A Highly Scalable Model for Network Attack Identification and Path Prediction

Sanjeeb Nanda
School of Computer Science
University of Central Florida,
Orlando, Florida 32816-2362
sanjeeb@earthlink.net

Narsingh Deo
School of Computer Science
University of Central Florida,
Orlando, Florida 32816-2362
deo@cs.ucf.edu

Abstract

The rapid growth of the Internet has triggered an explosion in the number of networked applications that leverage its capabilities. Unfortunately, many of them are intentionally designed to burden or destroy the capabilities of their peers and the supporting network infrastructure. Hence, considerable effort has been focused on detecting and predicting the breaches in security propagated by these malicious applications. However, the enormity of the Internet poses a formidable challenge to representing and analyzing such attacks on it using scalable models. Furthermore, the unavailability of complete information on network vulnerabilities makes the task of forecasting the systems that are likely to be exploited by such applications in the future even harder. This paper presents a technique to identify attacks on large networks using a highly scalable model, while filtering for false positives and negatives. It also forecasts the propagation of the security failures proliferated by attacks over time and their likely targets in the future.

1. Introduction

Directed graphs serve as an intuitive way to represent network attacks. Existing formulations [9], [10], [12] use vertices to represent a tuple of attributes comprised of a source system, target system, a vulnerability that exists on the target as a precondition, and the postcondition of an atomic attack from the source to the target using that vulnerability. Then, an arc exists from one exploit to another if only if an atomic attack can leverage the postcondition in the former to utilize the precondition in the latter, and the corresponding systems are connected. To produce such attack graphs, the set of vulnerabilities at each system in the network is first obtained using scanners such as Nessus, Saint, ISS' Internet Scanner and the CISCO Security Scanner. Then, a model checker such as NuSMV or Spin is applied to a graph on all possible exploits to generate every path that breaches an explicitly stated security condition that is stated as the goal. Finally, the result produced by the model checker is rendered using a visualization application such as GraphViz. However, such attack graphs suffer from two major drawbacks. First, they have very large orders and lack scalability. As a result, they cannot be used to model and visualize attacks on networks in practice. To alleviate the severity of this problem, mechanisms have been proposed to reduce the order of such graphs [1] and to represent them succinctly to facilitate their comprehension by users [4], [6]. Second, a given graph describes the attack paths comprised of sequences of exploits that breach a target security condition, and therefore, its use is confined to the analysis of attacks pertaining to the violation of that condition alone. To address this challenge, attack graphs have been constructed using a large set of attack goals [7].

However attack graphs merely provide a roadmap of the attacks that can occur based on the vulnerabilities exposed by the systems under consideration. In isolation, they cannot establish if an exploit corresponding to an intrusion detection system (IDS) alert truly constitutes an attack. There are various reasons that can make this determination difficult. Firstly, any given network-based IDS incorporates rules that are designed to search the payload of network packets for signatures that portend threat and generates an alert when a match is found. However, such rules are independent of the potentially differing sets of vulnerabilities admitted by various hosts on that network. That is, the payload of a packet arriving at a system may ostensibly exploit a vulnerability v, but that system may not admit v. For example, the Lion worm exploits vulnerabilities in Linux only. However, a typical IDS generates an alert even when this worm attempts to exploit a Windows-based system. However, such an exploit is guaranteed to have no ill effect on a targeted Windows system nor serve as an intermediate step towards harming any peers, and is therefore defined to be irrelevant. Secondly, an event by itself may have ambiguous implications. For example, a TCP-SYN packet received by a host may be a legitimate request by a client to open a streaming connection to that host, or it may constitute a deliberate TCP-SYN flooding being carried out by that client using spoofed IP addresses to achieve denial of service (DoS). An alert generated in response to the occurrence of the former event, termed as a false positive, is undesirable since they drain network administration resources away from the investigation of meaningful events. While methods to identify false positives such as TCP-SYN flooding already exist, they are based on time-consuming heuristics that track the activity of malicious clients over periodic intervals of time [11]. Thirdly, an event may be incorrectly judged to be benign. For example, an ftp request that appears to originate from a trusted system within a firewall to a peer within it may be considered harmless, and therefore, ignored by the existing set of IDS rules. However, in reality the actual source may reside outside the firewall, and the system appearing to initiate the ftp request may have been compromised earlier to allow a port-forwarding program on it to masquerade as the source. The lack of an alert in such an instance is termed as a false negative and it poses a far greater danger to the network than false positive or irrelevant alerts. It is therefore vital to correlate alerts and events to meaningfully identify attacks.

A number of methods have been proposed to meaningfully correlate alerts with attacks. One salient approach [13] fuses together alerts that are generated within a finite window of time and possess common values for a specified set of attributes to form a meta-alert. However, the drawback of this technique is that it can fuse together uncorrelated alerts when they have common attributes such as source and destination addresses, and can ignore related alerts that are spaced farther apart in time. Furthermore, it fuses meta-alerts constituting multi-step attacks in a similar manner with the same deficiencies. Another technique [7] creates paths comprised of observed events such that, each contains events whose corresponding exploits in the attack graph have a distance less than a specific threshold to other exploits whose corresponding events lie on the same path. The relevancy of a path to an attack is then obtained by first applying a moving average filter to the distances between the adjacent events in each path, and then calculating the ratio of the number of events in a given path to the sum of the filtered distances between all pairs of events in that path. The drawback of this scheme is that, it uses conventional non-aggregated attack graphs, which as stated earlier, have extremely large orders, which makes distance calculation between alerts computationally nontrivial.

Several methods have also been proposed to estimate the likelihood of an attack with specified goals. One prominent scheme [2] accomplishes this by modeling attacks using trees where each vertex represents a network/attacker state and each edge represents an exploit that leverages a non-unique vulnerability with a given probability. The probability of the goal state is defined to be the sum of the probabilities of each non-intersecting path that has the vertex representing that goal state as an end, where the probability of each path is the product of the probabilities of the edges comprising it. However, it does not address the mechanism by which the probability of an exploit can be derived. Another technique [5] computes the closure of the adjacency matrix A of the vertices of an attack graph, where A^k shows the clusters of systems that are at risk from one another after k attack steps. However, this indicates the reachability of an attack initiated from a given system to others in the network and not necessarily the path that is taken by an attack.

2. Description of the Model

Unlike all previously mentioned schemes, our model decouples attack graphs from the networked systems on which they are described. The motivation for this approach is as follows. By definition, each system is reachable from every other in a network. Hence, to execute the sequence of exploits needed to achieve its goal, a malware relies only on the existence of the preconditions necessary to execute each, regardless of which system provides each precondition. We then define an *attack graph* unique to an attack goal and an *attack path* in the following manner.

Definition 1. An *attack graph* is a directed graph $G(V, A)$ where each vertex v in V is a vulnerability and each arc (u, v) in A represents an action a originating from a system having vulnerability u that has been exploited to yield the condition necessary for a to exploit a system with vulnerability v. A unique vertex v_f in V represents the *end vulnerability* that can be exploited by the attack to realize the postcondition that immediately satisfies its goal. Thus an attack path is a unique path in G that contains v_f as its end.

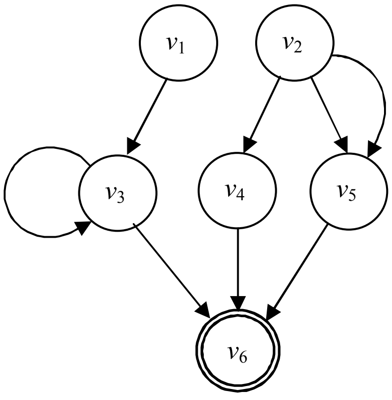

Figure 1. An attack graph with attack paths

The definition of an attack graph does not constrain an action to be atomic. An action a may be comprised of a sequence of sub-actions, a_1, a_2, …, a_m, where the postcondition resulting from the application of sub-action a_i is the precondition necessary for the application of sub-action a_{i+1}, where $1 \leq i \leq m - 1$. Figure 1 illustrates an example of an attack graph, where vertices v_1 and v_2 – each having in-degree of 0 – are starting points for an attack, whose goal is achieved by exploiting the vertex v_6. In this graph, the sequences of vulnerabilities v_1, v_3^+, v_6, and v_2, v_4, v_6, and v_2, v_5, v_6 represent attack paths, where v^+ denotes one or more occurrences of v. Each has vulnerability v_6 as its end, which can be exploited to yield the postcondition that satisfies the goal of the attack. While each of the paths described here have length three, it is possible for them to be larger. Note that, an attack graph admits self-loops and arcs having the same tail and head.

We now describe how the given model of an attack graph can be used to determine the likelihood of an alert corresponding to a multi-step attack. Suppose an IDS generates an alert corresponding to an exploit originating from system s_2 targeting vulnerability v_1 at time t_1 on system s_1. Then, we determine if s_1 admits v_1. If not, we discard that alert as irrelevant, else we find a predecessor of v_1 in a chosen attack graph G, v_2 say, and determine if s_2 admits v_2. If true, we determine if an alert has been seen at some time $t_2 < t_1$ for an exploit targeting v_2 on s_2. Suppose this is true, and the system from which that exploit originates is s_3. Then we find a predecessor of v_2 in G, v_3 say, and determine if s_3 admits v_3. If true, we again determine if an alert has been seen at some time $t_3 < t_2$ for an exploit targeting v_3 on s_3. We continue backtracking in this manner until one of the following three cases occurs.

(a) Upon examining an alert generated at time t_{k-1} corresponding to an exploit from system s_k targeting vulnerability v_{k-1} on system s_{k-1}, we backtrack to the predecessor of v_{k-1} in the chosen attack graph, v_k say, and find that s_k admits v_k, but v_k has no predecessor.

(b) After inspecting the alert corresponding to an exploit from system s_{k+1} targeting vulnerability v_k on system s_k, we backtrack to the predecessor of vulnerability v_k in the chosen attack graph, v_{k+1} say, and find that v_{k+1} is not admitted by s_{k+1}. There are three possible scenarios that can cause this:

  (b.1) s_{k+1} already admits the postcondition necessary for an exploit to target v_k on s_k. This would imply that the exploits targeting the sequence of vulnerabilities v_k, v_{k-1}, …, v_1 represent a legitimate multi-step attack targeting system s_1.

  (b.2) The chosen attack graph may not correspond to the attack that is ostensibly occurring.

  (b.3) The attack may be exploiting an unknown vulnerability on s_{k+1} to exploit the sequence of vulnerabilities v_k, v_{k-1}, …, v_1 thereafter.

(c) Upon examining an alert generated at time t_{k-1} for an exploit from system s_k targeting vulnerability v_{k-1} on system s_{k-1}, we backtrack to the predecessor of v_{k-1} in the chosen attack graph, v_k say, and find that s_k admits v_k. However, there is no alert originating from any system that targets v_k on s_k at time $t_k < t_{k-1}$.

3. Method to Correlate an Alert to an Attack

Using the previously described model of an attack graph, we can correlate an alert to an attack in the following manner. If (a) holds true and $v_1 = v_f$ in the attack graph being used, then we have found a complete sequence of exploits targeting each vulnerability in an attack path with v_f at its end. We refer to such a path as a *complete attack path* and its length as the *complete attack length.* It would then be intuitive to assume that such an occurrence indicates a legitimate multi-step attack that targets system s_1. In the event (b.2) holds true, we can substitute the attack graph being currently used with another that contains vulnerability v_1 and repeat the process of backtracking along the new graph to see if (a) or (b.1) holds. We may obtain such a graph after a number of substitutions. However, if no such attack graph is found, then we have to reconcile the matching of partial sequences of vulnerabilities v_k, v_{k-1}, …, v_1 for each examined attack graph with the likelihood that it represents a real attack. It is intuitively appealing that the greater the length of the sequence v_k, v_{k-1}, …, v_1 that is matched, the greater the likelihood that the alert targeting vulnerability v_1 corresponds to a legitimate multi-step attack. We refer to the value of k in this *matching attack subpath* as the *matching attack length*, and its difference from the complete attack length as the *remaining attack length*. Furthermore, if u_i, u_{i-1}, …, u_1 and v_j, v_{j-1}, …, v_1 are the partial sequence of vulnerabilities that are matched by employing the attack graphs G and H respectively, with $i > j$, then it is again intuitive to assume that G is the attack graph that best describes the perceived attack. Finally, (b.3) and (c) may be caused by the exploit on s_k being stealthy, or the IDS rules not being adequately comprehensive to detect such an exploit.

Algorithm 1 in Figure 3 uses the aforementioned model to determine if the alert corresponding to an exploit constitutes an attack. Step 1 is executed once for a given network, while Step 2 is executed each time an alert is generated by an IDS thereafter. If the alert is correlated to an attack, Step 2 returns the sequence of 2-tuples comprised of the systems and their corresponding vulnerabilities that have been iteratively targeted by that attack. Such a sequence forms an attack instance as defined next.

Definition 2. A *network* is a graph $G(U, E)$ where each vertex s in U is a system and each edge (u, v) in E represents a link between u and v. Let $H(V, A)$ be an attack graph. Given an attack path in H defined by the sequence of vulnerabilities v_1, v_2, …, v_k, an *attack instance* is the sequence of tuples (s_1, v_1), (s_2, v_2), …, (s_i, v_i) where $i \leq k$ and systems s_1, s_2, …, s_i admit v_1, v_2, …, v_i respectively as a matching attack subpath.

Figure 2 illustrates an example of an attack instance comprised of the tuples (s_a, v_a), (s_b, v_b), (s_c, v_b), and (s_d, v_c). In this instance, systems s_a, s_b, s_c, s_d admit an attack path comprised of the attack path v_a, v_b, v_b, v_c. The correlation factor *corr* in Algorithm 1 is defined such that it is well behaved. That is, it ensures that correlation is proportional to the ratio of the matching attack length to the complete attack length. At the same time, it is inversely proportional to the remaining attack length. This enables the correlation value of an attack to be proportional to the proximity of v_f to the vulnerability targeted by its most recently observed exploit. Finally, the threshold τ is a suitable real value in (1.0, 2.0).

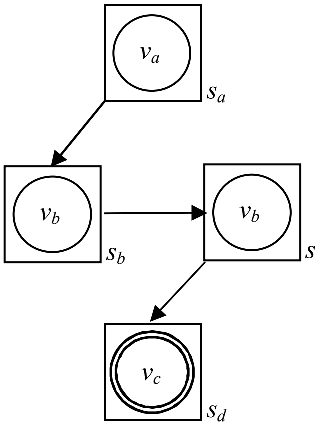

Figure 2. An attack instance with an attack subpath

Algorithm 1.

```
$Correlate$(vulnerability $v$, system $s$, time $t$, int $matchLen$)
{
    if ($v \neq null$) and ($s$ admits $v$) and (alert found for exploit
        targeting $v$ on $s$ at time $t_v < t$)
    {
        $e \leftarrow$ exploit at time $t_v$ for which alert is found;
        $t \leftarrow t_v$;   $P \leftarrow P + (v, s)$;   $s \leftarrow$ origin system of $e$;
        $matchLen \leftarrow matchLen + 1$;

        if ($v$ has predecessors in $G_i$)
        {
            for each predecessor $u$ of $v$ in $G_i$ do
            { $Correlate(u, s, t, matchLen)$; }
        }
        else { $Correlate(null, s, t, matchLen)$; }
    }
    else
    {   $corr \leftarrow \frac{matchLen}{matchLen + d(w, w_f)} + \frac{1}{d(w, w_f) + 1}$;
        if ($corr > maxCorr$)
        { $maxCorr \leftarrow corr$;   $P_{max} \leftarrow P$; }
    }
}
```

Step 1

```
$S$ = Systems in a subnet;
$V$ = Set of vulnerabilities admitted by all systems in $S$;
$n$ = Size of the set of attack goals admitted by $V$;

for attack goal $i \leftarrow 1$ to $n$ generate attack graph $G_i$ on $V$;
```

Step 2

```
$e \leftarrow$ exploit at time $t$ for which alert is raised;
$s \leftarrow$ system targeted by $e$;   $complete \leftarrow false$;
$w \leftarrow$ vulnerability targeted by $e$;
$P_{max} \leftarrow \phi$;   $maxCorr \leftarrow 0$;   $i \leftarrow 1$;   $t \leftarrow \infty$;

while ($i \leq n$)
{
    $d(w, w_f) \leftarrow$ distance of $w$ to end vulnerability $w_f$ in $G_i$;
    $P \leftarrow \phi$; $Correlate(w, s, t, -1)$;
}

if ($maxCorr > \tau$) return ($P_{max}$, $i$);
else return $null$;
```

Figure 3. Algorithm to identify an attack from an alert

Execution Step 2 of Algorithm 1 returns the tuple (P, i) if an attack instance P using the attack graph having index i is correlated to the investigated alert. Now, suppose that it returns (P_1, i_1) in response to an alert for an exploit e_1 at time t_1, and (P_2, i_2) in response to an alert for an exploit e_2 at time t_2, where $t_2 > t_1$. If $i_1 \neq i_2$ and P_1 is a subsequence of P_2, then P_2 must represent an attack that is different from P_1, since by Definition 1, each attack graph has a unique goal. On the other hand, if $i_1 = i_2$ then we obtain vindication of the algorithm's earlier result that suggested an attack with the goal of the attack graph having index i_1.

The depths of attack graphs are generally small since the average length of a sequence of vulnerabilities exploited to achieve a desired security breach is small. Furthermore, the system that is examined for the predecessor of a given vulnerability v is specified by the alert for the exploit targeting v, and therefore, its discovery is achieved in constant time using suitable data structures such as a hash table. Thus, the order of Algorithm 1 is dominated by the number of attack graphs examined. If their number equals n, then the algorithm has order $\mathbf{O}(n)$. Moreover, since the value of n is independent of the size of the network, the proposed model with its associated algorithm to identify attacks is very scalable.

Algorithm 1 identifies attack instances on a subnet when the vulnerabilities on all systems have been identified in Step 1. However, it does not identify the systems that are future targets of exploits by an identified attack. For example, suppose that (s_1, v_1), (s_2, v_2), …, (s_j, v_j) is an identified attack instance using the attack graph with the final goal vulnerability v_f, where systems s_1, s_2, …, s_j admit vulnerabilities v_1, v_2, …, v_j respectively, and $v_j \neq v_f$. Then Algorithm 1 does not identify the systems that are likely to be targeted next by that attack to realize its goal of exploiting v_f. For instance, the attack may next target systems s_a and s_b that admit vulnerabilities v_i and v_{j+1} using exploits from systems s_{i-1} and s_j respectively, for some $1 \leq i < j$, as shown in Figure 4. Next, we describe the method to identify such systems.

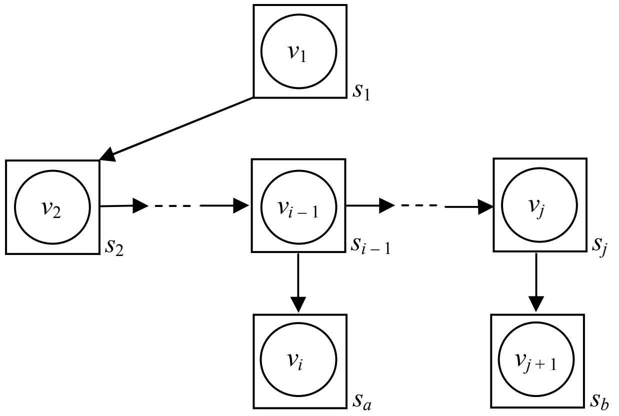

Figure 4. Future exploit targets of an identified attack

4. Predicting Future Targets of Attacks

To determine the systems that are likely to be targeted by an attack in the future utilizes the idea of matching the remaining attack length of an attack path in the direction of the goal. As described earlier, Algorithm 1 returns the index i of the attack graph corresponding to the matched attack instance. Now, if the remaining attack length is zero, then the system specified in the first tuple (v_1, s_1) of an attack instance is the system targeted by the attack. On the other hand, if the remaining attack length is nonzero, d say, then we iteratively find the sequence of systems s_{k+1}, s_{k+2}, … s_{k+d} such that, system s_j admits vulnerability v_j, for $k + 1 \leq j \leq k + d$, v_{j+1} is a child of v_j in the attack graph, v_{k+1} is the child of v_1, and $v_{k+d} = v_f$. However, in some iteration j, multiple systems may admit the vulnerability v_j that is a descendant of v_1. Therefore, to definitively identify the next system that is likely to be targeted, we determine that malware's past preferences when selecting a system to exploit under such circumstances.

Consider a malware that has just compromised system s within a private network by exploiting vulnerability u on it, and whose eventual goal is to exploit vulnerability v_f Furthermore, suppose that (u, v) is an arc in the attack graph with goal v_f. Since a malware propagates itself in the manner dictated by the flow of its coded logic, it can be expected to exhibit predictable traits in choosing the next system to exploit in the set of systems that all admit vulnerability v. For instance, that malware may be designed to next exploit, from s, the system that admits vulnerability v and belongs to the same private network as s. On the other hand, if s is a router, then that malware may be designed to immediately exploit another router that admits v. Similarly, it may choose the host having the lowest version of the operating system that admits vulnerability v. We now describe how such preferences are measured.

First, we define a *system state* as a tuple $z = (v, x_1, x_2, \ldots, x_m)$ in $V \times X_1 \times X_2 \times \ldots \times X_m$ of $m + 1$ independent variables that describes the state of a system, where V is the set of all vulnerabilities and X_i is the domain of characteristic x_i, $1 \leq i \leq m$, for some integer value m. For example, we may have $X_1 = \{a_{11}, a_{12}, a_{13}\}$, where a_{11} denotes a system having a private class C IP address of the form 192.168.b_1.b_2, with $0 \leq b_1, b_2 \leq 255$; a_{12} represents a system having a private class B IP address of the form 172.b_1.b_2.b_3, with $16 \leq b_1 \leq 31$ and $0 \leq b_2, b_3 \leq 255$; and a_{13} represents a system having a private class A IP address of the form 10.b_1.b_2.b_3, with $0 \leq b_1, b_2, b_3 \leq 255$. Similarly, we may have $X_2 = \{a_{21}, a_{22}, \ldots, a_{2r}\}$ as the set of r unique combinations of operating systems and version numbers that indicate their respective service packs or patches. Also, we may have $X_3 = \{a_{31}, a_{32}\}$, where a_{31} denotes a system that is a router and a_{32} denotes otherwise. In this manner, we identify $x_1, x_2, \ldots, x_m$ while ensuring that their respective domains do not omit any values that characterizes a system.

Since each variable comprising a system state has a finite domain, we must have a finite number of such states. Then, the conditional probability of an attack being in the system state z_i when the preceding system state is z_j, denoted as $p(z_i \mid z_j)$, may be derived in the following manner. For each pair of system states z_i and z_j we determine over all attack instances of the form (v_1, s_1), (v_2, s_2), …, (v_k, s_k), $k > 0$, the number of occurrences c_j where a system s_t is in state z_j, and the number of occurrences $c_{i,j}$ where a system s_t is in state z_j and system s_{t+1} is in state z_i, for $t < k$. Then, $p(z_i \mid z_j) = c_{i,j} / c_j$. We may represent the probabilities of such pairs of states using a matrix $P = (p)_{ij}$, where entry (i, j) equals $p(z_i \mid z_j)$. Now, given any system, its state must be a tuple in $V \times X_1 \times X_2 \times \ldots \times X_m$, since by construction, each domain is chosen such that each and every system is characterized by a unique value within it. As a result, a system's state is also unique. Hence a given pair of systems r and s have a unique pair of corresponding states z_r and z_s respectively. Then the probability of an attack exploiting vulnerability v on s from r, using its prior exploitation of vulnerability u on r is given by the conditional probability $p(z_s \mid z_r)$. For simplicity of notation we denote this simply as $p(s, r)$.

Algorithm 2 utilizes the function *FindTarget* to perform a depth-first search for the targeted system. In each stage of recursion, it searches the branches from a system r to each system s in decreasing order of the conditional probability value $p(s, r)$, where s admits the next vulnerability in the attack path towards v_f following that admitted by r. If a target is not found by proceeding down a branch, *FindTarget* backtracks to r and resumes its search on the branch having the next lower conditional probability value. It continues in this manner until a branch returns a target system, or all branches have been exhausted.

Algorithm 2.

```
FindTarget(system $s$, set of systems $R$, vulnerability $u$)
{
    if ($u = v_f$) { return $s$; }
    else
    {   $R \leftarrow R - s$;   $target \leftarrow null$;
        for each successor $v$ of $u$ in attack graph $G_i$ do
        {
            Let $R_v = \{r_1, \ldots, r_j\}$ be the set of systems in $R$
            that admit $v$, with $p(r_i, s) \geq p(r_{i+1}, s)$, $1 \leq i < j$;

            if ($R_v \neq$ empty and $target = null$)
            {   $i \leftarrow 1$;
                while ($target = null$ and $i < j$) do
                {   $target \leftarrow FindTarget(r_i, R_v, v)$;
                    $i \leftarrow i + 1$;
                }
            }
        }
    }

    return $target$;
}
```

Execution Step

```
$S$ = Systems in a subnet;   $R \leftarrow S$;
Suppose Algorithm 1 returns ($P$, $i$);
Let $P = (v_1, s_1), (v_2, s_2), \ldots, (v_k, s_k)$;

for $j \leftarrow 1$ to $k$
{ $R \leftarrow R - s_j$, where $s_j$ is a member of a tuple in $P$; }

$FindTarget(s_k, R, v_k)$;
```

Figure 5. Algorithm to identify the target of an attack

5. Conclusion

Our model for representing attack graphs is highly scalable and the given algorithms that use it to identify attacks and predict their future targets, are computationally efficient. However, they rely on the ability of the IDS to detect each exploit, and therefore, they are limited to detecting attacks that exhibit well-known exploits. Extending our model and algorithms to detect new types of attacks is an area of future research.

6. References

[1] P. Ammann, D. Wijesekera and S. Kaushik, "Scalable graph-based network vulnerability analysis," *Proceedings of the* 9th *ACM Conference on Computer and Communications Security*, pp. 217 – 224, 2002.

[2] J. Dawkins and J. Hale, "A systematic approach to multi-stage network attack analysis," *Proceedings of the Second IEEE International Information Assurance Workshop*, pp. 48 – 56, 2004.

[3] S. Mylavarapu, J. Zachary, D. Ettlich, J. McEachen and D. Ford, "A model of conversation exchange dynamics for detection," *The* 47th *Midwest Symposium on Circuits and Systems*, vol. 3, pp. 231 – 234, 2004.

[4] S. Noel, M. Jacobs, P. Kalapa and S. Jajodia, "Multiple coordinated views for network attack graphs," *Proceedings of the IEEE Workshop on Visualization for Computer Security*, pp. 99 – 106, 2005.

[5] S. Noel and S. Jajodia, "Understanding complex network attack graphs through clustered adjacency matrices," *Proceedings of the* 21st *Annual Computer Security Applications Conference*, pp. 160 – 169, 2005.

[6] S. Noel and S. Jajodia, "Managing attack graph complexity through visual hierarchical aggregation," *Proceedings of the* 2004 *ACM workshop on Visualization and data mining for computer security*, pp. 109 – 118, 2004.

[7] S. Noel, E. Robertson and S. Jojodia, "Correlating intrusion events and building attack scenarios through attack graph distances," *Computer Security Applications Conference*, pp. 350 – 359, 2004.

[8] S. Patton, W. Yurcik and D. Doss, "An Achilles' heel in signature based IDS: squealing false positives in Snort," *Fourth International Symposium on Recent Advances in Intrusion Detection*, 2001.

[9] C. Phillips and L. Swiler, "A graph-based system for network-vulnerability analysis," *Proceedings of the* 1998 *Workshop on New Security Paradigms*, pp. 71 – 79, 1998.

[10] R. Ritchey and P. Ammann, "Using model checking to analyze network vulnerabilities," *Proceedings of the* 2000 *IEEE Symposium on Security and Privacy*, pp. 156 – 165, 2000.

[11] K. Shah, S. Bohacek and A. Broido, "Feasibility of detecting TCP-SYN scanning at a backbone router," *Proceedings of the* 2004 *American Control Conference*, vol. 2, pp. 988 – 995, 2004.

[12] O. Sheyner, J. Haines, S. Jha, R. Lippmann and J. Wing, "Automated generation and analysis of attack graphs," *Proceedings of the* 2002 *IEEE Symposium in Security and Privacy*, pp. 272 – 284, 2002.

[13] F. Valeur, G. Vigna, C. Kruegel and R. Kemmerer, "A comprehensive approach to intrusion detection alert correlation," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 3, pp. 146 – 169, 2004.

[14] Y-M. Wang, Z-L. Liu, X-Y. Cheng and K-J. Zhang , "An analysis approach for multi-stage network attacks," *Proceedings of* 2005 *International Conference on Machine Learning and Cybernetics*, vol. 7, pp. 3949 – 3954, 2005.