainty δ in identifying the three attacks over the 100 repetitions drops to 0.73 in Figure 8a. The TPR drops to 0.50 and the FPR stays below 0.17.

In the second experiment, we ask about the expected accuracy when a scenario is identified with a specific certainty δ. For this reason, we blur the three attacks with a random $\beta \in [0, 1]$ and repeat this with a different β 10000 times. The average accuracy, i.e., fraction of correct classifications, over all repetitions was 80%. Figure 8b however plots the average accuracy depending on the certainty δ. During the repetitions, the scenario classifications were grouped in bins of 0.01 regarding their resultant certainty δ and the average accuracy is calculated per bin. In the figure we see the range of actually calculated certainties was in $[0.56, 1]$ and we achieved always 100% when the certainty $\delta \geq 0.91$, which is the case for 20% of the attack clusters. 53% of the clusters have been classified with a certainty $\delta \geq 0.79$. The average accuracy for classifications with this certainty was 90%.

The experiments show that the majority of attacks can be correctly classified even in worst-case situations. Also, false positives are expected only in scenarios with high uncertainty.

5.2 Real-World Results

We are now evaluating the performance of *GAC* on real-world data. Therefore, we apply the full algorithm to the data sets as introduced in Section 5.2.1. The results are presented in Sections 5.2.2 - 5.2.4.

5.2.1 Real-World Alert Data Set

The SANS Internet Storm Center operates a community-based collaborative monitoring system for Internet threats on a global level. This system is called DShield[3] and collects network incident logs, i.e., alerts, from various contributors. Every alert in DShield contains the IP and port of the source and the target, among others. However, the target IP is hashed for anonymity reasons. For this real-world evaluation, we correlate DShield alerts from the days given in Table 3. The table summarizes each set by the number of alerts per day, from both last and this year to enable a representative view, and additionally reports the number of distinct attacking and targeted IP addresses.

Our alert correlation algorithm performs on a finite set of alerts, not on an alert stream. Therefore, we split the data sets of 24 hours into chunks, each being a separate input to an individual processing with *GAC*. Our experience shows that chunks of 5000 subsequent alerts are a reasonable size as complexity increases heavily with graph size. However, because of the chunks, we can process the data set in parallel. We use 16 workers in parallel on a system with 2×8 2.2 GHz cores and 128 GB RAM. With this setup, it takes on average 1 to 1.5 minutes per chunk depending on the data set.

Table 3: Days of DShield sets and their statistics

Set	Day	Alerts	Sources	Targets
1	2016-08-22	4517498	395872	100675
2	2016-08-23	7238861	443981	116965
3	2016-08-24	5825579	427718	116835
4	2016-08-25	5125589	406530	100053
5	2017-08-11	3668198	281690	17531
6	2017-08-12	3937357	345382	17497
7	2017-08-13	4138175	329108	17155

As in Section 5.1, we use *GAC* with the parameter $\tau = 0.25$ to require at least one of four attributes to be equal between two alerts. CPM is based on the clique size $k = 15$ such that the smallest attack consists out of 15 similar alerts to filter false positive alerts. Next, we present the application of *GAC* to these DShield data sets. The results represent the aggregation over all chunks of the specific day. First, the clustering performance is presented. Then, we show results

[3]https://www.dshield.org/

of attack scenario identification and afterwards we present the outcome of multi-step detection.

5.2.2 Clustering

The outcome of clustering has two properties that are important to evaluate. First, the alerts that are not clustered and second, the grouping of the remaining into clusters. Table 4 shows a classification for the assignment of alerts and the fraction of nodes per class. An alert with assignment *None* was filtered and therefore not considered further. The assignment *Multi* is specific for the CPM clustering and describes alerts that are grouped into more than one cluster. Table 4 furthers shows the number of clusters as well as the average size and standard deviation among the clusters.

Table 4: Clustering statistics for DShield

	Assignment			Cluster Size	
Set	None	Multi	Clusters	Avg	Std
1	0.70%	9.33%	25111	195.83	776.88
2	0.84%	6.42%	31242	244.94	871.69
3	0.90%	7.33%	27235	228.02	836.50
4	0.71%	9.70%	29654	188.71	754.80
5	1.56%	21.22%	44664	98.62	459.29
6	1.37%	21.72%	49188	96.67	452.65
7	1.38%	19.67%	47289	103.78	477.97

The low fraction of unassigned alerts shows that *GAC* encompasses almost all alerts in its results. This is expected because the nature of distributed attacks causes a lot of alerts, whereas a targeted attack like advanced persistent threat (APT) causes only few alerts which are distributed along a large time frame. The large amount of clusters and the high standard deviation is because of the many different clusters and shows that *GAC* is not limited to a specific form of clusters.

5.2.3 Attack Scenario Identification

At this stage of *GAC*, each cluster is analyzed to detect the attack scenario. Table 5 describes the median value that was used as certainty δ, its average value, and standard deviation. The table also shows the fraction of clusters that are identified as `OtO`, `OtM`, and `MtO`.

Table 5: Scenario identification statistics for DShield

	Certainty Factor δ			Scenario		
Set	Med	Avg	Std	OtO	OtM	MtO
1	1.0	0.918	0.143	12.50%	20.75%	66.75%
2	1.0	0.911	0.147	14.53%	24.79%	60.68%
3	1.0	0.909	0.147	14.65%	22.63%	62.73%
4	1.0	0.923	0.140	11.37%	21.18%	67.45%
5	1.0	0.922	0.142	13.84%	18.37%	67.79%
6	1.0	0.930	0.137	12.32%	16.89%	70.79%
7	1.0	0.927	0.140	13.16%	17.56%	69.28%

From the certainty factor δ in Table 5 we can derive that the majority of values is expected to be between 1 and 0.75. This allows a high confidence in the identification, as Section 5.1.3 showed that this range achieves good results in theory. We manually verified a sample of labels. Most of the time, clusters whose certainty in scenario identification is $\delta < 0.75$ are caused by alerts with same destination port. The *Alert Flow Graph* then usually consists out of several unconnected components, where groups of IPs appear to represent unrelated attacks without any connection among them apart from the same destination port. When these clusters consist out of small components with a few hosts each, they are tagged with `OtO`. If they additionally include a large component of many hosts, this component usually follows the properties of `OtM` or `MtO` and the cluster is tagged respectively. We did not observe any worm-like attacks, i.e., `MtM`, because the hashed target IPs in DShield prevents to have unique identifiers for hosts independent whether they are attacker or target. However, we were able to label every cluster and the majority of scenarios ($> 85\%$) were identified as `OtM` or `MtO`.

5.2.4 Multi-Step Detection

The last stage of *GAC* assembles multi-stage attacks, i.e., calculates the similarity of IP addresses among the clusters to reveal relations between attacks. We illustrate the result in Figure 9 in form of a CDF plot. It shows the value distribution of the inter-cluster similarity. All sets show similar results. We included the sets 1, 4, and 5 in the figure.

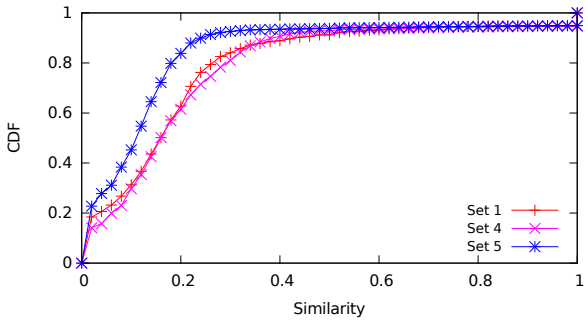


Figure 9: Cumulative distribution function (CDF) of similarity values during the multi-step detection in the DShield sets

From the plot we see that inter-cluster similarity is in almost all cases either 1 or < 0.5. Any two clusters rarely have a similarity ≥ 0.5 and < 1. So we can say that if two clusters have a high similarity, i.e., above 0.5, it is most probably 1. Despite the low value of about 5% high inter-cluster similarities, it shows that we can detect significant overlap in attacks with probably high precision. This is expected because it eases human analysis by pointing out these multi-stage attacks. In the DShield data sets, high similarities are usually caused by clusters that have both high similarity among attackers (sim_{AA}) and victims (sim_{VV}). We note that potentially not all multi-step attacks could be detected because the hashed target IP addresses in the data sets prevent to identify hosts that are both attacked and themselves attackers.

5.3 Discussion and Limitations

We conclude the evaluation of *GAC* with a discussion of the experiment results and the limitations of the approach. We

do this regarding the complexity and required resources as well as regarding the accuracy for specific applications.

The Clique Percolation Method (CPM) used for community clustering is a resource intensive task. Current implementations of this method do not scale well with the size of the input graph. This is why we processes large alert sets in chunks of up to 5000 consecutive alerts. This limits the number of nodes in our *Alert Similarity Graph* G_{attr} to 5000. Depending on how many alerts are similar the each other, i.e., how dense the graph is, the clustering might take 20 minutes on individual chunks. To compensate this, we have used parallel processing of individual chunks to increase the clustering throughput. This allows to constantly analyze a stream of alerts but leaves the delay until the results are available as a limitation. Another limitation, especially compared to simpler clustering algorithms such as AoI [6], is that *GAC* has an increased demand on the memory, i.e., RAM. Once the clustering is done, there is a more relaxed resource demand for the classification of attacks. The *Alert Flow Graph* G_{flow} itself might require a bit more memory than the alerts, however, the calculation of the scenario metrics can be calculated very fast. Using only the relevant information of each cluster, linking attack steps can be done efficiently with respect to CPU and RAM.

With community clustering, we found a method that allows a flexible definition of alert clusters. While other approaches such as [6, 26] form clusters based on attribute patterns, *GAC* does not enforce common attributes among all alerts in a cluster. This allows us to cluster alerts from a broader range of attacks as long as a sufficient amount of alerts are similar. *GAC* also overcomes the problem that we have seen with AoI during the experiments. Especially for distributed attacks, it might happen that correlation algorithms from related work split the actual attack up into smaller clusters, one for each individual attacker.

6. CONCLUSION

In this paper we have proposed *GAC*, an alert correlation algorithm that implements the three building blocks *alert clustering*, *context supplementation*, and *attack interconnection* with a graph-based approach. The community clustering parameter k works well for a wide range of values, especially for clustering alerts from distributed attacks. We outperform Attribute-oriented Induction, an efficient and powerful correlation algorithm proposed by Julisch [6], when it comes to distributed multi-step attacks.

We have evaluated how to identify distributed attack scenarios based on the node-degree among the hosts involved in malicious communication. The heuristic identification of attack scenarios with the certainty δ has a low false positive rate in theory. In the analysis of real alert data, the scenario of $\approx 80\%$ of the attacks have been identified with a certainty of $\geq 75\%$. This shows that alert analysis can significantly be eased by identifying distributed attacks with *GAC*. The interconnection of real attacks yield only low fraction of pairs that have high similarity among the hosts. The majority of similarities between two attacks is low and therefore most probably indicate unrelated attacks.

For future work, we plan to develop (semi-)automated methods for the clustering parameter k. Since *GAC* works on a finite set of alerts, we have seen large attacks whose alerts have been split to consecutive inputs. Thus, we want to investigate how to achieve a correlation of the outputs of different GAC instances. This would make *GAC* applicable for alert correlation within a collaborative and highly distributed IDS without a central correlation unit. Moreover, to protect privacy in a collaborative IDS, we would like to combine such a distributed version of *GAC* with anonymous publish-subscribe approaches like [2, 3].

7. REFERENCES

[1] D. Dagon, X. Qin, G. Gu, W. Lee, J. Grizzard, J. Levine, and H. Owen. Honeystat: Local worm detection using honeypots. In *International Workshop on Recent Advances in Intrusion Detection (RAID)*, 2004.

[2] J. Daubert, M. Fischer, S. Schiffner, and M. Mühlhäuser. Distributed and Anonymous Publish-Subscribe. In *International Conference on Network and System Security (NSS)*, 2013.

[3] J. Daubert, S. Schiffner, P. Kikiras, M. Mühlhäuser, and M. Fischer. AnonPubSub: Anonymous Publish-Subscribe Overlays. *Elsevier Computer Communications*, 2015.

[4] S. Haas and M. Fischer. GAC: Graph-Based Alert Correlation for the Detection of Distributed Multi-Step Attacks. In *ACM/SIGAPP Symposium On Applied Computing (SAC)*, 2018.

[5] S. Haas, S. Karuppayah, S. Manickam, M. Mühlhäuser, and M. Fischer. On the Resilience of P2P-based Botnet Graphs. In *IEEE Conference on Communications and Network Security (CNS)*, 2016.

[6] K. Julisch. Clustering intrusion detection alarms to support root cause analysis. *ACM Transactions on Information and System Security (TISSEC)*, 2003.

[7] J. Jung, V. Paxson, A. Berger, and H. Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *IEEE Symposium on Security and Privacy (SnP)*, 2004.

[8] L. Krämer, J. Krupp, D. Makita, T. Nishizoe, T. Koide, K. Yoshioka, and C. Rossow. AmpPot: Monitoring and defending against amplification DDos attacks. In *International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2015.

[9] K. Lee, J. Kim, K. H. Kwon, Y. Han, and S. Kim. DDoS attack detection method using cluster analysis. In *Expert Systems with Applications*, 2008.

[10] M. E. Locasto, J. J. Parekh, A. D. Keromytis, and S. J. Stolfo. Towards Collaborative Security and P2P Intrusion Detection. In *IEEE Workshop on Information Assurance and Security*, 2005.

[11] J. McHugh. Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory. *ACM Transactions on Information and System Security (TISSEC)*, 2000.

[12] P. Ning, Y. Cui, and D. S. Reeves. Constructing Attack Scenarios through Correlation of Intrusion Alerts. In *ACM Conference on Computer and*

Communications Security (CCS), 2002.

[13] P. Ning, D. Xu, C. G. Healey, and R. S. Amant. Building Attack Scenarios through Integration of Complementary Alert Correlation Methods. In *Network and Distributed System Security Symposium (NDSS)*, 2004.

[14] G. Palla, I. Derényi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society: Supplimentary Info. *Nature*, 2005.

[15] V. Paxson. Bro: a system for detecting network intruders in real-time. *Computer networks*, 31, 1999.

[16] K. Pei, Z. Gu, B. Saltaformaggio, F. Wang, Z. Zhang, L. Si, X. Zhang, and D. Xu. HERCULE: Attack Story Reconstruction via Community Discovery on Correlated Log Graph. In *Annual Computer Security Applications Conference (ACSAC)*, 2016.

[17] M. Roesch et al. Snort: Lightweight intrusion detection for networks. In *Lisa*, volume 99, 1999.

[18] S. Roschke, F. Cheng, and C. Meinel. A New Alert Correlation Algorithm Based on Attack Graph. In *Computational Intelligence in Security for Information Systems (CISIS)*, 2011.

[19] S. Salah, G. Maciá-Fernández, and J. E. Díaz-Verdejo. A model-based survey of alert correlation techniques. *Computer Networks*, 2013.

[20] S. Staniford, J. a. Hoagland, and J. M. McAlerney. Practical automated detection of stealthy portscans. *Journal of Computer Security*, 2002.

[21] X. Sun, J. Dai, P. Liu, A. Singhal, and J. Yen. Towards Probabilistic Identification of Zero-day Attack Paths. In *IEEE Conference on Communications and Network Security (CNS)*, 2016.

[22] A. Valdes and K. Skinner. Probabilistic Alert Correlation. In *Recent Advances in Intrusion Detection (RAID)*, 2001.

[23] E. Vasilomanolakis, C. Garcia Cordero, M. Mühlhäuser, and M. Fischer. SkipMon: A Locality-Aware Collaborative Intrusion Detection System. In *International Performance Computing and Communications Conference (IPCCC)*, 2015.

[24] E. Vasilomanolakis, S. Karuppayah, M. Mühlhäuser, and M. Fischer. Taxonomy and Survey of Collaborative Intrusion Detection. *ACM Computing Surveys*, 47(4), 2015.

[25] G. Yan, R. Lee, A. Kent, and D. Wolpert. Towards a bayesian network game framework for evaluating DDoS attacks and defense. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2012.

[26] C. V. Zhou, C. Leckie, and S. Karunasekera. Decentralized multi-dimensional alert correlation for collaborative intrusion detection. *Network and Computer Applications*, 2009.

[27] B. Zhu and A. A. Ghorbani. Alert Correlation for Extracting Attack Strategies. *Network Security*, 2006.

ABOUT THE AUTHORS:

Steffen Haas is a PhD student in the IT-Security and Security Management research group at the University of Hamburg since 2016. Previously, he worked as a research assistant at the University of Muenster from 2015 to 2016. Steffen received his Master degree at the TU Darmstadt in 2015 and graduated from the Baden-Wuerttemberg Cooperative State University in 2013. His research interests are various aspects of cyber-security, especially threats and appropriate countermeasures related to computer networks.

Mathias Fischer is an assistant professor at the University Hamburg since September 2016. Before that, he was an assistant professor at the University Münster (2015-16), a Postdoc at the International Computer Science Institute (ICSI) / UC Berkeley (2014-15), and Postdoc at the Center for Advanced Security Research Darmstadt (CASED) / TU Darmstadt from (2012-14). His research interests encompass IT and network security, resilient distributed systems, network monitoring, and botnets. Mathias received a PhD in 2012 and a diploma in computer science in 2008, both from TU Ilmenau.