

# Nexat: A History-Based Approach to Predict Attacker Actions

Casey Cipriano  
UCSB  
CS Department  
Santa Barbara, CA, USA  
ccipriano@gmail.com

Ali Zand  
UCSB  
CS Department  
Santa Barbara, CA, USA  
zand@cs.ucsb.edu

Amir Houmansadr  
UIUC  
ECE Department  
Urbana, IL, USA  
ahouman2@illinois.edu

Christopher Kruegel  
UCSB  
CS Department  
Santa Barbara, CA, USA  
chris@cs.ucsb.edu

Giovanni Vigna  
UCSB  
CS Department  
Santa Barbara, CA, USA  
vigna@cs.ucsb.edu

## ABSTRACT

Computer networks are constantly being targeted by different attacks. Since not all attacks are created equal, it is of paramount importance for network administrators to be aware of the status of the network infrastructure, the relevance of each attack with respect to the goals of the organization under attack, and also the most likely next steps of the attackers. In particular, the last capability, attack prediction, is of the most importance and value to the network administrators, as it enables them to provision the required actions to stop the attack and/or minimize its damage to the network's assets. Unfortunately, the existing approaches to attack prediction either provide limited useful information or are too complex to scale to the real-world scenarios.

In this paper, we present a novel approach to the prediction of the actions of the attackers. Our approach uses machine learning techniques to learn the historical behavior of attackers and then, at the run time, leverages this knowledge in order to produce an estimate of the likely future actions of the attackers. We implemented our approach in a prototype tool, called *Nexat*, and validated its accuracy leveraging a dataset from a hacking competition. The evaluations show that *Nexat* is able to predict the next steps of attackers with very high accuracy. In particular, *Nexat* achieves a 94% accuracy in predicting the next actions of the attackers in our prototype implementation. In addition, *Nexat* requires little computational resources and can be run in real-time for instant prediction of the attacks.

## Categories and Subject Descriptors

C.2.0 [Computer-Communication Networks]: General—*Security and protection*; C.2.3 [Computer-Communication Networks]: Network Operations—*Network monitoring*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACSAC '11 Dec. 5-9, 2011, Orlando, Florida USA  
Copyright 2011 ACM 978-1-4503-0672-0/11/12 ...\$10.00.

## General Terms

Algorithms, Design, Security

## Keywords

Attack prediction, situation awareness, machine learning

## 1. INTRODUCTION

Computer networks are constantly under various attacks. Network administrators deploy intrusion detection systems (IDS) in order to detect the occurrence of the *events* that may be part of an attack [1, 19, 24]. Unfortunately, while intrusion detection systems provide useful information about the attack-related events, they provide little information about the network's *situation* before and after any of the attacks. In order to better provision and respond to the attacks a network administrator needs the answers to the following questions: how are different events related to an attack? what is the impact of an attack on the network? and, the most important, what are the most likely next actions of the attackers? The answers to these questions and other similar questions provide a high-level understanding of the security situation of the computer networks, also referred to as *situation awareness*. Situation awareness has long been referred to as an important and critical aspect for cyber-defense [7, 6, 10, 23, 3, 26, 20, 14]. Endsley [6] proposes a general definition for situation awareness as "the perception of the elements of the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future." With respect to the cyber-defense scenario, situation awareness tries to contextualize cyber-attacks with respect to the network infrastructure and the mission flows in order to enable a network administrator to prioritize certain preventative measures and also determine the severity of different attacks [25]. Furthermore, situation awareness can provide estimates of the likely next steps of an attacker, assisting a network administrator in using appropriate preventative approaches instead of just reactive ones [21, 22].

Predicting the next actions of the attackers is an important, yet difficult, aspect of situation awareness. Existing approaches for attack prediction either provide limited predictive information [15, 21, 13] or are complex and can not be utilized for large-scale scenarios [18, 8, 22, 28]. As an example, Qin et al. uses Granger causality analysis in order to compute precursor alerts [21]. This approach provides limited accuracy in attack prediction as its per-

formance is very sensitive to the presence of unrelated background alerts. As another example, Geib et al. provide a plan prediction scheme that uses pre-determined attacker plans in order to predict future attacks [8]. This approach is not scalable, as it does not perform in a fully automated manner and requires continuous inputs from the network administrators in order to perform the predictions. Therefore, there is the need for effective situation awareness tools that provide attack prediction capabilities that are not only accurate, but also scalable to large-scale networks.

In this paper, we leverage *machine learning* to predict the next steps of attacks based on histories of previous attacks. We implemented our approach in a tool called *Nexat*, which is the focus of this paper. *Nexat* uses machine learning techniques to *learn* the past behavior of the attackers, and then uses this knowledge to predict the next steps of future attacks in real-time. More specifically, *Nexat* consists of three operational phases: the data extraction phase, the training phase, and the prediction phase. During the data extraction phase, *Nexat* extracts some information from the alerts previously generated by intrusion detection systems. *Nexat* uses this extracted information during the training phase in order to generate a knowledge base about the attackers' behavior. Finally, *Nexat* uses the generated models of the network and the attackers to make predictions on the next steps of a live stream of attack alerts. The attack predictions provided by *Nexat* can be used by decision makers, e.g., network administrators, to devise efficient defensive actions and to estimate the impact of a future attack on the critical assets of the organization.

We evaluated the performance of *Nexat* using a prototype implementation on a unique database of attack alerts. The alert database is collected by an intrusion detection system during a large-scale hacking competition, where a number of hacker teams tried to compromise the same target network by taking multiple steps. Our evaluation shows that *Nexat* is able to learn the attacking behavior of the attackers and use it to predict their future actions efficiently. In particular, by learning the attack histories of 32 teams *Nexat* was able to predict the actions of another hacker team with an average accuracy of 94%.

The rest of this paper is organized as follows: in Section 2, we give a detailed description of our attack prediction algorithm, *Nexat*. Section 3 evaluates a prototype implementation of *Nexat* by discussing its prediction efficiency. In Section 4 we discuss some of the related work. Finally, the paper is concluded in Section 5.

## 2. NEXAT: HISTORY-BASED ATTACK PREDICTION

In this section, we describe the design of *Nexat*, a tool that we develop for predicting attackers' likely next steps. *Nexat* uses a machine learning technique to predict the attacker actions based on the activity history of the attackers. More specifically, *Nexat*'s operation can be divided into three main phases: the data extraction phase, the training phase, and the prediction phase. Before describing these steps, we first provide a formal definition of some of the terms used later. Also, Table 1 summarizes all of the elements used in the design of *Nexat*, e.g., lists and hash tables, along with their symbols and definitions.

**DEFINITION 1. (Alert)** An alert is a set of information about a suspicious network event being observed and reported by an IDS system.

The set of information included in an alert depends on the type of the reporting IDS system; common examples of alert information are the "source address" and the "destination address" of the network flow associated with the event, and the time of its occurrence.

**DEFINITION 2. (Attack session)** An attack session is a sequence of alerts whose properties satisfy certain conditions. We define an alert  $A$  to be part of an attack session  $S$  if at least one of these properties holds:

- the destination of alert  $A$  is the same as the destination of a previous alert in  $S$ ,
- the source of alert  $A$  is the same as the source of a previous alert in  $S$ ,
- the source of alert  $A$  is a destination of a previous alert in  $S$ .

In addition,  $A$  should lie within a time window of  $w$  seconds from the last alert in  $S$ .

The properties used in specifying the attack sessions are meant to capture three common attack patterns, namely, *one-to-many*, *many-to-one*, and *island-hopping*. A single source attacking many unique targets is classified as a one-to-many pattern, whereas multiple sources attacking a single target is referred to as a many-to-one attack. Finally, island-hopping occurs when an attacker uses a previously-compromised target to attack another target. By looking for each of these attack patterns we establish a basis for finding all of the alerts that belong to a specific attack.

The *data extraction* phase takes as input a training set containing a history of previously-observed alerts. Using these alerts, *Nexat* extracts a list of all of the possible attack sessions by grouping together the alerts as will be described later. The rationale behind this classification is that alerts pertaining to the same attack session are very likely to be part of the same attack, and, hence, they are very likely to happen simultaneously in the future. During the *training phase*, *Nexat* analyzes the extracted attack sessions to identify the co-occurrence relationship amongst different alerts being involved in these sessions. This results in a knowledge base for *Nexat* which is used during the *prediction phase* to probabilistically predict the next actions of an attacker, once a live stream of alerts are being observed by an intrusion detection system. In the following, we provide the detailed description of these three phases.

### 2.1 Data Extraction Phase

In this phase, *Nexat* extracts attack sessions out of a list of alerts that are reported by one or more IDSs. This is done in two steps: generating the list of alerts, and extracting attack sessions from the alerts list.

#### Generating the alerts list

The current implementation of *Nexat* takes Snort alerts<sup>1</sup> as input and extracts four of its attributes, i.e., the name of the attack, the source address, the destination address, and the time the alert was generated. Table 2 shows some sample alerts generated by a Snort IDSs. *Nexat* constructs and maintains an *alerts list* that keeps the alerts collected by the IDSs in an ascending order of time. We assume that multiple IDSs are time-synchronized. Table 3 shows a sample alerts list. For the sake of simplicity, we use pseudonyms for the name of the alerts and also for the source and destination IP addresses.

#### Extracting the attack sessions

For any alert  $A$  in an alerts list  $AL$ , *Nexat* generates an attack session  $S$  initiating from  $A$  as defined in Definition 2. More specifically, in order to add an alert  $A_1$  to an existing attack session, one

<sup>1</sup>The code can easily be extended to take inputs from other IDSs as well.

Table 1: Different elements used in *Nexat*.

Name	Symbol	Description
Alert	$A$	An alert as defined in Definition 1.
Alerts list	$AL$	A list that contains the list of alerts collected from an IDS in ascending order of time.
Attack session	$S$	An attack session as defined in Definition 2.
List of attack sessions	$H$	A hash table keyed by alerts, containing the corresponding attack sessions.
Trained dataset	$T$	A hash table that contains the empirical probabilities trained during the training phase.
Power-set of an alert	$P(A)$	The power-set corresponding to an alert as defined in the text.
Results list	$R(T, I)$	A hash table that provides predictions on the next alert of an attack stream of $I$ based on the trained dataset $T$ .

Table 2: Example alerts generated by an Snort IDS.

Pseudonym	Description
A	DNS named authors attempt
B	(http_inspect) OVERSIZE REQUEST-URI DIRECTORY
C	SNMP trap TCP
D	WEB-CGI bb-hist.sh access
E	MISC source port 53 to <1024
F	BACKDOOR Wordpress backdoor feed.php code execution attempt

Table 3: An example list of alerts. The source and destination addresses are replaced by pseudonyms of the hosts.

Alert name	Attribute		
	Source address	Destination address	Time
A	9	6	1
G	2	4	2
B	8	7	4
D	7	1	7
K	3	4	8
J	2	5	9
E	1	6	10

of these three requirements should be met:  $A_1$ 's destination address is the same as the destination of an alert existing in  $S$ ,  $A_1$ 's source is the same as the source address of an alert in  $S$ , or  $A_1$ 's source is the same as the destination address of an alert in  $S$ . *Nexat* uses a sliding time window,  $w$ , for performing such correlation. This algorithm,  $\text{GetSession}(A \in AL, AL)$ , is depicted in Algorithm 2.1.  $\text{GetSession}(A \in AL, AL)$  goes over an alerts list  $AL$ , being sorted in ascending order of time, to find all of the existing attack sessions corresponding to an alert  $A \in AL$ . For any instance of such an attack session,  $S$ ,  $\text{GetSession}$  performs the following steps:

- For the attack session  $S$ ,  $\text{GetSession}$  creates and maintains two lists  $\text{SourceList}(S)$  and  $\text{TargetList}(S)$ , which contain the possible source address and destination addresses of the future alerts of  $S$ , respectively.
- $\text{GetSession}$  traverses the alerts list  $AL$  in descending or-

der of time starting from  $A$  in order to find and append related alerts to  $S$ . For any alert  $A_1$  which lies in a window  $w$  from the last alert in  $S$ ,  $\text{GetSession}$  adds it to  $S$  if either  $A_1$ 's source is in  $\text{SourceList}(S)$  or  $A_1$ 's destination is in  $\text{TargetList}(S)$ .

- For any alert  $A_1$  that is added to an attack session  $S$ , the destination of  $A_1$  is added to  $\text{TargetList}(S)$  and the source of  $A_1$  is added to both  $\text{SourceList}(S)$  and  $\text{TargetList}(S)$  lists (this enables *Nexat* to find the three possible types of attack, namely, one-to-many, many-to-one, and island-hopping, as mentioned before).
- $\text{GetSession}$  closes  $S$  if it does not find another alert from  $AL$  that satisfies the mentioned properties.

*Nexat* maintains a list of attack sessions,  $H$ , which is a hash table that is indexed by alert names.  $H$  stores lists of attack session sets corresponding to each alert. Upon finding an attack session  $S$  for an alert  $A \in AL$ , as described above, *Nexat* appends  $S$  to the line of  $H$  corresponding to  $A$ , i.e.,  $H[A]$ . Once this is done for all of the alerts in  $AL$  the hash table  $H$  will hold a set of attack sessions for each alert in the  $AL$ . This algorithm is sketched in Algorithm 2.2, where the  $\text{GetSession}(A, AL)$  algorithm evaluates the attack sessions for an alert  $A$ , as detailed before. The hash table  $H$  will serve as the input to the training phase, as described later. Table 4 shows a sample hash table  $H$ .

**Algorithm 2.2:** DATAEXTRACTIONPHASE( $AL$ )

```

 $H \leftarrow hashtable()$ 
for each  $A \in AL$ 
  do  $H[A].append(\text{GETSESSION}(A, AL))$ 

```

**Algorithm 2.1:** GETSESSION( $A \in AL, AL$ )

```

reversedList ← AL.reverse()    % X ← Y assigns the value of Y to X
attack.SessionSet ← set()
TargetList ← set()
SourceList ← set()
lastAlert ← A
TargetList.append([lastAlert.target, lastAlert.source])
SourceList.append(lastAlert.source)
for each previousAlert ∈ reversedList
  if previousAlert.time > lastAlert.time - w
    then {
      if (previousAlert.target ∈ TargetList
      or previousAlert.source ∈ SourceList)
        then {
          S.append(previousAlert.name)
          lastAlert ← previousAlert
          TargetList.append([previousAlert.source, previousAlert.target])
          SourceList.append(previousAlert.source)
        }
      else
        break
    }
return (S)

```

Table 4: A sample list of attack sessions,  $H$ .

Alert Name	Attack sessions
E	{A,B,D}, {A,B}, {A,C}, {B,D}, {B}
F	{B,C,D}, {B,C}, {C}
...	...

*An example run of the data extraction phase*

For the sake of clarity, here we mention a sample run of the data extraction phase. Let us suppose that *Nexat* received as input an alerts list  $AL$  as shown in Table 3. As can be seen,  $AL$  contains the name of the alerts along with their corresponding sources and destination addresses, and the time when they occurred. We show how *Nexat* proceeds to find the attack sessions for the last alert of  $AL$ , i.e.,  $E$ , for a time window size of  $w = 5sec$ .

*Nexat* traverses the  $AL$  list backwards, starting with alert  $J$ . Since  $J$  does not share any source or destination address similarities with  $E$ , it is not added to the attack session  $S$ . The situation is the same for alert  $K$ , since there is no shared destination or source information with  $E$ . However, as can be seen, the destination address of the alert  $D$  is the same as the source address of the last alert in  $S$ , i.e.,  $E$ . In fact, this can be an indication of an island-hopping attack. Since the time difference between the alerts  $D$  and  $E$  (the last alert in  $S$ ) is less than the time window  $w = 5sec$  the alert  $D$  is appended to the attack session  $S$ . In addition, the source address of  $D$  is added to  $SourceList(S)$  and  $TargetList(S)$ , and its destination address is added to  $TargetList(S)$ . Similarly, the alerts  $B$  and  $A$  are appended to  $S$  and their source and destination addresses are added to the corresponding lists. When *Nexat* reaches the end of the  $AL$  list (or the time window  $w$  from the last alert has elapsed),  $GetSession(E, AL)$  closes the attack session  $S$  and appends it to the line indexed  $E$  of the list of attack sessions, e.g.,  $H[E]$ . In this example, the attack session  $S = \{A, B, D\}$  is appended to the list of attack sessions shown in Table 4. The following is a real-world sample of an attack session that we observed in our evaluations:

{ "ICMP PING NMAP" , "(http\_inspect) BARE BYTE UNICODE ENCODING" , "WEB-PHP PHP function CRLF injection attempt" , "WEB-MISC .htpasswd access" , "(http\_inspect) DOUBLE ENCODING ATTACK" , "WEB-MISC encoded cross site scripting attempt" , "SHELLCODE x86 NOOP"}.

**2.2 Training Phase**

*Nexat* uses the list of attack sessions, i.e., the hash table  $H$ , generated during the data extraction phase to train about the co-occurrence of the alerts, i.e., to determine which alerts are more likely to occur in the same attack session. The training algorithm is depicted in Algorithm 2.3.

**Algorithm 2.3:** TRAININGPHASE( $H$ )

```

T ← hashtable()
for each A ∈ H
  U(A) ← list()
  for all S ∈ H[A]
    do U = U ∪ S
    for each x ∈ POWERSET(U) \ {∅}
      do {
        o ← 0
        for each S ∈ H[A]
          do if x ⊆ S
            then o ← o + 1
        T[x].append({o, A})
      }
total ← hashtable()
for each y ∈ T
  do { total[y] ← 0
        total ← total + T[y].o
      }
for each y ∈ T
  do T[y].o ← T[y].o / total[y]

```

During the training phase, the `TrainingPhase` algorithm uses the list of attack sessions  $H$ , created in the data extraction phase, to generate a *trained dataset*  $T$ . The trained dataset  $T$  is a hash table indexed by the set of alerts. For any set of alerts  $x \in T$ ,  $T$  keeps a list of pairs  $(p, A)$  such that  $p$  is the empirical probability that  $x$  is followed by the alert  $A$ . The generation of  $T$  is further described later. The training algorithm of *Nexat* is performed as follows:

Table 5: A sample trained dataset  $T$ .

(a) Before normalization		(b) After normalization	
{A}	{3, E}	{A}	{1, E}
{B}	{4, E}, {2, F}	{B}	{.66, E}, {.33, F}
{C}	{1, E}, {3, F}	{C}	{.25, E}, {.75, F}
{D}	{2, E}, {1, F}	{D}	{.66, E}, {.33, F}
{AB}	{2, E}	{AB}	{1, E}
{AC}	{1, E}	{AC}	{1, E}
{AD}	{1, E}	{AD}	{1, E}
{BC}	{2, F}	{BC}	{1, F}
{BD}	{1, E}, {1, F}	{BD}	{.5, E}, {.5, F}
{CD}	{1, F}	{CD}	{1, F}
{ABC}	{ $\emptyset$ }	{ABC}	{ $\emptyset$ }
{ABD}	{1, E}	{ABD}	{1, E}
{ACD}	{ $\emptyset$ }	{ACD}	{ $\emptyset$ }
{BCD}	{1, F}	{BCD}	{1, F}

- For each alert  $A \in H$ , `TrainingPhase` evaluates the union set of all of the attack sessions that  $H$  keeps for  $A$ . As an example, if for an alert  $A$  the table  $H$  returns two attack sessions of  $\{B, C\}$  and  $\{B, F\}$  the union set for  $A$  would be  $U(A) = \{B, C, F\}$ . In fact, the union set  $U(A)$  returns a set of possible precursor alerts for the alert  $A$ .
- `TrainingPhase` evaluates  $P(A)$ , which is the power-set of  $U(A)$  excluding the empty set  $\{\emptyset\}$  and limited to a maximum size of  $k$ . As an example, for the  $U(A) = \{B, C, F\}$  mentioned the corresponding power-set is  $P(A) = \{\{B\}, \{C\}, \{F\}, \{B, C\}, \{B, F\}, \{C, F\}, \{B, C, F\}\}$ . The limit  $k$  is used to bound the computational complexity of `Nexat` and is set to 3 in our implementation.
- For any set  $x \in P(A)$ , `TrainingPhase` evaluates  $N(x, A)$  to be the number of attack sessions,  $S \in H[A]$  that are a superset of  $x$ , i.e.,  $N(x, A) = |\{S | S \in H(A) \& x \subseteq S\}|$ . If  $N(x, A) > 0$  the pair  $(N(x, A), A)$  is appended to the line  $x$  of the hash table  $T$ , i.e.,  $T[x]$ .
- Finally, `TrainingPhase` normalizes the entries of the  $T$  table by dividing each  $N(\cdot, \cdot)$  by the sum of all of the  $N(\cdot, \cdot)$  entries in the same row of table  $T$ .

Table 5 shows a sample  $T$  hash table before and after normalization. For any set  $x \in T$ , a normalized pair of  $(p, A)$  indicates that with probability  $p$  the alert  $A$  proceeds the set of alerts in  $x$ . This is used in the prediction phase to predict the next steps of an attack, as described later.

It should be mentioned that limiting the size of the power-sets to  $k = 3$  puts a constraint on the storage and the computational resources required by `Nexat`. Increasing  $k$  improves the training process of `Nexat` at the cost of exponentially increasing the required resources.

### An example run of the training phase

We continue the example mentioned in Section 2.1 for the training phase of `Nexat`. The data extraction phase of `Nexat` generates the list of attack sessions  $H$  that is shown in Table 4. As can be seen,  $H$  keeps the attack session sets of  $\{A, B, D\}$ ,  $\{A, B\}$ ,  $\{A, C\}$ ,  $\{B, D\}$ ,  $\{B\}$  for the alert  $E$ , and the attack session sets of  $\{B, C, D\}$ ,  $\{B, C\}$ ,  $\{C\}$  for the alert  $F$ . So, the union set cor-

responding to  $E$  can be evaluated as  $U(A) = \{A, B, C, D\}$ , resulting in the corresponding powerset to be

$$P(E) = \{\{A\}, \{B\}, \{C\}, \{D\}, \{AB\}, \{AC\}, \{AD\}, \{BC\}, \{BD\}, \{CD\}, \{ABC\}, \{ABD\}, \{ACD\}, \{BCD\}\}$$

As mentioned before, the powerset  $P(\cdot)$  excludes the empty set and is limited to subsets with maximum size of  $k = 3$ . One can similarly find the powerset corresponding to the alert  $F$  as

$$P(F) = \{\{B\}, \{C\}, \{D\}, \{BC\}, \{BD\}, \{CD\}, \{BCD\}\}$$

`TrainingPhase` uses these powersets to generate the trained dataset  $T$  as described before. Table 5 shows the generated  $T$  before and after normalization.

## 2.3 Prediction phase

In the prediction phase, `Nexat` uses the trained dataset  $T$  generated during the training phase to make predictions on the live streams of attacking events. More specifically, for each alert in a live stream of alerts  $I$  returned by one or more IDSs, `Nexat` predicts the most likely forthcoming alerts.

The prediction algorithm of `Nexat` is illustrated in Algorithm 2.4. `PredictionPhase` uses a *results list*,  $R(T, I)$ , to store the predictions performed on  $I$  using the trained dataset of  $T$ .  $R(T, I)$  is a hash table indexed by the name of the alerts, where for each alert it stores a list of *result pairs* in the form of  $(p, \ell)$  where  $p$  is the probability and  $\ell$  is the length of a set of alerts. The `PredictionPhase` algorithm works as follows:

- Suppose that  $A_r$  is the last received alert in the live stream of alerts  $I$ . `PredictionPhase` evaluates the attack session  $S_r$  corresponding to  $A_r$  using the `GetSession( $A_r, I$ )` algorithm mentioned in Algorithm 2.1.
- `PredictionPhase` also evaluates  $P_r$ , the limited power-set of  $S_r$ , as defined before (excluding the empty set and limited to size  $k = 3$ ).
- For any  $x \in P_r$  and for any  $r \in T[x]$ , `PredictionPhase` appends the result pair of  $(r.p, length(x))$  to the line  $x$  of hash table  $R$ , i.e.,  $R[x]$ .  $r.p$  is the probability attribute of  $r$ , as defined in Section 2.2, and  $length(x)$  returns the length of the set  $x$ . This identifies all of the records in the trained dataset that are related to the current alert.
- Finally, `PredictionPhase` normalizes the probability attribute in the result pairs of the  $R$  table. In order to do this, `Nexat` evaluates the weighted sum of the probabilities for each row (corresponding to alert  $A$ ) of the table as  $Sum_A = \sum_{i=1}^{Num_A} p_i \times \ell_i$ , where  $Num_A$  is the number of result pairs corresponding to alert  $A$  in  $R$ . `PredictionPhase`, then, divides all of the probability attributes of the result pairs by  $Sum_A$  to produce the normalized weighted probabilities.

**Algorithm 2.4:** PREDICTIONPHASE( $T, I$ )

```

 $R \leftarrow \text{hash}()$ 
for each  $x \in \text{POWERSET}(\text{GETSESSION}(A_r, I)) \setminus \emptyset$ 
  do  $\left\{ \begin{array}{l} \text{for each } r \in T[x] \\ \text{do } R[r].\text{append}(\{r.p, \text{length}(x)\}) \end{array} \right.$ 
for each  $A \in R$ 
   $\left\{ \begin{array}{l} \text{sum} \leftarrow 0 \\ \text{for each } \text{record} \in R[A] \\ \text{do } \text{sum} \leftarrow \text{sum} + \text{record}.p * \text{record}.\ell \\ R[A] \leftarrow \text{sum} \end{array} \right.$ 
 $\text{sum} \leftarrow 0$ 
for each  $A \in R$ 
  do  $\text{sum} \leftarrow \text{sum} + R[A]$ 
for each  $A \in R$ 
  do  $R[A] \leftarrow R[A] / \text{sum}$ 

```

*Nexat* uses the weighted probabilities calculated above to predict the future alerts. In fact, by weighting the probabilities we give more significance to the longer attack sessions from the trained dataset. Table 6 shows a sample results list  $R$ .

*An example run of the prediction phase*

We will continue with the data provided from the training phase example. `PredictionPhase` uses the trained dataset  $T$  from the training phase to make predictions on a live stream of alerts  $I$ . Suppose that running the `GetSession` algorithm on the most recent alert of the stream  $I$  returns the set  $S_I = \{B, C, D\}$ . The power-set of  $S_I$ , as defined before, results in the set

$$\{\{B\}, \{C\}, \{D\}, \{BC\}, \{BD\}, \{CD\}, \{BCD\}\}$$

`PredictionPhase` extracts all of the corresponding records from  $T$  and imports them into the results list  $R$ , as shown in the Table 6. As can be seen, the entries of this hash table are the result pairs in the form of  $(p, \ell)$ , where  $p$  is the probability of the subset and  $\ell$  is the length of the subset. The columns of the table show the corresponding subsets for the sake of clarity. The table also shows the weighted sum for the alerts  $E$  and  $F$ . Using the weighted sums *Nexat* provides a predicted probability for any of the alerts to be the next alert of  $I$ . As the table shows, for the mentioned example *Nexat* provides a 78.5% prediction of  $F$  being the next alert, and a 21.5% prediction of  $E$  being the next alert.

### 3. EVALUATION

In this section, we evaluate the performance of the *Nexat* attack prediction tool described in this paper. We run a prototype implementation of *Nexat* over a large database of attack alerts gathered during the 2008 UCSB International Capture The Flag (iCTF) hacking competition [2], held on December 2008. The alert database contains 248,783 alerts that are generated by a snort intrusion detection system during the competition. A total number of 800 attackers from different educational institutions participated in the iCTF competition in the form of 40 hacking teams.

An identical copy of the same multi-host network was dedicated to each team, and the goal of the competition for the teams was to compromise their dedicated networks through an arbitrary number of attacking steps. The network was monitored by both signature-based and anomaly-based intrusion detection systems. The competing teams were allowed to use any hacking technique, however, they would lose points whenever their activities were detected by the intrusion detection systems. More information about the iCTF 2008 competition including the competition rules, the architecture

of the competition networks, and the list of the competitors is available online [2].

We use the database of alerts described above to evaluate the prediction performance of *Nexat*: we use the alerts generated as a result of attacking actions of one or more teams to train *Nexat*, and then use the trained *Nexat* to predict the next actions of other teams. In fact, this resembles the real-world scenario for attack prediction: a network administrator aims at predicting future attacks by having access to a history of attacks performed against the same network infrastructure. The conducted evaluations are described in the following.

Before explaining the experiments we define the accuracy metric that is used for the evaluations.

**DEFINITION 3. Accuracy:** We define the accuracy of *Nexat* in predicting an stream of alerts,  $I$ , to be the mean of the probabilities that *Nexat* predicts for all of the observed alerts. More specifically, if *Nexat* predicts the next alerts of  $I$  with probabilities  $p_i$  ( $i = 1, \dots, n$ ), the accuracy of *Nexat* is  $\sum_{i=1}^n p_i / n$ .

#### 3.1 Experiment 1: trained by one team

In this experiment, *Nexat* is trained using the Snort alerts corresponding to one of the hacking teams, and then the trained models are used to predict the actions of the other teams. To do this, we extracted the alerts corresponding to any of the teams from the mentioned alerts database. Teams who produced less than 100 alerts were ignored as this would not be sufficient data to provide for any meaningful training. This filtering process left us with 33 unique teams, each generating between 100 and 20000 alerts. This resulted into  $32^2$  series of predictions, i.e., we choose one team for training and try to predict actions of the other 32 teams.

Figure 1 depicts the results of the experiments for a time window of  $w = 5\text{sec}$ . Each point on the figure shows the accuracy of *Nexat*, as defined in Definition 3, in predicting the actions of one hacker team, being trained by the attack history of another team. The horizontal axis of the figure shows the name of the team that *Nexat* uses for training. The names assigned to the teams are not their actual names, but their ranking at the end of the competition, e.g., team 3 gained the third place in the competition. The empty columns of the figure correspond to the teams that produced less than 100 alerts, hence were not used to train *Nexat*. For any of the hacker teams used for training *Nexat*, the figure also shows the average accuracy of *Nexat* in predicting the actions of all the other teams. As can be seen, in 22 cases (out of a total of 33 cases) the mean accuracy of *Nexat* in predicting the actions of the other teams is more than 60%. This is significantly promising, because *Nexat* uses the history of *only* one hacker team during the training phase. We observe that in a few cases, using a single team for training *Nexat* results in very poor prediction performance, e.g., in the case of teams ranked 13 and 24. We call such teams *bad predictor* teams. In fact, the bad predictor teams take hacking actions that are different from the rest of the teams. We can see that most of the bad predictor teams are placed in the lower two-third of the competition's ranking table, i.e., only one of the first 12 teams is a bad predictor. We conclude that the more skilled a hacker team is the better prediction performance is achieved by using that team for training *Nexat*.

We also use a force-based algorithm to show the clusters of the hacker teams based on the results of the predictor, as shown in Figure 2. Each circle in the figure represents one of the hacker teams, where the number inside the circle is the rank of the team in the competition. Two teams are connected in the graph if the accuracy of *Nexat* in predicting each of them is more than 0.5, by using the

Table 6: A sample result list  $R$ .

Alerts	{B}	{C}	{D}	{BC}	{BD}	{CD}	{BCD}	Weighted sum	Predicted probability
E	(.66, 1)	(.25, 1)	(.66, 1)	(0,2)	(.5, 2)	(0,2)	(0,3)	2.583	0.215
F	(.33, 1)	(.75, 1)	(.33, 1)	(1, 2)	(.5, 2)	(1, 2)	(1, 3)	9.416	0.785

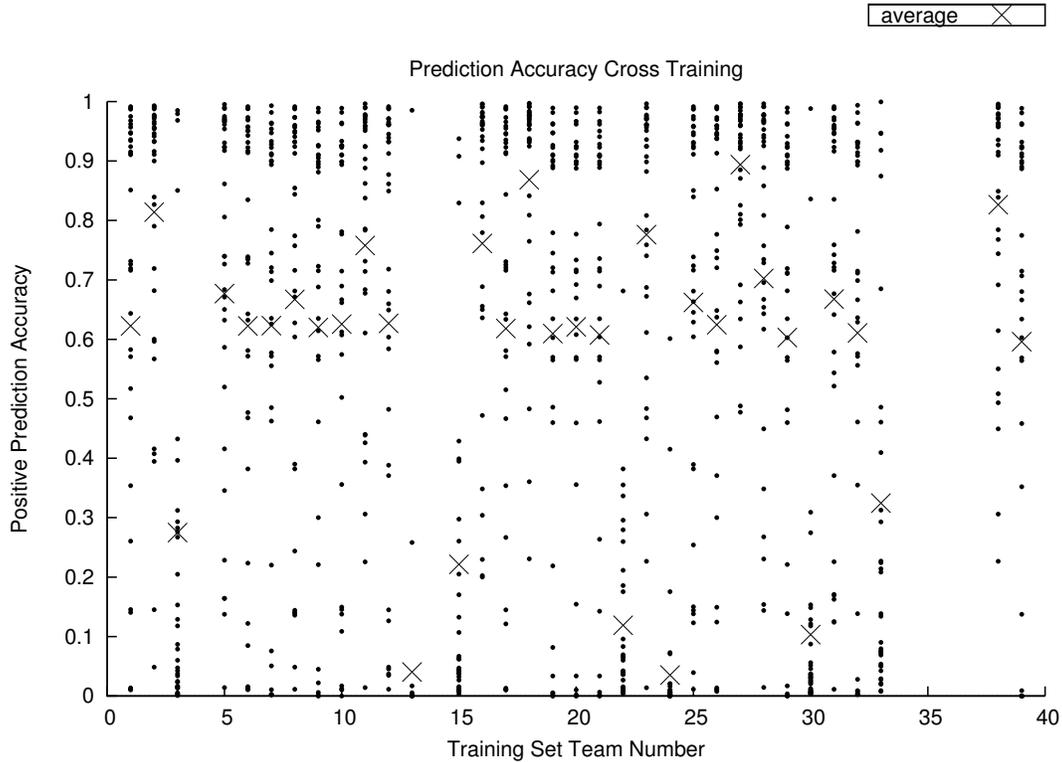


Figure 1: The prediction accuracy of *Nexat* using only one team for training.

other one for training. As can be seen, most of the teams are well connected in the graph, showing a high correlation between their actions. The teams which are less connected to the graph represent the bad predictor teams, as discussed before.

### 3.2 Experiment 2: trained by all the other teams

The results from the first experiment show how important a quality training set is to making accurate predictions. The teams whose training sets proved to be poor, likely had little to no overlap with the vast majority of teams in the competition, with respect to their attack behavior. In this experiment we try to improve the prediction accuracy of *Nexat* by using the histories of multiple hacker teams to train *Nexat*. This is unlike the first experiment where only one team's history was used for training.

Figure 3 shows the prediction accuracy of *Nexat* for each team, when the histories of all the other teams are used to train *Nexat*. In other words, each point on the graph shows the accuracy of *Nexat* in predicting the team shown in the horizontal axis, being trained by the other 32 teams. As can be seen, using the actions of multiple teams to train *Nexat* drastically improves its prediction performance. In particular, all the prediction accuracies are more than 80%, and the average prediction accuracy is 94%. This is due to the

fact that the training set has a much larger coverage of unique alerts, since it could combine the strategies of several different teams.

### 3.3 Effect of the time window on the results

As mentioned before, in the training phase of *Nexat* a time window of  $w$  is used in order to divide each complex compromise activity into distinct sessions. The value of  $w$  impacts the prediction accuracy of *Nexat* in different ways. By using a large value for  $w$ , *Nexat* can collect more of the alerts that correspond to the same complete attack, hence improving its training phase. This, however, increases the computational complexity of *Nexat* by generating more sets of alerts. Using a properly sized time window gives us a balance of prediction accuracy and computational complexity.

Figure 4 shows the average prediction accuracy of *Nexat* being trained on the history of team 25, for different values of the  $w$  parameter in the logarithmic scale. As can be seen, increasing  $w$  drastically improves the prediction accuracy for  $w \leq 10sec$ , however, it does not change significantly for the larger values of  $w$ . Considering the impact of  $w$  on the computational complexity, we choose  $w = 5$  in our implementation, as it makes a reasonable tradeoff between the complexity and the accuracy of *Nexat*.

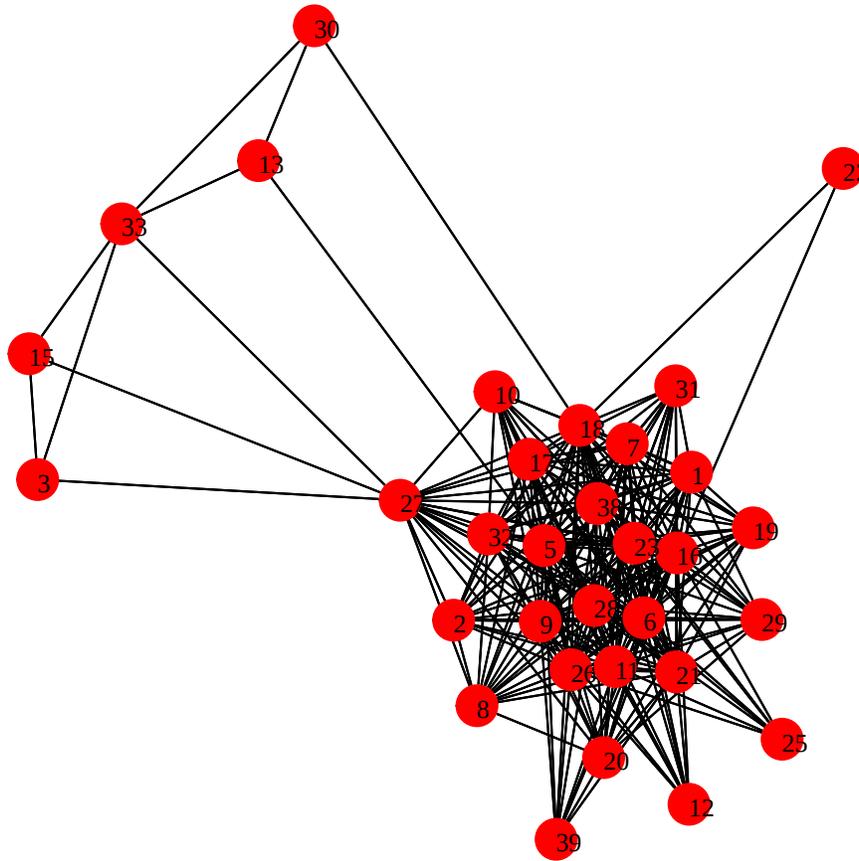


Figure 2: Clustering the hacker teams based on the prediction accuracy of *Nexat* (only one team is used for training).

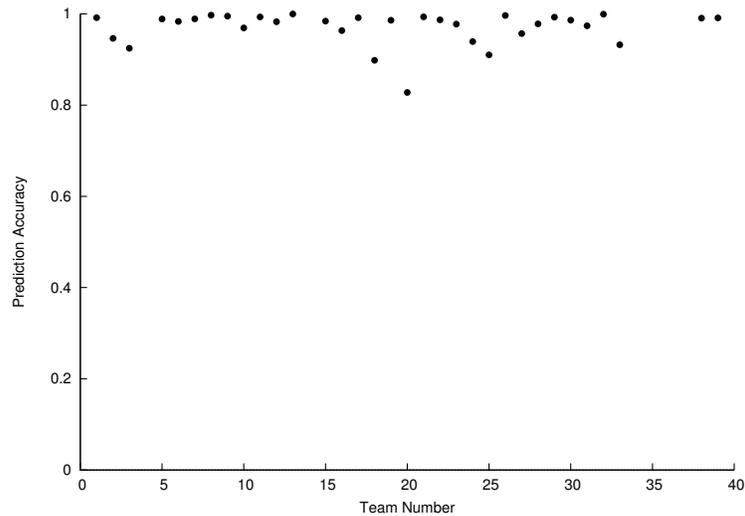


Figure 3: The prediction accuracy of *Nexat* using 32 teams for training.

### 3.4 Nexat’s resources

We ran the prototype implementation of *Nexat* on a laptop with a 2.53GHz processor, 4GB memory, and 100GB disk space. Our measurements show that *Nexat* is able to predict the attacks with

very reasonable resources, making it suitable to be deployed in real-time scenarios. *Nexat* performed the training phase for each team in about 1.7 seconds, using around 30 MB of RAM for each team. This resulted in less than one minute to train all of the com-

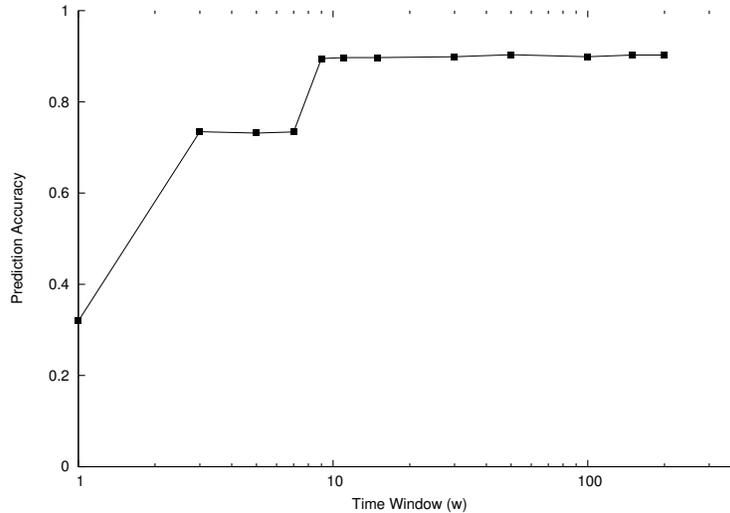


Figure 4: The effect of the time window  $w$  on the prediction accuracy.

peting teams. During the prediction phase, *Nexat* ran the data from each team against any of the profiles in about 2.4 seconds, using an average memory of 32MB. This resulted in a total prediction time of about one minute and a half for each of the teams. We expect that a customized implementation of *Nexat* on a powerful network monitoring device will result in much faster operation of *Nexat*.

#### 4. RELATED WORK

Several different approaches have been taken by researchers in order to analyze security alerts. Several researchers have considered alert correlation by grouping low-level alerts together in order to find cohesive groups or precursor of alerts [12, 11, 4, 5, 27, 16, 17]. As they rely on different levels of predefined knowledge of attack conditions and consequences, most of such approaches have limited effectiveness. In other words, they are not able to provide valid alert correlations for unknown attacks or unknown attack relations. Plan recognition has been a common approach to predicting attacker behavior [8]. Plan recognition approaches use pre-determined attacker plans in order to predict future attacks. Plan recognition approaches, however, are not scalable as they require inputs from a security professional in order to perform their predictions. This is in contrast to *Nexat*, which performs the prediction in a fully automated manner using machine learning techniques.

Alternatively, some research use statistical tools in order to analyze the security alerts. Qin et al. look for attack scenarios by correlating the alerts and finding their causality relations [21]. More specifically, they classify the alerts into a number of clusters and present each cluster as a time series of alerts. Also, the resulting alerts are ranked based on the expert knowledge. Finally, Granger causality analysis is used on the time series of the top-ranked alerts in order to compute precursor alerts. The Granger causality test gives statistical predictions on whether a time series of alerts is an adequate predictor of another time series of alerts. Unfortunately, the performance of this approach is very sensitive to the presence of unrelated background alerts. For instance, running their tool on a 5-day sample of the 2000 DARPA Intrusion Detection Scenarios dataset results in about 94% true causality detection and 14% false causality detection. The false positives are even higher for DEFCON 9 traces due to the presence of more background alerts.

Qin and Lee conduct probabilistic inference using casual networks to recognize the attack plans and make some predictions on the attacks [22]. More specifically, the authors develop a graph-based technique in order to correlate isolated attack scenarios, extracted from low-level alert correlation, based on their relations in attack plans. This is followed by probabilistic inference to evaluate the likelihood of the attack goals, hence predict potential future attacks using the generated causal networks. The authors evaluate their algorithms on the dataset of attack alerts of the DARPA's Grand Challenge Program of 2003. The paper only provides sample attack predictions with average 0.7 chance of prediction. This is in contrast to the higher accuracies of *Nexat*, as described in this section. Another main issue with this approach is that it is not scalable as the size of the attack trees and the causal networks grows very rapidly with the size of the dataset. Wang et al. also use attack graphs for analyzing the alerts and making predictions [28]. The paper differs from previous work by using a queue graph instead of the timing windows to analyze the recent alerts, requiring less memory and speeding up the attack correlation tasks. The basic queue graph approach is also extended to a unified method to identify the missing alerts and to predict future alerts. The authors provide sample successful alert prediction but point out the possibilities of false positives and false negatives in practice due to incomplete or inaccurate domain knowledge. Moreover, this approach is sensitive to the timing inaccuracies of the reporting IDS systems.

Some research use game theory in order to predict the next actions of cyber attacks [13, 9]. In order to make the prediction, the game theoretic approach requires knowledge about all of the possible actions of the attackers against the defending system in order to predict the best actions of the attackers in different scenarios. This information is not usually available to the defending systems which degrades the applicability of this approach in many scenarios.

#### 5. CONCLUSIONS

In this paper, we presented the design and implementation of *Nexat*, a tool for the prediction of attacker behavior. *Nexat* relies on machine-learning techniques to learn the attacker's behavior from previous alert histories and does not require any input from security

professionals to make predictions. Also, since *Nexat*'s prediction is not order-dependent, it can handle minor timing errors associated with the reporting from intrusion detection systems. We validate the prediction accuracy of *Nexat* through a prototype implementation. The results show that by training *Nexat* on a large database of attacker behavior it is possible to perform attack prediction with an average accuracy of 94%. Future work will include researching alternative machine learning algorithms in order to further improve the accuracy of the predictions as well as lowering the computational complexity.

## 6. ACKNOWLEDGEMENTS

This work was supported by the ARO under grant W911NF-09-1-0553, the National Science Foundation (NSF) under grants CNS-0845559 and CNS-0905537.

## 7. REFERENCES

- [1] IBM Internet Security Systems. <http://www.iss.net/>.
- [2] The 2008 UCSB International Capture The Flag (iCTF). [http://ictf.cs.ucsb.edu/archive/iCTF\\_2008/index.html](http://ictf.cs.ucsb.edu/archive/iCTF_2008/index.html), December 5th 2008.
- [3] Paul Barford, Marc Dacier, Thomas G. Dietterich, Matt Fredrikson, Jon Giffin, Sushil Jajodia, Somesh Jha, Jason Li, Peng Liu, Peng Ning, Xinming Ou, Dawn Song, Laura Strater, Vipin Swarup, George Tadda, Cliff Wang, and John Yen. Cyber SA: Situational Awareness for Cyber Defense. In Sushil Jajodia, Peng Liu, Vipin Swarup, and Cliff Wang, editors, *Cyber Situational Awareness*, volume 46 of *Advances in Information Security*, pages 3–13. Springer US, 2010.
- [4] F. Cuppens and A. Mieke. Alert Correlation in a Cooperative Intrusion Detection Framework. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2002.
- [5] H. Debar and A. Wespi. Aggregation and Correlation of Intrusion-Detection Alerts. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection*, 2001.
- [6] M. R. Endsley. Towards a Theory of Situation Awareness in Dynamic Systems. *Human Factors*, 37:32, 1995.
- [7] Mica R. Endsley. Design and Evaluation for Situation Awareness Enhancement. In *Proceedings of the Human Factors Society 32nd Annual Meeting*, volume 1 of *Aerospace Systems: Situation Awareness in Aircraft Systems*, pages 97–101, 1988.
- [8] C.W. Geib and R.P. Goldman. Plan Recognition in Intrusion Detection Systems. In *DARPA Information Survivability Conference & Exposition (DISCEX)*, 2001.
- [9] Wei Jiang, Zhi hong Tian, Hong li Zhang, and Xin fang Song. A Stochastic Game Theoretic Approach to Attack Prediction and Optimal Active Defense Strategy Decision. In *IEEE International Conference on Networking, Sensing and Control (ICNSC'08)*, pages 648–653, april 2008.
- [10] Gary Klein and Beth Crandall. Recognition-Primed Decision Strategies. Technical report ARI Research Note 96-36, United States Army Research Institute for the Behavioral and Social Sciences, April 1996. <http://handle.dtic.mil/100.2/ADA309570>.
- [11] C. Kruegel, W. Robertson, and G. Vigna. Using Alert Verification to Identify Successful Intrusion Attempts. *Practice in Information Processing and Communication (PIK)*, 27(4):219–227, October–December 2004.
- [12] C. Kruegel, F. Valeur, and G. Vigna. *Intrusion Detection and Correlation: Challenges and Solutions*, volume 14 of *Advances in Information Security*. Springer, 2005.
- [13] P. Liu and L. Li. A Game Theoretic Approach to Attack Prediction. Technical report, Penn State Cyber Security Group, 2002.
- [14] V. Mehta, C. Bartzis, H. Zhu, E. Clarke, and J. Wing. Ranking attack graphs. In *Recent Advances in Intrusion Detection*, 2006.
- [15] Sanjeeb Nanda and Narsingh Deo. The Derivation and Use of a Scalable Model for Network Attack Identification and Path Prediction. *JNW*, 3(4):64–71, 2008.
- [16] P. Ning, Y. Cui, , and D. Reeves. Analyzing Intensive Intrusion Alerts via Correlation. In *Proceedings of the International Symposium on the Recent Advances in Intrusion Detection*, 2002.
- [17] P. Ning, Y. Cui, and D. Reeves. Constructing Attack Scenarios through Correlation of Intrusion Alerts. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2002.
- [18] Steven Noel and Sushil Jajodia. Understanding complex network attack graphs through clustered adjacency matrices. In *21st Annual Computer Security Applications Conference (ACSAC 2005)*, pages 160–169. IEEE Computer Society, 2005.
- [19] V. Paxson. Bro: A system for detecting network intruders in real-time. In *Proceedings of the 7th USENIX Security Symposium*, 1998.
- [20] P. Porras, M. Fong, and A. Valdes. A Mission-Impact-Based Approach to INFOSEC Alarm Correlation. In *Proceedings of the International Symposium on the Recent Advances in Intrusion Detection*, 2002.
- [21] X. Qin and W. Lee. Statistical Causality Analysis of INFOSEC Alert Data. In *Proceedings of the Symposium on Recent Advances in Intrusion Detection (RAID)*, 2003.
- [22] X. Qin and W. Lee. Attack Plan Recognition and Prediction Using Causal Networks. In *Proceedings of the 20th Annual Computer Security Applications Conference*, 2004.
- [23] J. Rasmussen. Skills, Rules, and Knowledge; Signals, Signs and Symbols, and Other Distinctions in Humans Performance Models. *IEEE Transactions on Systems, Man and Cybernetics*, 13:257, 1983.
- [24] M. Roesch. Snort - Lightweight Intrusion Detection for Networks. In *Proceedings of the 13th Large Installation System Administration (LISA) Conference*, 1999.
- [25] J.J. Salerno, M.L. Hinman, and D.M. Boulware. A situation awareness model applied to multiple domains. In *Proceedings of SPIE*, volume 5813, pages 65–74, 2005.
- [26] G. Tadda, J.J. Salerno, D. Boulware, M. Hinman, and S. Gorton. Realizing situation awareness within a cyber environment. In *Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications 2006*, volume 6242. SPIE, 2006.
- [27] F. Valeur, G. Vigna, C. Kruegel, and R.A. Kemmerer. A Comprehensive Approach to Intrusion Detection Alert Correlation. *IEEE Transactions on Dependable and Secure Computing*, 1:146–169, 2004.
- [28] L. Wang, A. Liu, and S. Jajodia. Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts. *Computer Communications*, 29(15):2917–2933, 2006.