Towards Efficient Reconstruction of Attacker Lateral Movement

Florian Wilkens
Universität Hamburg
Germany
wilkens@informatik.uni-hamburg.de

Steffen Haas
Universität Hamburg
Germany
haas@informatik.uni-hamburg.de

Dominik Kaaser
Universität Hamburg
Germany
kaaser@informatik.uni-hamburg.de

Peter Kling
Universität Hamburg
Germany
kling@informatik.uni-hamburg.de

Mathias Fischer
Universität Hamburg
Germany
mfischer@informatik.uni-hamburg.de

ABSTRACT

Organization and government networks are a target of Advanced Persistent Threats (APTs), i.e., stealthy attackers that infiltrate networks slowly and usually stay undetected for long periods of time. After an attack has been discovered, security administrators have to manually determine which hosts were compromised to clean and restore them. For that, they have to analyze a large number of hosts.

In this paper, we propose an approach to efficiently reconstruct the lateral movement of attackers from a given set of indicators of compromise (IoCs) that can help security administrators to identify and prioritize potentially compromised hosts. To reconstruct attacker paths in a network, we link hosts with IoCs via two methods: k-shortest-paths and biased random walks. To evaluate the accuracy of these approaches in reconstructing attack paths, we introduce three models of attackers that differ in their network knowledge.

Our results indicate that we can approximate the lateral movement of the three proposed attacker models, even when the attacker significantly deviates from them. For insider attackers that deviate up to 75% from our models, the method based on k-shortest-paths achieves a true positive rate of 88% and can significantly narrow down the set of nodes to analyse to 5% of all network hosts.

ACM Reference Format:
Florian Wilkens, Steffen Haas, Dominik Kaaser, Peter Kling, and Mathias Fischer. 2019. Towards Efficient Reconstruction of Attacker Lateral Movement. In *Proceedings of the 14th International Conference on Availability, Reliability and Security (ARES 2019) (ARES '19), August 26–29, 2019, Canterbury, United Kingdom.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3339252.3339254

ARES '19, August 26–29, 2019, Canterbury, United Kingdom

ACM ISBN 978-1-4503-7164-3/19/08…$15.00
https://doi.org/10.1145/3339252.3339254

1 INTRODUCTION

Advanced Persistent Threats (APTs) pose a risk to enterprises and government organizations alike. They originate from highly skilled and well-funded attackers that aim to infiltrate networks in a stealthy manner during extended periods of time, e.g., to exfiltrate sensitive data [10].

After an attacker compromises a first host in a network, they can gradually expand their reach throughout the network via *lateral movement*. Once the breach has been detected, a security expert has to identify the hosts compromised to estimate the damage, clean these hosts, and fix potential security issues that enabled the attack in the first place. As the network infiltration can occur over and last for long time periods [15], this procedure is pretty difficult and error-prone. The indicators of compromise (IoC) are often too few to completely reconstruct the attacker movement. Thus, in the worst case the security expert has to clean every host in the network. The result is a high false-positive rate and slow response times to such security incidents. Therefore, new approaches are needed to reconstruct lateral movement from an incomplete set of IoCs and support the forensic process.

There are approaches that systematically model the security properties of a network before an attack has happened to quantify the existing attack vectors. Attack graphs can help to understand, how the attacker was able to compromise a specific host where the IoC was generated. However, it is difficult to retrace possible attack paths when only an incomplete set of IoCs is present.

Our contribution in this paper is twofold.

- First, we introduce three distinct attacker models to describe the lateral movement of APT intruders. These models describe the behavior of *regular, scanning attackers*, *directional attackers* with structural information about the target network, and full-blown *insider attackers*. The basis for the attacker models is a graph-based network model that quantifies attacker interest in different hosts. It consists of two node attributes and can be semi-automatically derived from existing network configurations.
- Second, we present a novel algorithm to efficiently reconstruct attacker lateral movement based on an incomplete set of IOCs. With the goal to assist manual security analyses, the algorithm outputs a candidate set of hosts that most likely have been compromised by the attacker and thus should be prioritized in the forensic process.

The remainder of this paper is structured as follows. In Section 2 we summarize related work. Section 3 introduces the formal models to quantify attacker interest in the network and three distinct attacker models. Section 4 describes our algorithm to reconstruct lateral movement for a given network and alert set. Section 5 illustrates the evaluation setup as well as the obtained results. We conclude our work in Section 6.

2 RELATED WORK

In this section, we summarize related work in the areas of *Advanced Persistent Threats*, *alert correlation*, and *attack graphs*.

APT campaigns expose different characteristics than conventional intrusion attempts. Ping et al. [5] describe four key characteristics for APT attacks: (1) targeting of specific organizations, (2) high level of organization and available resources, (3) formation of long-term campaigns with repeated intrusion attempts, and (4) stealthy operation and use of evasive attack techniques. Especially due to their stealthiness, APT attacks are most often discovered long after their start. Therefore, it is highly important to simplify the forensic process and support security administrators in identifying compromised hosts.

Ussath et al. [16] describe three main phases in their analysis of 22 different APT campaigns: (1) initial infection, (2) lateral movement, and (3) command and control activities. They point out that the lateral movement phase is especially hard to detect, since APT attackers often use legitimate credentials and standard OS tools. Hence, the resulting network traffic produced by these tools looks like legitimate one from the perspective of a network-based IDS. Therefore, our approach is based on abstract IoCs which can be traditional IDS alerts, but also policy violations and others.

Alert correlation describes the overall process to identify the underlying attacks on large sets of alerts or other IoCs. This can involve aggregation, clustering, and classifying of IoCs and generally results in some sort of label or description for a specific (sub-)set of the processed IOCs. Salah et al. [12] provide a good overview across the different methodologies and techniques applied. Clustering approaches as presented in [8] summarize a set of IoCs into a single higher-order IoC and thus reduce the number of alerts to process. This reduction is the prerequisite for applying an approach like ours that tries to trace attacker steps.

Attack graphs are a well-known technique to evaluate and quantify attack vectors in large networks. They consist of distinct *attack paths*, so called "kill chains", between multiple hosts and services of a network that the attacker compromises to ultimately gain access to the target host. Swiler et al. [11, 14] and Sheyner et al. [13] presented two of the earliest formalized models to build and use attack graphs to analyze vulnerabilities in a network. Their approaches are based on model checking and produce *complete* attack graphs that contain every possible path an attacker might use. However, the complexity of these graphs grows exponentially with the number of hosts in the network as vulnerabilities are modelled for each host. Therefore, complete attack graphs are unsuitable for use in large real-world networks.

Ammann et al. [4] developed a host-based attack graph with the notion of a *maximum level of penetration* to be more scalable. Instead of all possible exploit combinations on all hosts, their graph contains only the maximum access permission that can be obtained on each host. Multiple attack paths that lead to the same level of access are combined and attack paths with lower access levels are discarded completely. The result is a compact attack graph containing only worst-case scenarios. However, it can lead to suboptimal choices in which vulnerability should be repaired by the security administrator as the authors mention themselves. For APT attacks this might be problematic, as attackers do not necessarily obtain every access right they can. Instead they might only use a host to extend their reach into the network and thus stay undetected in such an attack graph.

Typical alert correlation systems operate on a *sliding window* of alerts to cope with the enormous influx of alerts and their resource constraints. In this setup, these alerts that belong to the same attack but occur across a large time-span cannot be correlated as the previous alert has already been discarded from memory when the new alert is generated. To address this, Wang et al. [17] designed *queue graphs* to correlate and predict intrusion alerts on the basis of attack graphs. Their approach is able to correlate alerts across arbitrary time-spans that are realized through the same exploit (but potentially different hosts). Thus, queue graphs could be useful to detect multistage attacks that stay in the network for extended periods of time and might actively delay lateral movement to prevent detection. However, as the approach is limited to attacks using the same exploit in all steps of the attack, it might not be fully applicable for APT attacks.

In general, attack graphs are usually used as a preventive measure to quantify and eliminate important attack vectors on a network. We use the idea to link IoCs together to reconstruct lateral movement and aid the forensic recovery process.

3 FORMAL MODEL

Detecting APT attacks, especially with respect to the attacker's lateral movement, requires a thorough understanding of how an attacker potentially spreads within the network in question. Therefore, we propose formal models for both the network and different classes of attackers. This is the prerequisite to quantify attacker interest in different hosts and to reconstruct their lateral movement through the network.

3.1 Network Model

The network model is based on a directed reachability graph $G_R = (V, E)$ as this matches the possibilities an attacker has during lateral movement. Hosts are modeled as vertices V while edges E represent connectivity between them. The model uses a simplified definition of connectivity that does not consider advanced routing configurations. That is, if any connection can be established from host u to host v, G_R contains the edge (u, v). This is in contrast to other approaches which usually consider connectivity based on port/service.

Furthermore, we define two vertex attributes, *Importance* and *Vulnerability*, that represent properties of the corresponding host that are visible to an attacker. This includes security properties and appliances such as network IDSs, host sensors, and other endpoint protection solutions.

The *Importance* $\mathcal{I}(v) \in (0, 1]$ represents the value of the host for the network. Hosts which run core services such as application and database servers are therefore rated high while regular workstations typically receive a low value. The lower bound explicitly excludes 0 as any compromised host represents a security risk and therefore should not be ignored. $\mathcal{I}$ can be semi-automatically derived from installed software when applying the approach to an existing network. Each software is rated with a certain value which ultimately results in a cumulative *Importance* score for each host. An attacker can estimate $\mathcal{I}$ by scanning a host for open ports or monitoring network traffic to deduce which services are running on the host.

The *Vulnerability* $\mathcal{V}(v) \in (0, 1]$ represents the difficulty for an attacker to compromise the host. This attribute is influenced by a variety of factors such as patch level of the operating system and installed software, known vulnerabilities matching the system configuration, and security policies implemented in the network. The lower bound explicitly excludes 0 to account for zero-day exploits as they can be used to compromise even the most secure hosts. Similarly to the $\mathcal{I}$, $\mathcal{V}$ can be derived semi-automatically from the installed applications, and automated vulnerability scans such as Nessus[1] or other CVSS[2] based scanners. The *Vulnerability* also allows to account for the human factor in IT-security by increasing the value for all hosts that are regularly used by employees, e.g., workstations or even employee-owned smart devices. An attacker can also estimate $\mathcal{V}$ by deducing software versions from banners, raw network packets or other fingerprinting techniques.

$$C(v) = \mathcal{I}(v) \cdot \mathcal{V}(v) \tag{1}$$

Based on these attributes we can assign a *Criticality* $C(v) \in (0, 1]$ value to each host in the network as given in Equation 1. This value represents a prioritization score of the attacker. As these scores can be compared among the hosts, they are used in our attacker model to approximate which hosts an attacker most likely tried to compromise in each lateral movement step.

We chose the product over a linear combination of $\mathcal{I}$ and $\mathcal{V}$ to highlight hosts which are both *important* and *vulnerable* as especially *critical*. It is therefore important to compute accurate values for both $\mathcal{I}$ and $\mathcal{V}$ when applying our approach to a real network. While a certain mismatch between the knowledge base of the security administrator and the attacker can be expected (and addressed in our approach), a more accurate estimation is expected to boost our algorithm's performance.

3.2 Attacker model

Based on the formal network model we can define three attacker models to capture the distinct behavior of different classes of attackers. We differentiate between *basic attackers*, *directional attackers* and *insider attackers*. These three attacker classes differ mainly in their knowledge about the target network.

In our model, all attackers start from a random *vulnerable* host in the network and target an *important* host, e.g., the production database or a crucial gateway system. This closely resembles real attack scenarios, where the initial point of compromise is generally hard to influence by the attacker, but most often is one of the more *vulnerable* hosts. The attackers then move laterally through the network and aim to find the best path to the target host. The *Criticality* $C(v)$ hereby models the cost function an attacker tries to maximize when deciding which host to compromise next. Depending on the attacker's knowledge this results in either a straight path or tree like movement to find the target host.

While these models describe an "idealized" attacker of the respective class, a real intruder will probably deviate from the expected path. This can either be explained by the different knowledge bases of attacker and security administrator mentioned in Section 3.1 or additional reasons specific to each class of attackers.

Basic attackers. These are intruders with strong technical skill that lack detailed information about the target network. To find the best path to the target host, basic attackers have to gradually explore the network while prioritizing hosts with high *Criticality*. Therefore, they generally compromise large parts of the network before reaching the target host. Their lateral movement forms a *broad tree-like graph* that is visualized in Figure 1a.

Reason for *deviation* from the best path, besides the aforementioned mismatch of knowledge, could be exploits (possibly even zero-days) that enable the attacker to compromise an otherwise secure host.

The behavior of basic attackers can be modeled by Dijkstra's algorithm [6]. The link distance function $d(u, v)$ can be modeled by taking the inverse of the *Criticality* of the destination node as given in Equation 2. However, in contrast to most applications of Dijkstra's algorithm, the lateral movement consists of all nodes that were explored instead of the shortest path only. This represents the explorative nature of the lateral movement of basic attackers.

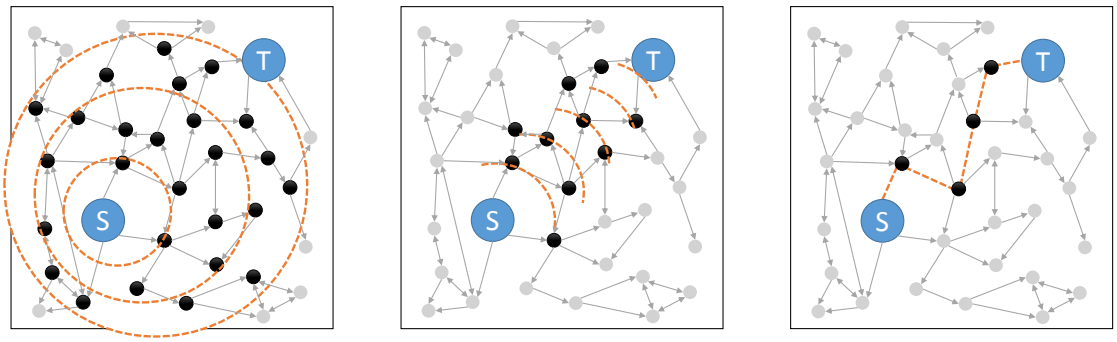
$$d(u, v) = \frac{1}{C(v)} \tag{2}$$

Directional attackers. Intruders with limited, structural information on the network are referred to as directional attackers. They also have to gradually explore the network to find the best path to the target node, but can use their knowledge to make better decisions which host to compromise next. An attacker from this class might know in which subnet the target host is located as well as how subnets are connected. They can then use this information to disregard a host with otherwise high *Criticality* and instead compromise another host that is reachable in different subnet close to the target host. Thus, their lateral movement graph is usually a *medium-sized tree* as multiple branches have to be explored to find the optimal path to the target. Figure 1b shows an example graph for this attacker class.

The behavior of directional attackers can be modeled by the A^* algorithm [9]. The heuristic hereby represents the attacker knowledge and limits the search space the attacker has to explore. It is therefore highly important to choose an appropriate function that closely approximates real behavior.

We chose an euclidean heuristic based on node coordinates assigned by a multidimensional scaling algorithm. Each node gets embedded in a two dimensional plane based on their connecting edges. This results in (x, y) coordinates for each node which represent an abstract location of the host in the network. This could

[1] https://www.tenable.com/products/nessus/nessus-professional
[2] https://www.first.org/cvss/



(a) Basic attacker (b) Directional attacker (c) Insider attacker

Figure 1: Lateral movement graphs for different attacker models. Orange/dashed represents the area of the lateral movement graph, black nodes have been compromised while gray nodes remain noncompromised.

represent a particular subnet or even real geographic coordinates. The heuristic then just calculates the euclidean distance between the target host and current host. The lateral movement graph consists of all nodes that were explored by the A* algorithm.

Directional attackers might *deviate* from the best path for the same reasons as basic attackers, but especially if the heuristic does not accurately model the knowledge of the real attacker.

Insider attackers. We refer to intruders that know the complete network topology, as well as the security properties of each individual host, as insider attackers. Their total knowledge enables them to traverse the network directly from the initial point of compromise to the target host in the most efficient way possible. As a result their lateral movement graph is usually a *simple path* through the network (visualized in Figure 1c). However, insider attackers might still *deviate* from the optimal path for various reasons such as

- (zero-day) exploits (which by definition cannot be considered when calculating host vulnerability),
- legitimate credentials,
- when attackers try to avoid security appliances,
- or if an attacker explicitly tries to cover their tracks by, e.g., partially behaving like an attacker of a different class

This class of attacker can be modeled by any shortest path algorithm that accepts custom edge costs. In the concrete case of directed reachability graphs, Dijkstra's algorithm is again sufficient and the resulting shortest path represents the attacker's lateral movement.

In the next section we will present algorithms to efficiently approximate the lateral movement of these three attacker classes from an incomplete set of IoCs.

4 RECONSTRUCTING LATERAL MOVEMENT

After an APT attack has been discovered, it is essential to efficiently identify which hosts were compromised during the attacker's extended presence in the network to recover quickly and harden the network against future attacks. We present an approach to compute a candidate set of hosts which most likely have been compromised based on the IoCs that could be obtained from the attack.

We assume preprocessed IoCs that have been verified and attributed to the APT attack in question. In particular, we assume that there are no false positives in the IoC set. For alerts from IDSs this usually involves some kind of aggregation or correlation such as presented in [8]. For simplicity, we also assume a IoC were detected on the first and last node in the lateral movement graph, i.e., entry point and target host are known.

The considered problem can then be stated as follows: Given a weighted reachability graph G_R that has been annotated according to our network model (see: Section 3.1) and a set of strictly ordered IoCs I^*, we aim to identify the target set of hosts the attacker compromised on its way to the target host V_t. As the lateral movement cannot be fully reconstructed without human verification, the algorithm returns an approximated candidate set of compromised hosts V_c. Additionally, V_c should be ordered, such that the nodes that are most likely compromised, can be inspected first.

Figure 2 shows the two variants of the algorithm that we designed for this problem. The basic idea is to approximate V_t by repeatedly iterating the IoC pairs in I^* and calculating possible shortest paths between them, until the result set is large enough. V_c then contains an approximation of the unknown target set of compromised nodes V_t. The desired size of V_c is given by the parameter $\tau \in (0, 1]$ and expressed as a percentage of all nodes in the graph $|G_R|$. Lastly, V_c should also be ordered, to help prioritize hosts during manual security analysis.

K-shortest Paths. The algorithm based on k-shortest paths is shown in Figure 2a. For the concrete implementation we chose Yen's k-shortest paths algorithm [18] as it is able to compute multiple shortest paths with increasing length, while keeping state between iterations to avoid costly recomputations. The distance function required by the algorithm is the same as given in Equation 2 for the attacker models (see: Section 3.2). The result set V_t is an append-only list of nodes and thus directly fulfills our order requirement. Nodes are added in the order they are discovered by the k-shortest paths algorithm. The `Update` function first checks whether a new node n is already present in V_t before appending it.

Random walks. The second algorithm is based on random walks and is summarized in Figure 2b. Our expectation is that random

```
Input  : G_R, I*, τ
Output : V_c
1  i ← 0
2  V_c ← ∅
3  while |V_c| < τ · |G_R| do
4      foreach (u, v) ∈ I* do
5          nodes ← K-Shortest-Path(G_R, u, v, i)
6          Update(V_c, nodes)
7      end
8      i ← i + 1
9  end
10 return V_c
```

(a) Variant based on k-shortest paths

```
Input  : G_R, I*, τ, limit
Output : V_c
1  V_c ← ∅
2  for i ← 0 to limit do
3      foreach (u, v) ∈ I* do
4          hitcounts ← RandomWalk(G_R, u, v)
5          if v in hitcounts then
6              Update(V_c, hitcounts)
7          end
8      end
9  end
10 return OrderAndTruncate(V_c, τ · |G_R|)
```

(b) Variant based on random walks

Figure 2: Algorithms for reconstruction of attacker lateral movement

walks that are biased according to the criticality of neighboring nodes are more robust against attackers that deviate from the proposed attacker models.

Each random walk simulates a single attacker which moves laterally through the network starting from u to v. `RandomWalk` returns after either v was hit or the random walk timed out, performing more than a predefined amount of maximum steps. The return value is a set of node hit counts that contains each node that was visited by the random walk. If v is in this set, the walk reached the target node and V_c gets `Updated` with the new hit counts. Instead of exiting the loop when enough results have been obtained, this algorithm takes a `limit` parameter. This parameter sets the number of random walks that are performed and thus directly influences the runtime of this variant. As each random walk roughly takes the same amount of time, the `limit` parameter offers fine-grained control over the runtime. In contrast the variant on k-shortest paths is harder to predict, as the runtime of each iteration takes more time as the shortest paths get longer.

After the loop was exited, V_c contains the cumulative hit counts for all nodes that were hit in at least one random walk. V_c is then ordered by hit counts descending and truncated to the desired size of $\tau \cdot |G_R|$. The result is an ordered list of the nodes that were hit the most in all successful random walks.

$$P_t(u, v) = \frac{1}{|S(u)|} \tag{3}$$

$$P'_t(u, v) = \frac{C(v)}{\sum_{s \in S(u)} C(s)} \tag{4}$$

In order to adapt the random walks to our attacker model, we modify the transition probability $P_t(u, v)$ to be *biased* towards C. $P_t(u, v)$ describes how likely a random walk transitions from node u to node v. As shown in Equation 3 the conventional, *unbiased* function simply factors in the out-degree of u and assigns each possible successor $s \in S(u)$ the same probability.

The biased transition function prioritizes nodes with higher *Criticality* as given in Equation 4. The probability is essentially a "relative *Criticality*" value normalized by the sum of *Criticalities* of all neighbors. This resembles the basic assumption of all three of our attacker models: Each attacker generally compromises hosts with higher values for C to find the best combination of *Vulnerability* and *Importance* out of all available hosts.

5 EVALUATION

We implemented both variants of the algorithm in a Python-based prototype and evaluated the performance in different scenarios. In this section, we first describe the general experiment setup and data sets that were used followed by a discussion of the results.

5.1 Setup and Data Sets

There are no data sets openly available for either real enterprise network topologies and few data sets containing IoC data. Large organizations fear the knowledge contained in these sets might enable future attacks. Therefore, we evaluated our approach on data sets consisting of two types of generated graphs as well as IoC sets from synthetic lateral movement.

We evaluated our algorithm on standard preferential attachment graphs as they are structurally similar to Internet or data center topologies [7], [1]. While these networks do not exactly match our targeted enterprise networks, the graphs should provide a decent approximation. The graphs were generated using the Barabási-Albert algorithm [2] with the parameter m set to 4—i.e. 4 edges were added from a newly generated node to the existing graph.

Node Attributes. After the graphs have been generated, they have to be prepared for the algorithm following our network model (see: Section 3.1). We assign *Importance* and *Vulnerability* attributes to all nodes to represent the security properties of the corresponding hosts. Both attributes are sampled from certain probability distributions to approximate real-world networks.

In a typical enterprise network the *important* services (such as web servers, SSH gateways, or databases) are usually deployed on a

small number of hosts, while the majority of the IP address space is occupied by regular workstations with relative low impact on the overall network function. Therefore, we sampled $\mathcal{I}$ from a *gamma distribution* with $\alpha = 4$ and $\beta = 0.075$ and limited the result to a maximum of 1 to fit our model. The corresponding probability density function is shown in Figure 3. The mode of this distribution is about 0.3 that matches an average workstation, while hosts with an attribute value of more than 0.6 are sampled less likely to account for typical server setups.

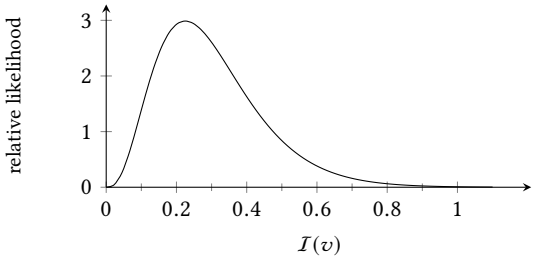


Figure 3: Probability distribution for $\mathcal{I}(v)$ ($\alpha = 4$, $\beta = 0.075$)

We assume that the *Vulnerability* of hosts is distributed inversely to their *Importance*. The majority of workstations are more likely to be outdated, uniquely configured or set up with a local administrative account controlled by the user. All of these factors increase $\mathcal{V}$ for these hosts compared to centrally provisioned server machines with automatic security updates. As a result, we sample $\mathcal{V}$ from $1 - gamma(\alpha = 4$ and $\beta = 0.075)$ and limit the value to the required interval $(0, 1]$.

As mentioned previously in Section 3.2 our implementation of the directional attacker requires node coordinates for the euclidean heuristic. We used Gephi[3] with a plugin based on [3] to perform multidimensional scaling and embed the nodes into the two-dimensional plane.

Attack generation. After G_R has been prepared with node attributes, we generate synthetic attacks based on lateral movement along the three attacker classes. Moreover, we use a *deviation* parameter $\delta \in [0, 1]$ that determines how much the synthetic attack deviates from the idealized attacker model. δ is applied on the graph by selectively setting the edge cost of a percentage of edges to $2^{63} - 1$, the maximum value of an 64-bit integer. For $\delta = 0.2$ this means that the cost for every fifth edge is set to this value, effectively removing it from the graph for the shortest path calculation. This simulates an attacker that deviates in 20% of cases, because 20% of edges in the graph will (most likely) not be on the shortest path to the target node.

The start and end nodes of each lateral movement graph are chosen based on our assumption that an attack starts on a *vulnerable* host and targets an *important* one (see: Section 3.2). To account for this we calculated all shortest paths between the nodes with the top 10% of *Vulnerability* $\mathcal{V}$ and the top 10% of *Importance* $\mathcal{I}$ according to the algorithm for the respective attacker class. We then choose the lateral movement graph from one of the 10% longest paths to have enough nodes to reconstruct later. For insider attackers that is just the path itself—for both other attacker classes the lateral movement graph consists of all visited nodes. Lastly, the result stored as an ordered set of nodes.

[3]https://gephi.org/

Alert placement. After the lateral movement has been generated, IoCs were randomly placed on the set of nodes. We introduce another node attribute called *Detectability* $\mathcal{D}(v) \in [0, 1]$ that indicates how likely a compromise is detected on the host. When applying the approach to a production network $\mathcal{D}$ can be derived from the installed security appliances in the network and on the hosts. For our synthetic networks, we assume a balanced security setup with some hosts protected by IDSs and some honeypots deployed in the network (which carry a high detectability). Therefore, we sample $\mathcal{D}$ from a *normal distribution* with $\mu = 0.5$ and $\sigma = 0.15$. To place the alerts a random number from $[0, 1]$ is generated for each node in the lateral movement graph. If that number exceeds $\mathcal{D}$, an IoC is placed on the node. Additionally, the first and last node always receive an IoC—following our assumption that point of entry and target of the attack are known.

Data sets. For our experiments, we generated two data sets differing mainly in the size of the preferential attachment graphs. Table 1 gives an overview across both data sets. Next to the number of nodes in G_R we also highlight the number of compromised nodes $|V_t|$ for each attacker class on average and the average number of alerts placed on these nodes $|I^*|$. The numbers give a rough overview about the dimensions of an attack from the respective attacker class. Basic attackers compromise about 75% of nodes on average while directional attacker reach the target node after compromising $46 - 50\%$ respectively. Insider attackers only need to compromise $2 - 4\%$ of all nodes as they can take the shortest path directly. Initially, all experiments were performed on both data sets but we saw that network size (at least between 200 and 500 nodes) had no significant impact on the results. Therefore, all graphs in the following section were obtained from the result set with $|V| = 200$ with 50 repetitions to reduce runtime.

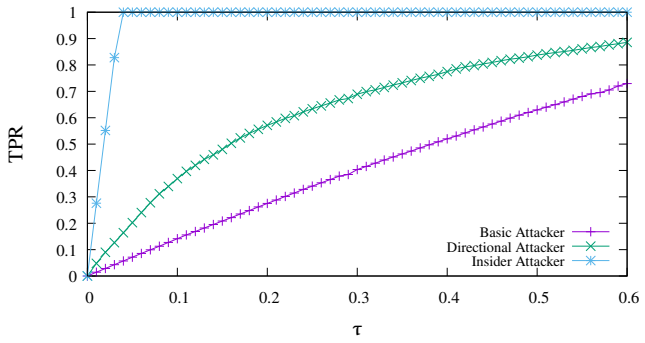
5.2 Metrics

To evaluate our algorithm, we performed multiple experiments based on the setup described in the previous section. In the following, we treat the two algorithms as binary classifiers, i.e., the hosts that are contained in V_t are classified as compromised, while the remaining ones are classified as noncompromised. This is formalized in Equation 5. The *true positive rate (TPR)* is calculated by dividing the number of correctly labeled host by the total number of compromised hosts V_c.

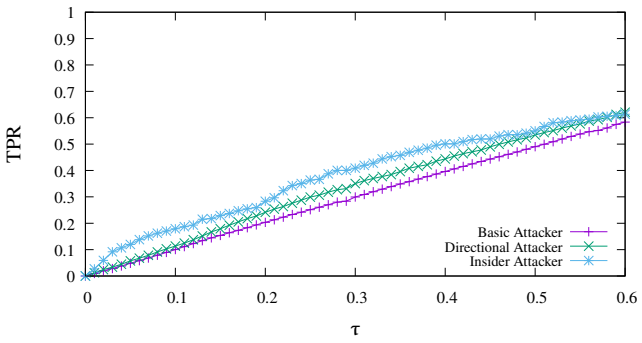
$$TPR = \frac{|V_t \cap V_c|}{|V_c|} \tag{5}$$

$$FPR = \frac{|V_t \setminus V_c|}{|V \setminus V_c|} \tag{6}$$

As shown in Equation 6 the *false positive rate (FPR)* is calculated by dividing the number of incorrectly classified hosts by the number of noncompromised host (all hosts that are not in V_c). Based on this, we can perform standard statistic analyses to determine the performance of the two variants of the algorithm and finally derive a recommended value for τ. Each experiment was repeated 50 times



(a) K-Shortest Paths



(b) Random Walks

Figure 4: Performance analysis for idealized attackers ($\delta = 0$)

	Basic attacker		Directional attacker		Insider attacker	
$\|V\|$	$\|V_c\|$	$\|I^*\|$	$\|V_c\|$	$\|I^*\|$	$\|V_c\|$	$\|I^*\|$
200	151	76	93	47	8	5
500	383	185	270	131	11	6

Table 1: Data sets

for both variants—one based on *k-shortest paths* and the other based on *random walks*.

5.3 Experiments

In this section, we present the experiments we conducted to evaluate our approach and discuss the results that we obtained.

Performance for idealized attackers. In the first experiment we aim to analyse the performance of both algorithm variants for idealized attackers that follow our attacker models exactly (c.f. Section 3.2). The result can be treated as ground-truth to validate the core functionality of our approach. To achieve this, we set the *deviation* δ to zero and measure the TPR for varying values of τ. Hereby, values greater than 0.6 were not evaluated for τ as it is typically not useful nor feasible to analyse more than 60% of the whole network. Additionally, this limit reduces the runtime of single experiments considerably. The results are shown in Figure 4. The plots show the average TPR across 50 different graphs for each attacker class and $\tau \in [0, 0.6]$.

The reconstruction of lateral movement is most accurate for the insider attacker and least accurate for the basic attacker. The results indicate that both variants are capable to approximate the set of compromised hosts effectively with the implementation based on *k-shortest paths* performing slightly better for both basic and directional attackers and substantially better for insider attackers. This is expected as the insider attacker strictly compromises hosts on the shortest path that is the first path returned by the k-shortest paths algorithm. While directional attacker movement cannot be fully reconstructed, the TPR is promising especially for lower values of τ. For the basic attacker the TPR is increasing slightly better than linear with τ. The random walk based variant performs strictly inferior for all three attacker classes with almost linear performance. This is somewhat expected as δ is set to zero for this experiment, favoring k-shortest paths. However, we expected a better TPR especially for insider attackers and a higher maximum TPR for $\tau = 0.6$ and all three attacker classes.

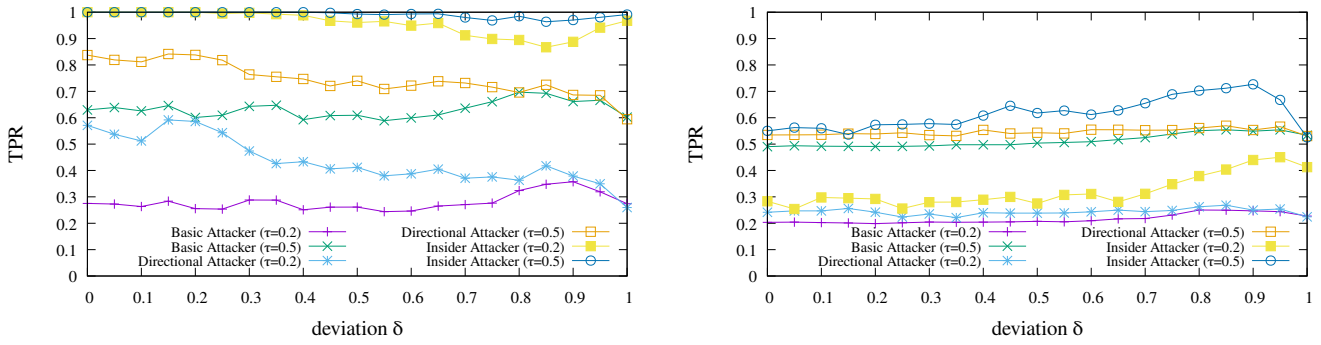
Performance for deviating attackers. Next, we aim to analyze the performance of our algorithm for more realistic attackers that deviate from the idealized models. Although the models capture expected attacker behavior, it is unlikely that a real attacker follows them exactly. Even if that were the case, we still expect some deviation, because of the different knowledge bases of attacker and security administrator (see: Section 3.1).

Therefore, in the second experiment we evaluate the influence of the deviation on the performance of both algorithm variants. The deviation parameter δ is varied in $[0, 1]$ and we measure the TPR for τ set to 0.2 and 0.5 per attacker class respectively. The results are shown in Figure 5. The plots show the average TPR across the 50 runs for $\delta \in [0, 1]$.

The graphs indicate that δ has a small negative impact on the TPR for both intermediate and insider attackers, while basic attackers are only minimally affected. The variant based on k-shortest paths is able to reconstruct 90% of insider lateral movement with τ set to 0.2 even when the attacker deviates 80% from the expected behavior. For both other attacker classes, the TPR basically scales linearly with the size of τ as the lines for 0.2 and 0.5 are basically offset by a fixed size on the y-axis. This is expected, when we look at the graphs of these attacker classes in Figure 4. Additionally, the TPR is worse for the directional attacker and high values of δ while the performance for the basic attacker is nearly unaffected.

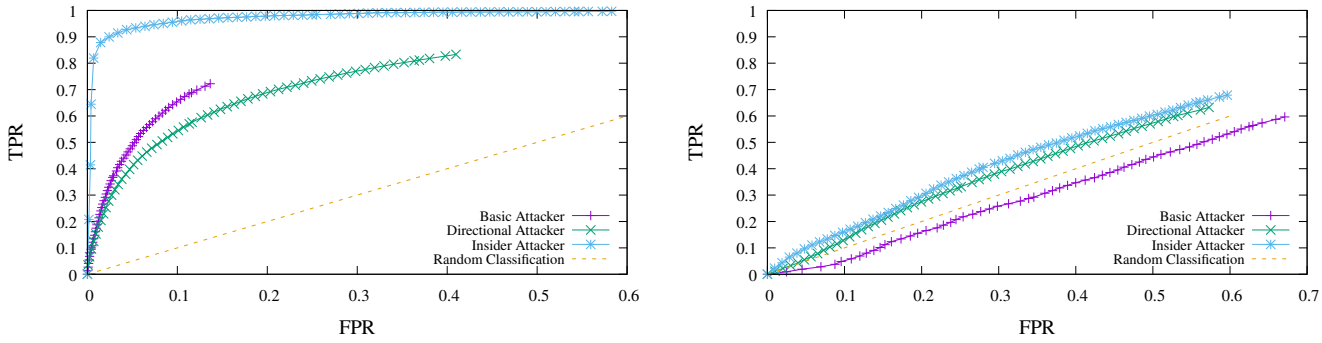
The performance of the random walk-based variant is again strictly inferior. However, it is even less affected by deviation and even sees an increase in TPR for the insider attacker and high values of δ. The influence of τ also appears to be mostly linear.

This poor performance is unexpected, as we explicitly designed this variant to reconstruct lateral movement graphs that do not closely follow the attack model. One reason for this could be how the deviation parameter δ is applied to the graph. As mentioned



(a) K-Shortest Paths **(b) Random Walks**

Figure 5: Deviation influence on classification performance



(a) K-Shortest Paths **(b) Random Walks**

Figure 6: ROC curves for $\tau \in [0, 0.6]$ and $\delta \in [0, 0.75]$

previously in Section 5.1 δ increases the cost for a percentage of the edges to $2^{63} - 1$, so that they are not chosen by the attacker. As the edges are chosen randomly, a low value for δ could result in the modification of edges that would not have been part of the lateral movement graph anyway. In these cases, the attacker compromises the same hosts as if there were no deviation. However, this mostly relates to the insider attacker, as this class naturally compromises the least amount of hosts compared to the other models (see: Table 1). Thus, it is least affected by our implementation of the deviation.

Recommendations for τ. In the final experiment we aim to derive useful values for τ for real-world production use. As mentioned previously τ greatly influences the runtime of the algorithm and thus should be carefully chosen. If the chosen value for τ is too low, the candidate set V_t is too small to contain all compromised hosts. High values of τ increase runtime of our algorithm and are also expected to increase the false positive rate (FPR). To approximate real attackers we include runs with a deviation up to 0.75 in this experiment.

We visualize the effect of τ on the performance of our approach in a receiver operating characteristic (ROC) curve in Figure 6. Each data point represents a value of τ in $[0, 0.6]$. The plots also contain a graph representing a random classifier, i.e., a function that randomly classifies hosts as compromised or noncompromised. All points above this line imply a classification performance better than random. Thus, a perfect classifier would produce a result at the $(0, 1)$ coordinate. Random classification is of course not a perfect approximation of real security administrators. However, it serves as a good baseline to compare both implementations.

The k-shortest paths variant achieves great performance for insider attackers with 0.88 TPR and only 0.01 FPR for $\tau = 0.05$. The performance for directional attackers is slightly less good. However, it still offers a big improvement over random classification especially for low values of τ. The graph for basic attackers also indicate a good performance trend. The ratio between TPR and FPR favors the TPR, i.e., more compromised hosts are correctly classified than wrongly classified. For low values of τ the candidate set V_c is too small to contain all compromised nodes and thus cannot offer a high TPR. The same applies for the directional attacker, although the gradient is nearly identical to the graph of the basic attacker, it decreases faster starting from $\tau = 0.15$.

The variant based on random walks offers a linear performance. For insider and directional attackers the TPR is slightly better and

for basic attackers inferior to random guessing. For basic attackers this could be leveraged by inverting the classification and thus achieving a slightly better performance. However, as k-shortest paths outperformed random walks for every attacker class and deviation, we did not implement this. While these results could be expected from the previous experiment, they ultimately disqualify the random walks based variant as a useful implementation of our approach. Therefore, all recommendations that follow are given for the k-shortest paths variant.

The recommended value for τ differs depending on the amount of resources available and how many false positives should be tolerated. Also, in real-world scenarios a security administrator would not run our approach once and analyze all results but rather start from the beginning of the candidate set and rerun the algorithm once a new IoC has been verified.

A sensible choice for insider attackers is $\tau = 0.05$ as this achieves 0.88 TPR with only 0.01 FPR. If more false positives can be tolerated, $\tau = 0.35$ achieves 0.99 TPR with a still relatively low FPR of 0.16.

For directional attacker the ideal value for τ should be between 0.1 and 0.2. Values below 0.1 contain too few results to achieve a high enough TPR while value above 0.2 start to see increased FPRs. This results in a TPR of 0.27 for $\tau = 0.1$ and 0.47 for $\tau = 0.2$. While this is certainly not enough to properly identify all compromised hosts, the resulting candidate set should contain enough compromised machines to yield more IoCs which can then be fed back into a future iteration of the algorithm.

It is generally difficult to detect all hosts that were compromised by a basic attacker, as they usually compromise many hosts in the network (see: Table 1). The choice of τ offers multiple points to consider. At $\tau = 0.23$ the algorithm achieves a TPR of 0.3 with only 0.02 FPR. If more false positives can be tolerated, $\tau = 0.54$ achieves 0.66 TPR at the cost of 0.1 FPR. Finally, as so many hosts are compromised anyways, it might even make sense to chose $\tau = 0.6$. This results in 0.72 TPR at still only 0.13 FPR.

6 CONCLUSION

In this paper, we introduced three attacker models to describe intruder behavior as well as a novel algorithm to reconstruct attacker lateral movement from an incomplete set of IoCs. The variant based on k-shortest paths has promising results and is especially effective in reconstructing lateral movement of insider attackers.

We implemented and evaluated two variants of the algorithm for idealized attackers, robustness against deviating attackers, and finally provided recommended values for τ, the size of the resulting candidate set, to enable real-world use-cases. The implementation based on k-shortest paths achieves a TPR of 90% for attackers that deviate up to 75% from our idealized model with a candidate set of the size of 5% of the total number of hosts.

For future work, we plan to extend our proposed attacker model to account for budgeted or otherwise resource constrained attackers and adapt the algorithm accordingly. This would allow for reconstruction of unsuccessful attacks (that were aborted due to constraints). Similarly the heuristic of the directional attacker could be adapted or the directional attacker could even be split into different attacker classes that use distinct heuristics to model other complex attackers.

REFERENCES

[1] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication - SIGCOMM '08*. ACM Press.

[2] Réka Albert and Albert-László Barabási. 2002. Statistical mechanics of complex networks. *Reviews of Modern Physics*.

[3] Algorithmics Group, University of Konstanz. 2009. MDSJ: Java Library for Multidimensional Scaling. (2009).

[4] Paul Ammann, Joseph Pamula, Ronald Ritchey, and Julie Street. 2005. A host-based approach to network attack chaining analysis. *Annual Computer Security Applications Conference (ACSAC)*.

[5] Ping Chen, Lieven Desmet, and Christophe Huygens. 2014. A Study on Advanced Persistent Threats. *Communications and Multimedia Security*.

[6] Edsger W Dijkstra. 1959. A note on two problems in connexion with graphs. *Numerische mathematik*.

[7] Albert Greenberg et al. 2009. VL2: A Scalable and Flexible Data Center Network. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication - SIGCOMM '09*. ACM Press.

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

[9] Peter E Hart, Nils J Nilsson, and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*.

[10] NIST. 2011. Managing Information Security Risk: Organization, Mission, and Information System View. *Nist Special Publication*.

[11] Cynthia Phillips and Laura Painton Swiler. 1998. A graph-based system for network-vulnerability analysis. *Workshop on New security paradigms*.

[12] Saeed Salah, Gabriel Maciá-Fernández, and Jesús E. Díaz-Verdejo. 2013. A model-based survey of alert correlation techniques. *Computer Networks*.

[13] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing. 2002. Automated generation and analysis of attack graphs. *IEEE Symposium on Security and Privacy (S&P)*.

[14] L. P. Swiler, C. Phillips, D. Ellis, and S. Chakerian. 2001. Computer-attack graph generation tool. *DARPA Information Survivability Conference and Exposition DISCEX*.

[15] Symantec. 2016. Internet Security Threat Report 2016. *Internet Security Threat Report*.

[16] Martin Ussath, David Jaeger, Feng Cheng, and Christoph Meinel. 2016. Advanced persistent threats: Behind the scenes. In *2016 Annual Conference on Information Science and Systems (CISS)*. IEEE.

[17] Lingyu Wang, Anyi Liu, and Sushil Jajodia. 2006. Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts. *Computer Communications*.

[18] Jin Y. Yen. 1971. Finding the K Shortest Loopless Paths in a Network. *Management Science*.