Building Evidence Graphs for Network Forensics Analysis

Wei Wang, Thomas E. Daniels
Department of Electrical and Computer Engineering
Iowa State University
Ames, Iowa 50010
{weiwang, daniels}@iastate.edu

Abstract

In this paper, we present techniques for a network forensics analysis mechanism that includes effective evidence presentation, manipulation and automated reasoning. We propose the evidence graph as a novel graph model to facilitate the presentation and manipulation of intrusion evidence. For automated evidence analysis, we develop a hierarchical reasoning framework that includes local reasoning and global reasoning. Local reasoning aims to infer the roles of suspicious hosts from local observations. Global reasoning aims to identify group of strongly correlated hosts in the attack and derive their relationships. By using the evidence graph model, we effectively integrate analyst feedback into the automated reasoning process. Experimental results demonstrate the potential and effectiveness of our proposed approaches.

1 Introduction

With the increasing scale and impact of cyber attacks, various network security techniques have been developed to fight against invisible attackers. These techniques address different aspects of security needs. For example, firewalls are generally used for prevention of attacks while IDS systems are mainly used for detection of attacks. However, these defensive mechanisms are not sufficient to eliminate cyber attack threats. Because we can never block all paths for cyber crimes, effective investigation techniques would help us collect, analyze and present evidences of cyber attacks to hold the attackers responsible for their malicious actions. These investigation methods belong to the realm of network forensics.

More formally, network forensics is a subfield of digital forensics where evidence is captured from networks and interpretation is substantially based on knowledge of cyber attacks. It aims to locate the attackers and reconstruct their attack actions through analysis of intrusion evidence. Below we summarize two major technical challenges facing network forensics analysis:

- Current sources of intrusion evidence such as IDS alerts are not well adapted for forensics investigation. Forensics analysts are often buried in large volume of low-level event logs. There exists much redundancy and unrelated background noise while parts of useful information may be missing or incomplete.
- Cyber attacks are increasingly sophisticated. There exist more multi-stage attacks consisting of several evolving phases and spanning over large number of hosts, which increases the difficulty of analysis.

In view of these challenges, the investigation needs of cyber attacks have not been met by available technologies. Current practices in network forensics analysis are to manually examine logs, a time-consuming and error prone process [13]. We argue that network forensics analysis mechanisms should meet the following requirements:

- Short response times: Large volume of irrelevant information and increasingly complex attack strategies make manual analysis impossible in a timely manner. Automated evidence analysis would produce an immediate impact on law enforcement's ability to reduce response times.
- Friendly interface: Intrusion evidence and analysis results should be presented in an intuitive approach. The ad-hoc nature of cyber attacks indicates that expert opinion and out-of-band information must be efficiently integrated into the automated reasoning process.

In this paper we present a prototype network forensics analysis mechanism that integrates novel techniques for evidence presentation, interaction and automated reasoning. We propose the evidence graph to model intrusion evidence. Based on the evidence graph, we develop a hierarchical reasoning framework for automated evidence analysis.

Major objectives of network forensics analysis can be summarized into two fundamental problems: attack group identification and attack scenario reconstruction. Attack scenario reconstruction is the process of inferring step-wise actions taken by the attacker to achieve his malicious objective. Attack group identification is the task of discovering the group of hosts involved in the attack and determining the roles of each host in the group. Common roles in an attack group include:

- *Attackers* are sources of primary attack actions in the scenario of interest.
- *Victims* are target hosts that were compromised in the attack.
- *Stepping Stones* are compromised hosts that are later used to attack other hosts and distract the analyst's attention from the real attacker.
- *Background Attackers* are hosts that initiate malicious activities but are irrelevant to the primary attack of interest.

Our evidence analysis mechanism focus on identifying members of an attack group and their relationships, which would help to answer questions like:

- How likely is a specific host relevant to the attack?
- What is the role the host played in the attack?
- How strongly are two hosts M and N connected in the attack ?

In summary, our work aims to provide the following contributions to network forensics analysis:

1. A flexible pre-processing mechanism that reduces redundancy in intrusion alerts;
2. A novel graph model that facilitates effective presentation and interaction with intrusion evidence;
3. A hierarchical reasoning framework for automated intrusion evidence analysis.

The rest of the paper is organized as follows. The next section presents the architecture of our forensics analysis mechanism and describes each component. Section 3 proposes our hierarchical reasoning framework. In section 4, we discuss related work. In section 5, we provide experimental results in support of our approach. Section 6 concludes this paper and discusses future work.

2 Network Forensics Analysis Mechanism

In this section, we present the components that support our approach in network forensics analysis. Figure 1 shows the architecture of our network forensics analysis mechanism.

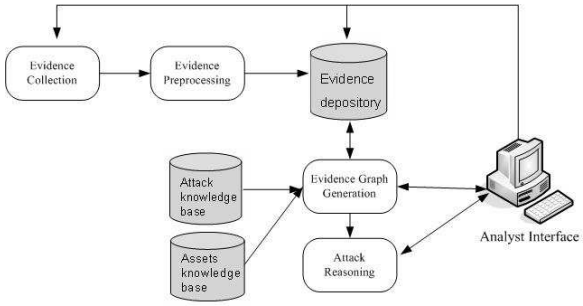

Figure 1. Architecture of analysis mechanism

- *Evidence collection module* collects intrusion evidence from networks and hosts under investigation.
- *Evidence preprocessing module* parses certain types of evidence like intrusion alerts into desired format and reduce the redundancy in low level evidence by aggregation.
- *Attack knowledge base* provides prior knowledge of known exploits.
- *Assets knowledge base* provides prior knowledge of the networks and hosts under investigation.
- *Evidence graph manipulation module* generates and updates the evidence graph by retrieving intrusion evidence in the depository.
- *Attack reasoning module* performs automated reasoning based on evidence graph.
- *Analyst interface module* provides visualization of evidence graph and reasoning results to the analyst and passes analyst feedbacks to the graph generation and reasoning module.

In the initial phase, the intrusion evidence collected are pre-processed and stored into the evidence depository. Next, the graph generation module constructs the evidence graph with evidence retrieved from the depository. Following that, the reasoning module performs automated inference based on the evidence graph and present results to the analyst. Through the interface module, the analyst could provide expert opinions and out-of-band information in two

approaches: (1) directly edit the evidence graph; (2) send queries to retrieve specific evidence. The reasoning process is then performed on the updated evidence graph for improved results.

2.1 Sources of Intrusion Evidence

Evidence for network forensics investigation can be classified into two categories: primary evidence and secondary evidence. Primary evidence refers to information that directly indicates attacks or security policy violations. Secondary evidence refer to information that does not directly represent attacks but could provide complementary information for investigation. Secondary evidence comes from extensive sources and in a much higher volume. Generally, primary evidence is the starting point of forensic investigation and provides the basis for searches towards secondary evidence. Querying the secondary evidence usually has two objectives: to discover hidden suspicious events and to evaluate the trustworthyness of primary evidence. In our current prototype, we use network IDS alerts as the primary evidence; raw network flow logs and host logs are used as secondary evidence.

2.2 Evidence Preprocessing

In our prototype, IDS alerts are used as the source of primary evidence. We define a simplified template derived from IDMEF [12] to capture essential intrusion alert attributes. The result is denoted as raw alert. Format of raw alert is $\{AlertID, Classification, SrcIP, DesIP, DetectTime, HyperID\}$.

The large amount of redundancy in raw alerts makes it difficult to analyze the underlying attacks in an efficient manner. For example, a single event often generates many duplicate alerts in a short period. We use alert aggregation based on similarity of attributes and context requirements to merge raw alerts into hyper alerts. Format of hyper alerts is $\{HyperID, Classification, SrcIP, DesIP, StartTime, EndTime, Count\}$. The alert aggregation process aims to remove the duplicates and generate hyper alerts that are easy to analyze without losing granularity of important information.

Each hyper alert has a one-to-many relationship with raw alerts. In the hyper alert template, the $Count$ field records the number of raw alerts that are merged into the hyper alert for statistical evaluation. The $HyperID$ field in the raw alert template records the unique identification number of the hyper alert it merged into. The index mapping between raw alerts and hyper alerts enable analysts to backtrack and examine alerts in finer scale.

We apply a flexible alert aggregation algorithm based on the Leader-Follower model. In essence it adapts the similarity-based alert correlation method proposed by Valdes and Skinner [21]. The aggregation criteria is to combine alerts that have the same source-destination pair, belong to the same attack class and whose time stamp falls in a self-extending time window. The time window of hyper alerts is self-extending in that if time stamp of the raw alert falls outside the [start time, end time] window of hyper alert but the difference is within a predefined limit T, corresponding bound of the hyper alert time window is updated with the time stamp of the raw alert. This implies that we are able to merge continuous duplicate raw alerts that span over a long period into a single hyper-alert with a proper T.

The Leader-Follower alert aggregation procedure is shown in algorithm 1.

input : A set of raw alerts $r_1 \dots r_n$, time limit T
output: A set of hyper alerts $h_1 \dots h_m$
begin
  $h_1 \leftarrow r_1$;
  $m \leftarrow 1$;
  for $i \leftarrow 2$ **to** n **do**
    $merged \leftarrow 0$;
    for $j \leftarrow 1$ **to** m **do**
      if $r_i.sourceaddr = h_j.sourceaddr$ && $h_j.destaddr = r_i.destaddr$ && $h_j.class = r_i.class$ && $h_j.starttime - T \leqslant r_i.detecttime \leqslant h_j.endtime + T$ **then**
        $h_j.starttime \leftarrow min(h_j.starttime, r_i.detecttime)$;
        $h_j.endtime \leftarrow max(h_j.endtime, r_i.detecttime)$;
        $r_i.hyperid \leftarrow h_j.id$;
        $h_j.count \leftarrow h_j.count + 1$;
        $merged \leftarrow 1$;
        $break$;
      end
    end
    if $merged = 0$ **then**
      $m \leftarrow m + 1$;
      $h_m \leftarrow r_i$;
      $h_m.count \leftarrow 1, h_m.HyperID \leftarrow m$;
    end
  end
end

Algorithm 1: Leader-Follower alert aggregation

In practice, one important variant that affects the results of aggregation is the evaluation of attack class. It is common to observe that multiple different attack classes are defined for attacks exploiting the same vulnerability or having similar result. When we wish to ignore such trivial differences, we may consider the class of attack on a higher abstraction level. For example, a "SCAN Nmap TCP" alert

and a "SCAN Nmap XMAS" alert generated by Snort can be merged into one hyper alert with the same abstracted class "SCAN Attack". We note that alert abstraction requires expert knowledge and defining an appropriate abstract class is largely a manual process.

The Leader-Follower process is flexible to tailor the aggregation criteria for specific contexts. For example, by evaluating traces of attacks we discover that a large portion of raw alerts is triggered by port scan activity, which is usually of little significance. Therefore we use a strategy that only cares for hosts that are either source or target of large number of scan-related alerts. The former often represents an attacker while the latter often indicates a potential victim of attack. Consequently the flexible aggregation criteria uses an abstract type that represents all scan-related alerts and only require match of either $sourceaddr$ or $destaddr$.

2.3 Evidence Graph Structure

In the following, we define the evidence graph model as the foundation of our analysis mechanism. Functionalities of the evidence graph model include:

1. The evidence graph provides the analyst an intuitive visualization of observed evidence;
2. The evidence graph provides a convenient interface for the analyst to interact with the evidence and add expert feedback.
3. The evidence graph provides the basis for automated reasoning procedure.

Definition 1. An evidence graph is a quadruple G= (N, E, S, R), where N is the set of nodes, E is the set of directed edges, S is the set of labels that indicate the states of nodes and R is the set of labels that indicate the attributes of edges. In the evidence graph, each node represents a host of forensic investigation interest and each edge represents an observed intrusion evidence.

Each node in the evidence graph is characterized by the following labels:

1. Host: Identification of the suspicious host.
2. States: States of the node is defined by a set of fuzzy variables F=$\{Attacker,\ Victim,\ Stepping\ Stone,\ Affiliated\}$. The fuzzy variable $Attacker(AT)$ indicates the belief that the current node is a source of attack. The fuzzy variable $Victim(VI)$ indicates the belief that the current node is a target compromised in the attack. The fuzzy variable $Stepping\ Stone(SS)$ indicates the belief that the current node is controlled by another host and used as a stepping stone in the attack. The fuzzy variable $Affiliated(AF)$ indicates the belief that the current node has suspicious interactions with an attacker, victim or stepping stone host. Note that these states are not mutually exclusive. For example, a victim host that was compromised in an attack could be used as a storage relay to transfer stolen files. Therefore states of the host will evolve from "victim" to both "victim" and "affiliated".
3. Time stamps: Each state variable is associated with two time stamps: $T_{activate}$ records the initial time the state is activated above a certain threshold and T_{latest} records the latest time of update.

Each edge in the evidence graph is represented by the following labels:

1. General attributes: The set of general attributes of an edge depends on the specific type of intrusion evidence. For network IDS alerts, we define the set of attributes as source/target IP address, time stamp and classification. Time stamp of the edge is an interval [start time, end time].
2. Weight: Weight is a fuzzy value $w \in [0, 1]$ that represents the impact of evidence. For example, a port scan alert that has little seriousness is assigned a weight of 0.1 while a buffer overflow attack that could gain root privilege on the target system is assigned a weight of 0.8. In our prototype, we model the known attacks and assign the weight of IDS alerts based on expert knowledge.
3. Relevancy: Relevancy value represents the belief that the attack indicated by the evidence would successfully achieve expected impact on the target host. Specifically, there could be three cases behind an alert: relevant true positive, false positive and non-relevant true positive [14]. Relevant true positive refers to alerts that truly represent an attack and the attack achieves its expected impact. False positive refers to alerts that identify a legitimate event as an alert by mistake. Non-relevant true positive refers to alerts that truly represent an attack but the attack does not achieve its expected impact. We define relevancy value for these three cases as follows:

$$r = \begin{cases} 0, & \text{false/non-relevant true positive;} \\ 0.5, & \text{unable to verify;} \\ 1 & \text{relevant true positive.} \end{cases} \quad (1)$$

The process to check the relevancy of an alert is denoted as alert verification. Alert verification is no trivial problem. In our prototype, we compare the prerequisites of an attack with target host's configuration. If prerequisites of the attack are completely satisfied, the relevancy value is assigned as 1. If contradicting

configuration is found, the relevancy value is assigned as 0; otherwise the relevancy value is assigned as 0.5. This approach could rule out attacks that are bound to fail because the target host is not vulnerable. However it cannot guarantee that attacks tagged as relevant are truly successful, because the attack could still fail because of an incorrect parameter. Recently Kruegel and Robertson [14] proposed an active verification mechanism that checks for traces that match the attack's expected outcome on the victim host, but effectiveness of the method needs further study.

4. Host Importance: Host importance $h \in [0, 1]$ is an optional fuzzy parameter to relate importance of evidence with certain hosts. We observe that same events may represent different suspiciousness when associated with different hosts. For example, a port scan that targets a highly valued file server should be assigned higher importance than one targets a public web server.

We calculate priority score for an edge to indicate the overall importance of the intrusion evidence. The priority score $p(e)$ of an edge e is calculated as the product of its weight, relevancy and host importance:

$$p(e) = w(e) \times r(e) \times h(e) \quad (2)$$

As an example, a Windows DCOM buffer overflow attack is observed to initiate from host s against host t. By prior knowledge we know that host t is a Linux server. Therefore the Windows DCOM buffer overflow will have no impact on target t because it is not vulnerable to the exploit. As described before, weight of the edge $w(e)$ comes from the attack knowledge base; relevancy of the edge $r(e) = 0$. Thus priority score of the edge $p(e) = 0$.

Weight and priority score are defined for different implications in our reasoning process. We use $p(e)$ in reasoning for attack group and scenario identification. In the above example, the failed attack has $p(e) = 0$, which indicates that it has little significance in the attack scenario because the attacker did not achieve his goal. On the other hand, we use $w(e)$ in reasoning for host s' state because although the attack is unsuccessful, it still indicates malicious intent of host s and consequently host s' "$Attacker$" state should be updated.

2.4 Building Evidence Graph

To construct the evidence graph, the sequence of intrusion evidence is processed in time order, starting from the first evidence in record and moving towards the latest evidence. Evidence with time intervals is added to the graph in order of the start time in their interval. For each evidence, we evaluate which nodes in the current evidence graph it will affect and create nodes that do not exist, then create the edge accordingly. The algorithm for building the evidence graph is listed as follows.

input : Stream of evidence in time order
output: Evidence graph G
begin
  foreach *evidence* E *in stream* **do**
    foreach *host* V *affected by* E **do**
      if V *does not exist in* G **then**
        `CreateNode` (G, V);
      end
    end
    `CreateEdge` (G, E);
    foreach *host* V *affected by* E **do**
      `UpdateNode` (E, V);
    end
  end
end

Algorithm 2: Constructing an evidence graph

The $UpdateNode$ function updates the states of node by causal reasoning via Rule-Based Fuzzy Cognitive Maps(RBFCM), which we will describe in section 3.

2.5 Edit Operations on Evidence Graph

The evidence graph provides an effective means for the forensics analyst to incorporate expert knowledge and out-of-band information into automated analysis. General operations on the evidence graph includes the following:

1. Insert a new node n: This represents adding a new suspicious host to the evidence graph.

2. Remove a node n: This represents removing an irrelevant host from the evidence graph. Note that removing a node implies removing all the edges connecting to it.

3. Update a node n: This represents changing one or more state values of the node.

4. Insert a new edge e: This represents adding new intrusion evidence between existing nodes in the evidence graph.

5. Remove an edge e: This represents removing irrelevant evidence from the evidence graph. We consider an edge as irrelevant when its priority value is below a certain threshold.

6. Update an edge e: This represents changing weight, relevancy or host importance of evidence represented by the edge.

3 Hierarchical Reasoning Framework

Based on the evidence graph, we develop a hierarchical reasoning framework for automated evidence analysis. In this section we describe two levels of the framework: local reasoning and global reasoning.

3.1 Local Reasoning

The objective of local reasoning is to infer the states of a host from local observations. We argue that it is essential to keep track of host states for the following reasons:

1. Host states provide context for evaluating evidence. There is no absolute "suspicious value" of events. Actions of attackers are often represented by events that do not seem suspicious when examined individually without context. For example, legitimate file transfer connections associated with a healthy host generally do not indicate suspicious activity. However, an outbound ftp connection initiated from a victim host is likely to be a data exfiltration attempt that leads back to a host controlled by the attacker. Therefore, host states provides the context to discover hidden events for further investigation.

2. Host states derived with local observations provide an initial view of the roles the host may play in an attack. The evolution of host states helps to display the advancing stages of an attack to the forensics analyst.

The complexity of host systems and cyber attacks makes it difficult to reach a precise statement about host states. Consequently we believe that the fuzzy logic method could be an effective approach towards the problem. In the current prototype,we develop causal inference via Rule-Based Fuzzy Cognitive Maps(RBFCM) to model the states of nodes.

Fuzzy Cognitive Maps(FCM) are actually fuzzy directed graphs that combine neural networks and fuzzy logic to predict changes of the system. [3]. Nodes in a FCM are concepts that change over time and edges represent the causality link between nodes. Weight of the edge measures how much one concept impacts the other. FCM has been used for decision support in many different domains, including network security and intrusion detection systems [19]. However, FCMs are limited to model simple monotonic causal relations and cannot cooperate with traditional fuzzy rules [3]. We also observe that it requires significant training to get appropriate weight values.

Rule Based Fuzzy Cognitive Maps(RBFCM) are an evolution of FCM. A RBFCM is essentially a standard rule based fuzzy system plus feedback and mechanisms to deal with causal relations [4]. Compared with the generic FCM, RBFCM is better adapted for modelling complex dynamic systems because changes to the concept are not simply determined by the weight of the edges, but are defined by the fuzzy rules relating the concepts and inputs. This is an important advantage to incorporate non-symmetric and non-monotonic causal relationships.

As shown in figure 2, a RBFCM consists of fuzzy concepts and fuzzy rule bases. In our context, concepts are the defined states $\{Attacker, Victim, Stepping\ Stone, Affiliated\}$. Fuzzy rule bases consist of "$IF...Then...$" fuzzy rules that define how each concept is affected by values of other concepts and incoming new evidence.

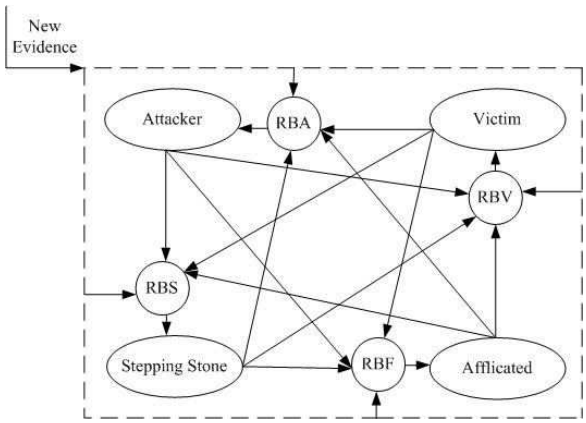

Figure 2. RBFCM model for local reasoning

In the RBFCM shown in figure 2, fuzzy rules are used to map multiple inputs (current value of states and new evidence) to the output (updated value of states). The states are updated in an incremental manner. State values at time $t+1$ are determined by the states at time t and new evidences observed during the time interval $[t, t+1)$.The fuzzy rules are designed from expert knowledge. For example:

If $BackOrifice$ is detected
Then $Victim$ state is activated highly.

If exploit of weight w initiates from $self$
Then $Attacker$ state value is increased by w.

If $Victim$ state is activated highly and
$Attacker$ state is activated highly and
$AT(T_{activate}) > VI(T_{activate})$
Then $Stepping\ Stone$ state is activated highly.

If ftp connection to host n is detected and
$Victim(n) = high$ and
$T_{ftp} - T_{activate}(VI) < T_{limit}$
Then $Affiliated$ state value is activated medium.

To monotonically map the concept value into a normalized range [0,1], we apply the sigmoid function in the following form where c is a positive constant.

$$f(x) = \frac{1}{1 + e^{-cx}} \tag{3}$$

3.2 Global Reasoning

The global reasoning process aims to identify a set of highly correlated hosts that belong to the attack of interest and derive their relationships by refining the local state estimates in the scenario context. Global reasoning is based on the assumption that during the procedure of attack, there must be a strong correlation between members of the attack group, and this correlation is exhibited through certain evidence observable in the network.

Based on the evidence graph, we approach the global reasoning task as a group detection problem, which is to discover potential members of an attack group given the intrusion evidence observed. The attack group detection procedure works in two different phases: (1) create new attack groups by generating seed for the group and (2) expand existing groups by discovering hidden members.

1. Seed Selection

   The first phase of group detection is to select certain node as seed of the attack group. Usually there are two approaches for seed generation.

   The first approach is to select seed based on states and context of nodes. This is basically an empirical procedure. To start the investigation, it is natural to select the victim host as the initial seed. Another example is that if a host in the trusted domain has Attacker state activated in local reasoning, it would be a good candidate as the initial seed.

   The second approach is to select seed based on graph metrics. We have been evaluating a set of measures to rank the list of nodes in the evidence graph for selecting important nodes that are highly suspicious as initial seeds. In our current prototype, we use the degree of a node degree(v) in the evidence graph as a simple metric. The measure of degree(v) represents the fanout of a host in the given time interval, which indicates the number of intrusion evidence associated with the host. A node with higher degree often has greater probability to be an important host that could be used as initial seed of the attack group.

2. Group Expansion

   In group expansion process, we expect to discover hidden members of the attack group that have strong correlation with the initial seeds.

Based on the evidence graph, we transform the problem space into a correlation graph $\hat{G} = (V, E)$. $\hat{G}$ has the same set of nodes with the evidence graph G while each undirected edge e represents the correlation between two neighbor nodes. Weight of the edge indicates the strength of correlation. For a pair of neighbor nodes (s, t) in $\hat{G}$, let $E(s, t)$ denote the set of incident edges between s and t in the evidence graph G, then correlation score $C(s, t)$ is computed as the sum of priority value of edges in $E(s, t)$.

$$C(s,t) = C(t,s) = \sum_{e \in E\{s,t\}} p(e) \tag{4}$$

For convenience, we define the reciprocal of correlation score as the distance between two neighbor nodes s and t. Smaller distance value represents stronger correlation between the two nodes.

$$d(s,t) = d(t,s) = \frac{1}{C(s,t)} \tag{5}$$

We are investigating different approaches to refine the distance evaluation process with logical and temporal constraints. For example, we can apply a time window so that repeated same class of exploits within the time window are only counted once when calculating the correlation score by (4). Another option is to make the priority score decay with time, i.e. $p(e)' = p(e) \times e^{\alpha t} (\alpha < 0)$ so that to give recent events higher weight in the correlation score. The decay factor α depends on the classification of evidence.

Group expansion is an iterative process. First, we identify all external neighbors of current seed members as the list of candidate nodes. Secondly, a ranked list is formed based on the distance between each candidate node to current group members; Finally, the ranked list is cut at a predefined threshold and nodes within the distance threshold are added as new seed members of the group. If no candidate node is within the distance threshold, the group expansion procedure terminates. The procedure is listed in algorithm 3.

The $FindNeighbors$ function returns neighbor nodes that are not current seed members. In the $GetDistance$ function, we can (1) evaluate the distance between the candidate node to its nearest seed member or (2) view the current seed group as a "super seed" and evaluate the distance as aggregation of distances to all seed members. The $RankCandidates$ function ranks the candidate nodes by distance value, candidate nodes whose distance exceed the predefined threshold are discarded.

input : Evidence graph G, initial seed node v_0 and distance threshold d
output: The group of nodes $group$
begin
$group \leftarrow v_0$;
$neighbors \leftarrow \emptyset$;
$candidates \leftarrow \emptyset$;
repeat
foreach *node* v *in the set* $group$ **do**
$neighbors \leftarrow$ `FindNeighbour` (G, v);
$candidates \leftarrow candidates \cup neighbors$;
end
foreach *node* v *in the set* $candidates$ **do**
$v.distance \leftarrow$ `GetDistance` $(v, group)$;
end
$new \leftarrow$`RankCandidates` $(candidates, d)$;
$group \leftarrow group \cup new$;
until *no new member is found*;
end

Algorithm 3: Attack group expansion

Due to the vast difference in attack traces, selection of the threshold is largely an empirical process: the analyst could compare the results of a set of thresholds and pick the most suitable one based on expertise. Lowering the threshold generally leads to higher rate of false positives while raising the threshold may result in higher rate of false negatives. Moreover, the ranking list often explains more than trying to find the best cut-off threshold.

4 Related Work

To our knowledge, little work has been done in automated network forensics analysis. Most mature forensics investigation tools like EnCase [11] and Safeback [17] focus on capture and analysis of evidence from storage media on a single host. ForNet [18] is a novel distributed logging mechanism that focuses on network forensics evidence collection rather than evidence analysis.

Our work extends current work in several areas into the network forensics analysis mechanism.

Intrusion Detection Systems: Intrusion detection techniques are generally classified into two categories: anomaly detection and misuse detection [9]. IDS' are an important source of evidence for forensics analysis. However we cannot solely rely on IDS' as they only catch known attacks or unusual behavior. Also, the high volume and low quality of IDS alerts makes it difficult for forensics investigators to identify a clear picture of the attack. We incorporate aggregated IDS alerts into our evidence graph model and evaluate their effects in reasoning process.

Attribution Techniques: Attack attribution techniques aim to locate the true origin of attack flows. IP spoofing and stepping stone connections are two common techniques attackers use to conceal their origin [8]. Therefore, attribution techniques generally fall into two classes: stepping stone detection [22, 23] and IP traceback [2, 20]. Attribution methods can be integrated into our reasoning process as evidence sources.

Alert Correlation: As intrusion alerts only reflect elementary steps in an attack, alert correlation methods aim at reconstructing the attack scenario by linking alerts that satisfy certain relationships together. Past work on alert correlation include attribute similarity based [6, 21], pre-defined scenario based [7, 10], pre/post condition based [5, 16] and methods that based on multiple information sources [15]. These methods are complementary to our approach. We extend a simple and flexible attribute-based alert aggregation mechanism derived from [21] in our evidence preprocessing module. Pre/post condition based and predefined scenario based methods can be leveraged in our local and global reasoning process.

5 Experiments

We evaluate our proposed techniques through several multi-stage attack scenarios. We use Snort as the network IDS sensor to generate intrusion alerts and use TcpDump to collect raw network traffic in the testbed. Evidence collected are stored into a MySQL database. We implemented a set of Perl scripts to aggregate intrusion alerts, extract flow information and automatically integrate prior knowledge in reasoning process. We develop an application based on LEDA [1] to manipulate evidence graphs and reasoning results.

5.1 Scenario Setup

We implemented a small scale multi-stage attack scenario is in our testbed. Roles of hosts in the attack group are shown in table 1. In addition to two stepping stones, the attacker also uses one third party host that has public ftp service as the relay for attack tools and exfiltrated data.

Table 1. Role configuration of hosts

Attacker	192.168.21.3
Stepping Stone 1	192.168.25.3
Stepping Stone 2	192.168.22.4
Victim	192.168.23.4
FTP Relay	192.168.24.4

To bring more variety to the scenario, multiple exploits, backdoor programs and scanning tools are used. In addition, we generate mixed background traffic including web

traffic, ftp, sftp, ssh and telnet throughout the whole process of attack to simulate a more realistic network environment. Random attacks not related to the primary attack scenario are also included in the background traffic to obscure the primary attack. The attack process lasts about 40 minutes and the raw network flow trace is about 300MB in size.

The attack scenario includes the following steps:

1. Samba remote buffer overflow attack against stepping stone 1 from attacker.
2. Download attack tools from ftp relay to stepping stone 1 and start an Netcat backdoor on stepping stone 1.
3. Windows DCOM remote buffer overflow attack against stepping stone 2 from stepping stone 1.
4. Download attack tools from ftp relay to stepping stone 2 and starts a backdoor on stepping stone 2.
5. Frontpage Server 2000 buffer overflow attack against the victim from stepping stone 2.
6. Download backdoor program from ftp relay to victim and starts backdoor on victim.
7. Transfer data from the victim to the ftp relay and close backdoor connections.

5.2 Evidence Preprocessing

Throughout the attack process, Snort reported a huge number of 7501 alerts. With our flexible alert aggregation procedure, the alerts information is highly condensed. We apply a time limit T of 60 seconds for the self-extending time window. The result is 4 hyper-alerts related to port scan activity and 17 hyper-alerts which represent other exploits. The aggregated hyper alerts are illustrated in figure 3.

5.3 Evidence Graph Construction

Based on the primary evidence intrusion alerts, our graph generation module generated the initial evidence graph shown in figure 3. The number attached with each edge denotes the sequence of corresponding evidences in time order. Note that there's no exploit activities involved with the ftp relay, therefore it does not show up in the initial evidence graph.

To get a clearer view, the analyst can apply filter conditions to remove irrelevant evidences. On the other hand, the analyst can specify queries for secondary evidence to provide a more comprehensive view of what's going on in the network. For example, based on figure 3, we can ignore all port scan attacks and find out all file transfer connections to hosts in the evidence graph during a specified time window. The updated evidence graph is showing in figure 4. We notice that several potentially suspicious hosts include the ftp relay are brought up in the updated evidence graph. Specific alerts in figure 3 are replaced by abstractions from the knowledge base to offer a high level picture in figure 4.

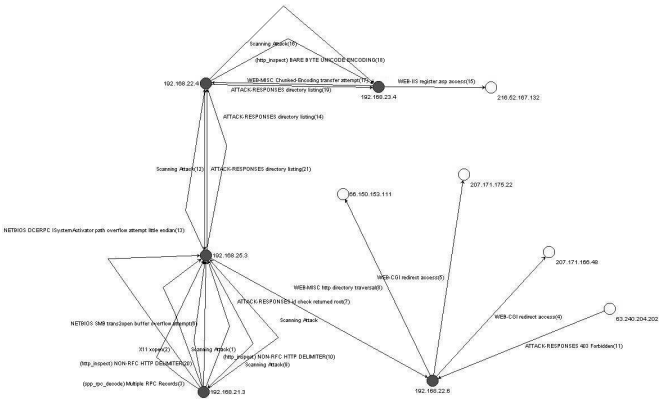

Figure 3. Evidence Graph from primary evidence

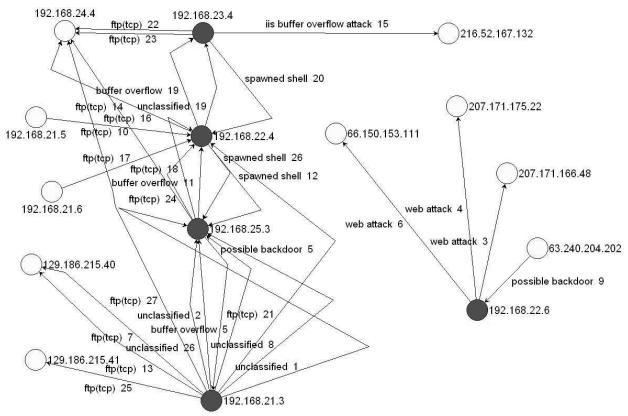

Figure 4. Enriched Evidence Graph with secondary evidence

5.4 Local Reasoning

In the first step of analysis, the analyst would examine the states of hosts from local reasoning process. Based on the RBFCM-based local reasoning procedure, states of nodes in the evidence graph are inferred and shown in table 2. The hosts that have "$Attacker$" state activated are highlighted in figure 3 and figure 4.

Table 2. Local Reasoning Results

Host	degree	AT	VI	SS	AF
192.168.22.4	12	0.85	0.85	0.87	0.84
192.168.25.3	12	0.85	0.80	0.94	0.84
192.168.21.3	11	0.80	0	0	0.84
192.168.23.4	6	0.69	0.85	0	0.82
192.168.24.4	5	0	0	0	0.84
192.168.22.6	4	0.85	0.50	0	0
129.186.215.40	2	0	0	0	0.81
129.186.215.41	1	0	0	0	0.69
192.168.21.5	1	0	0	0	0.70
192.168.21.6	1	0	0	0	0.70
207.171.166.48	1	0	0.67	0	0
207.171.175.22	1	0	0.67	0	0
66.150.153.111	1	0	0.67	0	0
216.52.167.132	1	0	0.69	0	0
63.240.204.202	1	0	0.71	0	0

Table 3. Distance between pair of nodes

Host 1	Host 2	Distance
192.168.25.3	192.168.22.4	0.23
192.168.21.3	192.168.25.3	0.24
192.168.23.4	192.168.22.4	0.43
192.168.23.4	192.168.24.4	0.59
192.168.21.3	129.186.215.40	0.67
63.240.204.202	192.168.22.6	1.11
192.168.21.3	192.168.22.4	1.18
192.168.21.6	192.168.22.4	1.19
192.168.22.4	192.168.24.4	1.19
192.168.21.5	192.168.22.4	1.19
192.168.21.3	192.168.24.4	1.25
192.168.23.4	216.52.167.132	1.25
192.168.21.3	129.186.215.41	1.27
192.168.25.3	192.168.24.4	1.27
192.168.22.6	207.171.175.22	1.43
192.168.22.6	207.171.166.48	1.43
192.168.22.6	66.150.153.111	1.43

5.5 Global Reasoning

First, we use degree of a node as the metric to generate the initial seed for attack group. From table 2 we can see that host 192.168.22.4 has the highest degree, thus we choose it as the initial seed for attack group.

In the next step, we expand the group by evaluating the distance between candidate nodes to seeds of the group. Distances between neighbor pairs are shown in table 3. Starting from the seed, the attack group is expanded incrementally. With distance threshold 1, the result attack group is shown in the correlation graph (figure 5). The highlighted oval node is the initial seed of the attack group and the highlighted square nodes are members added to the attack group during the group expansion process.

By observing the states of members in the attack group we can see that host 192.168.21.3 has only $Attacker$ state activated. Hosts 192.168.25.3,192.168.22.4 both have $Stepping\ Stone$ state activated. Host 192.168.23.4 has both $Attacker$ and $Victim$ state activated but not $Stepping\ Stone$. Further examination of the activation time of states clearly suggests that host 192.168.21.3 is the initial start point of attack and hosts 192.168.25.3,192.168.22.4 are used as stepping stones. Both 129.186.215.40 and 192.168.24.4 are affiliated with attacker and stepping stones, which suggest that more investigation is needed to find out whether they are truly related to the attack. Intuitively from the graph we can see that 192.168.24.4 seems more suspicious because it is affiliated with more members in the attack group than 129.186.215.40 does. Although 192.168.22.6 is labelled as an attacker in local reasoning process, it is unrelated to the main attack group. Thus we tend to regard it as a background attacker. The results show that with sufficient evidence, our automated analysis mechanism is effective in discovering attack group and high-level scenario of the attack.

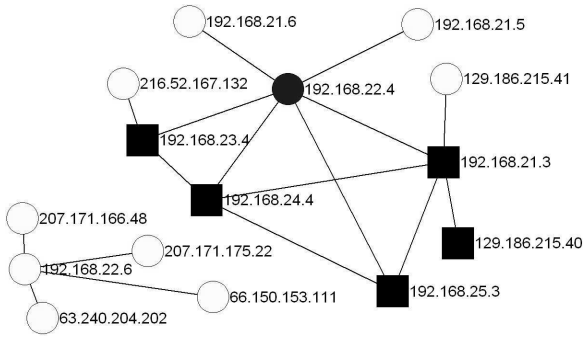

Figure 5. Correlation graph with attack group

We also evaluated our techniques with an external dataset generated by a larger scale multi-stage attack. Six hosts are true attackers that are responsible for primary attacks and 10 hosts are background attackers that initiate malicious activities not related to the primary attack scenario. Our analysis mechanism correctly identifies most of the attack group and steps. One attacker and two background attackers are absent from the evidence graph because exploits originated from the two attackers evaded detection. Therefore results of our analysis mechanism are limited by the completeness and accuracy of collected evidence.

6 Conclusions and Future Work

In this paper, we developed a network forensics analysis mechanism to reason about attack group and scenario.

We proposed the evidence graph as a novel graph model for presentation and manipulation of intrusion evidence. Based on the evidence graph, we further proposed a hierarchical reasoning framework for automated evidence analysis. By local reasoning with RBFCM we come to know the possible roles of suspicious hosts in a local view. In global reasoning, we identify the set of highly correlated hosts in an attack group and refine the roles in local reasoning results to fit into the scenario context. The combination of local and global reasoning results provides a high level picture of attacks represented by the observed intrusion evidence to the analyst. We developed a prototype tool and initial experimental results have demonstrated the potential use of our proposed methods.

This work is only the starting point of our efforts towards network forensics analysis. In future work, we will refine current approaches in local and global reasoning process. We will also investigate methods to automate the process for hypothesizing missing evidence and validating hypotheses. In the next step, we will work with industrial and government agencies to evaluate our techniques with more realistic experiments.

Acknowledgment

Thanks to DOI contract No. NBCHC030107 for initial funding of this research and laboratory as well as the external datasets. We would also like to thank the anonymous reviewers for their helpful feedback.

References

[1] LEDA graph library. Algorithmic Solutions Software, http://www.algorithmic-solutions.com/enleda.htm.

[2] S. M. Bellovin. Internet draft: ICMP traceback messages. Available at http://www.ietf.org/internet-drafts/draft-bellovin-itrace-00.txt, March 2000.

[3] J. P. Carvalho and J. A. B. Tome. Rule Based Fuzzy Cognitive Maps and Fuzzy Cognitive Maps - A Comparative Study. In *Proceedings of the 18th International Conference of the North American Fuzzy Information Processing Society(NAFIPS99)*, New York, 1999.

[4] J. P. Carvalho and J. A. B. Tome. Rule Based Fuzzy Cognitive Maps: Fuzzy Causal Relations. In *Proceedings of the 8th International Fuzzy Systems Association World Congress(IFSA99)*, Taiwan, 1999.

[5] F. Cuppens and A. Miege. Alert Correlation in a Cooperative Intrusion Detection Framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, May 2002.

[6] O. Dain and R. Cunningham. Building scenarios from a heterogeneous alert stream. In *Proceedings of the 2001 IEEE workshop on Information Assurance and Security*, pages 231–235, 2001.

[7] O. Dain and R. Cunningham. Fusing a heterogeneous alert stream into scenarios. In *Proceedings of the 2001 ACM workshop on Data Mining for Security Applications*, pages 231–235, 2001.

[8] T. E. Daniels. *Reference Models for the Concealment and Observation of Origin Identity in Store-and-Forward Networks*. PhD thesis, Purdue University, West Lafayette,Indiana, 2002.

[9] H. Debar, M. Dacer, and A. Wespi. A revised taxonomy for intrusion-detection systems. In *IBM Research Report*, 1999.

[10] H. Debar and A. Wespi. Aggregation and Correlation of Intrusion-Detection Alerts. In *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection(RAID)*, October 2001.

[11] EnCase Forensic Tool. Available at http://www.guidancesoftware.com.

[12] Intrusion Detection Message Exchange Format. Internet draft available at http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-14.txt.

[13] Institute for Security Technology Studies. Law enforcement tools and technologies for investigating cyber attacks: Gap analysis report. http://www.ists.dartmouth.edu, February 2004.

[14] C. Kruegel and W. Robertson. Alert Verification: Determing the success of intrusion attempts. In *Proceedings of the 1st Workshop on the Detection of Intrusions and Malware Vulnerability Assessment (DIMVA)*, Dortmund, Germany, July. 2004.

[15] B. Morin, L. Me, H. Debar, and M. Ducasse. M2D2: A Formal Data Model for IDS Alert Correlation. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection*, pages 115–137, 2002.

[16] P. Ning, Y. Cui, and D. S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *9th ACM Conference on Computer and Communicaitons Security*, November 2002.

[17] SafeBack Bit Stream Backup Software. Available at http://www.forensics-intl.com/safeback.html.

[18] K. Shanmugasundaram, N. Memon, A. Savant, and H. Bronnimann. ForNet: A Distributed Forensics Network. In *Proceedings of the Second International Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security*, St. Petersburg, Russia, 2003.

[19] A. Siraj, S. M.Bridges, and R. B.Vaughn. Fuzzy cognitive maps for decision support in an intelligent intrusion detection system. Technical report, Department of Computer Science, Mississippi State University, 2001.

[20] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, B. Schwartz, S. T. Kent, and W. T. Strayer. Single-packet ip traceback. *IEEE/ACM Trans. Netw.*, 10(6):721–734, 2002.

[21] A. Valdes and K. Skinner. Probablistic alert correlation. In *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection(RAID)*, October 2001.

[22] K. Yoda and H. Etoh. Finding a connection chain for tracing intruders. In *Proceedings of the 6th European Symposium on Research in Computer Security (ESORICS 2000)*, Toulouse, France, Oct. 2000.

[23] Y. Zhang and V. Paxson. Detecting stepping stones. In *Proceedings of the 9th USENIX Security Symposium*, pages 171–184, Denver, USA, Aug. 2000.