Scientia Iranica, Vol. 15, No. 6, pp. 605–619

SCIENTIA
IRANICA

Using Binary Particle Swarm Optimization for Minimization Analysis of Large-Scale Network Attack Graphs

M. Abadi[1] **and S. Jalili**[1,*]

The aim of the minimization analysis of network attack graphs (NAGs) is to find a minimum critical set of exploits so that by preventing them an intruder cannot reach his goal using any attack scenario. This problem is, in fact, a constrained optimization problem. In this paper, a binary particle swarm optimization algorithm, called SwarmNAG, is presented for the minimization analysis of large-scale network attack graphs. A penalty function method with a time-varying penalty coefficient is used to convert the constrained optimization problem into an unconstrained problem. Also, a time-varying velocity clamping, a greedy mutation operator and a local search heuristic are used to improve the overall performance of the algorithm. The performance of the SwarmNAG is compared with that of an approximation algorithm for the minimization analysis of several large-scale network attack graphs. The results of the experiments show that the SwarmNAG outperforms the approximation algorithm and finds a critical set of exploits with less cardinality.

Keywords: Particle swarm optimization; Constrained optimization; Penalty function method; Local search; Network attack graph.

INTRODUCTION

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

Phillips and Swiler [2] proposed the concept of attack graphs, where each node represents a possible attack state. Edges represent a change of state caused by a single action taken by the intruder.

1. *Department of Computer Engineering, Tarbiat Modares University, P.O. Box 14115-143, Tehran, Iran.*
*. *To whom correspondence should be addressed. E-mail: sjalili@modares.ac.ir*

Sheyner et al. [3] and Jha et al. [4,5] used a modified version of the model checker NuSMV [6] to produce attack graphs. Ammann et al. [7] introduced a monotonicity assumption and applied it to develop a polynomial algorithm to encode all of the edges in an attack graph without actually computing the graph itself. These attack graphs are essentially similar to [2], where any path in the graph, from an initial node to a goal node, shows a sequence of exploits that an intruder can launch in order to reach his goal.

Noel et al. [8] presented a number of techniques for managing attack graph complexity through visualization.

Mehta et al. [9] presented a ranking scheme for the nodes of an attack graph. The rank of a node shows its importance, based on factors like the probability of an intruder reaching that node. Given a ranked attack graph, the system administrator can concentrate on relevant subgraphs to figure out how to start deploying security measures.

The aim of the minimization analysis of attack graphs is to find a minimum critical set of exploits

that completely disconnect the initial nodes and the goal nodes of the graph. Sheyner et al. [3] and Jha et al. [4,5] showed that this problem is, in fact, NP-hard. They proposed an approximation algorithm that can find an approximately-optimal set of exploits that must be prevented to thwart an intruder. While it is currently possible to generate very large and complex network attack graphs, relatively little work has been done regarding their analysis.

Particle swarm optimization (PSO) is a swarm intelligence method that models social behaviour to guide swarms of particles towards the most promising regions of the search space [10,11] and has proved to be efficient at solving engineering problems [12-15].

The problem of the minimization analysis of network attack graphs is, in fact, a constrained optimization problem, in which the objective is to find a solution with minimum cardinality, and the constraint is that the solution must be critical (i.e., it must hit all attack scenarios). The most common approach in solving constrained optimization problems is the use of a penalty function method, which adds a penalty to the objective function in order to discourage infeasible areas of the search space being searched [16].

In this paper, a binary PSO algorithm, called SwarmNAG, is presented for the minimization analysis of large-scale network attack graphs (NAGs). The performance of the SwarmNAG is also compared with that of the approximation algorithm proposed by Sheyner et al. [3] and Jha et al. [4,5], in order to analyze several large-scale network attack graphs.

The remainder of this paper is organized as follows: First, an overview of PSO is provided, then, the network security model is introduced followed by a description of network attack graphs. There is a presentation of the SwarmNAG followed by a description of the different measures used to evaluate its performance. Finally, the experimental results are reported, followed by some conclusions.

PARTICLE SWARM OPTIMIZATION

Particle swarm optimization (PSO) is a population based stochastic optimization algorithm developed by Kennedy and Eberhart [10]. It was inspired by the social behavior of flocks of birds when searching for food. In PSO, the potential solutions, called particles, fly through the problem space looking for better regions. The position of a particle is influenced by its best visited position and the position of the best particle in its neighborhood. When the neighborhood of a particle is the entire swarm, the best position in the neighborhood is referred to as the global best particle and the resulting algorithm is referred to as a *gbest* PSO. When smaller neighborhoods are used, the algorithm is generally referred to as a *lbest PSO*.

The performance of each particle is measured by a predefined fitness function, which is related to the problem to be solved. Each particle in the swarm has a current position, x_i, a velocity (rate of position change), v_i, and a personal best position, y_i. The personal best position of particle i shows the best fitness reached by that particle at a given time. Let f be the objective function to be maximized, then, the personal best position of a particle at iteration or time step t is updated as follows:

$$y_i(t) = \begin{cases} y_i(t-1) & \text{if } f(x_i(t)) \leq f(y_i(t-1)) \\ x_i(t) & \text{if } f(x_i(t)) > f(y_i(t-1)) \end{cases} \quad (1)$$

For the *gbest* model, the global best position is determined from the entire swarm by selecting the best personal best position. This position is denoted by $\hat{y}$.

The equation that manipulates the velocity is called the *velocity update equation* and is stated as follows:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)(y_{ij}(t) - x_{ij}(t)) + c_2 r_{2j}(t)(\hat{y}_j(t) - x_{ij}(t)), \quad (2)$$

where $v_{ij}(t+1)$ is the velocity updated for the jth dimension, $j = 1, 2, \cdots, d$. c_1 and c_2 are the acceleration constants, where the first moderates the maximum step size towards the personal best position of the particle, while the second moderates the maximum step size towards the global best position in just one iteration. $r_{1j}(t)$ and $r_{2j}(t)$ are two random values in the range $[0, 1]$ which give the PSO algorithm a stochastic search property.

The velocity update equation consists of the following three components:

- The *inertia component*, which serves as a memory of the previous flight direction, i.e. movement in the immediate past;
- The *cognitive component*, which quantifies the performance of particle i relative to past performances. In a sense, the cognitive component resembles individual memory of the position that was best for the particle;
- The *social component*, which quantifies the performance of particle i relative to a group of particles. The effect of the social component is that each particle is also drawn towards the best position found by the particle's neighbourhood.

Velocity updates on each dimension can be clamped with a user defined maximum velocity, $V_{\max}$, which would prevent them from exploding, thereby causing premature convergence [17,18].

Each particle updates its position using the following equation:

$$x_i(t+1) = x_i(t) + v_i(t+1). \quad (3)$$

In swarm terminology, particle i is flying to its new position, $x_i(t+1)$. After the new position is calculated for each particle, the iteration counter increases and the new particle positions are evaluated. This process is repeated until some convergence criteria are satisfied.

Binary Particle Swarm Optimization

Kennedy and Eberhart [19] have adapted the PSO to search in binary spaces. For the binary PSO, the elements of x_i, y_i and $\hat{y}$ can only take the values 0 and 1. The velocity, v_i, is interpreted as a probability to change a bit from 0 to 1, or from 1 to 0, when updating the position of particles. Therefore, the velocity vector remains continuous-valued. Since each v_{ij} is a real value, a mapping needs to be defined from v_{ij} to a probability in the range $[0,1]$. This is done using a sigmoid function to squash velocities into the $[0,1]$ range. The sigmoid function is defined as follows:

$$sig(v) = \frac{1}{1+e^{-v}}. \quad (4)$$

The equation for updating positions is then replaced by the following probabilistic update equation:

$$x_{ij}(t+1) = \begin{cases} 0 & \text{if } r_{3j}(t) \geq sig(v_{ij}(t+1)) \\ 1 & \text{if } r_{3j}(t) < sig(v_{ij}(t+1)) \end{cases} \quad (5)$$

where $r_{3j}(t)$ is a random value in the range $[0,1]$.

In binary PSO, the meaning and behavior of velocity clamping differ substantially from the real-valued PSO [16]. With the velocity interpreted as a probability of change, velocity clamping sets the minimal probability for a bit to change its value from 0 to 1, or from 1 to 0. For example, if $V_{\max} = 4$ and v_{ij} is clamped with $V_{\max}$, then $sig(v_{ij}) = 0.982$ is the probability of x_{ij} to change to 1, and 0.018 the probability to change to 0. Velocity clamping, therefore, has a meaning very similar to the mutation rate in genetic algorithms [16].

In this paper, the *gbest* model of binary PSO is used for the minimization analysis of network attack graphs.

NETWORK SECURITY MODEL

The network security model is a tuple (S, H, C, T, E, R, IDS), where S is a set of services, H is a set of hosts connected to the network, C is a relation expressing connectivities among hosts, T is a relation expressing trust between hosts, E is a set of individual known exploits that an intruder can use to construct attack scenarios, R is a model of an intruder and IDS is a model of the intrusion detection system.

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.

Hosts

Each host $h \in H$ is a tuple, $(id, svcs, plvl, vuls)$, where id is a 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 host-specific vulnerable components. For simplicity, only three privilege levels are considered: None, user and root.

Network Connectivities

Network connectivities are expressed as a relation, $C \subseteq H \times H \times P$, where P is a set of port numbers. Each network connectivity $c \in C$ is a triple, (h_s,h_t,p), where h_s is the source host, h_t is the target host and p is the target port number. 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 may log in from host h_s to 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 host from which the exploit is launched, h_t is the host targeted by the exploit and $post$ specifies the effects of the exploit on the network.

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 and information gathered with port scans, etc.

Intrusion Detection System

Exploits are classified as being detectable or undetectable, with respect to the intrusion detection system (IDS). If an exploit is detectable, it will trigger an alarm

when executed on a host or network segment monitored by the IDS.

NETWORK ATTACK GRAPHS

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 $L : A \rightarrow E$ is a labelling 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, $v_1, v_2, \cdots, v_m$, such that $v_i \in V$ and $(v_i, v_{i+1}) \in A$, where $1 \leq i < m$. 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.

A typical process for generating a network attack graph is shown in Figure 1. First, vulnerability scanning tools, such as Nessus [20], determine the vulnerabilities of individual hosts. Using this vulnerability information, along with exploit templates, intruder goals and other information about the network, such as connectivity between 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.

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

For each exploit, $e_j \in E$, the *total hit value*, $hv_t(e_j)$, is defined as being the number of attack scenarios that are hit by e_j.

$$hv_t(e_j) = |\{S_k \in S | e_j \in S_k\}|. \quad (6)$$

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

$$hs(U) = \{S_k \in S | e_j \in S_k \text{ for some } e_j \in U\}. \quad (7)$$

An exploit, e_j, is *redundant*, with respect to U, if $hs(U \backslash \{e_j\}) = hs(U)$.

For each exploit, $e_j \notin U$, the *partial hit value*, $hv_p(e_j, U)$, is defined as being the number of attack scenarios that are hit by e_j, but that are not hit by any exploit in U.

$$hv_p(e_j, U) = |\{S_k \in S | e_j \in S_k \wedge S_k \notin hs(U)\}|. \quad (8)$$

A subset of exploits, $CR \subseteq E \backslash I$, is *critical* if, and only if, the intruder cannot reach his goal when the exploits in CR are removed from his arsenal. Equivalently, CR 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, $e_j \in CR$. A critical set of exploits is *minimal* if it contains no redundant exploit.

A critical set of exploits, CR, is *minimum* if there is no critical set of exploits, CR', such that $|CR'| < |CR|$.

The aim of the minimization analysis of a network attack graph is to find a minimum critical set of exploits that must be prevented to guarantee no possible attack scenario. To prevent an exploit, the security analyst may change the firewall configuration or patch the vulnerabilities that made this exploit possible.

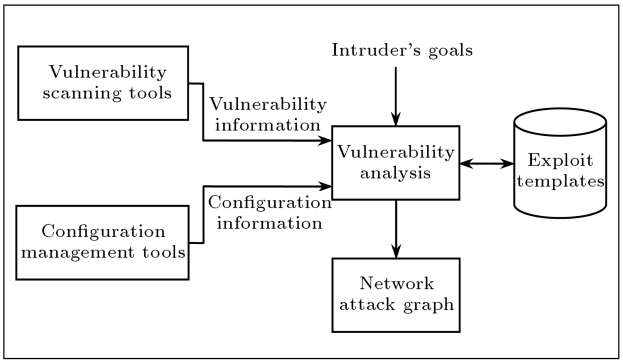


Figure 1. The process of generating a network attack graph.

SWARMNAG

In this section, SwarmNAG, a binary PSO algorithm for the minimization analysis of large-scale network attack graphs, is presented. The aim of the minimization analysis of a network attack graph is to find a minimum critical set of exploits. Any solution must be a critical set and its cardinality must be minimal.

Figure 2 shows the pseudo-code of the SwarmNAG algorithm. The first step is to initialize the swarm and control parameters, then, repeated iterations of the algorithm are executed until some termination condition is met (e.g., a maximum number of iterations is reached). Within each iteration, if each particle's current position, x_i, does not represent a critical set of exploits, a greedy mutation operator is applied to it with probability P_g. Then, redundant exploits of x_i are eliminated. After that, with probability P_l, a local search heuristic is applied to x_i, in order to improve it. Then, the particle's personal best position, y_i, is updated. The global best position, $\hat{y}$, is then determined from the entire swarm by selecting the best personal best position. Finally, the velocity and the position of each particle are updated, using Equations 2 and 5.

It should be mentioned that, in Figure 2, $U(0, 1)$ is a uniform random number between 0 and 1.

```
procedure SwarmNAG
  Set parameters, create and initialize the swarm
  while termination condition not met do
    for each particle i do
      if U(0,1) < P_g then
        Apply the greedy mutation operator to x_i;
      end
      Eliminate redundant exploits of x_i;
      if U(0,1) < P_l then
        Apply local search heuristic to x_i;
      end
      Update the personal best position y_i using Equation 1;
    end
    Update the global best position ŷ;
    for each particle i do
      Update the velocity v_i using Equation 2;
      Update the position x_i using Equation 5;
    end
  end
end SwarmNAG
```

Figure 2. The SwarmNAG algorithm.

Problem Representation

Let $E = \{e_1, e_2, \cdots, e_n\}$ be the set of preventable exploits. Each particle position, x_i, corresponds to an n-bit vector, $(x_{i1}, x_{i2}, \cdots, x_{in})$, and represents a subset of exploits, $E_i \subseteq E$, in which the exploit $e_j \in E_i$ if, and only if, the element $x_{ij} = 1$.

$$E_i = \{e_j \in E | x_{ij} = 1\}. \tag{9}$$

Let $S = \{S_1, S_2, \cdots, S_l\}$ be the set of attack scenarios represented by the network attack graph. The attack scenario $S_k \in S$ is hit by the particle position x_i, if $S_k \cap E_i \neq \emptyset$. The set of attack scenarios hit by x_i is denoted by A_i.

$$A_i = \{S_k \in S | S_k \cap E_i \neq \emptyset\}. \tag{10}$$

The particle position x_i represents a critical set of exploits if all attack scenarios are hit by it.

The aim of the minimization analysis of a network attack graph is to find a minimum critical set of exploits. This problem is a constrained optimization problem, in which the objective is to find a solution with minimum cardinality, and the constraint is that the solution must be critical (i.e., it must hit all attack scenarios). Hence, the SwarmNAG uses the following objective function to evaluate the fitness of each particle position x_i:

$$f(x_i) = z(x_i) + \lambda.h(x_i), \tag{11}$$

where $z(x_i)$ is the number of elements, x_{ij}, in particle position x_i, which are zero. The higher the value of $z(x_i)$, the smaller the cardinality of the set of exploits represented by x_i:

$$z(x_i) = |E| - |E_i|. \tag{12}$$

$h(x_i)$ is the number of attack scenarios hit by particle position x_i:

$$h(x_i) = |A_i|, \tag{13}$$

and λ is the penalty coefficient. If λ is too small, not enough emphasis is placed on preventing violation of the constraint. Hence, non-critical solutions may then be found. On the other hand, if λ is too large, the algorithm may get trapped in local optima.

Accordingly, a time-varying penalty coefficient is used, where an initially small penalty coefficient is linearly increased to a large value:

$$\lambda(t) = \lambda(0) + (\lambda(t_{\max}) - \lambda(0))\frac{t}{t_{\max}}, \tag{14}$$

where $t_{\max}$ is the maximum number of iterations for which the algorithm is executed, $\lambda(0)$ is the initial penalty coefficient, $\lambda(t_{\max})$ is the final penalty coefficient and $\lambda(t)$ is the penalty coefficient at iteration t. Note that $\lambda(0) < \lambda(t_{\max})$. Typically, the penalty coefficient is set to $0.1 \leq \lambda \leq 1.9$.

Time-Varying Velocity Clamping (TVVC)

In binary PSO, the velocity is interpreted as a probability of change. Hence, the velocity clamping sets the minimal probability for a bit to change its value [16].

If $V_{\max}$ is a small value, it provides a bigger chance for a bit to change its value (i.e., exploring the search space), while, if $V_{\max}$ is large, it allows particles to converge on a solution (i.e., exploiting the search space). Accordingly, a time-varying velocity clamping is used:

$$V_{\max}(t) = V_{\max}(0) + (V_{\max}(t_{\max}) - V_{\max}(0))\frac{t}{t_{\max}}, \tag{15}$$

where $t_{\max}$ is the maximum number of iterations, $V_{\max}(0)$ is the initial velocity clamping, $V_{\max}(t_{\max})$ is the final velocity clamping and $V_{\max}(t)$ is the velocity clamping at iteration t.

In following sections, the effect of time-varying velocity clamping on the performance of the SwarmNAG will be shown.

Greedy Mutation

At each iteration, if each particle's current position, x_i, does not represent a critical set of exploits, a greedy mutation operator is applied to it with probability P_g.

```
procedure GreedyMutation (x_i)
  E_i = {e_j ∈ E|x_ij = 1};
  if x_i does not represent a critical set of exploits then
    Choose an exploit e_k ∉ E_i such that it has the maximum
    partial hit value hv_p(e_k, E_i);
    E_i = E_i ∪ {e_k};
    x_ik = 1;
    v_ik = V_max;
  end if;
  return x_i;
end procedure
```

Figure 3. The greedy mutation operator.

As shown in Figure 3, the greedy mutation first chooses an exploit, $e_k \notin E_i$, that has the maximum partial hit value, $hv_p(e_k, E_i)$, then adds it to E_i and changes the value of its corresponding element, x_{ik} of x_i, to 1.

The greedy mutation uses heuristic information and helps the algorithm to choose exploits that have more hits with attack scenarios.

Elimination of Redundant Exploits

The set of exploits represented by particle position x_i may contain redundant exploits, which must be eliminated.

Let E_i be the set of exploits represented by x_i and A_i be the set of attack scenarios hit by x_i. For each exploit, e_j, the *exclusive hit value*, $hv_x(e_j, E_i, A_i)$, is defined as being the number of attack scenarios, $S_k \in A_i$, that are hit by e_j, but that are not hit by any exploit in $E_i \backslash \{e_j\}$. The exploit, e_j, is called *candidate redundant*, with respect to E_i, if $hv_x(e_j, E_i, A_i) = 0$. The set of candidate redundant exploits of E_i is denoted by R_i.

$$R_i = \{e_j \in E_i | hv_x(e_j, E_i, A_i) = 0\}. \quad (16)$$

The exclusive hit value is used to define the *selection value*, $sv(e_j, E_i)$, of a candidate redundant exploit, $e_j \in R_i$.

$$sv(e_j, E_i) = \sum_{e_k \in E_i \backslash \{e_j\}} hv_x(e_k, E_i \backslash \{e_j\}, A_i). \quad (17)$$

The selection value is used to evaluate the candidate redundant exploits of a set of exploits, in order to choose a candidate redundant exploit for removal from it.

In Figure 4, an algorithm is presented, which can be used to eliminate redundant exploits of x_i. The algorithm is based on the idea that it is good to remove an exploit, e_k, from E_i, if e_k is a candidate redundant exploit and hits attack scenarios that are hit by too many other exploits in E_i. Hence, the algorithm removes at each step a candidate redundant exploit

```
procedure Eliminate Redundants (x_i)
  E_i = {e_j ∈ E|x_ij = 1};
  R_i = {e_j ∈ E_i|hv_x(e_j, E_i, A_i) = 0};
  while R_i ≠ ∅ do
    Choose an exploit e_k ∈ R_i such that it has
    the minimum selection value sv(e_k, R_i);
    E_i = E_i\{e_k};
    x_ik = 0;
    v_ik = −V_max;
    R_i = {e_j ∈ E_i|hv_x(e_j, E_i, A_i) = 0};
  end while;
  return x_i;
end procedure
```

Figure 4. The procedure of eliminating redundant exploits.

that has the minimum selection value. This is repeated until a set of exploits without redundant exploits is obtained.

Local Search Heuristic

It has been shown in many empirical studies that global optimization algorithms lack exploitation abilities in later stages of the optimization process. This is also true for the basic PSO, as shown in [21-23]. However, it provides mechanisms to balance exploration and exploitation through proper setting of the inertia weight, acceleration coefficients and velocity clamping. Many variations of the basic PSO have been proposed to address this problem [16]. Most of them first allow the algorithm to explore new regions and, when a good region is located, allow the algorithm to exploit the search space to refine solutions. This is a sequential approach to balancing exploration and exploitation.

Another approach is to embed a local optimizer in between iterations of the global search heuristics. By doing this, exploration and exploitation occur in parallel [16]. Such hybrids of local and global search heuristics have been studied extensively in the evolutionary computation paradigm [24] and are generally referred to as memetic algorithms [25].

Al-Kazemi and Mohan [26] implemented a basic hill-climbing heuristic in their multi-phase PSO. Particle positions are only updated if the new position improves on the fitness of the previous position. Yin [27] used a basic hill-climbing heuristic within a discrete PSO to find the optimal set of polygons to approximate digital curves. In this approach, each vertex of the polygons is adjusted sequentially to see if a better fitness is obtained.

In the SwarmNAG, a local search heuristic is probabilistically applied to the current position of each particle to improve them, before their personal best

```
procedure LocalSearch (x_i)
    E_i = {e_j ∈ E|x_ij = 1};
    while improvement is possible do
        Choose an exploit e_k ∉ E_i such that g(e_k) > 0;
        E_i = E_i ∪ {e_k};
        x_ik = 1;
        v_ik = V_max;
        Eliminate redundant exploits of x_i;
    end while;
    return x_i;
end procedure
```

Figure 5. The local search heuristic procedure.

positions are updated. The probability of a local search heuristic is denoted by P_l.

The local search heuristic is based on the following idea: Given a particle position, x_i, and its corresponding subset of exploits, E_i, suppose there is an exploit, $e_k \notin E_i$, such that $E_i \cup \{e_k\}$ contains at least two exploits other than e_k, say $e_{j_1}, \cdots, e_{j_l}$, with $l \geq 2$ that are redundant. Then, $(E_i \backslash \{e_{j_1}, \cdots, e_{j_l}\}) \cup \{e_k\}$ is a better subset of exploits than E_i. The gain of exploit e_k, with respect to E_i, is $g(e_k) = l - 1$. In this case, e_k is called a *candidate dominant* exploit.

As shown in Figure 5, the local search heuristic first chooses a candidate dominant exploit e_k and changes its corresponding element, x_{ik}, to 1. It then eliminates the redundant exploits of the new position, using the algorithm already presented for eliminating redundant exploits. This process is repeated until no further improvement is possible.

PERFORMANCE MEASURES

This section presents two different measures used to evaluate the performance of the SwarmNAG.

Accuracy

Accuracy refers to the quality of the solution obtained, which is represented by the global best solution. The accuracy of the swarm at iteration t is simply the fitness of the global best position,

$$\text{accuracy}(t) = f(\hat{y}(t)), \tag{18}$$

where $\hat{y}(t)$ is the global best position at iteration t.

Diversity

Diversity is an important measure that may be used to describe the amount of exploration a PSO algorithm still performs and to detect stagnation situations. Large diversity implies that a large area of the search space can be explored. In simple terms, diversity can be defined as the degree of dispersion of particles. A diversity measure is defined based on the Hamming distance between particle positions in the swarm,

$$\text{diversity}(t) = \frac{2}{n_s(n_s-1)} \sum_{i=1}^{n_s} \sum_{j=i+1}^{n_s} H(x_i(t), x_j(t)), \tag{19}$$

where n_s is the swarm size and $H(x_i(t), x_j(t))$ is the number of different bits between the particle positions, x_i and x_j, at iteration t.

EXPERIMENTS

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

Sample Network Attack Graph

Consider the network shown in Figure 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. One firewall separates the internal network from the DMZ and another firewall separates the DMZ from the rest of the Internet.

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 [20], can be used to find the vulnerabilities of each host.

In Table A1 of Appendix A, different types of services and vulnerabilities available on the network hosts are introduced.

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 and the Squid web proxy is a full-featured

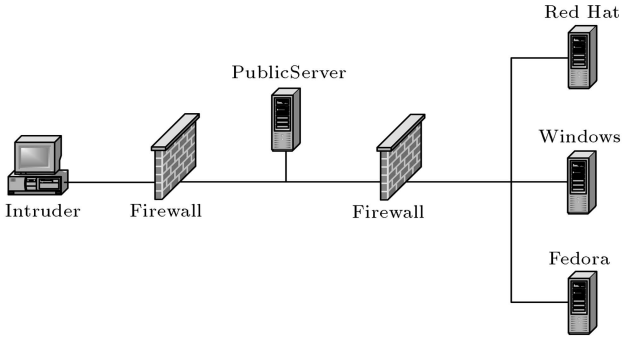


Figure 6. An example network.

web proxy cache that 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 among 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 host and h_t is the target host. Every entry has five Boolean values. These values are 'T', if host h_s can connect to host h_t on the ports of http, licq, ftp, ssh and smtp, respectively.

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

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 Appendix B, the description of each generic exploit is given in Table B1, and in Table B2 each generic exploit is represented by its preconditions and postconditions. More information about each of the exploits is available in [28]. Before an exploit can be used, its preconditions must be met. Each exploit will increase the network vulnerability if it is successful.

Among the ten generic exploits shown in Table B2, the first eight generic exploits require a pair of hosts and the last two generic exploits require only one host. Therefore, there are $8*5*4+2*4 = 168$ exploits in total, 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 by 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 consists of between 4 to 9 exploits.

Experimental Results

The SwarmNAG was applied for the minimization analysis of the above network attack graph. To evaluate the performance of the algorithm, several experiments were performed. In the first experiment, it was assumed that all exploits are preventable. Therefore, the aim was to find a minimum critical set of exploits among 168 exploits. Using the SwarmNAG, the following minimum critical set of exploits was found:

CR = {iis_r2r(Intruder,PublicServer),
exchange_r2u(Intruder, PublicServer)}.

In the second experiment, it was 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 SwarmNAG, the following minimum critical set of exploits was found:

CR = {licq_r2u(PublicServer, Fedora),
licq_r2u(RedHat, Fedora),
script_r2u(PublicServer, Windows),
ftp_rhosts(PublicServer, Fedora),
ftp_rhosts(RedHat, Fedora)}.

It should be mentioned that the exact cardinality of the minimum critical set of exploits for this network

Table 1. Network connectivity information.

Host	Intruder	PublicServer	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
PublicServer	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

attack graph is 5, so, the above critical set of exploits found by the SwarmNAG is minimum. While using the approximation algorithm proposed by Sheyner et al. [3] and Jha et al. [4,5], the following minimum critical set of exploits was found:

CR = {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 that SwarmNAG can find a critical set of exploits with less cardinality.

In the experiments, the parameters were set to $c_1 = 2$ and $c_2 = 2$, which are values commonly used in the binary PSO literature. The swarm size was set to $m = 15$ and the maximum number of iterations was set to 150. The penalty coefficient was set to $0.1 \leq \lambda \leq 1.9$ and the velocity clamping was set to $2 \leq V_{\max} \leq 4.5$. The probability of greedy mutation and the probability of local search were set to $P_g = 0.90$ and $P_l = 0.90$, respectively.

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. 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 [29].

In order to further evaluate the performance of the SwarmNAG, 12 large-scale network attack graphs, denoted by $\text{NAG}_1, \text{NAG}_2, \cdots, \text{NAG}_{12}$, were generated. For each network attack graph, different values for the cardinalities of E and S were considered, where E is the set of known exploits and S is the set of attack scenarios represented by the network attack graph. In $\text{NAG}_1, \cdots, \text{NAG}_6$, attack scenarios consist of between 3 to 9 exploits, while in $\text{NAG}_7, \cdots, \text{NAG}_{12}$, attack scenarios consist of between 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 large-scale network attack graph.

Experimental Results

The SwarmNAG was applied for the minimization analysis of the above large-scale network attack graphs. 10 runs of each algorithm were performed, with different random seeds, and the best cardinality and the average cardinality of the critical sets of exploits obtained from these 10 runs were reported. The approximation algorithm proposed by Sheyner et al. [3] and Jha et

Table 2. Large-scale network attack graphs.

Network Attack Graph	Cardinality of the Set of Exploits ($\|E\|$)	Cardinality of the Set of Attack Scenarios ($\|S\|$)	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	600	6000	6.03
NAG_6	600	8000	5.95
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}	600	6000	7.53
NAG_{12}	600	8000	7.55

Table 3. The cardinality of critical set of exploits.

Network Attack Graph	SwarmNAG		SwarmNAG Without LS		Approximation Algorithm [3-5]
	Best	Average	Best	Average	
NAG_1	44	45.1	45	46.3	50
NAG_2	88	89.3	90	90.9	98
NAG_3	177	180.2	181	185.1	197
NAG_4	198	202.1	206	208	221
NAG_5	269	273	282	283.6	296
NAG_6	294	297.2	306	309.7	317
NAG_7	39	39.8	39	40.5	45
NAG_8	81	82.3	81	83.8	91
NAG_9	159	162.4	165	168.3	182
NAG_{10}	181	183.8	185	189.4	200
NAG_{11}	243	246.1	252	255.3	267
NAG_{12}	264	266.3	273	277.2	293

al. [4,5] was also applied to analyze the above network attack graphs. Table 3 shows the results.

As shown in Table 3, the SwarmNAG outperforms the approximation algorithm and finds a critical set of exploits with less cardinality. Also, the SwarmNAG performs better than the SwarmNAG without the local search heuristic.

In the experiments, the parameters were set to $c_1 = 2$ and $c_2 = 2$, which are values commonly used in binary PSO literature. The swarm size was set to $m = 15$, the penalty coefficient was set to $0.1 \leq \lambda \leq 1.9$ and the velocity clamping was set to $2 \leq V_{\max} \leq 4.5$. The probability of greedy mutation and the probability of local search were set to $P_g = 0.90$ and $P_l = 0.90$, respectively. Also, the maximum number of iterations was set to 150 for the minimization analysis of NAG_1 and NAG_7, 300 for the minimization analysis of NAG_2 and NAG_8, 600 for the minimizatoin analysis of NAG_3, NAG_4, NAG_9 and NAG_{10}, and 900 for the minimizatoin analysis of NAG_5, NAG_6, NAG_{11} and NAG_{12}.

Figures 7 and 8 show the progress of the number of attack scenarios hit by the global best position of the best run and the number of exploits corresponding to that position in the experiments for the minimizatoin analysis of NAG_4 and NAG_{11}, respectively. The number of attack scenarios hit by the global best position is expected to be as large as possible, while the number of exploits corresponding to that position is expected to be as small as possible.

As mentioned before, diversity is an important measure that may be used to describe the amount of exploration a PSO algorithm performs. Large diversity implies that a large area of the search space can be explored.

Figures 9 to 11 show the average diversity of the SwarmNAG and the SwarmNAG without TVVC, obtained from 10 runs of the SwarmNAG and 10 runs of the SwarmNAG without TVVC in the experiments for the minimizatoin analysis of NAG_3, NAG_8 and NAG_{11}, respectively. For the SwarmNAG without TVVC, the velocity clamping was fixed to $V_{\max} = 4$.

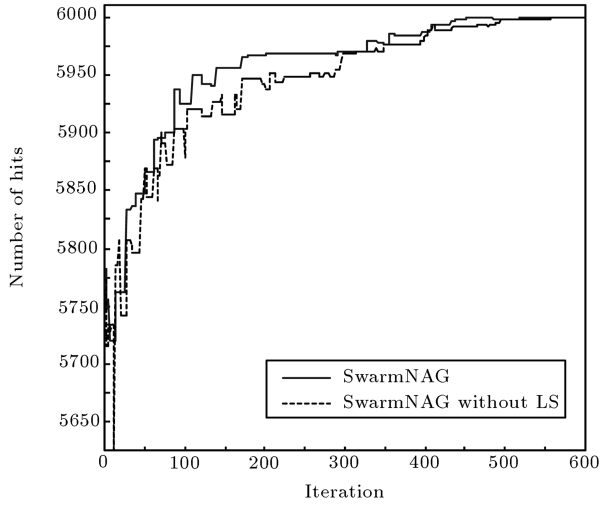
As Figures 9 to 11 show, the SwarmNAG explores the search space better than the SwarmNAG without TVVC.

Figure 12 shows the progress of the average number of attack scenarios hit by the global best position and the average number of exploits corresponding to that position, obtained from 10 runs of the SwarmNAG and 10 runs of the SwarmNAG without TVVC in the experiment for the minimizatoin analysis of NAG_{11}.

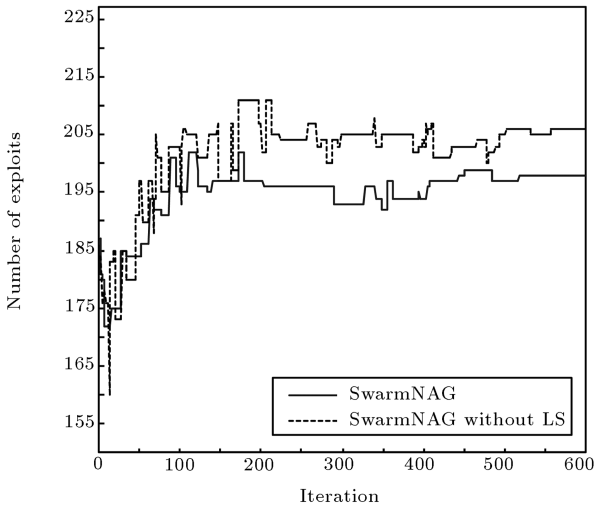
As the above figures show, the SwarmNAG performs better than the SwarmNAG without TVVC and finds a critical set of exploits with less cardinality.

CONCLUSIONS

Each attack scenario is a sequence of exploits launched by an intruder towards a particular goal. The collection of possible attack scenarios in a computer network can be represented by a directed graph, called a network attack graph (NAG). In this directed graph, each path, from an initial node to a goal node, corresponds to an attack scenario. The aim of the minimization analysis of a network attack graph is to find a minimum critical set of exploits that completely disconnect the initial nodes and the goal nodes of the graph. This problem is, in fact, a constrained optimization problem, the objective of which is to find a solution with minimum

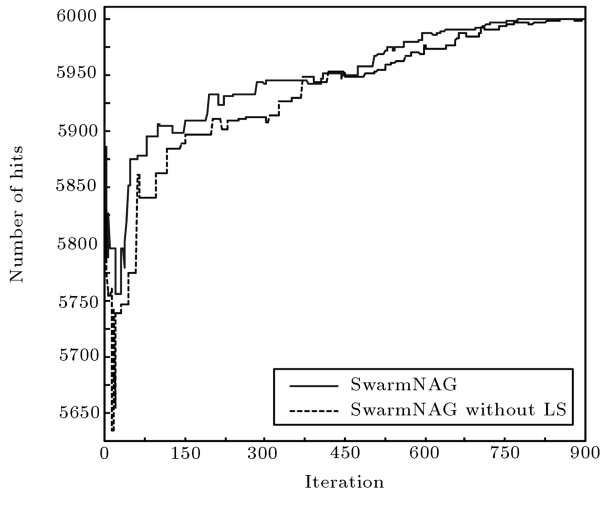


(a) The number of attack scenarios hit by the global best position of the best run

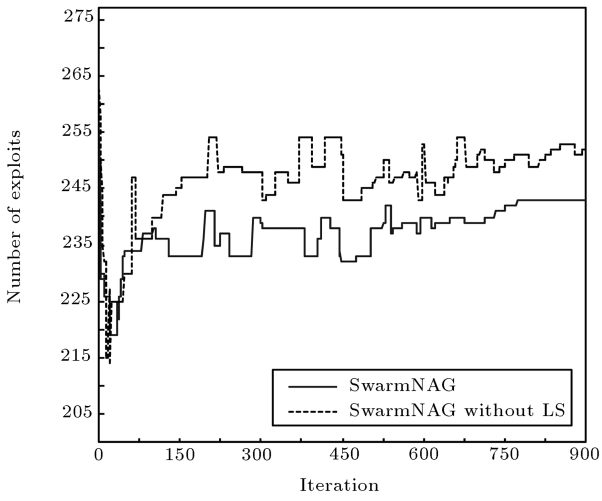


(b) The number of exploits corresponding to the global best position of the best run

Figure 7. Comparison of the performance of the SwarmNAG and the SwarmNAG without local search heuristic for the minimizatoin analysis of NAG_4.



(a) The number of attack scenarios hit by the global best position of the best run



(b) The number of exploits corresponding to the global best position of the best run

Figure 8. Comparison of the performance of the SwarmNAG and the SwarmNAG without local search heuristic for the minimization analysis of NAG_{11}.

cardinality and the constraint is that the solution must be critical.

In this paper, a binary PSO algorithm, called SwarmNAG, was presented, for the minimizatoin analysis of large-scale network attack graphs. A penalty function method with a time-varying penalty coefficient was used to convert the constrained optimization problem into an unconstrained one. Also, a time-varying velocity clamping, a greedy mutation operator and a local search heuristic were used to improve the overall performance of the algorithm. The results of applying the above algorithms were reported, in order to analyze several large-scale network attack graphs. The approximation algorithm proposed by Sheyner et al. [3] and Jha et al. [4,5] was also applied, to analyze the above large-scale network attack graphs. On average, the cardinality of critical sets of exploits found by the SwarmNAG was 8.89% less than the cardinality of critical sets of exploits found by the approximation algorithm.

The results of the experiments show that the SwarmNAG can be successfully used for the minimization analysis of network attack graphs.

ACKNOWLEDGMENTS

This work was supported in part by ITRC.

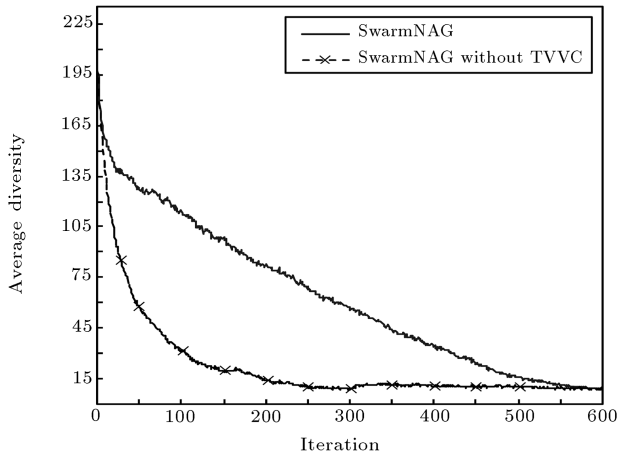


Figure 9. Comparison of the average diversity of the SwarmNAG and the SwarmNAG without TVCC in the experiments for the minimization analysis of NAG_3.

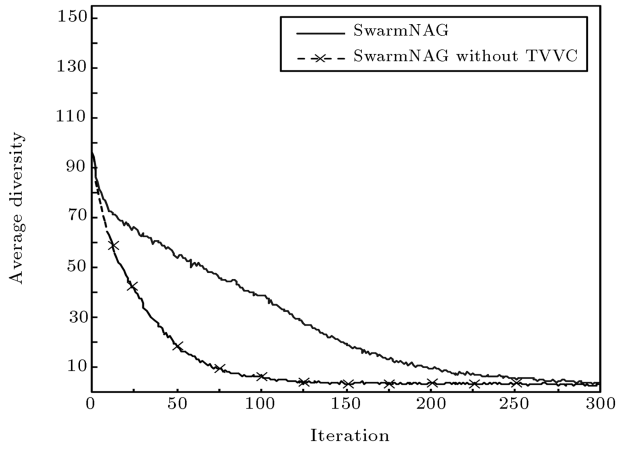


Figure 10. Comparison of the average diversity of the SwarmNAG and the SwarmNAG without TVCC in the experiments for the minimization analysis of NAG_8.

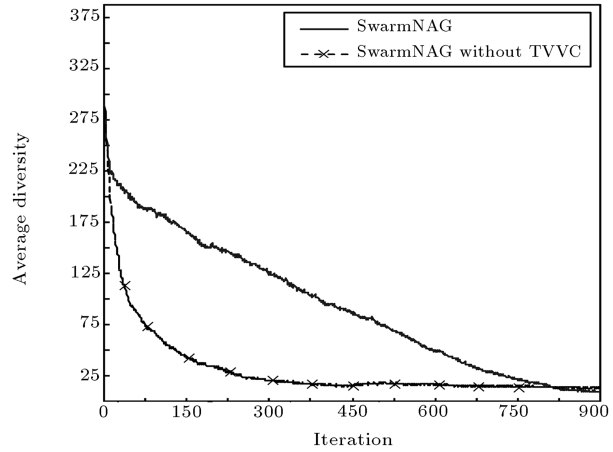
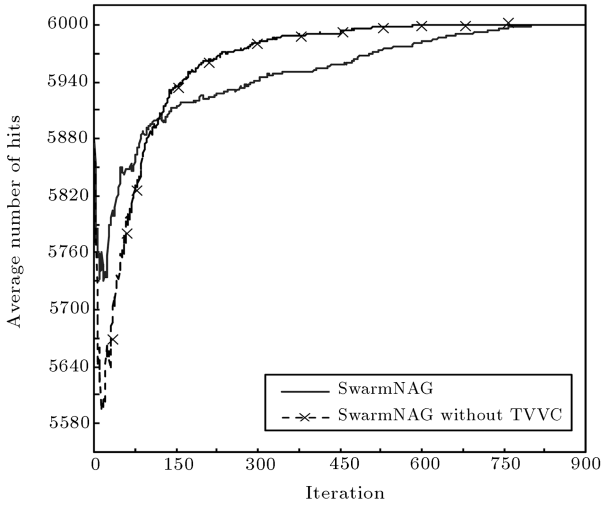
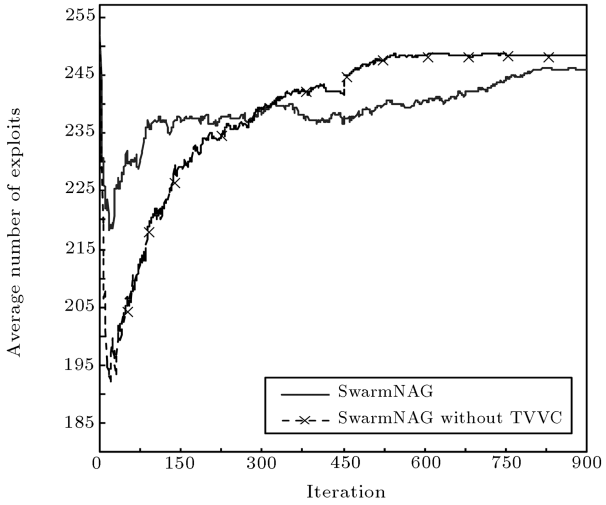


Figure 11. Comparison of the average diversity of the SwarmNAG and the SwarmNAG without TVCC in the experiments for the minimization analysis of NAG_{11}.



(a) The average number of attack scenarios hit by the global best position



(b) The average number of exploits corresponding to the global best position

Figure 12. Comparison of the performance of the SwarmNAG and the SwarmNAG without TVVC for the minimization analysis of NAG_{11}.

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*, Shiraz, Iran, pp. 81-86 (2005).

2. Phillips, C. and Swiler, L.P. "A graph-based system for network-vulnerability analysis", In *Proceedings of the 1998 Workshop on New Security Paradigms*, Charlottesville, VA, USA, pp. 71-79 (1998).

3. 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, Berkeley, CA, USA, pp. 273-284 (2002).

4. Jha, S., Sheyner, O. and Wing, J.M. "Minimization and reliability analysis of attack graphs", Technical Report CMU-CS-02-109, School of Computer Science, Carnegie Mellon University (2002).
5. Jha, S., Sheyner, O. and Wing, J.M. "Two formal analyses of attack graphs", In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, Cape Breton, Nova Scotia, Canada, pp. 49-63 (2002).
6. NuSMV "NuSMV: A new symbolic model checker", http://afrodite.itc.it:1024/~nusmv/.
7. 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*, Washington DC, USA, pp. 217-224 (2002).
8. 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)*, Minneapolis, MN, USA, pp. 99-106 (2005).
9. Mehta, V., Bartzis, C., Zhu, H., Clarke, E.M. and Wing, J.M. "Ranking attack graphs", In *Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID 2006)*, Hamburg, Germany, pp. 127-144 (2006).
10. Kennedy, J. and Eberhart, R.C. "Particle swarm optimization", In *Proceedings of the IEEE International Joint Conference on Neural Networks*, Perth, Australia, pp. 1942-1948 (1995).
11. Kennedy, J., Eberhart, R.C. and Shi, Y., *Swarm Intelligence*, Morgan Kaufmann, San Mateo, CA, USA (2001).
12. Parsopoulos, K.E. and Vrahatis, M.N. "Particle swarm optimizer in noisy and continuously changing environments", *Artificial Intelligence and Soft Computing*, M.H. Hamza, Ed., IASTED/ACTA Press, Anaheim, CA, USA, pp. 289-294 (2001).
13. Vlachogiannis, J.G. and Lee, K.Y. "A comparative study on particle swarm optimization for optimal steady-state performance of power systems", *IEEE Transactions on Power Systems*, **21**(4), pp. 1718-1728 (2006).
14. Vlachogiannis, J.G. "Constricted local-neighborhood particle swarm optimization with passive congregation applied in reactive power and voltage control", *Electric Power Components and Systems*, **34**(5), pp. 509-520 (2006).
15. Hereford, J.M., Siebold, M. and Nichols, S. "Using the particle swarm optimization algorithm for robotic Search applications", In *Proceedings of the 2007 IEEE Swarm Intelligence Symposium*, Honolulu, HI, USA, pp. 53-59 (2007).
16. Engelbrecht, A.P., *Fundamentals of Computational Swarm Intelligence*, John Wiley & Sons, Hoboken, NJ, USA (2005).
17. Eberhart, R.C., Simpson, P. and Dobbins, R., *Computational Intelligence PC Tools*, Academic Press, San Diego, CA, USA (1996).
18. Shi, Y. "Particle Swarm Optimization", *IEEE Connections*, **2**(1), pp. 8-13 (2004).
19. Kennedy, J. and Eberhart, R.C. "A discrete binary version of the particle swarm algorithm", In *Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics*, Orlando, FL, USA, pp. 4104-4109 (1997).
20. Deraison, R. "Nessus Vulnerability Scanner", http://www.nessus.org.
21. Shi, Y. and Eberhart, R.C. "Empirical study of particle swarm optimization", In *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, Washington DC, USA, pp. 1945-1950 (1999).
22. Hendtlass, T. and Randall, M. "A survey of ant colony and particle swarm meta-heuristics and their application to discrete optimisation problems", In *Proceedings of the Inaugural Workshop on Artificial Life*, Adelaide, Australia, pp. 15-25 (2001).
23. Braendler, D. and Hendtlass, T. "The suitability of particle swarm optimisation for training neural hardware", *In Proceedings of the 15th International Conference on Industrial and Engineering, Applications of Artificial Intelligence and Expert Systems*, Cairns, Australia, LNCS 2358, pp. 190-199 (2002).
24. Eiben, A.E. and Smith, J.E., *Introduction to Evolutionary Computing*, Springer-Verlag, Berlin, Germany (2003).
25. Krasnogor, N., Aragon, A. and Pacheco, J. "Memetic Algorithms", *Metaheuristic Procedures for Training Neural Networks*, E. Alba and R. Martí, Eds., Springer-Verlag, Berlin, Germany, pp. 225-248 (2006).
26. Al-Kazemi, B. and Mohan, C.K. "Multi-phase discrete particle swarm optimization", In *Proceedings of the 4th International Workshop on Frontiers in Evolutionary Algorithms*, Research Triangle Park, NC, USA, pp. 622-625 (2002).
27. Yin, P.-Y. "A discrete particle swarm algorithm for optimal polygonal approximation of digital curves", *Journal of Visual Communication and Image Representation*, **15**(2), pp. 241-260 (2004).
28. CVE list; CVE: Common Vulnerabilities and Exposures: http://www.cve.mitre.org.
29. 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)*, Tucson, AZ, USA, pp. 72-84 (2005).

APPENDIX A

Description of Vulnerabilities

APPENDIX B

Description of Exploits

Table A1. 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)	SSH 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
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

Table B1. 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 ssh 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

Table B2. Exploit templates.

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