Attack Pattern Discovery in Forensic Investigation of Network Attacks

Ying Zhu, *Member, IEEE*

Abstract

Network attacks on systems perpetrated by remote hackers rarely occur in isolation; when a successful or merely detected attack occurs, it is often desirable to reconstruct the *context* of this security breach: all the events that lead up to and are related to the breach. We mine the logs of recent network traffic data to find these contexts of attacks — we call them *attack patterns*. We propose an iterative algorithm for discovering attack patterns; the logs are scanned to identify coherent groups of events (called *bubbles*) that are likely to constitute attacks, and via a feedback mechanism, the degrees of belief that the bubbles are attack instances are propagated to the next iteration in order to refine the search for bubbles related to the ones already found. Our simulations verify that the algorithm achieves accuracy in discovering attack patterns. Our attack pattern discovery has the additional advantage of being an unsupervised algorithm, e.g., it does not require *a priori* user-defined thresholds.

Index Terms

network forensics, security, attack patterns, suspicion feedback

I. Introduction and Motivation

The scenario we consider in this paper is the investigation that follows the occurrence of disruptive or suspected network attacks on one or more connected systems. The occurrence could be attacks succeeding in crashing or disrupting the systems, or network activities that are highly likely to be attempts at attacking. It is well-known that network attacks typically do not occur in isolation [1]: the activities that cause damage or detection are not stand-alone, because these attacks are impossible to carry out without some detailed required information of the target systems; therefore, they are always, by necessity, preceded by a few stages of exploratory probings by the attackers in order to obtain as much information as possible on the target system and thereby find vulnerabilities. Consider the following simple scenario. A potential attacker A is trying to target an organization with a block of IP addresses. First, A will gather sufficient information to find vulnerabilities, by doing a ping sweep of the organization's network followed by port scans. A ping sweep of the IP address block is simply probing each IP address to see which systems are alive. Then a port scan can be carried out on each live system, which involves connecting to a large range

Ying Zhu is with the Faculty of Engineering and Applied Science (e-mail: ying.zhu@uoit.ca), University of Ontario Institute of Technology, Oshawa, Ontario, Canada, L1H7K4.

of port numbers to find TCP and UDP ports that are listening and the corresponding services they are used for. Suppose A is able to determine that one of the IP addresses X runs a flavor of UNIX and has a ToolTalker Database (TTDB) server owned by `root` listening at port 32775, which is a remote procedure call (RPC) service that has a known vulnerability. Now A can simply execute a buffer overflow attack on port 32775 to get this program to execute an xterm and display it back on A's screen, which means A has gained root access to the system X. The awareness of the attack usually occurs when A is doing something damaging as the root, and this is the occurrence that gets investigated afterwards.

Investigation into these visible breaches of security (occurrences of malicious attacks or attempts) is essential, so that there is more information learned than simply that the systems crashed or the network went down. For instance, in the above imaginary scenario, X could be used by A as an entry point into the entire organization's network, and after the unknown compromise of X, A proceeded to bring down the entire network. If no investigation is conducted afterwards, it would not have been known that X was the vulnerable point. It is critical to gather this sort of evidence. Since network attacks rarely occur in isolation, the sequence of network events leading up to the final breach as well as any others that cooperate with or contribute to these attack activities should be discovered. The information obtained is useful beyond the current investigation — it can be archived and used to prevent similar future intrusions, and it can also be used to identify previously unknown security weaknesses in the system. It is becoming increasingly important to the IT industry and organizations at large to preserve and use network traffic as a form of digital evidence [2], [3], [4]. Network forensic analysis provides valuable information that form the foundation of accurate conclusions and good decisions relating to an incident. The algorithm we proposed in this paper to discover attack patterns can be used in gathering such information.

A final clarification concerns the broader context for our attack pattern discovery algorithm. Our focus is on the digital *forensic* task of discovering attack patterns, rather than a real-time monitoring system that analyzes live network traffic data and triggers alerts, examples of which are found in works such as [5], [6], [7], [8]. In real-time monitoring systems, efficiency is of crucial importance — packets must be processed at least as fast as they are arriving; any decrease in efficiency results in the highly undesirable scenario of packets being dropped. In contrast, we assume that the network traffic data have already arrived at a previous point in time and have been saved in static log files that are available for us to examine. Our algorithm is iterative; the logged network events are scanned more than once in the algorithm. Efficiency, though important, was not our top priority when we designed the algorithm, since one of the underlying assumptions was that the data is historical and static, not growing dynamically at real-time. Our concern was mainly in the accuracy of discovering attack patterns. Therefore, it is not conclusive whether our algorithm would be suitable for real-time analysis of network data. There may be potential to use it for such purposes, but that is outside the scope of this paper.

II. Related Work

To the best of our knowledge, there are no previous work specifically addressing the attack pattern discovery problem in this paper. There are a number of open-source and commercial tools available that can aid in network forensics investigations, e.g., [9], [10], [11], [12], [13]. The functionalities and capabilities of these tools encompass: packet capturing, packet logging, reconstruction of files sent over the network, user-defined threshold-based detection of anomalous network activities, signature-based intrusion detection. The packet capturing and logging functionalities could be used to generate the logs that are scanned in our algorithm. The anomaly and intrusion detection methods utilize user-defined thresholds which are not sufficiently flexible for our purposes, as we will elaborate in the later sections. They also have the objective of detecting a single attack at a time and triggering an alert, which is different from our problem of mining the logs to find all the network activities that are correlated to an attack that was already detected.

A generalized network monitoring tool was proposed in [14] that provides network traffic analysis and builds statistical event records. It has the attractive feature of making the filtering and analysis process internally programmed rather than statically coded into the monitoring application. This is a generalized tool that includes a programming language for filtering, to be used by system administrators to choose the types of traffic to be recorded.

Novel payload attribution methods were proposed in [15], [16], [17] to obtain a reduced size digest of packets seen in the past which supports the capability of answering queries about whether any packets seen so far contained certain payload excerpts. These work present an excellent viable alternative to logging raw network traffic. They allow the log size to be considerably reduced (the data reduction ratio is on the order of $100:1$ in [17]), while maintaining high accuracy in answering queries on the payload information from packet history. Payload attribution methods are complementary to our algorithm; the smaller-sized packet digests could be logged instead of the raw packets.

Recently, as a reaction to the prevalence of security attacks on the Internet, there have been initiatives to set up information system traps on the Internet to attract, solicit and monitor malicious attacks — these traps are called “honeypots”. The honeypot data can then be analyzed and mined for valuable information on the patterns behind these attacks, which help in developing defenses against them. A number of honeypots have been successful in collecting data from malicious Internet activities, e.g., [18], [19], [20], [21], [22], [23]. They have also made efforts to analyze the honeypot data to mine for useful information. Several studies have been conducted to apply automatic knowledge discovery techniques to honeypot data with the objective of gaining deeper insight. A clustering algorithm is proposed in [24] to identify attack tools behind the observed attacks; each cluster gathers the IP addresses of all the sources that use the same attack tool. The work in [25] focuses on the identification of inter-relationships between these clusters to obtain additional characteristics of the attack tools used, e.g., association of some attack tools with certain geographical locations. Another clustering technique is presented in [26] that analyzes

the time series of the network traces and finds groups of traces that are similar in time signatures, thereby identifying various worms and botnets. The authors in [27] proposed a method that integrates clustering structure visualization together with outlier detection, in order to provide a big picture of honeypot data patterns and to detect new botnets as they are deployed. All these works have a different focus than our study: Their objective is to group or cluster together the IP sources that use the same attack tool or belong to the same botnet. Our problem is to find *chains* or sequences of network events that lead up to some final security breach. Even though the term "attack pattern" is used in [26] as well, it has a different meaning; in the context of [26], an attack pattern refers to an attack time signature of a specific attack such as a worm or a type of botnet attack. In contrast, we define an attack pattern to be a sequence of attacks that could be of various types but correlated to the security breach.

Another closely related work that is very interesting is on Internet situational awareness by Yegneswaran et al. [28]. Their focus is on integrating honeypot data into daily network security monitoring, in order to accurately classify the Internet traffic and provide real-time situational awareness. This is distinct from our purpose of *forensic* analysis of network logs.

There is a body of work on network traffic classification. For instance, algorithms are described in [29] to construct traffic clusters of conspicuous consumption; this method of traffic characterization can be used to classify new traffic patterns such as worms. In [30], the authors present a method to classify network flows by identifying behavior patterns of hosts at three levels: the social, the functional and the application level, doing so all at the transport layer, with no knowledge of the source applications of the flows. Kannan et al. [31] propose algorithms based on Poisson processes that mine network connections to identify groups of related connections, each group corresponding to a distinct application-layer user-initiated session. Our work is distinct from these works in two folds: (1) We only do forensic mining of network logs, after a security breach has already occurred. (2) We are grouping or classifying network events, rather than network flows, into related groups or patterns, and the grouping is done over time and based on a distinct malicious attacker intent instead of grouping across flows occuring at the same time based on a distinct application or session.

III. Problem Setup

Our approach to the network attack pattern discovery problem is sketched in this section, then presented in further detail in the later sections. We also set up the problem here and discuss the assumptions we make.

Suppose that there are one or more attackers, at one or more source IP addresses, targeting a network of multiple systems (multiple IP's) or a single system (one IP). It is assumed that on any target system, a packet sniffer tool captures all the raw network packets, and the TCP/UDP headers and other relevant information from the data payloads are recorded in a log file stored locally. Every system stores one big log file that stores all this network traffic data, which can be viewed as a data table; each line or

entry we call an event, containing fields that hold the captured packet header and occasionally selected payload information. This can be easily done using a packet sniffer and some pre-processing of the data it captures. The log is a rolling one over time; when a new event is to be logged and maximum log size has been reached, the oldest event is removed to make room for the new one.

When an attack on the target system(s) is successful enough to crash, disrupt or generally catch attention, we will refer to this event as the *breach* that triggers the investigation. Upon the occurrence of an breach, our goal is to mine the logs of the target systems to discover and interconnect all the events that have been network attacks and activities that lead up to and are correlated in some way to the final breach, based on the hindsight knowledge of the breach. We call this group of interconnected attack events an *attack pattern*.

A. Problem definition

An *attack definition* is a formal description of an attack signature. Section IV-A provides the details of attack definitions. We assume that a list of attack definitions is provided.

In this paper, we further characterize attacks by their co-occurrence relations using an *attack graph*.

Definition 1 (Attack graph): An attack graph is an undirected graph $G = (V, E)$, where the nodes V are the types of attacks, and $\{a, a'\} \in E$ if the attacks a and a' are likely to occur in conjunction. An edge $\{a, a'\}$ may be assigned a weight $0 \leq p_{\text{co-occurrence}}(a, a') \leq 1$ representing the probability of one attack occurring given the other has occurred.

A bubble is a collection of network events that is believed to be an attack.

Definition 2 (Bubbles): A bubble is completely characterized as a tuple $x = (\text{ID}, a, E, susp)$ where ID is the identifier of the source of the events, E is the set of network events, a is the suspected type of network attack, and $susp \in [0, 1]$ is the likelihood that the events in E constitute an attack of the type a.

At this point, it is sufficient to say that the suspicion score is a numeric measure of the belief that the events in E form an attack instance of type a. The initial suspicion score $susp$ of a bubble x is a function of the events E and the attack type a — the details of its computation are deferred to Section IV-B. Subsequently, the suspicion score is adjusted in every iteration of our algorithm from the suspicion feedback, as explained in detail in Section V-D.

Given a bubble x, we denote its ID as $x.\text{ID}$, its attack type as $x.a$, its events as $x.E$ and the suspicion score as $x.susp$.

The identifier $x.\text{ID}$ corresponds to the source of the attack, so it is not unique to a specific bubble. Different bubbles $x_1 \neq x_2$ can have the same sources, i.e. $x_1.\text{ID} = x_2.\text{ID}$.

Our problem can be stated as: Given a list of attack definitions, an attack graph, and a log file of network events, we wish to:

- extract bubbles which have high degrees of suspicion of being attacks, and
- group suspicious bubbles that are interconnected into an attack pattern.

This paper presents an algorithm for discovery of attack patterns by iteratively solving the above sub-problems. The extracted bubbles generate new connections which, via a feedback mechanism, are used to facilitate further refinement in the suspicious bubble extraction.

IV. Attacks and Suspicion Scores

In this section, we begin the description of our algorithm with the formulation of attack definitions (Sec. IV-A) and computation of the initial suspicion scores of events (Sec. IV-B). The details of the main algorithm will be given in Sec. V.

A. Attack definition

We first discuss how we formulate attack signatures in a unified way into *attack definitions*. An *attack signature* refers to a special sequence of events that can be distinguished from normal network traffic and signalize likely instances of specific attacks. For example, a port scan involves the attacker trying to connect to a large number of destination ports including those that are not listening, so an appropriate attack signature for port scans is: ”Packets are sent from the same source IP to a large number of distinct destination ports within a short period of time.”

To discover attack patterns, the events recorded in the log should be compared with the attack signatures, and if there is a match between some events and a certain attack signature, then these events are believed to be an instance of this attack. However, the exact way to implement a given attack signature is not immediately obvious. For instance, the abovementioned signature for port scans can be implemented by absolute thresholds on number of ports and time interval, which is a common practice in intrusion detection systems (IDS).

However, our objective is distinct from that of an IDS; we do not view network events as mutually independent. With the starting point of the security breach, we are mining for events that are suspicious *and* correlated with the breach. This means that we must be *biased* somehow towards related events even though they would have been deemed not suspicious by the absolute thresholds method. Another strike against thresholds is they yield an absolute boolean result that would potentially miss some events which only appear innocuous in isolation but are suspicious in the context of other events already linked to the breach.

The above reasons motivated us to express attack signatures in a formulation we call *attack definition* that has the properties: (1) By matching events to an attack definition, a continuous score is returned that represents the likelihood of being an attack instance. (2) It is a unified formulation that can be used for a wide variety of attack signatures.

Our attack definition formulation uses a syntax modeled loosely after SQL queries; it consists of a bunch of key-value pairs:

- The `name` key maps to the name of the type of attack defined, e.g., “port scan”.

port scan	Packets to a large number of distinct ports in a short time period from the same source IP.
	`name` : *port scan* `identified-by` : *source_ip* `property` : [`field`=*port*, `agg`=*count distinct*, `obj`=*max*], [`field`=*timestamp*, `agg`=*range*, `obj`=*min*]
password	Many failed authentication attempts in a short time from same source IP.
	`name` : *password* `identified-by` : *source_ip* `where` : *error* : [*"contains", "failed authentication"*] `property` : [`field`=*username*, `agg`=*count*, `obj`=*max*], [`field`=*timestamp*,`agg`=*range*,`obj`=*min*]
rpc.ttdbserverd	Data sent to port 32775 contains "`xterm`".
	`name` : *rpc.ttdbserverd* `identified-by` : *source_ip* `where` : *port* : [*"=", "32775"*] AND *data* : [*"contains", "xterm"*]

Fig. 1. Examples of attack definitions

- The `identified-by` key is equivalent to the "group-by" in SQL, mapping to a list of one or more field from the log entries and events with the same values in those fields are grouped together, denoted by ID.
- The `where` key is equivalent to the "where" clause in SQL, mapping to a condition on the values of specified fields that an event must satisfy to be suspected as part of this attack.
- The `property` key maps to one or more properties, each consisting of a field i, an aggregation function α_i on that field, and an associated objective function ω. The aggregation function α_i is applied to the values of the field i in the events identified by (or grouped by) ID and returns a single aggregate value. The objective function ω is evaluated on the aggregation result; it is used in the process of demarcating candidate bubbles, and in the calculation of the suspicion score.

For example, we give the attack definition for the attack signature of port scans in Fig. 1. Events are "identified-by" or grouped by their source IP because in a port scan all the packets are sent from the same IP. In the "property" there are two fields to be aggregated: count the number of distinct ports and find the range of the timestamps. The number of ports is to be maximized while the time range is to be minimized, when searching for bubbles suspected to be port scans.

B. Computing initial suspicion scores from attack definitions

Given a sequence of $E = e_{i_1}^{n}$ of events, and given an attack definition a, it is easy to partition E into groups each of which has the same ID and select only those events that satisfy the `where` condition. So without loss of generality, we assume for now that all events in E have the same ID (say, same source IP) and satisfy the `where` condition. We can compute the likelihood of E constituting an instance of attack type a using its aggregation and objective function; this is the initial *suspicion score* for E, $susp(E, a)$. To do this, we express the objective function ω as a probability distribution. The specific probability distribution chosen depends on the types of objective functions and the fields involved.

- When the objective function is to `maximize` the count of some field in the network events, we use a Gaussian distribution to express it.

  The rationale is as follows: The timespan of a set of events is divided into a sequence of small subintervals, such that in each subinterval at most one event occurs which either does not increase or increases by one the count of the field of interest. This means that each subinterval can be viewed as a Bernoulli trial (i.e., a success is when the count increases by one), hence the sequence of subintervals corresponds to a sequence of Bernoulli random variables. The count we are maximizing is the sum of this sequence of random variables. And the well-known Central Limit Theorem [32] states that the probability distribution of sums of random variables can be approximated by a Gaussian distribution.
- When the objective function is to `minimize` the timespan of events, we use a geometric distribution.

  The rationale is as follows: Our underlying purpose in formulating the objective of minimizing timespan is to find a coherent set of events that constitute a single attack; if the timespan of some events is too large, then they likely do not belong to the same attack. Let t^0 denote the time of the first event e^0. Starting at t^0, divide time into small subintervals such that in each subinterval, at most one event is logged. Let e' denote the last event that belongs to the set of events constituting the same attack beginning with e^0. Each subinterval can be viewed as an independent Bernoulli trial: it is a success if the event e' is logged, a failure otherwise (either no event or an event that's not e' is logged). Finding the timespan between e^0 and e' is then equivalent to counting the number of failures before a success occurs in a sequence of independent Bernoulli trials, which corresponds to the geometric random variable [32].

The parameters that define a Gaussian distribution are its mean and variance, they can simply be estimated from training data of typical events logged for the specific attack. The geometric distribution has one defining parameter that determines both its mean and variance, which can similarly be estimated from training data. Note that the geometric is a discrete distribution and the aggregation function may return continuous values, but in all the cases we have encountered in the context of this problem, the continuous values can be easily discretized.

It helps to use an illustrating example to support our reasoning so far. In the attack definition for port scans(see Fig. 1), there are two objectives: to maximize the number of ports and to minimize the timespan. We use a Gaussian distribution for the first objective and a geometric distribution for the second objective. The rigorous rationale for doing so is given above. Intuition also agrees: it is easy to see that a small number of ports should not be given a high score; while as the number of ports becomes too large, the likelihood also increases that these events are not occurring in a short time period, and therefore should be penalized by a decreasing score — a Gaussian distribution fits here. Its parameters (mean and variance) can simply be estimated using some training data of typical log events from port scans. For minimizing the range of event timestamps, we should give a great deal of preference to small timespans and exponentially penalize increasing timespans after a certain point. Again, intuition agrees with the

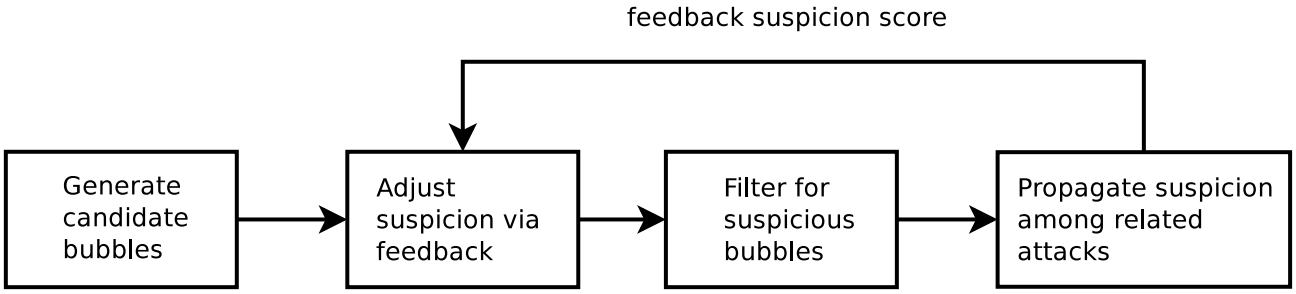

Fig. 2. Overview of the algorithm

rigorous reason explained above that a geometric distribution fits this objective, provided the timespans are first discretized. To discretize, we simply take the timespan value of the first two consecutive events in E as one unit of time, $u = \textit{timestamp}(e_2) - \textit{timestamp}(e_1)$, and any given timespan value t can be discretized to $\lceil t/u \rceil$.

Now given an attack definition a, we have a probability distribution expressing the objective function in its property, let it also be denoted $\omega(.)$, and it takes as its input a return value of the aggregation function α_f. We compute the suspicion score for a sequence of events E (recall that E is assumed to have the same ID and satisfy the `where` condition of a) as $susp(E, a) = \omega(\alpha_f(E))$; and since it is a probability, its value is in $[0, 1]$. If a has more than one property, i.e., multiple α_i and ω_i, the suspicion is simply the product of the probabilities from all the properties (which remains a probability, as desired),

$$susp(E, a) = \prod_i \omega_i(\alpha_i(E)). \tag{1}$$

For those attack definitions that have no property and hence no aggregation nor objective functions, any events that match the `where` conditions would receive a suspicion score of 1.

V. Iterative Attack Pattern Discovery

We now present the details of the main algorithm. The aim of our algorithm is to discover bubbles with high suspicion values. Recall that an *attack bubble* is defined as a collection of network events originating from a common source that represents a likely attack. Once some bubbles of high suspicion have been identified, an iterative feedback mechanism allows the algorithm to iteratively search for more hidden but related attack bubbles, thus allowing the reconstruction of the complete attack sequence in its entirety. The algorithm is depicted in Figure 2.

The overall iterative attack pattern discovery algorithm is shown in Algorithm 1. ITERATIVE-ATTACK-PATTERN-DISCOVERY maintains a dynamic table $susp\_feedback$ of feedback suspicion scores between 0 and 1 for each combination of ID, and attack type, a. It utilizes GENERATE-CANDIDATE-BUBBLES (to be presented in Section V-A) over all types of known attacks to compile a collection of candidate bubbles. Each bubble represents a potential attack, with the suspicion score being the degree of belief. So, a low suspicion score $x.susp \approx 0$ implies that the bubble x is probably not an attack, whereas a high suspicion

Algorithm 1 ITERATIVE-ATTACK-PATTERN-DISCOVERY

Require: log file
  $\mathbf{Attacks}$ is the list of attack definitions
  $\mathbf{G}$ is the attack graph
  $susp\_feedback \leftarrow \vec{0}$
  $C \leftarrow \bigcup_{a \in \mathbf{Attacks}}$ GENERATE-CANDIDATE-BUBBLES(a)
  $B \leftarrow \emptyset$
  while B changes **do**
    ADJUST-SUSPICION$(C, susp_feedback)$
    $B \leftarrow B \cup$ FILTER-SUSPICION-BUBBLES$(C, susp_feedback)$
    $susp\_feedback \leftarrow$ PROPAGATE-SUSPICION(B, G)
  end while

Algorithm 2 GENERATE-CANDIDATE-BUBBLES(a)

Require: log file
  a is a single attack definition.
  score $\leftarrow$ empty table
  duration $\leftarrow$ empty table
  for $t = 0, \Delta t, 2\Delta t, 3\Delta t, \ldots$ **do**
    for ID seen in seen in log **do**
      score$[\mathrm{ID}, a, t] \leftarrow$
        $\max_{\tau > 0} susp(E(\mathrm{ID}, t \to t + \tau), a)$
      duration$[\mathrm{ID}, a, t] \leftarrow$
        $\mathrm{argmax}_{\tau > 0} susp(E(\mathrm{ID}, t \to t + \tau), a)$
      if score$[\mathrm{ID}, a, t] <$ score$[\mathrm{ID}, a, t - \Delta t]$ **then**
        $E_C = E(\mathrm{ID}, t - \Delta t + \mathrm{duration}[\mathrm{ID}, a, t])$
        yield bubble $C = (E_C, \mathrm{ID}, t - \Delta t, a, \mathrm{score}[\mathrm{ID}, a, t])$
      end if
    end for
  end for

score $x.susp \approx 1$ implies that the bubble is strongly believed to be an attack. The events associated with a bubble is the evidence that supports the belief of attack.

The scores in the table $susp\_feedback$ is the degree of belief that an attack a co-occurred with other attacks which have already been discovered in the previous iteration. The feedback suspicion scores are initialized to zeros, but will be computed based on discovered attacks and an attack graph G; this is done by the function PROPAGATE-SUSPICION (Section V-C. The function ADJUST-SUSPICION (Section V-D) updates the suspicion scores of the candidate bubbles based on the suspicion feedback. Only the bubbles with relatively high suspicion scores will be classified as attack bubbles. The function FILTER-SUSPICION-BUBBLES performs the classification without any supervision (Section V-B). The filtered bubbles will be used to determine the feedback suspicion scores for the next iteration.

The final set of attack bubbles is obtained when the iteration converges.

A. Generating candidate bubbles

Every candidate bubble consists of one or more events that are usually close in time, and can be thought of as events that form a coherent unit that corresponds to some single activity. For example, an event that logs a remote procedure call (RPC) to an RPC service running on the target system is a candidate bubble, or several events that record a series of remote login attempts from the same source IP constitutes a candidate

Algorithm 3 FILTER-SUSPICIOUS-BUBBLES(C)

(B_1, B_2) = 2-Means clustering of $\{(x.susp, x) : x \in C\}$
return B_i where B_i has a higher mean

bubble. Because the log is comprised of network packet-level entries, with each entry corresponding to a single TCP or UDP packet, this initial step of generating candidate bubbles is necessary for delineating events related to the same unit of activity from the raw log. A candidate bubble defines some coherent set of events that could potentially match an attack definition, even if the probability is remote.

Definition 3 (Window of attack): Let $E(\mathrm{ID}, t_1 \to t_2)$ be all the events in the log with the identifier ID with timestamp between t_1 to t_2. Define a scores $\mathrm{score}(\mathrm{ID}, a, t)$ and $\mathrm{duration}(\mathrm{ID}, a, t)$ as follows.

$$\begin{aligned} \mathrm{score}(\mathrm{ID}, a, t) &= \max_{\tau > 0} susp(E(\mathrm{ID}, t \to t + \tau), a) \\ \mathrm{duration}(\mathrm{ID}, a, t) &= \mathrm{argmax}_{\tau > 0}\, susp(E(\mathrm{ID}, t \to t + \tau), a) \end{aligned}$$

The measure $\mathrm{score}(\mathrm{ID}, a, t)$ is the likelihood that an attack of type a started at time t, with identifier ID. The most likely attack duration is given by $\mathrm{duration}(\mathrm{ID}, a, t)$. As the objective functions of the property timespan, $\lim_{T \to \infty} \omega_{\mathrm{timespan}}(T) = 0$, and that $susp(E, a) = \prod_i \omega_i(\alpha_i(E))$, it is guaranteed that

$$\lim_{\tau \to \infty} \mathrm{score}(\mathrm{ID}, a, t) = 0$$

Hence, the duration $\mathrm{duration}(\mathrm{ID}, a, t)$ must remain finite.

The candidate bubbles are simply the local maxima of $\mathrm{score}(\mathrm{ID}, a, t)$ over time for all possible combinations of (ID, a). Given a local maximum of $\mathrm{score}(\mathrm{ID}, a, t^*)$, the candidate bubble is given by

$$x = (\mathrm{ID}, a, E(\mathrm{ID}, t^* \to t^* + \mathrm{duration}(\mathrm{ID}, a, t^*)), \mathrm{score}(\mathrm{ID}, a, t^*))$$

In Algorithm 2, we look for local maxima by discretizing the time dimension with time step Δt.

B. Filtering suspicious bubbles

The set of candidate bubbles delineate units of coherent network activities that have an extremely wide range of suspicion, even those that are only tenuously probable to be attack activities are included. Every candidate bubble also has an associated suspicion score, which is updated at each iteration of the attack pattern discovery algorithm and represents the current suspicion that this candidate bubble is part of the attack pattern. The procedure for updating the current suspicion score of the candidates is discussed in detail later. At each iteration, we classify the candidate bubbles according to their suspicion scores into two classes, filtering the suspicious bubbles from rest.

To do so, we apply the 2-means clustering algorithm (k-means clustering with $k = 2$) to the set of candidate bubbles and return the top cluster as the bubbles, as shown in Algorithm 3.

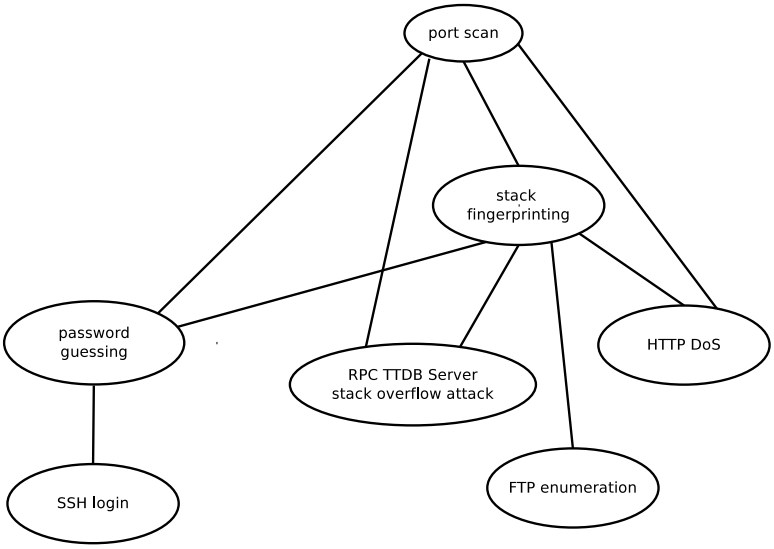

Fig. 3. An example attack graph.

C. Propagating suspicion

Given a set of bubbles, B, we wish to use the attack graph to infer other likely types of attacks that may not have been detected in B (A simple example of an attack graph is given in Figure 3). This information will be the feedback to the next iteration of bubble generation. The suspicion is propagated from the set of bubbles B to a subset of the nodes of the attack graph G, and then further propagated throughout to the rest of G. The propagation is done as follows.

Definition 4 (Suspicion propagation): Given a set of bubbles B and an attack graph $G = (V, E)$, define $V_0 = \{x.\text{attack} : x \in B\}$. Define the propagated suspicion on the nodes of G as:

$$s_0(a) = \begin{cases} \max\{x.susp : x \in B \text{ and } x.\text{attack} = a\} & \text{if } a \in V_0 \\ 0 & \text{otherwise.} \end{cases}$$

$$s_i(a) = \begin{cases} s_0(a) & \text{if } a \in V_0 \\ \max\{s_{i-1}(a')\} \cdot p_{\text{co-occurrence}}(a, a') : a' \in \text{NEIGHBOURS}_G(a)\} & \text{otherwise.} \end{cases}$$

where $\text{NEIGHBOURS}_G(a)$ is the adjacent nodes of a in the graph G. The propagated suspicion of a node $a \in V$ is defined as the fixed point of $s_i(a)$.

The fixed point is guaranteed to exist.

Proposition 1: If for all edges of G, $\{a, a'\}$, $0 \leq p_{\text{co-occurrence}}(a, a') < 1$, then there always exists $k > 0$ such that for all $a \in V$, $s_k(a) = s_{k-1}(a)$.

The function PROPAGATE-SUSPICION, shown in Algorithm 4, performs the propagation as defined in Definition 4 for each ID.

D. Adjusting suspicion scores

After the first iteration, we obtain a set of bubbles which are essentially evidence we have found so far. We reason that this evidence should be used for further refining the search to discover correlated attack

Algorithm 4 PROPAGATE-SUSPICION(B, G)

Require: B is the set of bubbles
Require: G is the attack graph
  $susp\_feedback$ = empty table
  for ID in distinct ID's in B **do**
    $susp\_init$ = empty table
    $susp\_propagate$ = empty table
    $B(\mathrm{ID}) \leftarrow \{x \in B : \mathrm{ID}[x] = \mathrm{ID}\}$
    for $a \in \mathrm{Nodes}(G)$ **do**
      $B(\mathrm{ID}, a) = \{x \in B(\mathrm{ID}) : \mathrm{Attack}[x] = a\}$
      if $B = \emptyset$ **then**
        $susp\_init[a] = 0$
      else
        $susp\_init[a] = \max_{x \in B(\mathrm{ID},a)} x.susp$
      end if
    end for
    while True **do**
      for $a \in \mathrm{Nodes}(G)$ **do**
        if $susp\_init[a] = 0$ **then**
          $N(a) =$ neighbours of a in G
          $susp\_propagate[a] = \max_{x \in N(a)} susp\_propagate[x] \cdot p_{\mathrm{co-occurrence}}(x, a)$
        else
          $susp\_propagate[a] = susp\_init[a]$
        end if
      end for
      break **if** values in $susp\_propagate$ did not change.
    end while
    $susp\_feedback[\mathrm{ID}] = susp\_propagate$
  end for

activities in the next iteration. This is repeated in each iteration until no more new bubbles are generated, i.e., no more new evidence is found. The required task is to somehow propagate the suspicion scores of the bubbles from the previous iteration to the next iteration. We accomplish this by updating the suspicion scores of the set of candidates.

The existing suspicion score of a candidate bubble should be updated by combining it with a *feedback suspicion*, $susp\_feedback$. The feedback suspicion is propagated from the suspicion scores of bubbles generated in the previous iteration. It is a mechanism for incorporating bias derived from the evidence (bubbles) already found into the next round of bubble search, in order to find bubbles correlated to evidence already discovered. The mathematical function for combining existing suspicion score and suspicion feedback to obtain the updated suspicion must have the properties: (1) the result is again a value between 0 and 1; (2) when feedback is low, it should not affect the suspicion score; (3) when feedback is high, the suspicion score should be boosted by the feedback. The following function satisfies all three properties:

$$\mathbf{G}_{\mathrm{feedback}}(x_{\mathrm{feedback}}, x_{\mathrm{susp}}) = \frac{1 - \exp(-c \cdot x_{\mathrm{feedback}} \cdot x_{\mathrm{susp}})}{1 - \exp(-c \cdot x_{\mathrm{feedback}})},$$

where c is a constant; we set c to 10 in our implementation. The function $\mathbf{G}_{\mathrm{feedback}}$ is shown in Figure 4

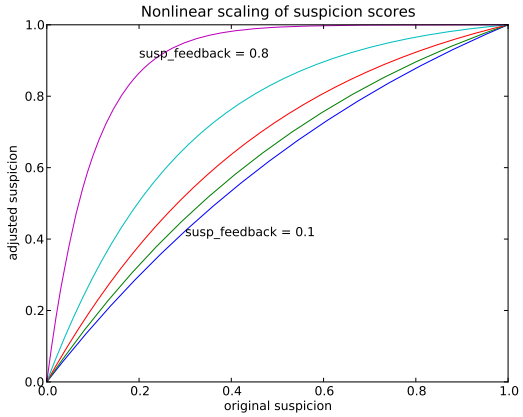

Fig. 4. Nonlinear transfer functions used to rescale suspicion using feedback suspicions

for different values of feedback suspicion, x_{feedback}.

In the first iteration, the existing suspicion is computed only using the attack definitions, as discussed in Sec. IV-B; the suspicion feedback is initialized to 0 for the candidate bubbles. This reflects that initially no evidence exists to refine the search by biasing towards certain types of activities.

Algorithm 5 ADJUST-SUSPICION$(C, \mathit{susp_feedback})$

```
for x ∈ C do
    ID ← ID[x]
    a ← attack[x]
    if susp_feedback[ID][a] is defined then
        x.susp = G_feedback(susp_feedback[ID][a], x.susp)
    end if
end for
```

VI. Simulations and Experiments

A. Simulation

In order to evaluate the performance of the proposed network forensic algorithms, we have set up a simulated environment consisting of several profiles modeling different types of network activities. The activity profiles simulate network traffic patterns ranging from normal network usage over protocols such as HTTP, to malicious network attacks of varying degree. The types of attacks simulated by activity profiles include *port scan*, *ping sweep*, *password attack*, *ruser buffer overflow attack* and *HTTP DoS*.

Each activity profile is described by several attributes including: type of network traffic it generates (e.g. ICMP packets, HTTP requests etc.), generation of packets, start time of the activity, the duration of the activity and any other protocol specific attributes. The possible values of these attributes can be deterministically specified (e.g. type of network traffic), or given as a probabilistic distribution over some sample space (e.g. packet intervals are from a Gamma distribution [33], [34]). For instance, a HTTP denial of service attack session can be modeled by the network profile with the attributes:

- network activity: HTTP
- interval: Gamma(10, 0.5 seconds)
- start: Uniform(0, 1 hr)
- duration: Uniform(1 hr, 2 hr)
- method: Sample(GET, POST)
- header-size: Gamma(5, 1024 bytes)

The profile simulates a session of HTTP DoS attack with a stream of HTTP request events. The events are separated by intervals which are sampled from a Gamma distribution [35], [34] with parameters degree of 10 and expected value of 0.5 seconds. The attack session is scheduled to start at a random time after the start of the simulation, and lasts between one to two hours (according to one security blog of real observed DDoS attacks [36], the average attack duration is a little over two hours). The duration is a random variable sampled from a uniform distribution over one to two hours. The profile also contains two HTTP specific attributes. The HTTP request methods vary between GET and POST, and in case of POST, the request headers also have a payload. The payload size is a random variable from the Gamma distribution with degree of 5 and expected value of 1 kilobytes. Figure VI-A shows the complete listing of the profiles used in the simulation. We omitted the start time as it is determined by the simulation engine. In addition to the attack profiles, we also have simulate normal network traffic from various sources.

The simulation instantiates large volumes of normal profiles, and inject a small number of attack profiles. All the profiles are executed asynchronously. These generated network events are sent to an event log for forensic processing. The simulation environment allows us to evaluate the performance of the proposed network forensics framework in terms of accuracy (precision and recall) and runtime.

B. Experimental Evaluation

We have simulated 48 hours of the network traffic using 1000 instances of the non-attack profiles, and 200 attack profiles. The attack profiles are made to be correlated according to an attack graph, as shown in Figure 3.

Figure 6 shows the likelihood of a HTTP DoS attack bubble at different stages of the log scan. The middle spike shows the most likely starting time and duration of a HTTP DoS attack: starting at $t = 50$ with duration $\Delta t = 14$, and the suspicion score is 0.67.

The naive method for filtering out the suspicious bubbles would be to apply a threshold cut-off to the suspicion scores. Such simple thresholding is problematic for two reasons. First, it is difficult to determine a priori the appropriate threshold value. Second, to support after-fact forensic queries, it is important that we achieve high precision and recall. Precision and recall are two commonly used metrics for measuring the correctness or *accuracy* of classification algorithms [37]. The precision and recall measures are defined as usual (p. 138 of [37]). Let $\mathbf{B}(t)$ be the bubbles extracted at time t, and $\mathbf{A}(t)$ be the actual attacks that have take place at time t.

Attack: Port scan	
network activity	ICMP packet
interval	Gamma(8, 1.0 sec)
duration	Uniform(10 sec, 100 sec)
ports	[22, 80, 8080, 21]

Attack: HTTP denial of service	
network activity	HTTP
interval	Gamma(10, 0.5 sec)
duration	1 hour
method	Sample(GET, POST)
header-size	Gamma(5, 1024 bytes)

Attack: Password guessing	
network activity	SSH LOGIN
interval	Gamma(10, 2 sec)
start	100
duration	1 hour
user	Sample user-dictionary
login-success	Bernoulli(0.001)

Normal: HTTP User	
network activity	HTTP
interval	Gamma(2, 10.0 sec)
duration	Uniform(30 sec, 5 min)
method	CHOOSE(GET, POST)
header-size	Uniform(100 bytes, 1024 bytes)

Normal: SSH login	
network activity	SSH LOGIN
interval	Gamma(2, 30 minutes)
duration	24 hour
user	Sample system-users
login-success	Bernoulli(0.9)

Normal: network traffic	
network activity	ICDM
interval	Gamma(2, 5 sec)
duration	24 hour

Fig. 5. Representative network profiles used in the simulation

$$\begin{aligned} \text{precision}(t) &= \frac{|\mathbf{A}(t) \cap \mathbf{B}(t)|}{|\mathbf{B}(t)|} \\ \text{recall}(t) &= \frac{|\mathbf{A}(t) \cap \mathbf{B}(t)|}{|\mathbf{A}(t)|} \end{aligned}$$

We further define the overall precision and recall as the average precision and recall over time respectively. Let T be the total duration of the entire simulation.

$$\begin{aligned} \text{precision} &= \frac{1}{T}\int_T \text{precision}(t)dt \\ \text{recall} &= \frac{1}{T}\int_T \text{recall}(t)dt \end{aligned}$$

Figure 7 shows the precision and recall if we utilized a simple threshold method to extract the attack bubbles without suspicion feedback. Note that regardless of the threshold value for the likelihood score, it is impossible to achieve high precision and recall simultaneously. If the cutoff is too low, then the

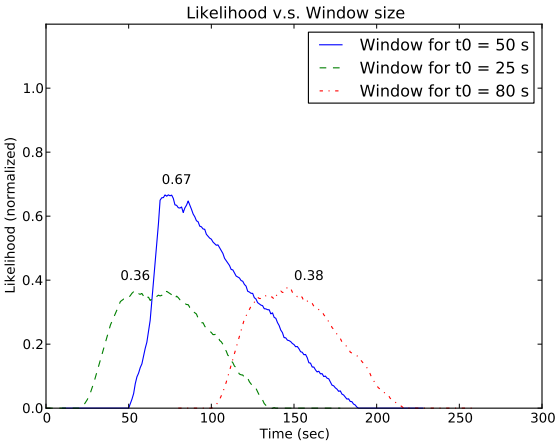

Fig. 6. Scores for different candidate bubbles with varying starting time t and duration τ as in Algorithm 2.

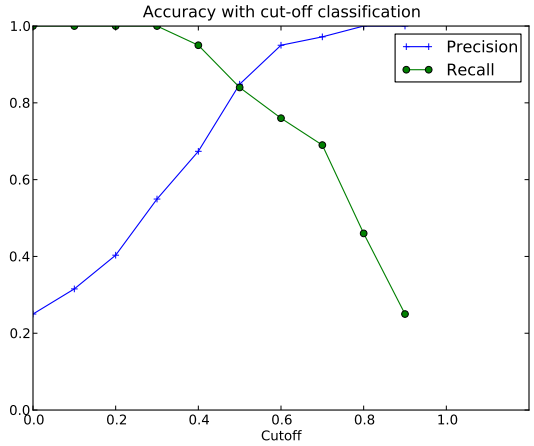

Fig. 7. Accuracy (precision and recall) of attack analysis using simple cutoff.

precision is poor (too many false positives): normal network activities are classified as attacks. If the cutoff is set too high, then recall becomes very poor (too many false negatives): many attacks are missed during the investigation. The accuracy is extremely sensitive to the choice of the threshold.

When suspicion feedback is utilized, we observe a significant improvement in the accuracy of the attack detection. Figure 8 shows the improvements to *both* the precision and recall during the iterations of the suspicion feedback.

Figure 9 shows the precision and recall for various threshold when feedback is used. We observe that even with very high threshold values, the algorithm still performs with high recall value ($> 90\%$). In comparison with the case without suspicion feedbacks (Figure 7), the a good choice of the cut-off value is much easier to make. The two-cluster method as discussed in Algorithm 3 yields a cut-off value of 0.63. The corresponding precision and recall measure are both well over 90%.

In Figure 10, we present $\mathrm{precision}(t)$ and $\mathrm{recall}(t)$ over the entire log file. We observe that the algorithm maintains accurate throughout.

While real-time performance less of a certain in the case of network forensic investigate, our algorithm performs well to support interactive user response. Figure 11 shows the distribution of time (in seconds)

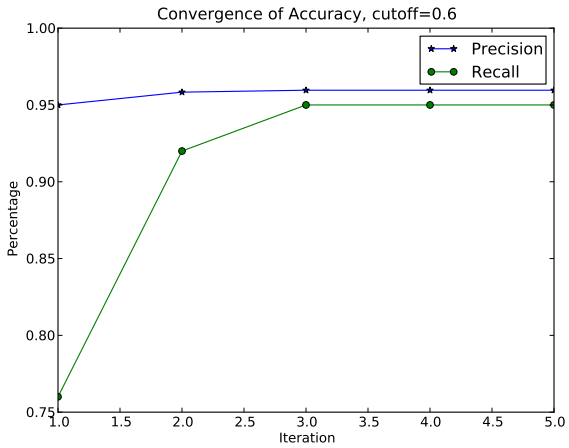

Fig. 8. Convergence of attack suspicion feedback.

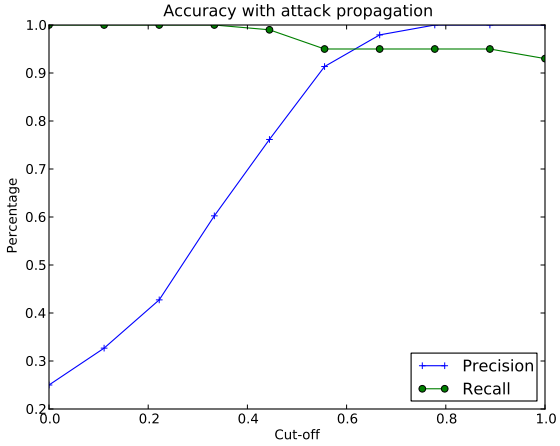

Fig. 9. Accuracy (precision and recall) of attack analysis using suspicion feedback.

to process each window of the event log. The window size is set to be 1000 events.

Remark: We make a final note on the general relevance of the simulations and effectiveness of our algorithm. One concern that may arise is that the adversary can easily obtain the knowledge of all the assumptions and internal mechanisms of our algorithm, including the random variables and parameters, and then somehow use this knowledge to make it more difficult for our algorithm to discover their attack patterns. While this is a valid point, we observe that one of the two major principles behind all the research done on intrusion detection systems is *signature detection* (the other one is anomaly detection) [38]. Signature detection is based on finding network traffic that behaves similarly as some previously defined pattern signature of a known intrusion. Numerous detection tools and proposals have adopted this approach, examples can be found in [13], [39], [40], [1], [41]. Our proposed algorithm also makes use of a signature detection-based mechanism to find potential attack bubbles. The risk that the adversary will have knowledge of the attack signatures and algorithms used to detect the attacks is a risk shared by all the previous works on intrusion detection systems based on the signature detection principle. There are reasons why the signature detection principle is useful for detecting attacks despite this risk. First, a

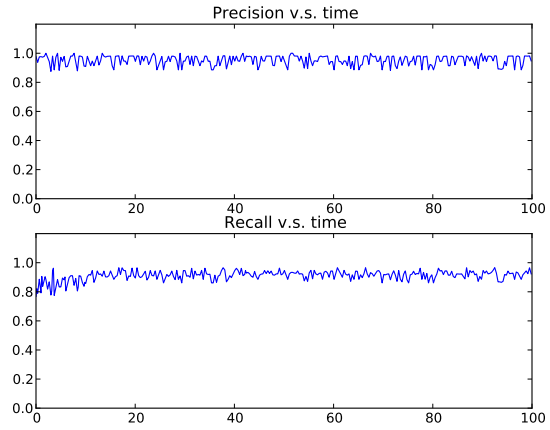

Fig. 10. Accuracy over time.

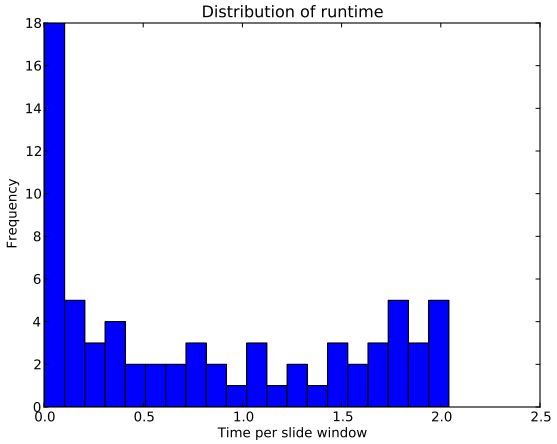

Fig. 11. Distribution of runtime to process 1000 events.

majority of attackers do not write hacking programs from scratch. They typically use existing software tools to help their efforts, such as `nmap` [1], [42]. The behavior of these software tools can then be captured by the pattern signatures. Second, many types of attacks must exhibit certain tell-tale signs that are unavoidable. For instance, an HTTP DDoS attack must involve sending lots of packets to port 80; even if the attacker knows beforehand that this is something being watched for, it cannot be circumvented.

VII. Conclusion

In this paper, we have identified the problem of discovering the context of network events related to a security breach, by mining the logs of network traffic data. We proposed an iterative algorithm to solve the problem. The algorithm scans the log to find coherent groups of events that are likely to be certain recognizable attack instances, and then uses a feedback mechanism to propagate these likelihoods or suspicion scores to the next iteration, thereby increasingly refining the search for events or attacks related to the ones already found. Our simulations verify the accuracy of the algorithm in discovering the attack patterns.

REFERENCES

[1] S. McClure, J. Scambray, and G. Kurtz, *Hacking Exposed: Network security secrets and solutions*, 5th ed. McGraw-Hill, 2005.
[2] E. Casey, "Network traffic as a source of evidence: tool strengths, weaknesses, and future needs," *Elsevier Journal of Digital Investigation*, vol. 1, pp. 28–43, 2004.
[3] V. e. a. Corey, "Network forensics analysis," *IEEE Internet Computing*, vol. 6, no. 6, 2002.
[4] B. H., "The discipline of internet forensics," *Communications of the ACM*, vol. 46, no. 8, August 2003.
[5] V. Paxson, "Bro: a system for detecting network intruders in real-time," *Computer Networks*, no. 31, pp. 2435–2463, 1999.
[6] M. Roesch, "Snort – lightweight intrusion detection for networks," in *Proceedings of the Thirteenth Systems Administration Conference (LISA 1999)*, November 7–12 1999, Seattle, WA.
[7] A. Lakhina, M. Crovella, and C. Diot, "Mining anomalies using traffic feature distributions," in *Proceedings of SIGCOMM'05*, August 21–26, 2005, Philadelphia, PA.
[8] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated worm fingerprinting," in *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI'04)*. USENIX, 2004, San Fransisco, CA.
[9] "Netdetector," http://www.niksun.com/product.php?id=4.
[10] "Networkminer," http://networkminer.wiki.sourceforge.net/NetworkMiner.
[11] "Netintercept," http://sandstorm.net/products/netintercept.
[12] "Wireshark," http://www.wireshark.org.
[13] "Snort," http://www.snort.org.
[14] R. M. et al., "Implementing a generalized tool for network monitoring," in *Proceedings of the Eleventh Systems Administration Conference (LISA 1997)*, 1997, San Diego, CA.
[15] K. Shanmugasundaram, H. Bronnimann, and N. Memon, "Payload attribution via hierarchical Bloom filters," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2004, Washington DC.
[16] C. Cho, S. Lee, C. Tan, and Y. Tan, "Network forensics on packet fingerprints," in *Proceedings of the 21st IFIP Information Security Conference (SEC 2006)*, 2006, Karlstad, Sweden.
[17] M. Ponec, G. P., W. J., and B. H., "New payload attribution methods for network forensic investigations," *ACM Transactions on Information and System Security*, vol. 13, no. 2, pp. 15:2–15:32, February 2006.
[18] "The leurre.com project," http://www.leurrecom.org.
[19] *The Honeynet Project (ed.): Know Your Enemy: Learning about Security Threats*, 2nd ed. Addison Wesley Professional, 2004.
[20] N. Provos, "A virtual honeypot framework," in *Proc. of the 13th USENIX security symposium*, 2004, San Diego, CA.
[21] P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. Freiling, "The nepenthes platform: an efficient approach to collect malware," in *Proc. of the 9th international symposium on recent advances in intrusion detection (RAID)*, September 2006, Hamburg, Germany.
[22] T. Werner, "Honeytrap," http://honeytrap.mwcollect.org.
[23] J. Riordan, D. Zamboni, and Y. Duponchel, "Building and deploying billy goat, a worm-detection system," in *Proc. of the 18th annual FIRST conference*, 2006, Baltimore, Maryland.
[24] F. Pouget and M. Dacier, "Honeypot-based forensics," in *Proc. of AusCERT2004, Brisbane, Australia*, May 23–27 2004.
[25] F. Pouget, M. Dacier, J. Zimmerman, A. Clark, and G. Mohay, "Internet attack knowledge discovery via clusters and cliques of attack traces," *Journal of Information Assurance and Security*, vol. 1, pp. 21–32, 2006.
[26] O. Thonnard and M. Dacier, "A framework for attack patterns' discovery in honeynet data," *Digital Investigation*, vol. 8, pp. S128–S139, 2008.
[27] H. Jin, O. de Vel, K. Zhang, and N. Liu, "Knowledge discovery from honeypot data for monitoring malicious attacks," in *Proc. of the 21st Australian Joint Conference on Artificial Intelligence: Advances in Artificial Intelligence*, 2008, Auckland, New Zealand, pp. 470–481.
[28] V. Yegneswaran, P. Barford, and V. Paxson, "Using honeypots for internet situational awareness," in *Fourth ACM SIGCOMM Workshop on Hot Topics in Networking (Hotnets IV)*, 2005, College Park, Maryland.
[29] C. Estan, S. Savage, and G. Varghese, "Automatically inferring patterns of resource consumption in network traffic," in *Proc. of SIGCOMM'03*, August 25–29, Karlsruhe, Germany 2003.
[30] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "Blinc: Multilevel traffic classification in the dark," in *Proc. of SIGCOMM'05*, August 21–26, Philadelphia, Pennsylvania 2005.
[31] J. Kannan, J. Jung, V. Paxson, and C. Koksal, "Semi-automated discovery of application session structure," in *Proc. of the Sixth ACM SIGCOMM Conference on Internet Measurement (IMC'06)*, 2006, Rio de Janeiro, Brazil, pp. 119–132.
[32] A. Leon-Garcia, *Probability and Random Processes for Electrical Engineering*, 2nd ed. Addison-Wesley, 1994.
[33] E. Doron and A. Wool, "Wda: A web farm distributed denial of service attack attenuator," *Computer Networks*, vol. 10, no. 1016, May 2010.
[34] Y. Ohsita and S. Ata, "Detecting distributed denial-of-service attacks by analyzing tcp syn packets statistically," in *Proc. IEEE GLOBECOM, Dallas, TX*, 2004, p. 20432049.
[35] J. A. Gubner, *Probability and random processes for electrical and computer engineers*. Cambridge University Press, 2006.
[36] "Georgia ddos attacks a quick summary of observations," http://asert.arbornetworks.com/2008/08/georgia-ddos-attacks-a-quick-summary-of-observations/.
[37] D. L. Olson and D. Delen, *Advanced Data Mining Techniques*, 1st ed. Springer, 2008.
[38] S. Axelsson, "Intrusion detection systems: A survey and taxonomy," Department of Computer Engineering, Chalmers University of Technology, Tech. Rep., March 2000.
[39] "Tripwire," http://www.tripwire.com/.
[40] W. Venema, "Tcp wrapper: Network monitoring, access control, and booby traps," in *Proc. of the 3rd USENIX UNIX Security Symposium*, September 14–16 1992, pp. 85–92.
[41] Northcutt, Cooper, Fearnow, and Frederick, *Intrusion Signatures and Analysis*. Sams, 2001.
[42] "Nmap," nmap.org.