An Ant Colony Optimization Algorithm for Network Vulnerability Analysis

M. Abadi* and S. Jalili**

Abstract: Intruders often combine exploits against multiple vulnerabilities in order to break into the system. Each attack scenario is a sequence of exploits launched by an intruder that leads to an undesirable state such as access to a database, service disruption, etc. The collection of possible attack scenarios in a computer network can be represented by a directed graph, called network attack graph (NAG). The aim of minimization analysis of network attack graphs is to find a minimum critical set of exploits that completely disconnect the initial nodes and the goal nodes of the graph. In this paper, we present an ant colony optimization algorithm, called AntNAG, for minimization analysis of large-scale network attack graphs. Each ant constructs a critical set of exploits. A local search heuristic has been used to improve the overall performance of the algorithm. The aim is to find a minimum critical set of exploits that must be prevented to guarantee no attack scenario is possible. We compare the performance of the AntNAG with a greedy algorithm for minimization analysis of several large-scale network attack graphs. The results of the experiments show that the AntNAG can be successfully used for minimization analysis of large-scale network attack graphs.

Keywords: Ant Colony Optimization, Metaheuristic, Network Attack Graph, Network Vulnerability Analysis.

1 Introduction

Our society has become increasingly dependent on computer networks and the trend towards larger networks will continue. Each network host runs different software packages and supports several modes of connectivity. Despite the best efforts of software architects and developers, network hosts inevitably contain a number of vulnerabilities. Hence, it is not feasible for a network administrator to remove all vulnerabilities present in the network hosts. Therefore, the recent focus in security of such networks is on analysis of vulnerabilities globally, finding exploits that are more critical, and preventing them to thwart an intruder.

When evaluating the security of a network, it is rarely enough to consider the presence or absence of isolated vulnerabilities [1]. This is because intruders often combine exploits against multiple vulnerabilities in order to reach their goals. For example, an intruder might exploit the vulnerability of a particular version of ftp to overwrite the .rhosts file on a victim host. In the next step, the intruder could remotely log in to the victim. In a subsequent step, the intruder could use the victim host as a base to launch another exploit on a new victim, and so on.

Dacier *et al.* [2] propose the concept of privilege graphs. Each node in the privilege graph represents a set of privileges owned by a user or a set of users. Edges represent vulnerabilities that can be exploited. Privilege graphs are then explored to construct attack state graphs, which represent different ways in which an intruder can reach a certain goal, such as root privilege on a host.

Phillips and Swiler [3] propose the concept of attack graphs in a more general way, where each node represents a possible attack state. Edges represent a change of state caused by a single action taken by the intruder.

Sheyner *et al.* [4] use a modified version of the model checker NuSMV [5] to produce attack graphs.

Ammann *et al.* [6] present a scalable attack graph representation. These attack graphs are essentially similar to [3], where any path in the graph from an initial node to a goal node shows a sequence of exploits that an intruder can launch to reach his goal.

Noel *et al.* [7, 8] present a number of techniques for managing network attack graph complexity through visualization.

Iranian Journal of Electrical & Electronic Engineering, 2006.

* M. Abadi is a PhD student at the Department of Computer Engineering, Tarbiat Modares University, Tehran 14115-143, Iran.
** S. Jalili is with the Department of Computer Engineering, Tarbiat Modares University, Tehran 14115-143, Iran.
E-mail: abadi@modares.ac.ir, sjalili@modares.ac.ir.

The aim of minimization analysis of network attack graphs is to find a minimum critical set of exploits that must be prevented to guarantee no attack scenario is possible. Sheyner *et al.* [4] and Jha *et al.* [9, 10] show this problem is in fact *NP*-hard. They propose a greedy algorithm that can find an approximately-optimal set of exploits, which must be prevented to thwart an intruder.

Ant Colony Optimization (ACO) [11, 12] is a metaheuristic method that is inspired by the behavior of real ants. The underlying idea is that by using very simple means of communications, a group of ants is able to find shortest paths between the nest and the food sources [13]. Along the way, ants leave a chemical substance called *pheromone*. If no pheromone trails are available, ants move essentially at random, but in the presence of pheromone, they have a tendency to follow the trail. In fact, ants probabilistically prefer paths that are marked by strong pheromone concentrations. Choices between different paths occur when several paths intersect. Then, ants choose the path to follow by a probabilistic decision biased by the amount of pheromone. Because ants in turn leave pheromone on the path they are following, this behavior results in a self-reinforcing process leading to the formation of paths marked by strong pheromone concentrations [14]. This behavior also enables ants to find shortest paths between the nest and the food sources.

ACO has been successfully applied to a large number of combinatorial optimization problems such as the traveling salesman problem [15], scheduling problems [16], and routing problems in telecommunication networks [17].

While it is currently possible to generate very large and complex network attack graphs, relatively little work has been done to analyze them.

In this paper, we present an ant colony optimization algorithm, called AntNAG, for minimization analysis of large-scale network attack graphs. A local search heuristic has been used to improve the overall performance of the algorithm. The aim is to find a minimum critical set of exploits that completely disconnect the initial nodes and the goal nodes of the graph. We also compare the performance of this algorithm with the greedy algorithm proposed by Sheyner *et al.* [4] and Jha *et al.* [9, 10] for minimization analysis of a sample network attack graph and several large-scale network attack graphs.

The remainder of this paper is organized as follows: Section 2 introduces our network security model. Section 3 describes network attack graphs. Section 4 presents AntNAG, an ant colony optimization algorithm for minimization analysis of large-scale network attack graphs. Section 5 reports the experimental results. Section 6 discusses the time complexity of the AntNAG with and without the local search heuristic, and finally Section 7 draws some conclusions.

2 Network Security Model

Our network security model is a tuple (S, H, N_c, T, E, R), where S is a set of services, H is a set of network hosts, N_c is a relation expressing connectivities between network hosts, T is a relation expressing trust relationships between network hosts, E is a set of individual known exploits that intruder can use to construct attack scenarios, and R is a model of intruder.

Services

Each service $s \in S$ is a pair (svn, p), where svn is the service name and p is the port on which the service is listening.

Network Hosts

Each network host $h \in H$ is a tuple (id, $svcs$, $plvl$, $vuls$), where id is the unique host identifier, $svcs$ is a set of services running on the host, $plvl$ is the level of privilege that the intruder has on the host, and $vuls$ is a set of vulnerable components available on the host.

Network Connectivities

Network connectivities are modeled as a relation $N_c \subseteq H \times H \times P$, where P is a set of port numbers. Each network connectivity $c \in N_c$ is a triple (h_s, h_t, p), where h_s is the source network host, h_t is the target network host, and p is the target port number. It is important to note that the connectivity relation incorporates the network elements such as firewalls that restrict the ability of one host to connect to another.

Trust Relationships

Trust relationships are modeled as a relation $T \subseteq H \times H$, where $T(h_t, h_s)$ indicates that a user can log in from the network host h_s to the network host h_t without authentication.

Exploits

Each exploit $e \in E$ is a tuple (pre, h_s, h_t, $post$), where pre is a list of conditions that must hold before launching the exploit, h_s is the network host from which the exploit is launched, h_t is the network host targeted by the exploit, and $post$ specifies the effects of exploit on the network.

To prevent an exploit, the security analyst may change the firewall configuration or patch the vulnerabilities that made this exploit possible. An exploit $e \in E$ is *inevitable* if its prevention is not feasible or incurs high cost. The set of inevitable exploits is denoted by I.

Intruder

The intruder has some information about the target network, such as known vulnerabilities, user passwords, etc.

3 Minimization Analysis

Let E be the set of exploits. A network attack graph is a tuple $G = (V, A, V_0, V_f, L)$, where V is the set of nodes, A is the set of directed edges, $V_0 \subseteq V$ is the set of initial nodes, $V_f \subseteq V$ is the set of goal nodes, and $L : A \to E$ is a labeling function where $L(a) = e$ if and only if an edge $a = (v, v')$ corresponds to an exploit e. A path π in G is a sequence of nodes $v_1, v_2, ..., v_m$, such that $v_i \in V$ and $(v_i, v_{i+1}) \in A$, where $1 \le i < m$. The label of path π is a subset of the set of exploits E. Each attack scenario corresponds to a complete path that starts from an initial node and ends in a goal node.

Let $E = \{e_1, e_2, ..., e_n\}$ be the set of exploits, I be the set of inevitable exploits, and $S = \{S_1, S_2, ..., S_l\}$ be the set of attack scenarios represented by the network attack graph G. The attack scenario $S_j \in S$ is hit by the exploit $e_i \in E$ if $e_i \in S_j$.

For each exploit $e_i \in E$, we define the *total hit value* $hv_t(e_i)$ to be the number of attack scenarios that are hit by e_i.

$$hv_t(e_i) = \left|\{S_j \in S \mid e_i \in S_j\}\right| \tag{1}$$

Let $U \subseteq E$ be a subset of exploits and $hs(U)$ be the set of attack scenarios hit by some exploits of U.

$$hs(U) = \{S_j \in S \mid e_i \in S_j \text{ for some } e_i \in U\} \tag{2}$$

An exploit e_i is *redundant* with respect to U if $hs(U \setminus \{e_i\}) = hs(U)$.

A subset of exploits $C \subseteq E \setminus I$ is *critical* if and only if all attack scenarios are hit by some exploits of it. Equivalently, C is critical if and only if every complete path from an initial node to a goal node of the network attack graph has at least one edge labeled with an exploit $e_i \in C$. A critical set of exploits is *minimal* if it contains no redundant exploit.

A critical set of exploits C is *minimum* if there is no critical set of exploits C' such that $|C'| < |C|$. In general, there can be multiple minimum critical set of exploits. We define the cardinality of a critical set of exploits C to be the number of exploits of C.

A typical process for finding a minimum critical set of exploits is shown in Fig. 1.

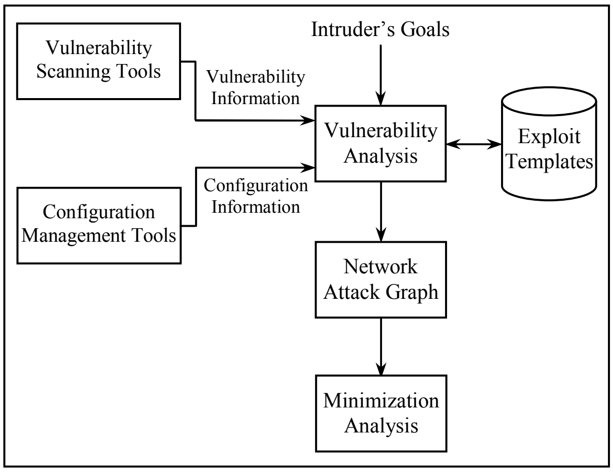

Fig. 1 Minimization analysis of network attack graphs.

First, vulnerability scanning tools, such as Nessus [18], determine vulnerabilities of individual network hosts. Using this vulnerability information along with exploit templates, intruder's goals, and other information about the network, such as connectivity between network hosts, a network attack graph is generated. In this directed graph, each complete path from an initial node to a goal node corresponds to an attack scenario. The minimization analysis of the network attack graph determines a minimum critical set of exploits that must be prevented to guarantee no attack scenario is possible.

4 AntNAG

In this section, we present AntNAG, an ant colony optimization algorithm for minimization analysis of large-scale network attack graphs.

Each ant incrementally constructs a critical set of exploits. To each exploit e_i is associated a pheromone trail τ_i that indicates the desirability of including that exploit into an ant's solution.

Fig. 2 shows the pseudo-code of the AntNAG algorithm. The first step is to set parameters and initialize pheromone trails. Then repeated iterations of the algorithm are run until some termination condition is met (e.g., a maximum number of iterations is reached). Within each iteration, each ant starts with an empty set and constructs a critical set of exploits by incrementally adding exploits until all attack scenarios are hit. The critical sets of exploits constructed by ants may contain redundant exploits, which are eliminated. After that, the iteration-best solution is improved by a local search heuristic. Finally, the pheromone trails are updated using a global updating rule.

4.1 Solution Construction

Each ant incrementally constructs a critical set of exploits using the algorithm shown in Fig. 3. At each construction step during iteration t, each ant k chooses an exploit from the set of preventable exploits to add to the partial solution $C^k(t)$. The probability with which ant k chooses an exploit e_i is as follows [12]:

procedure AntNAG
    Set parameters, initialize pheromone trails;
    while termination condition not met **do**
        for each ant k **do**
            Construct a critical set of exploits $C^k(t)$;
            Eliminate redundant exploits of $C^k(t)$;
        end for;
        Apply local search heuristic to the iteration-best solution $C^{ib}(t)$;
        Apply global pheromone trail update;
    end while;
end procedure

Fig. 2 The AntNAG algorithm.

$$p_i^k(t) = \begin{cases} \dfrac{[\tau_i(t)]^\alpha}{\sum_{e_j \in N^k(t)} [\tau_j(t)]^\alpha} & \text{if } e_i \in N^k(t) \\ 0 & \text{if } e_i \notin N^k(t) \end{cases} \quad (3)$$

where $\tau_i(t)$ is the amount of pheromone on the exploit e_i at iteration t and $N^k(t) \subseteq E \setminus I$ is the set of preventable exploits which ant k has not yet chosen. α is a positive constant used to amplify the influence of pheromone trails. Large values of α give high importance to the pheromone trails on exploits, which may lead to rapid convergence to sub-optimal critical sets of exploits.

procedure ConstructCriticalSet(k)
  $C^k(t) = \varnothing$;
  while ant k has not constructed a critical set **do**
    Probabilistically choose an exploit e_i from the set of preventable exploits;
    if e_i hits some attack scenarios that are not hit by any exploit of $C^k(t)$ **then**
      $C^k(t) = C^k(t) \cup \{e_i\}$;
      Apply local pheromone trail update;
    end if;
  end while;
  return $C^k(t)$;
end procedure

Fig. 3 The algorithm for constructing a critical set of exploits.

After choosing the exploit e_i, it will be added to the partial solution $C^k(t)$ if it hits some attack scenarios of S that are not hit by any exploit of $C^k(t)$.

Ants update the pheromone trails while constructing a solution. After adding an exploit e_i to the partial solution of ant k, the pheromone trail τ_i is updated using the local updating rule [12],

$$\tau_i(t) = (1-\xi)\tau_i(t) + \xi.\tau_0 \quad (4)$$

where $0 \le \xi \le 1$ is the local evaporation rate and τ_0 is the lower pheromone trail limit. The value of τ_0 is set to be the same as the initial value for the pheromone trails.

The effect of the local updating rule is that each time an ant chooses an exploit, the pheromone trail on the exploit is reduced, so that the exploit becomes less desirable for the following ants. In other words, the local updating rule has the effect of lowering the pheromone trails on visited exploits so that they will be chosen with a lower probability by the other ants in their steps for constructing a critical set of exploits. This allows an increase in the exploration of exploits that have not been visited yet.

The pheromone trails can never fall below τ_0, because the initial value for them is set to the value of τ_0 and the local updating rule always adds an amount of pheromone greater than or equal to τ_0. Having a lower pheromone trail limit has the advantage that all exploits have a nonzero probability of being included in a critical set of exploits. This causes the algorithm not to show a premature stagnation behavior (i.e., ants do not follow the same path and hence do not construct the same critical set of exploits.

4.2 Minimal Solutions

The critical set of exploits constructed by an ant may not be minimal. In other words, it may contain redundant exploits, which must be eliminated.

procedure EliminateRedundantExploits($C^k(t)$)
  $R^k(t) = \{e_j \in C^k(t) \mid hv_x(e_j, C^k(t)) = 0\}$;
  while $R^k(t) \neq \varnothing$ **do**
    Choose $e_i \in R^k(t)$ such that it has the minimum selection value $sv(e_i, C^k(t))$;
    $C^k(t) = C^k(t) \setminus \{e_i\}$;
    $R^k(t) = \{e_j \in C^k(t) \mid hv_x(e_j, C^k(t)) = 0\}$;
  end while;
  return $C^k(t)$;
end procedure

Fig. 4 The algorithm for eliminating redundant exploits.

Let $C^k(t)$ be the critical set of exploits constructed by an ant k. For each exploit e_i, we define the *exclusive hit value* $hv_x(e_i, C^k(t))$ to be the number of attack scenarios that are hit by e_i, but that are not hit by any exploit of $C^k(t) \setminus \{e_i\}$.

If an attack scenario is already hit by several other exploits of $C^k(t)$, then extra hitting by an exploit $e_i \in C^k(t)$ has no relevant effect. Hence, the exploit e_i is called *candidate redundant* with respect to $C^k(t)$ if $hv_x(e_i, C^k(t)) = 0$. The set of candidate redundant exploits of $C^k(t)$ is denoted by $R^k(t)$.

$$R^k(t) = \{e_j \in C^k(t) \mid hv_x(e_j, C^k(t)) = 0\} \quad (5)$$

The exclusive hit value is used to define the *selection value* $sv(e_i, C^k(t))$ of a candidate redundant exploit $e_i \in R^k(t)$.

$$sv(e_i, C^k(t)) = \sum_{e_j \in C^k(t) \setminus \{e_i\}} hv_x(e_j, C^k(t) \setminus \{e_i\}) \quad (6)$$

A low value of $sv(e_i, C^k(t))$ means that the candidate redundant exploit e_i hits attack scenarios that are hit by too many other exploits of $C^k(t)$, and hence it is a good candidate redundant exploit to be removed from $C^k(t)$.

Accordingly, the selection value is used to evaluate candidate redundant exploits of a critical set of exploits in order to choose a candidate redundant exploit to be removed from it.

In Fig. 4, an algorithm is presented, which can be used to eliminate redundant exploits of $C^k(t)$. The algorithm is based on the idea that it is good to remove an exploit e_i from $C^k(t)$ if e_i is a candidate redundant exploit and hits attack scenarios that are hit by too many other exploits of $C^k(t)$. Hence, the algorithm removes at each step a candidate redundant exploit that has the minimum selection value. This is repeated until a minimal critical set of exploits is obtained.

4.3 Local Search Heuristic

After the minimal critical sets of exploits are constructed, the iteration-best solution (i.e., the minimal critical set of exploits constructed by the iteration-best ant) is improved by a local search heuristic.

The local search heuristic is based on the following idea. Given the iteration-best solution $C^{ib}(t)$, suppose there is an exploit $e_j \notin C^{ib}(t)$ such that $C^{ib}(t) \cup \{e_j\}$ contains at least two exploits other than e_j, say $e_{i_1},...,e_{i_l}$, with $l \geq 2$ that are redundant. Then $(C^{ib}(t) \setminus \{e_{i_1},...,e_{i_l}\}) \cup \{e_j\}$ is a better critical set of exploits than $C^{ib}(t)$. The gain of the exploit e_j with respect to $C^{ib}(t)$ is $g(e_j) = l - 1$. In this case, we call e_j a *candidate dominant* exploit.

procedure LocalSearch($C^{ib}(t)$)
  $D^{ib}(t) = \left\{e_j \notin C^{ib}(t) \mid g(e_j) > 0\right\}$;
  while $D^{ib}(t) \neq \varnothing$ **do**
    Choose $e_i \in D^{ib}(t)$ such that it has the highest gain $g(e_i)$;
    $C^{ib}(t) = C^{ib}(t) \cup \{e_i\}$;
    Eliminate redundant exploits of $C^{ib}(t)$;
    $D^{ib}(t) = \left\{e_j \notin C^{ib}(t) \mid g(e_j) > 0\right\}$;
  end while;
  return $C^{ib}(t)$;
end procedure

Fig. 5 The local search heuristic.

As shown in Fig. 5, the local search heuristic starts with the iteration-best solution $C^{ib}(t)$ and chooses a candidate dominant exploit having the highest gain. Then the chosen candidate dominant exploit is added to $C^{ib}(t)$ and the set of redundant exploits are removed from the resulting critical set of exploits. This is repeated until no better critical set of exploits is obtained.

4.4 Global Pheromone Trail Update

The last step in an iteration of the AntNAG is the updating of pheromone trails using the following global updating rule:

$$\tau_i(t+1) = (1-\rho)\tau_i(t) + \rho.\Delta\tau_i^{ib}(t) \quad \forall e_i \in C^{ib}(t) \qquad (7)$$

where $0 < \rho \leq 1$ is the global evaporation rate, $C^{ib}(t)$ is the minimal critical set of exploits constructed by the iteration-best ant, and $\Delta\tau_i^{ib}(t)$ is the amount of pheromone deposited by the iteration-best ant on the exploit e_i at iteration t of the algorithm. $\Delta\tau_i^{ib}(t)$ is defined as follows:

$$\Delta\tau_i^{ib}(t) = |E| - |C^{ib}(t)| \qquad (8)$$

It is important to note that in the global updating rule, both evaporation and new pheromone deposit are only applied to the exploits of $C^{ib}(t)$. Also, the deposited pheromone is discounted by a factor ρ; this results in the new pheromone trail being a weighted average between the old pheromone value and the amount of the pheromone deposited. For small values of ρ, the existing pheromone trails on exploits evaporate slowly, while the influence of the iteration-best critical set of exploits is dampened. On the other hand, for large values of ρ, the previous pheromone deposits evaporate rapidly, but the influence of the iteration-best critical set of exploits is emphasized.

5 Experiments

In order to evaluate the performance of the AntNAG, we performed our experiments over a sample network attack graph and several randomly generated large-scale network attack graphs.

5.1 Sample Network Attack Graph

Consider the network shown in Fig. 6. There are three target hosts called *RedHat*, *Windows* and *Fedora* on an internal network, and a host called *PublicServer* on an isolated demilitarized zone (DMZ) network.

A number of services are running on each of the hosts of *RedHat*, *Windows*, *Fedora*, and *PublicServer*. Also, each of the above hosts has a number of vulnerabilities. Vulnerability scanning tools, such as Nessus [18], can be used to find the vulnerabilities of each host.

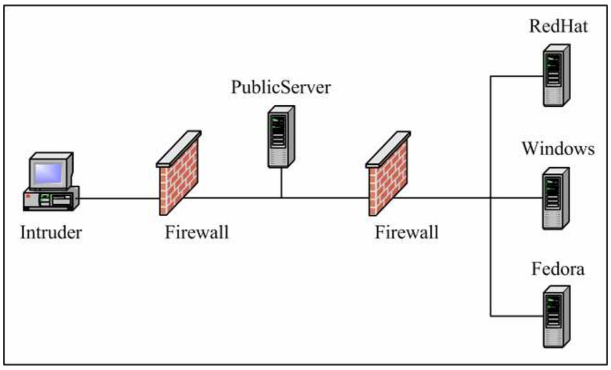

Fig. 6 An example network.

Different types of services and vulnerabilities available on the network hosts are introduced in Table A.1 of Appendix A.

The *RedHat* host on the internal network is running ftp and ssh services. The *Fedora* host is running several services: LICQ chat software, Squid web proxy, ftp and a database. The LICQ client lets Linux users exchange text messages over the Internet. The Squid web proxy is a full-featured web proxy cache. It stores requested Internet objects on a system closer to the requesting site than to the source. Web browsers can then use the local Squid cache as a proxy server, reducing access time as well as bandwidth consumption. The *PublicServer* host on the DMZ network is running IIS and Exchange services. The connectivity information between the network hosts is shown in Table 1. In this Table, each entry corresponds to a pair of (h_s, h_t) in which h_s is the source network host and h_t is the target network host. Every entry has five boolean values. These values are 'T' if the network host h_s can connect to the network host h_t on the ports of *http*, *licq*, *ftp*, *ssh*, and *smtp*, respectively.

The intruder launches his attack starting from a single network host, called *Intruder*, which lies on the outside network. His goal is to disrupt the *database* service on the network host *Fedora*. To achieve this goal, the intruder should gain the *root* privilege on this network host.

Table 1 Network connectivity information.

Host	Intruder	Server	RedHat	Windows	Fedora
Intruder	F,F,F,F,F	T,F,F,F,T	F,F,F,F,F	F,F,F,F,F	F,F,F,F,F
Server	F,F,F,F,F	T,F,F,F,T	F,F,T,T,F	F,F,F,F,F	T,T,T,F,F
RedHat	F,F,F,F,F	T,F,F,F,T	F,F,T,T,F	F,F,F,F,F	T,T,T,F,F
Windows	F,F,F,F,F	T,F,F,F,T	F,F,T,T,F	F,F,F,F,F	T,T,T,F,F
Fedora	F,F,F,F,F	T,F,F,F,T	F,F,T,T,F	F,F,F,F,F	T,T,T,F,F

There are *wdir*, *fshell*, and *sshd_bof* vulnerabilities on the *RedHat* host, *scripting* vulnerability on the *Windows* host, *wdir*, *fshell*, *squid_conf*, and *licq_ivv* vulnerabilities on the *Fedora* host, and *iis_bof* and *exchange_ivv* on the *PublicServer* host. Also, *at* and *xterm* programs on the *RedHat* and *Fedora* are vulnerable to buffer overflow.

The intruder can use ten generic exploits. In Table B.1 of Appendix B, each generic exploit is represented by its preconditions and postconditions. The description of each generic exploit is given in Table B.2 of Appendix B. More information about each of the exploits is available in the CVE List [19], which is a publicly available list or dictionary of standardized identifiers for common vulnerabilities and exposures.

Among the ten generic exploits shown in Table B.1, the first eight generic exploits require a pair of network hosts while the last two generic exploits require only one network host. Therefore, there are totally (8 * 5 * 4) + (2 * 4) = 168 exploits, which the intruder can try. Each attack scenario for the above network consists of a subset of these 168 exploits. For example, consider the following attack scenario:

(1) $iis\_r2r(Intruder, PublicServer)$
(2) $squid\_ps(PublicServer, Fedora)$
(3) $licq\_r2u(PublicServer, Fedora)$
(4) $xterm\_u2r(Fedora, Fedora)$

The intruder first launches the *iis_r2r* exploit to gain *root* privilege on the *PublicServer* host. Then he uses the *PublicServer* host to launch a port scan via the vulnerable Squid web proxy running on the *Fedora* host. The scan discovers that it is possible to gain *user* privilege on the *Fedora* host with launching the *licq_r2u* exploit. After that, a simple local buffer overflow gives the intruder *root* privilege on the *Fedora* host. The attack graph for the above network consists of 164 attack scenarios. Each attack scenario contains from 4 to 9 exploits.

5.1.1 Experimental Results

We applied the AntNAG for minimization analysis of the above network attack graph. To evaluate the performance of the algorithm, we performed several experiments.

In the first experiment, we assumed that the set of inevitable exploits is empty, i.e., all exploits are preventable. Therefore, the aim was to find a minimum critical set of exploits among 168 exploits. Using the AntNAG, the following minimum critical set of exploits was found:

$$C = \{\, iis\_r2r(Intruder, PublicServer), \\ exchange\_r2u(Intruder, PublicServer) \,\}$$

In the second experiment, we assumed that the generic exploits *iis_r2r*, *exchange_r2u*, and *xterm_u2r* are inevitable, i.e., the prevention of them is not feasible or incurs high cost. Therefore, the aim was to find a minimum critical set of exploits among 124 exploits. Using the AntNAG, the following minimum critical set of exploits was found:

$$C = \{\, licq\_r2u(PublicServer, Fedora), \\ licq\_r2u(RedHat, Fedora), \\ script\_r2u(PublicServer, Windows), \\ ftp\_rhosts(PublicServer, Fedora), \\ ftp\_rhosts(RedHat, Fedora) \,\}$$

While using the greedy algorithm proposed by Sheyner *et al.* [4] and Jha *et al.* [9, 10], the following minimum critical set of exploits was found:

$$C = \{\, script\_r2u(PublicServer, Windows), \\ at\_u2r(Fedora, Fedora),$$

sshd_r2u(*PublicServer*, *RedHat*),
ftp_rhosts(*PublicServer*, *RedHat*),
squid_ps(*PublicServer*, *Fedora*),
ftp_rhosts(*PublicServer*, *Fedora*)}

The second experiment shows the AntNAG can find a critical set of exploits with less cardinality.
In the experiments, the AntNAG parameters were set to $\tau_0 = 1$, $\alpha = 1$, $\xi = 0.1$, and $\rho = 0.1$. The number of ants was set to $m = 10$ and the maximum number of iterations was set to $t_m = 50$.

5.2 Large-Scale Network Attack Graphs

A large computer network builds upon multiple platforms, runs different software packages and supports several modes of connectivity. Despite the best efforts of software architects and developers, each network host inevitably contains a number of vulnerabilities. Several factors can make network attack graphs larger so that finding a minimum critical set of exploits becomes more difficult. An obvious factor is the size of the network under analysis. Our society has become increasingly dependent on computer networks and the trend towards larger networks will continue. For example, there are enterprises today consisting of tens of thousands of network hosts. Also, less secure networks clearly have larger network attack graphs. Each network host might have several exploitable vulnerabilities. When considered across a large enterprise, network attack graphs become potentially large [20].

Table 2 Large-scale network attack graphs.

Network Attack Graph	Cardinality of the Set of Exploits (n)	Cardinality of the Set of Attack Scenarios (l)	Average Cardinality of Attack Scenarios
NAG_1	100	1000	5.93
NAG_2	200	2000	6.01
NAG_3	400	4000	5.99
NAG_4	400	6000	5.99
NAG_5	800	8000	6.01
NAG_6	800	10000	6.04
NAG_7	100	1000	7.56
NAG_8	200	2000	7.55
NAG_9	400	4000	7.52
NAG_{10}	400	6000	7.48
NAG_{11}	800	8000	7.48
NAG_{12}	800	10000	7.50

In order to further evaluate the performance of the AntNAG, we randomly generated 12 large-scale network attack graphs, denoted by NAG_1, NAG_2, ..., NAG_{12}. For each network attack graph, we considered different values for the cardinalities of E and S, where E is the set of known exploits and S is the set of attack scenarios represented by the network attack graph. In NAG_1, ..., NAG_6, attack scenarios contain from 3 to 9 exploits while in NAG_7, ..., NAG_{12}, attack scenarios contain from 3 to 12 exploits.
Table 2 shows the cardinality of the set of known exploits, the cardinality of the set of attack scenarios, and the average cardinality of attack scenarios for each generated network attack graph.

5.2.1 Experimental Results

We applied the AntNAG for minimization analysis of the above large-scale network attack graphs. We performed 10 runs of the algorithm with different random seeds and reported the best cardinality and the average cardinality of critical sets of exploits obtained from these 10 runs. We also applied the greedy algorithm proposed by Sheyner *et al.* [4] and Jha *et al.* [9, 10] for minimization analysis of the above network attack graphs.
As shown in Table 3, the AntNAG outperforms the greedy algorithm and finds critical sets of exploits with less cardinality. Also, the AntNAG performs significantly better than the AntNAG without the local search heuristic. On average, the cardinality of critical sets of exploits found by the AntNAG and the AntNAG without the local search heuristic are, respectively, 9.98% and 7.20% less than the cardinality of critical sets of exploits found by the greedy algorithm.

Table 3 Cardinality of critical set of exploits found by the AntNAG and the greedy algorithm.

Network Attack Graph	AntNAG		AntNAG without LS		Greedy Algorithm [4, 9, 10]
	Best	Average	Best	Average	
NAG_1	44	44.5	44	44.9	50
NAG_2	87	88.5	91	92.7	98
NAG_3	177	178.9	182	182.9	197
NAG_4	198	199.3	202	203.7	221
NAG_5	358	361.1	373	376.1	397
NAG_6	373	380.1	396	397.8	417
NAG_7	39	39.3	39	39.5	45
NAG_8	81	81.8	84	85.3	91
NAG_9	159	161.7	164	165.4	182
NAG_{10}	180	181.8	184	185.6	200
NAG_{11}	326	329.1	341	343	362
NAG_{12}	346	348.8	361	365.4	388

In the experiments, the AntNAG parameters were set to $\tau_0 = 1$, $\alpha = 1$, and $\xi = 0.1$. The number of ants was set to $m = 15$ and the maximum number of iterations was set to $t_m = 100$. For minimization analysis of NAG_1

and NAG_7, ρ was set to 0.1, for minimization analysis of NAG_2, NAG_3, NAG_4, NAG_8, NAG_9 and NAG_{10}, ρ was set to 0.05, and for minimization analysis of the other network attack graphs, ρ was set to 0.025.

Figures 7 to 10 show the progress of the average cardinality of the global-best solution (i.e., the best critical set of exploits found from the first iteration of the algorithm), obtained from 10 runs of the AntNAG and 10 runs of the AntNAG without the local search heuristic for minimization analysis of NAG_4, NAG_6, NAG_9, and NAG_{11}, respectively. The cardinality of the global-best solution is expected to be as small as possible. As the figures show, the local search heuristic is essential for the construction of high-quality critical sets of exploits. This is because after improving the iteration-best solution by the local search heuristic, pheromone trails are updated on the exploits of the locally optimized solution.

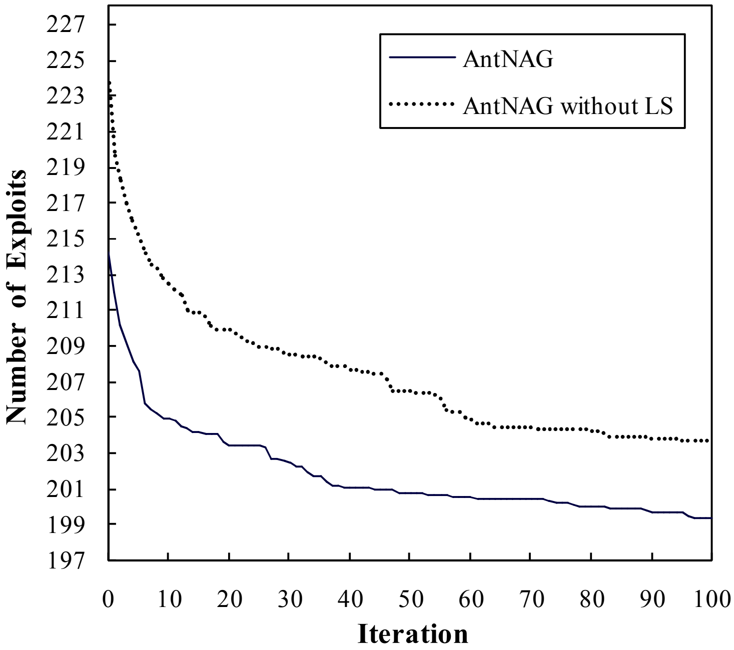

Fig. 7 Progress of the average cardinality of the global-best solution of the AntNAG and the AntNAG without the local search heuristic on NAG_4.

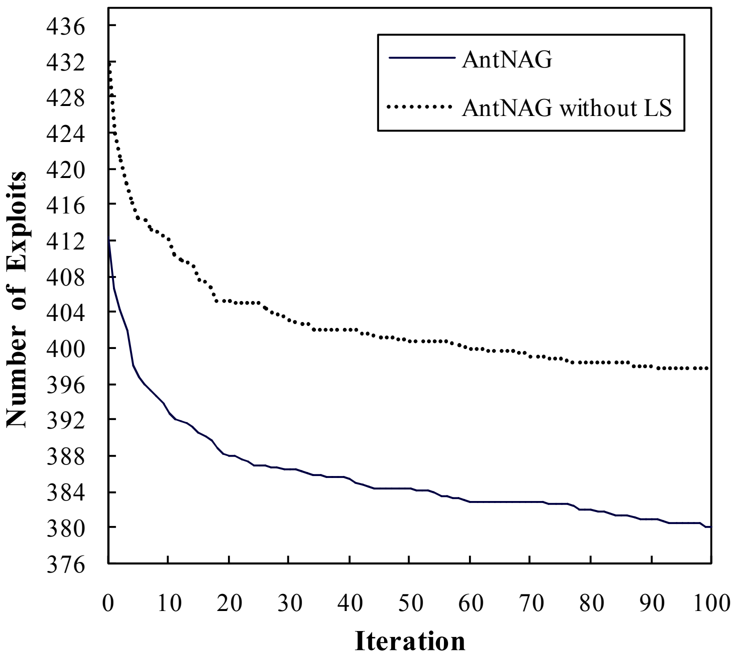

Fig. 8 Progress of the average cardinality of the global-best solution of the AntNAG and the AntNAG without the local search heuristic on NAG_6.

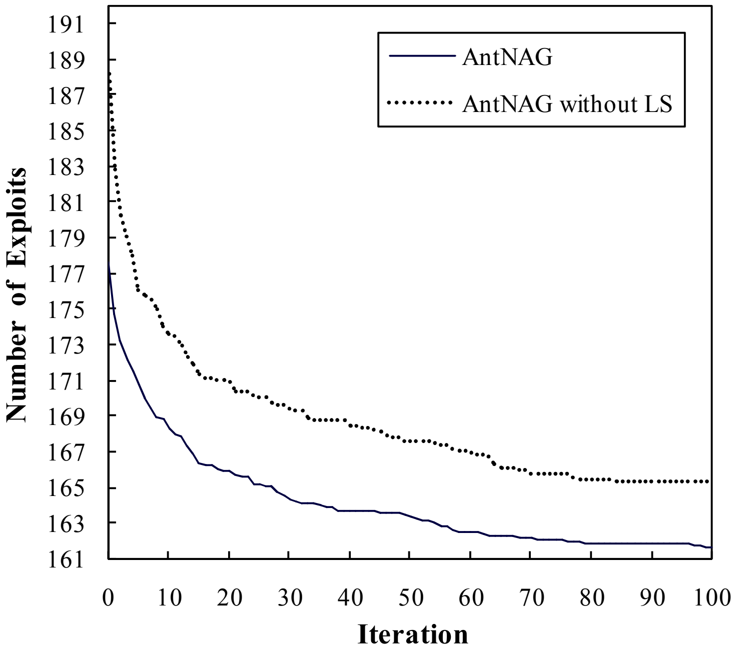

Fig. 9 Progress of the average cardinality of the global-best solution of the AntNAG and the AntNAG without the local search heuristic on NAG_9.

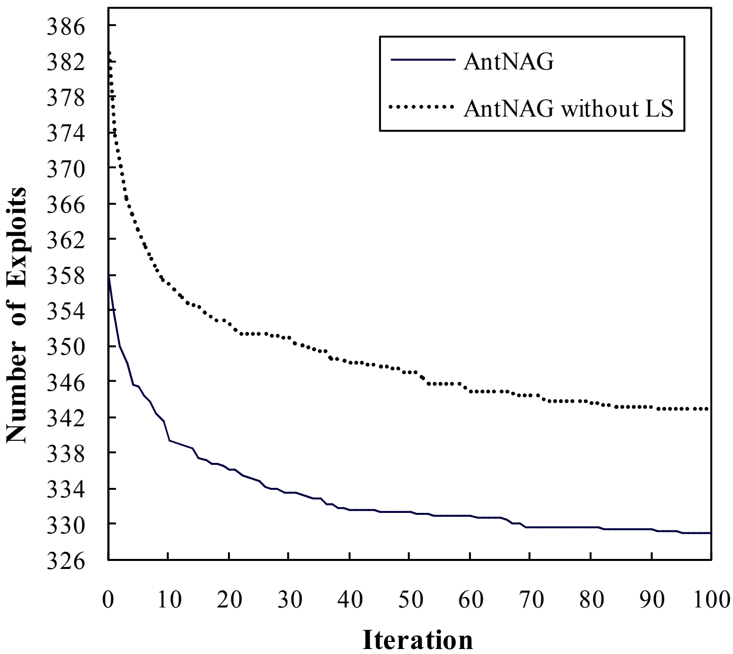

Fig. 10 Progress of the average cardinality of the global-best solution of the AntNAG and the AntNAG without the local search heuristic on NAG_{11}.

Figures 11 to 14 show the effect of the different values of the global evaporation rate, ρ, on the performance of the AntNAG. The results were obtained from 10 runs of the AntNAG for minimization analysis of NAG_3, NAG_6, NAG_{10}, and NAG_{11}. The figures suggest that by decreasing the value of the global evaporation rate ρ, the average cardinality of the global-best solution will decrease. For small values of ρ, the existing pheromone trails on exploits evaporate slowly, while the influence of the iteration-best solution is dampened.

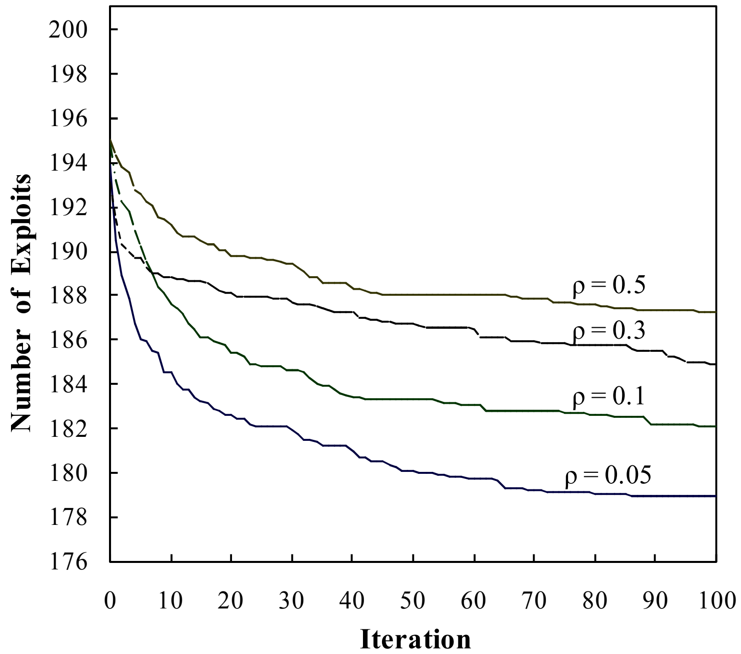

Fig. 11 Effect of the global evaporation rate on the performance of the AntNAG on NAG_3.

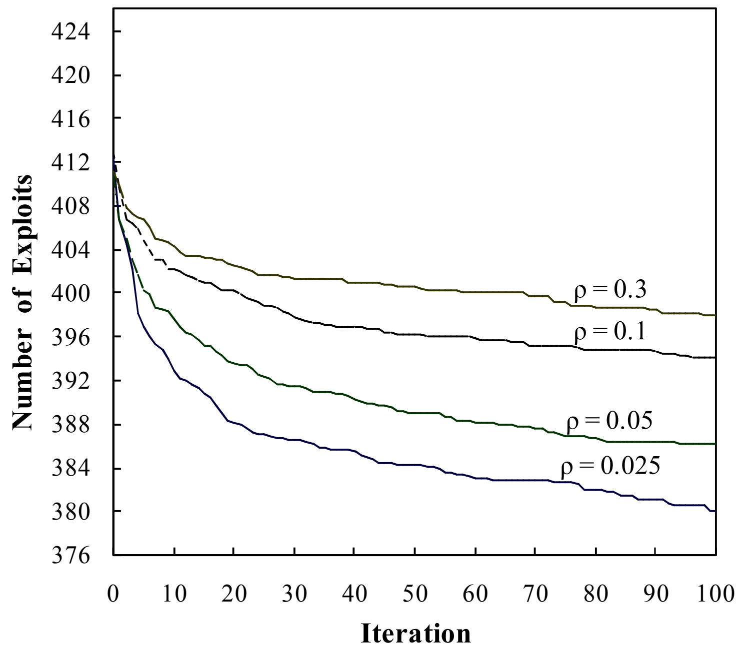

Fig. 12 Effect of the global evaporation rate on the performance of the AntNAG on NAG_6.

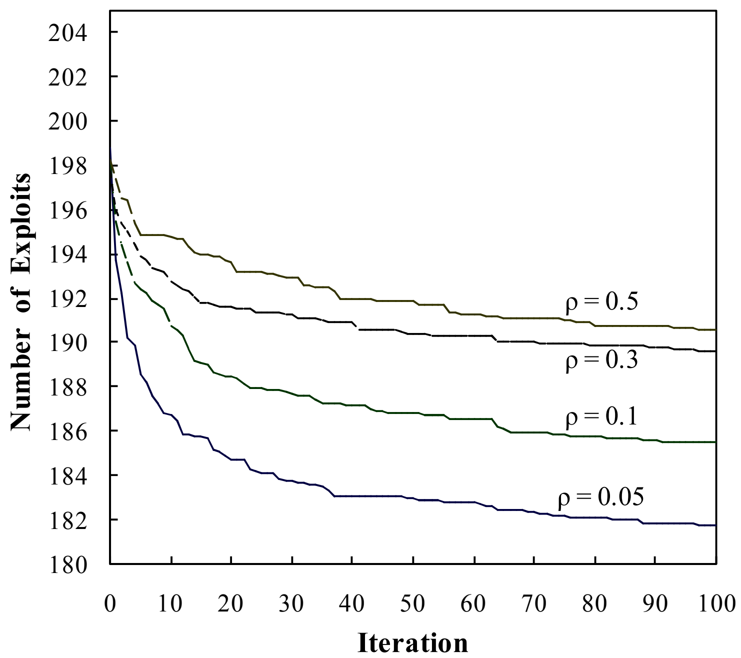

Fig. 13 Effect of the global evaporation rate on the performance of the AntNAG on NAG_{10}.

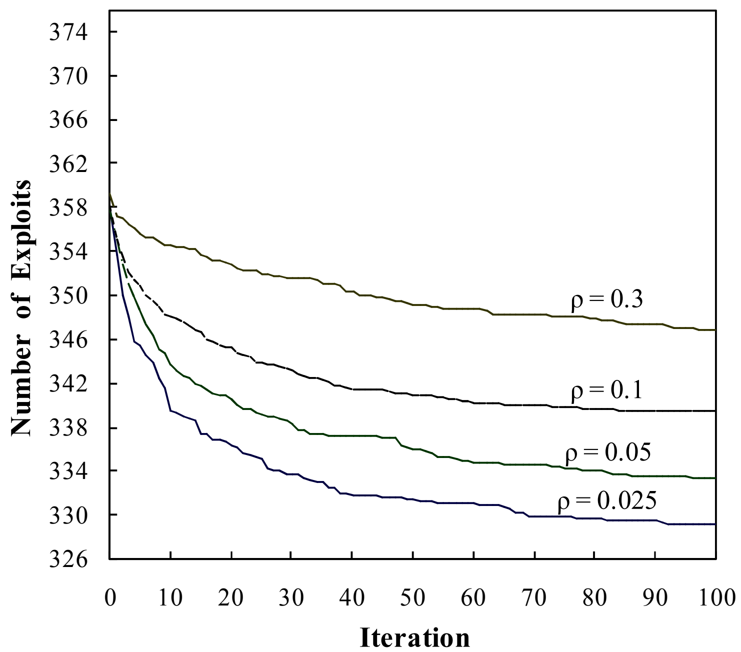

Fig. 14 Effect of the global evaporation rate on the performance of the AntNAG on NAG_{11}.

It should be noted that the best value of ρ is different for network attack graphs with different scales. This is because, as defined in Eq. (8), the amount of pheromone deposited by the iteration-best ant is obtained as a function of the cardinality of the set of exploits. The larger the cardinality of the set of exploits, the lower the value of ρ is chosen.

Figures 15 and 16 show the effect of the number of ants, m, on the performance of the AntNAG, obtained from 10 runs of the AntNAG for minimization analysis of NAG_4 and NAG_{12}. As the figures show, when using a very small number of ants, the algorithm shows a premature stagnation behavior. This is because the fewer the number of ants, the less the exploration ability of the algorithm, and consequently the less information about the search space is available to all ants.

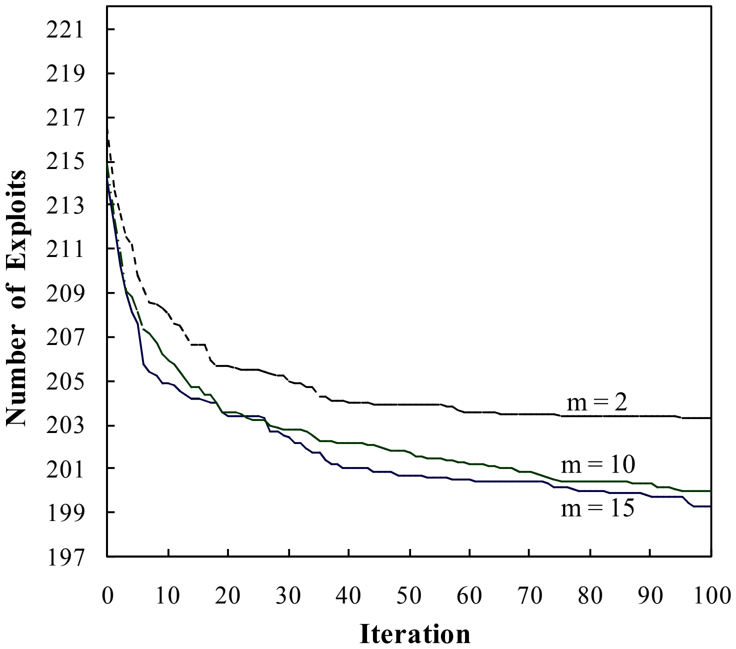

Fig. 15 Effect of the number of ants on the performance of the AntNAG on NAG_4.

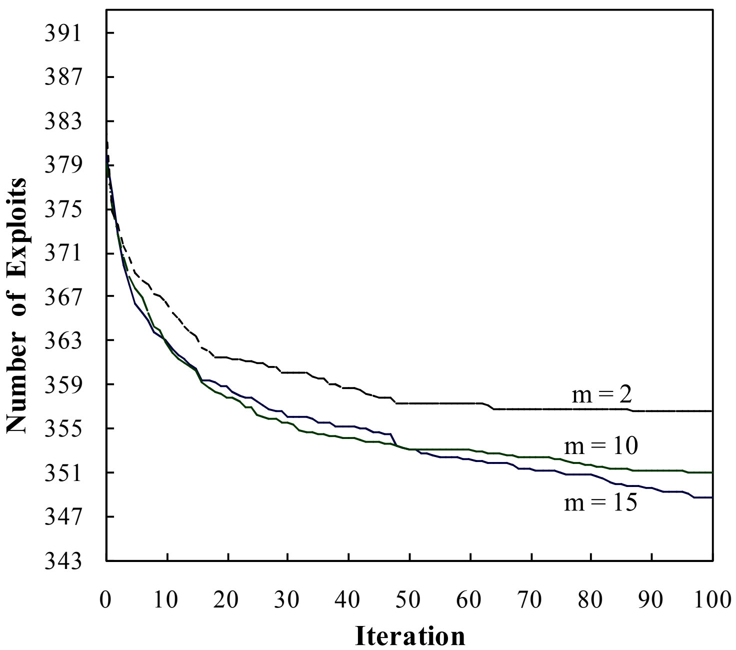

Fig. 16 Effect of the number of ants on the performance of the AntNAG on NAG_{12}.

6 Time Complexity

Let m be the number of ants and t_m be the maximum number of iterations. At each iteration t of the AntNAG, each ant k starts with an empty set and constructs a critical set of exploits $C^k(t)$ by incrementally adding exploits until all attack scenarios are hit. The construction of $C^k(t)$ takes $O(n \cdot l)$ time, where n is the cardinality of the set of preventable exploits and l is the cardinality of the set of attack scenarios. The redundant exploits of $C^k(t)$ are then eliminated using the algorithm in Fig. 4, which runs in $O(n^2 \cdot l)$ time. After that, the iteration-best solution $C^{ib}(t)$ is improved by the local search heuristic in Fig. 5, which runs in $O(n^3 \cdot l)$ time. Finally, the pheromone trails are updated using the global updating rule, which takes $O(n)$ time. The overall time complexity of the AntNAG is therefore $O(t_m \cdot n^3 \cdot l)$. Strictly speaking, it is $O(t_m \cdot n^2 \cdot l \cdot (n+m))$, but we usually set m to a small value. Hence, the time complexity can be simplified to $O(t_m \cdot n^3 \cdot l)$. Similarly, the overall time complexity of the AntNAG without the local search heuristic is $O(t_m \cdot m \cdot n^2 \cdot l)$. It should be noted that the time complexity of the greedy algorithm proposed by Sheyner *et al*. [4] and Jha *et al*. [9, 10] is $O(n^2 \cdot l)$.

7 Conclusions

Each network attack graph represents the collection of possible attack scenarios in a computer network. Each attack scenario is a sequence of exploits that leads to an undesirable state. An example of an undesirable state is one that the intruder gains root privilege on a critical network host. The aim of minimization analysis of a network attack graph is to find a minimum critical set of exploits, which must be prevented to thwart an intruder. This problem is in fact *NP*-hard.

In this paper, we presented an ant colony optimization algorithm, called AntNAG, for minimization analysis of network attack graphs. We reported the results of applying this algorithm for minimization analysis of a sample network attack graph and 12 large-scale network attack graphs. We also applied the greedy algorithm proposed by Sheyner *et al*. [4] and Jha *et al*. [9, 10] for minimization analysis of the above network attack graphs. On average, the cardinality of critical sets of exploits found by the AntNAG and the AntNAG without the local search heuristic were, respectively, 9.98% and 7.20% less than the cardinality of critical sets of exploits found by the greedy algorithm. The results of experiments show that the AntNAG performs significantly better than the AntNAG without the local search heuristic and finds a critical set of exploits with less cardinality. This shows the significance of the local search heuristic in the AntNAG.

Acknowledgments

This work was supported in part by ITRC.

References

[1] Abadi M. and Jalili S., “Automatic discovery of network attack scenarios using SPIN model checker”, *In Proceedings of the International Symposium on Telecommunications*, pp. 81-86, Shiraz, Iran, September 2005.

[2] Dacier M., Deswarte Y., and Kaâniche M., “Quantitative assessment of operational security: Models and tools”, *Technical Report, LAAS Research Report 96493, Laboratory for Analysis and Architecture of Systems*, May 1996.

[3] Phillips C. and Swiler L. P., “A graph-based system for network-vulnerability analysis”, *In Proceedings of the 1998 Workshop on New Security Paradigms*, pp. 71-79, Charlottesville, VA, USA, September 1998.

[4] Sheyner O., Haines J., Jha S., Lippmann R., and Wing J. M., “Automated generation and analysis of attack graphs”, *In Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pp. 273-284, Berkeley, CA, USA, May 2002.

[5] NuSMV. “NuSMV: A New Symbolic Model Checker”. http://afrodite.itc.it:1024/nusmv/.

[6] Ammann P., Wijesekera D., and Kaushik S., “Scalable, graph-based network vulnerability analysis”, *In Proceedings of the 9th ACM Conference on Computer and Communications Security*, pp. 217-224, Washington, DC, USA, November 2002.

[7] Noel S., Jacobs M., Kalapa P., and Jajodia S., “Multiple coordinated views for network attack graphs”, *In Proceedings of the IEEE Workshop on Visualization for Computer Security (VizSEC 2005)*, pp. 99-106, Minneapolis, MN, USA, October 2005.

[8] Noel S. and Jajodia S., “Understanding complex network attack graphs through clustered adjacency matrices”, *In Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC 2005)*, pp. 160-169, Tucson, Arizona, USA, December 2005.

[9] Jha S., Sheyner O., and Wing J. M., “Minimization and reliability analysis of attack graphs”, *Technical Report, School of Computer Science, Carnegie Mellon University*, February 2002.

[10] Jha S., Sheyner O., and Wing J. M., “Two formal analyses of attack graphs”, *In Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pp. 49-63, Cape Breton, Nova Scotia, Canada, June 2002.

[11] Dorigo M. and Di Caro G., “The ant colony optimization meta-heuristic”, In D. Corne, M. Dorigo, and F. Glover (Eds.), *New Ideas in Optimization*, pp. 11-32, McGraw Hill, London, UK, 1999.

[12] Dorigo M. and Stützle T., *Ant Colony Optimization*, MIT Press, Cambridge, MA, USA, 2004.

[13] Dorigo M., Di Caro G., and Gambardella L. M., “Ant algorithms for discrete optimization”, *Artificial Life*, Vol. 5, No. 2, pp. 137-172, 1999.

[14] Dorigo M. and Stützle T., “The ant colony optimization metaheuristic: Algorithms, applications and advances”, In F. Glover and G. Kochenberger (Eds.), *Handbook of Metaheuristics*, Vol. 57, pp. 251-285, Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[15] Stützle T. and Dorigo M., “ACO algorithms for the traveling salesman problem”, In K. Miettinen et al. (Eds.), *Evolutionary Algorithms in Engineering and Computer Science*, pp. 163-183, John Wiley & Sons, New York, NY, USA, 1999.

[16] Merkle D., Middendorf M., and Schmeck H., “Ant colony optimization for resource constrained project scheduling”, *In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2000)*, pp. 893-900, Las

Vegas, NV, USA, July 2000.
[17] Di Caro G. and Dorigo M., "AntNet: Distributed stigmergetic control for communications networks", *Journal of Artificial Intelligence Research (JAIR)*, Vol. 9, pp. 317-365, 1998.
[18] Deraison R., "Nessus Vulnerability Scanner". http://www.nessus.org/.
[19] CVE List, "CVE: Common Vulnerabilities and Exposures". http://cve.mitre.org/.
[20] Ammann P., Pamula J., Ritchey R., and Street J., "A host-based approach to network attack chaining analysis", In *Proceedings of the 2005 Annual Computer Security Applications Conference (ACSAC 2005)*, pp. 72-84, Tucson, AZ, USA, December 2005.

Appendix A. Description of Vulnerabilities

Table A.1 Types of services and vulnerabilities running on the network hosts.

$iis\_bof(h)$	IIS web server has buffer overflow vulnerability on host h
$exchange\_ivv(h)$	Exchange mail server has input validation vulnerability on host h
$squid\_conf(h)$	Squid web proxy is misconfigured on host h
$licq\_ivv(h)$	LICQ client has input validation vulnerability on host h
$sshd\_bof(h)$	sshd server has buffer overflow vulnerability on host h
$scripting(h)$	HTML scripting is enabled on host h
$ftp(h)$	ftp service is running on host h
$wdir(h)$	ftp home directory is writable on host h
$fshell(h)$	ftp user has executable shell on host h
$ssh(h)$	ssh service is running on host h
$xterm\_bof(h)$	*xterm* program has buffer overflow vulnerability on host h
$at\_bof(h)$	*at* program has buffer overflow vulnerability on host h
$database(h)$	database service is running on host h

Appendix B. Description of Exploits

Table B.1 Exploit templates.

Exploit	Preconditions	Postconditions
$iis\_r2r(h_s, h_t)$	$iis\_bof(h_t)$ $C(h_s, h_t, http)$ $plvl(h_s) \geq$ user $plvl(h_t) <$ root	$\neg iis(h_t)$ $plvl(h_t) :=$ root
$exchange\_r2u(h_s, h_t)$	$exchange\_ivv(h_t)$ $C(h_s, h_t, smtp)$ $plvl(h_s) \geq$ user $plvl(h_t) =$ none	$plvl(h_t) :=$ user
$squid\_ps(h_s, h_t)$	$squid\_conf(h_t)$ $\neg scan$ $C(h_s, h_t, http)$ $plvl(h_s) \geq$ user	$scan$
$licq\_r2u(h_s, h_t)$	$licq\_ivv(h_t)$ $scan$ $C(h_s, h_t, licq)$ $plvl(h_s) \geq$ user $plvl(h_t) =$ none	$plvl(h_t) :=$ user
$script\_r2u(h_s, h_t)$	$scripting(h_t)$ $C(h_t, h_s, http)$ $plvl(h_s) \geq$ user $plvl(h_t) =$ none	$plvl(h_t) :=$ user
$sshd\_r2r(h_s, h_t)$	$sshd\_bof(h_t)$ $C(h_s, h_t, ssh)$ $plvl(h_s) \geq$ user $plvl(h_t) <$ root	$\neg ssh(h_t)$ $plvl(h_t) :=$ root
$ftp\_rhosts(h_s, h_t)$	$ftp(h_t)$ $wdir(h_t)$ $fshell(h_t)$ $\neg T(h_t, h_s)$ $C(h_s, h_t, ftp)$ $plvl(h_s) \geq$ user	$T(h_t, h_s)$
$rsh\_r2u(h_s, h_t)$	$T(h_t, h_s)$ $plvl(h_s) \geq$ user $plvl(h_t) =$ none	$plvl(h_t) :=$ user
$xterm\_u2r(h_t, h_t)$	$xterm\_bof(h_t)$ $plvl(h_t) =$user	$plvl(h_t) :=$ root
$at\_u2r(h_t, h_t)$	$at\_bof(h_t)$ $plvl(h_t) =$user	$plvl(h_t) :=$ root

Table B.2 Description of generic exploits.

Exploit	Description
iis_r2r	Buffer overflow vulnerability in the IIS web server allows remote intruders to gain *root* shell on the target network host
exchange_r2u	The OLE component in the Microsoft Exchange mail server does not properly validate the lengths of messages for certain OLE data, which allows remote intruders to execute arbitrary code
squid_ps	The intruder can use a misconfigured Squid web proxy to conduct unauthorized activities such as port scanning

licq_r2u	The intruder can send a specially crafted URL to the LICQ client to execute arbitrary commands on the target network host
script_r2u	Microsoft Internet Explorer allows remote intruders to execute arbitrary code via malformed Content-Type and Content-Disposition header fields that cause the application for the spoofed file type to pass the file back to the operating system for handling rather than raise an error message
sshd_r2r	Buffer overflow vulnerability in the sshd server allows remote intruders to gain *root* shell on the target network host
ftp_rhosts	Using ftp vulnerability, the intruder creates a .rhosts file in the ftp home directory, creating a remote login trust relationship between his network host and the target network host
rsh_r2u	Using an existing remote login trust relationship between two hosts, the intruder logs in from one machine to another, getting a *user* shell without supplying a password
xterm_u2r	Buffer overflow vulnerability in the *xterm* program allows local users to gain *root* shell on the target network host
at_u2r	Buffer overflow vulnerability in the *at* program allows local users to gain *root* shell on the target network host

Mahdi Abadi is a Ph.D. student of computer engineering at Tarbiat Modares University. He received the M.Sc. degree in software engineering from that university in 2001, and the B.Sc. degree in software engineering from Ferdowsi University of Mashhad in 1998. His main research interests are network security, intrusion detection, evolutionary algorithms, and data mining. Currently, Mahdi Abadi works on his Ph.D. thesis on network vulnerability analysis.

Saeed Jalili received the Ph.D. degree from Bradford University in 1991 and the M.Sc. degree in computer science from Sharif University of Technology in 1979. Since 1992, he has been assistant professor at the Tarbiat Modares University. His main research interests are machine learning, intrusion detection, security protocol verification, and software runtime verification.