6

A Memetic Particle Swarm Optimization Algorithm for Network Vulnerability Analysis

Mahdi Abadi and Saeed Jalili
Tarbiat Modares University
Tehran, Iran

1. Introduction

As computer networks continue to grow, it becomes increasingly more important to automate the process of evaluating their vulnerability to attacks. 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. This is because intruders often combine exploits against multiple vulnerabilities in order to reach their goals (Abadi & Jalili, 2005). 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.

(Phillips & Swiler, 1998) 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. (Sheyner et al., 2002) used a modified version of the model checker NuSMV (NuSMV, 2010) to produce attack graphs. (Ammann et al., 2002) introduced a monotonicity assumption and used 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 (Phillips & Swiler, 1998), 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., 2005) presented a number of techniques for managing attack graph complexity through visualization. (Mehta et al., 2006) presented a ranking scheme for the nodes of an attack graph. 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.

(Ou et al., 2006) presented logical attack graphs, which directly illustrate logical dependencies among attack goals and configuration information. Their attack graph generation tool builds upon MulVAL (Ou et al., 2005), a network security analyzer based on logical programming.

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.

(Sheyner et al., 2002) and (Jha et al., 2002) showed this problem is in fact *NP*-hard. They proposed an approximation algorithm, ApproxNAG, that can find an approximately-optimal set of exploits, which must be prevented to thwart an intruder. (Abadi & Jalili, 2006) and (Abadi & Jalili, 2008) presented an ant colony optimization algorithm, AntNAG, and a genetic algorithm, GenNAG, for minimization analysis of network attack graphs.
While it is currently possible to generate very large and complex network attack graphs, relatively little work has been done for analysis of them.
Particle swarm optimization (PSO) (Kennedy & Eberhart, 1995) is a population based stochastic optimization algorithm that was inspired by social behaviour of flocks of birds when they are searching for food.
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 (Shi & Eberhart, 1999); (Hendtlass & Randall, 2001); (Braendler & Hendtlass, 2002), however, it provides mechanisms to balance exploration and exploitation through proper settings of the inertia weight, acceleration coefficients and velocity clamping. Many variations of the basic PSO have been proposed to address this problem (Engelbrecht, 2005). 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 (Engelbrecht, 2005).
Another approach is to embed a local optimizer in between the iterations of the global search heuristics. By doing this, exploration and exploitation occur in parallel (Engelbrecht, 2005). Such hybrids of local and global search heuristics have been studied elaborately in the evolutionary computation paradigm (Eiben & Smith, 2003), and are generally referred to as *memetic algorithms* (Krasnogor et al., 2006). While evolutionary algorithms take inspiration from biological evolution, memetic algorithms mimic cultural evolution. The term *meme* refers to a unit of cultural information that can be transmitted from one mind to another after reinterpretation and improvement that in the context of combinatorial optimization corresponds to local search.
In this paper, we present a memetic PSO algorithm, called ParticleNAG, for minimization analysis of large-scale network attack graphs (NAGs). We also compare the performance of ParticleNAG with ApproxNAG (Sheyner et al., 2002); (Jha et al., 2002), AntNAG (Abadi & Jalili, 2006), and GenNAG (Abadi & Jalili, 2008) for minimization analysis of several large-scale network attack graphs.
The remainder of this paper is organized as follows: Section 2 provides an overview of PSO, Section 3 introduces our network security model, and Section 4 describes the process of minimization analysis of network attack graphs. Section 5 presents ParticleNAG. Section 6 reports the experimental results and finally Section 7 draws some conclusions.

2. Particle swarm optimization

Particle swarm optimization (PSO) is a population based stochastic optimization. It was inspired by social behaviour of flocks of birds when they are searching for food. In PSO, the potential solutions, called *particles*, fly through the problem space exploring for better regions. The position of a particle is influenced by the best position visited by itself and the position of the best particle in its neighbourhood. When the neighbourhood of a particle is the entire swarm, the best position in the neighbourhood is referred to as the *global best particle*, and the

resulting algorithm is referred to as a *gbest* PSO. When smaller neighbourhoods are used, the algorithm is generally referred to as a *lbest* PSO (Kennedy et al., 2001).

The performance of each particle is measured using 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

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

For the *gbest* model, the best particle is determined from the entire swarm by selecting the best personal best position. This position is denoted as $\hat{y}$. The equation that manipulates the velocity is called the *velocity update equation* and is stated as

$$\begin{aligned} v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)(y_{ij}(t) - x_{ij}(t)) + \\ c_2 r_{2j}(t)(\hat{y}_j(t) - x_{ij}(t)) \end{aligned} \tag{2}$$

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

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 (Eberhart et al., 1996); (Shi, 2004). Each particle updates its position using the following equation:

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{3}$$

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 is satisfied.

(Kennedy & Eberhart, 1997) have adapted PSO to search in binary spaces. For 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 by using a sigmoid function to squash velocities into a [0,1] range. The sigmoid function is defined as

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

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

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

where $r_{3j}(t)$ is a random value in the range [0,1].
In binary PSO, the meaning and behaviour of velocity clamping differ substantially from real-valued PSO. With the velocity interpreted as a probability of change, velocity clamping, $V_{\max}$, sets the minimal probability for a bit to change its value from 0 to 1, or from 1 to 0 (Engelbrecht, 2005).
In this paper, we use the *gbest* model of binary PSO for minimization analysis of network attack graphs.

3. Network security model

Our network security model is a tuple (*S*, *H*, *C*, *T*, *E*, *M*, *R*), where *S* is a set of services, *H* is a set of hosts connected to the network, *C* is a relation expressing connectivity between hosts, *T* is a relation expressing trust between hosts, *E* is a set of individual known exploits that intruder can use to construct attack scenarios, *M* is a set of countermeasures that must be implemented to prevent exploits, 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.

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, we only consider three privilege levels: *none*, *user*, and *root*.

Network Connectivity

Network connectivity is modelled 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 network elements such as firewalls that restrict the ability of one host to connect to another.

Trust Relationships

Trust relationships are modelled 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 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.

Countermeasures

To prevent an exploit $e \in E$, the security analyst must implement a suitable countermeasure $m \in M$, such as

- changing the firewall configuration
- patching the vulnerability that made this exploit possible
- deploying a host-based or network-based intrusion detection and prevention system
- modifying the configuration of network services and applications
- deleting user accounts
- changing access rights
- setting up a virtual private network (VPN)

Intruder

The intruder has some knowledge about the target network, such as known vulnerabilities, user passwords, and information gathered with port scans. The intruder's knowledge is modelled as a relation $R \subseteq ID \times PW \times VUL \times INF$, where ID is a set of host identifiers, PW is a set of user passwords, VUL is a set of known vulnerabilities, and INF is a set of information gathered through port scans and operating system identification techniques.

4. Minimization analysis of network attack graphs

Let $E = \{e_1, e_2, ..., e_n\}$ be the set of exploits, $I \subseteq E$ be the set of inevitable exploits, $M = \{m_1, m_2, ..., m_p\}$ be the set of countermeasures, and $prv : M \to 2^{E \setminus I}$ be a function. An exploit $e_j \in prv(m_i)$ if and only if implementing the countermeasure m_i prevents the exploit e_j.

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 labelling function, where $L(a) = e_j$ if and only if an edge $a = (v, v')$ corresponds to an exploit $e_j \in E$. 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 $S = \{S_1, S_2, ..., S_l\}$ 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 $e_j \in S_k$.

Definition 1. Total Hit Value

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

$$hv_t(e_j) = \left| \{ S_k \in S \mid e_j \in S_k \} \right| \tag{6}$$

Definition 2. Redundant Exploit

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

$$hs(U) = \{ S_k \in S \mid e_j \in S_k \text{ for some } e_j \in U \} \tag{7}$$

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

Definition 3. Partial Hit Value

Let $U \subseteq E$ be a subset of exploits. For each exploit $e_j \notin U$, the partial hit value $hv_p(e_j, U)$ is defined to be the number of attack scenarios that are hit by e_j, but that are not hit by any exploit in U.

$$hv_p(e_j, U) = \left| \left\{ S_k \in S \mid e_j \in S_k \wedge S_k \notin hs(U) \right\} \right| \tag{8}$$

Definition 4. Exclusive Hit Value

Let $U \subseteq E$ be a subset of exploits. For each exploit $e_j \in U$, the exclusive hit value $hv_x(e_j, U)$ is defined to be the number of attack scenarios that are hit by e_j, but that are not hit by any exploit in $U \setminus \{e_j\}$.

Definition 5. Critical Set of Exploits

A subset of exploits $CE \subseteq E \setminus I$ is critical if and only if all attack scenarios are hit by the exploits in it. Equivalently, CE is critical if and only if every complete path from an initial node to a goal node of the network attack graph G has at least one edge labelled with an exploit $e_j \in CE$.

Definition 6. Minimal Critical Set of Exploits

A critical set of exploits CE is minimal if it contains no redundant exploit.

Definition 7. Minimum Critical Set of Exploits

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

Definition 8. Critical Set of Countermeasures

A subset of countermeasures $CM \subseteq M$ is critical if and only if all attack scenarios are prevented by implementing the countermeasures in it. Equivalently, CM is critical if and only if every complete path from an initial node to a goal node of the network attack graph G has at least one edge labelled with an exploit $e_j \in es(CM)$, where $es(CM)$ is the set of exploits prevented by implementing the countermeasures in CM.

$$es(CM) = \bigcup_{m_i \in CM} prv(m_i) \tag{9}$$

Definition 9. Minimal Critical Set of Countermeasures

A critical set of countermeasures CM is minimal if it contains no redundant countermeasure.

Definition 10. Minimum Critical Set of Countermeasures

A critical set of countermeasures CM is minimum if there is no critical set of countermeasures CM' such that $|CM'| < |CM|$.

In general, there can be multiple minimum critical set of exploits/countermeasures. We can now state formally two problems: MCEP and MCCP (Sheyner et al., 2002); (Jha et al., 2002).

Definition 11. Minimum Critical Set of Exploits Problem (MCEP)

Given a network attack graph G and a set of exploits E, find a minimum critical subset of exploits $CE \subseteq E \setminus I$ for G.

Definition 12. Minimum Critical Set of Countermeasures Problem (MCCP)

Given a network attack graph G, a set of exploits E, and a set of countermeasures M, find a minimum critical subset of countermeasures $CM \subseteq M$ for G.

There is a trivial reduction from MCEP to MCCP, and vice versa. Given an instance (G, E) of MCEP, we can construct an instance (G, E, M) of MCCP where $M = \{\{e_j\} \mid e_j \in E\}$.
A typical process for solving MCEP or MCCP is shown in Fig. 1. First, vulnerability scanning tools, such as Nessus (Deraison, 2010), determine vulnerabilities of individual hosts. Using this vulnerability information along with exploit templates, intruder's 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. The minimization analysis of the network attack graph determines a minimum critical set of exploits/countermeasures that must be prevented/implemented to guarantee no attack scenario is possible.

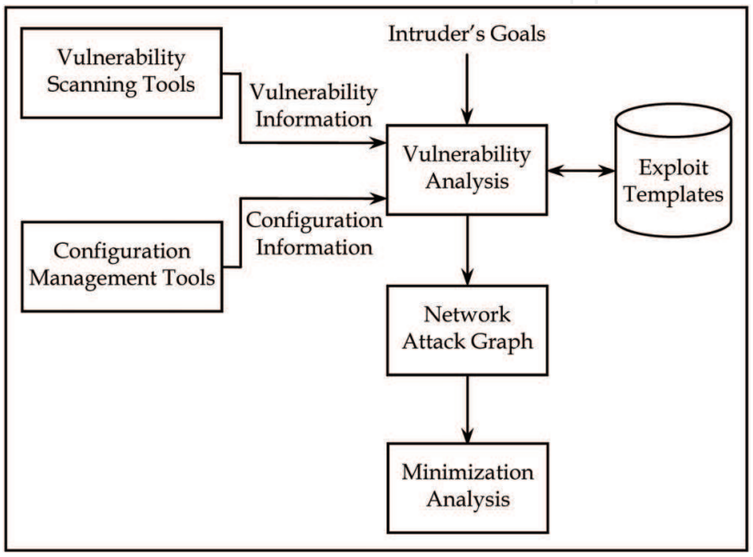


Fig. 1. Minimization analysis of network attack graphs

4. ParticleNAG

In this section, we present ParticleNAG, a memetic particle swarm optimization algorithm for minimization analysis of large-scale network attack graphs. The aim of minimization analysis of network attack graphs is to find a minimum critical set of exploits/ countermeasures. This problem 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).
Fig. 2 shows the pseudo-code of ParticleNAG. 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 repair algorithm is applied to it. Then redundant exploits of x_i are eliminated. After that, x_i is improved by a local search heuristic procedure. Then the particle's personal best position y_i is updated using equation (1). 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).

```
procedure ParticleNAG
    Set parameters, create and initialize the swarm
    while termination condition not met do
        for each particle i do
            if x_i does not represent a critical set of exploits then
                Apply the greedy repair procedure to x_i ;
            end if
            Eliminate redundant exploits of x_i ;
            Apply the local search heuristic to x_i ;
            Update the personal best position y_i ;
        end for
        Update the global best position ŷ ;
        for each particle i do
            Update the velocity v_i ;
            Update the position x_i ;
        end for
    end while
end ParticleNAG
```

Fig. 2. The ParticleNAG algorithm

5.1 Problem representation

Let $E = \{e_1, e_2, ..., e_n\}$ be the set of preventable exploits. Each particle position x_i corresponds to an n-bit vector $(x_{i1}, x_{i2}, ..., x_{in})$ 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 $x_{ij} = 1$.

$$E_i = \{ e_j \in E \mid x_{ij} = 1 \} \tag{10}$$

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

The particle position x_i represents a critical set of exploits if all attack scenarios are hit by it. The aim of minimization analysis of network attack graphs is to find a minimum critical set of exploits. So ParticleNAG uses the following fitness function to evaluate the quality of x_i:

$$f(x_i) = |E| - |E_i| \tag{11}$$

5.2 Greedy repair

The set of exploits represented by a particle position x_i may not be critical. In other words, it may not hit all attack scenarios.

Let E_i be the set of exploits represented by a particle position x_i. As shown in Fig. 3, the greedy repair algorithm chooses at each step an exploit $e_k \in E$ such that $e_k \notin E_i$ and it maximizes the partial hit value $hv_p(e_k, E_i)$. It then adds e_k to E_i and changes its corresponding element x_{ik} to 1. This is repeated until a critical set of exploits is obtained.

procedure GreedyRepair (x_i)
  $E_i = \{e_j \in E \mid x_{ij} = 1\}$;
  while x_i does not represent a critical set of exploits **do**
    Choose an exploit $e_k \in E$ such that $e_k \notin E_i$ and it maximizes the partial hit value $hv_p(e_k, E_i)$;
    $E_i = E_i \cup \{e_k\}$;
    $x_{ik} = 1$;
    $v_{ik} = V_{\max}$;
  end while
  return x_i;
end GreedyRepair

Fig. 3. The greedy repair procedure

5.3 Greedy elimination

The critical set of exploits represented by a particle position x_i may contain redundant exploits, which must be eliminated. Let E_i be the critical set of exploits represented by x_i. The exploit e_j is called *candidate redundant* with respect to E_i if $hv_x(e_j, E_i) = 0$. The set of candidate redundant exploits of E_i is denoted by R_i.

$$R_i = \{ e_j \in E_i \mid hv_x(e_j, E_i) = 0 \} \tag{12}$$

For each candidate redundant exploit $e_j \in R_i$, the *selection value* $sv(e_j, E_i)$ is calculated as

$$sv(e_j, E_i) = \sum_{e_k \in E_i \setminus \{e_j\}} hv_x(e_k, E_i \setminus \{e_j\}) \tag{13}$$

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.

procedure GreedyElimination (x_i)
  $E_i = \{ e_j \in E \mid x_{ij} = 1 \}$;
  $R_i = \{ e_j \in E_i \mid hv_x(e_j, E_i) = 0 \}$;
  while $R_i \neq \varnothing$ do
    Choose an exploit $e_k \in R_i$ that maximizes the selection value $sv(e_k, E_i)$;
    $E_i = E_i \setminus \{e_k\}$;
    $x_{ik} = 0$;
    $v_{ik} = -V_{\max}$;
    $R_i = \{ e_j \in E_i \mid hv_x(e_j, E_i) = 0 \}$;
  end while
  return x_i;
end GreedyElimination

Fig. 4. The greedy elimination procedure

In Fig. 4 an algorithm is presented, which can be used to eliminate redundant exploits of x_i. Let E_i be the critical set of exploits represented by 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, at each step, the algorithm chooses a candidate redundant exploit e_k from R_i that maximizes the selection value $sv(e_k, E_i)$. It then removes e_k from E_i and changes its corresponding element x_{ik} to 0. This is repeated until a minimal critical set of exploits is obtained.

5.4 Local search heuristic

Combining global and local search is a strategy used by many successful global optimization approaches.

In ParticleNAG, a local search heuristic is applied to the current position of each particle to improve them before their personal best positions are updated. The local search heuristic is based on the following idea: given a particle position x_i and its corresponding critical set of exploits E_i, suppose there is an exploit $e_j \in E$ such that $e_j \notin E_i$ and $E_i \cup \{e_j\}$ contains at least two exploits other than e_j, say $e'_1, ..., e'_r$, with $r \geq 2$ that are redundant. Then we conclude that $(E_i \setminus \{e'_1, ..., e'_r\}) \cup \{e_j\}$ is a better critical set of exploits than E_i. The gain of the exploit e_j with respect to E_i is $g(e_j, E_i) = l - 1$. In this case, we call e_j a *candidate dominant* exploit.

---

procedure LocalSearch(x_i)
  $E_i = \{ e_j \in E \mid x_{ij} = 1 \}$;
  while improvement is possible **do**
    Choose an exploit $e_k \in E$ such that $e_k \notin E_i$ and $g(e_k, E_i) > 0$;
    $E_i = E_i \cup \{e_k\}$;
    $x_{ik} = 1$;
    $v_{ik} = V_{\max}$;
    Eliminate redundant exploits of x_i;
  end while
  return x_i;
end LocalSearch

Fig. 5. The local search heuristic procedure

As shown in Fig. 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 in Section 5.3 for eliminating redundant exploits. This process is repeated until no further improvement is possible.

6. Experiments

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

6.1 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. 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 (Deraison, 2010) can be used to find the vulnerabilities of each host.

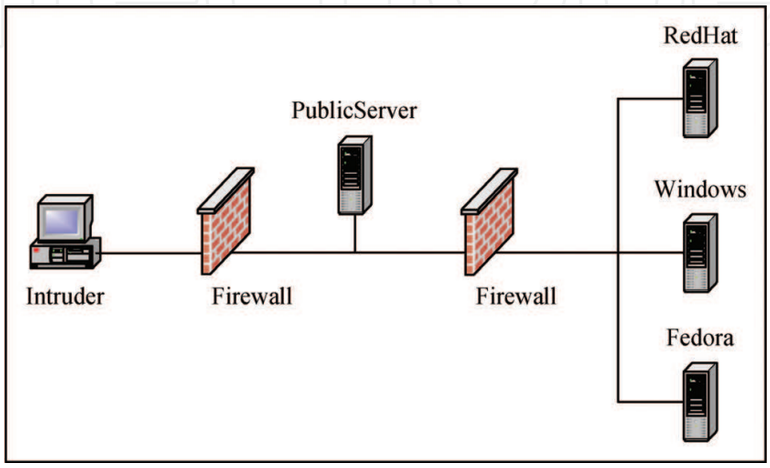


Fig. 6. An example network

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

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

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. 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 2. 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.

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

Table 2. Network connectivity information

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*. 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, described as follows:

- *iis_r2r*
  Buffer overflow vulnerability in the Microsoft IIS web server allows remote intruders to gain root shell on the target 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 host.
- *script_r2u*
  Microsoft Internet Explorer allows remote intruders to execute arbitrary code via malformed Content-Disposition and Content-Type 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.

- *ssh_r2r*
  Buffer overflow vulnerability in the SSH server allows remote intruders to gain root shell on the target 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 host and the target 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 host.
- *at_u2r*
  Buffer overflow vulnerability in the *at* program allows local users to gain root shell on the target host.

In Table 3, each generic exploit is represented by its preconditions and postconditions. More information about each of the exploits is available in (NVD, 2010). 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 3, 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 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 consists of between 4 to 9 exploits.

Experimental Results

We applied ParticleNAG 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 all exploits are preventable. Therefore, the aim was to find a minimum critical set of exploits among 168 exploits. Using ParticleNAG, the following minimum critical set of exploits was found:

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

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 3. Exploit templates

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 ParticleNAG, the following minimum critical set of exploits was found:

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

It should be mentioned that the exact cardinality of the minimum critical set of exploits for this network attack graph is 5, so the above critical set of exploits found by ParticleNAG is minimum. While using ApproxNAG (Sheyner et al., 2002); (Jha et al., 2002), the following minimum critical set of exploits was found:

$$CE = \{\ script\_r2u(PublicServer, Windows),$$
$$at\_u2r(Fedora, Fedora),$$
$$sshd\_r2u(PublicServer, RedHat),$$
$$ftp\_rhosts(PublicServer, RedHat),$$
$$squid\_ps(PublicServer, Fedora),$$
$$ftp\_rhosts(PublicServer, Fedora)\ \}$$

The second experiment shows ParticleNAG can find a critical set of exploits with less cardinality.

In the experiments, the parameters were set to $c_1 = 2$, $c_2 = 2$, and $V_{\max} = 4$, which are values commonly used in the binary PSO literature. The swarm size was set to $m = 10$ and the maximum number of iterations was set to $t_{\max} = 50$.

6.2 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/countermeasures becomes more difficult. An obvious factor is the size of the network under analysis. Our society has become increasingly dependent on networked computers and the trend towards larger networks will continue. For example, there are enterprises today consisting of tens of thousands of hosts. Also, less secure networks clearly have larger network attack graphs. Each network host might have several exploitable vulnerabilities. When considered across an enterprise, especially given global internet connectivity, network attack graphs become potentially large (Ammann et al., 2005).

In order to further evaluate the performance of ParticleNAG, we randomly generated 14 large-scale network attack graphs, denoted by NAG_1, NAG_2, ..., NAG_{14}. For each network attack graph, we considered different values for the cardinalities of E and S, where E is the set of preventable exploits and S is the set of attack scenarios represented by the network attack graph.

In NAG_1, ..., NAG_7, attack scenarios consists of between 3 to 9 exploits, while in NAG_8, ..., NAG_{14}, attack scenarios consists of between 3 to 12 exploits. Table 4 shows the cardinality of the set of preventable exploits, the cardinality of the set of attack scenarios, and the average cardinality of attack scenarios for each generated large-scale network attack graph.

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	200	2000	6.01
NAG_2	400	4000	5.99
NAG_3	400	6000	5.99
NAG_4	600	6000	6.03
NAG_5	600	8000	5.95
NAG_6	800	8000	6.01
NAG_7	1000	10000	6.05
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
NAG_{13}	800	8000	7.48
NAG_{14}	1000	10000	7.47

Table 4. Large-scale network attack graphs

Experimental results

We applied ParticleNAG 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 ApproxNAG (Sheyner et al., 2002); (Jha et al., 2002), AntNAG (Abadi & Jalili, 2006), and GenNAG (Abadi & Jalili, 2008) for minimization analysis of the above network attack graphs. As shown in Table 5, ParticleNAG outperforms all the algorithms referenced above and finds a critical set of exploits with less cardinality. On average, the cardinalities of critical sets of exploits found by ParticleNAG, AntNAG, GenNAG are, respectively, 10.77, 9.21, and 8.95 percent less than the cardinality of critical set of exploits of exploits found by ApproxNAG. Accordingly, we conclude that ParticleNAG is more efficient than ApproxNAG, AntNAG, and GenNAG.

In ParticleNAG experiments, the parameters were set to $c_1 = 2$, $c_2 = 2$, and $V_{max} = 4$, which are values commonly used in the binary PSO literature. The swarm size was set to $m = 20$ and the maximum number of iterations was set to $t_{max} = 100$.

Network Attack Graph	ParticleNAG		AntNAG		GenNAG		ApproxNAG
	Best	Average	Best	Average	Best	Average	
NAG_1	87	87.3	88	88.6	87	88.8	98
NAG_2	175	176.5	177	178.9	176	179.0	197
NAG_3	194	196.6	197	199.6	197	200.2	221
NAG_4	264	265.9	268	270.7	264	271.3	296
NAG_5	287	288.4	291	293.7	291	293.8	317
NAG_6	351	352.8	356	360.9	358	361.3	397
NAG_7	439	442.8	448	451.7	449	453.9	503
NAG_8	80	80.8	81	82.1	81	82.0	91
NAG_9	158	159.6	159	161.9	161	162.5	182
NAG_{10}	178	179.4	179	181.9	180	182.8	200
NAG_{11}	239	240.8	242	244.7	244	245.6	267
NAG_{12}	257	259	262	264.4	263	265.6	293
NAG_{13}	322	323.6	325	329.1	327	331.2	362
NAG_{14}	401	404	409	413.1	410	414.9	450

Table 5. The cardinality of critical set of exploits found by ParticleNAG, AntNAG, GenNAG, and ApproxNAG

Figures 7 to 10 show the progress of the average cardinality of the global best position of ParticleNAG, the global best solution of AntNAG, and the best chromosome of GenNAG in the experiments for minimization analysis of NAG_4, NAG_7, NAG_{12}, and NAG_{14}, respectively. As it can be seen in these figures, ParticleNAG is able to quickly converge to a good solution for large-scale network attack graphs and can maintain the balance between the exploration and exploitation reasonably well in comparison to AntNAG and GenNAG.

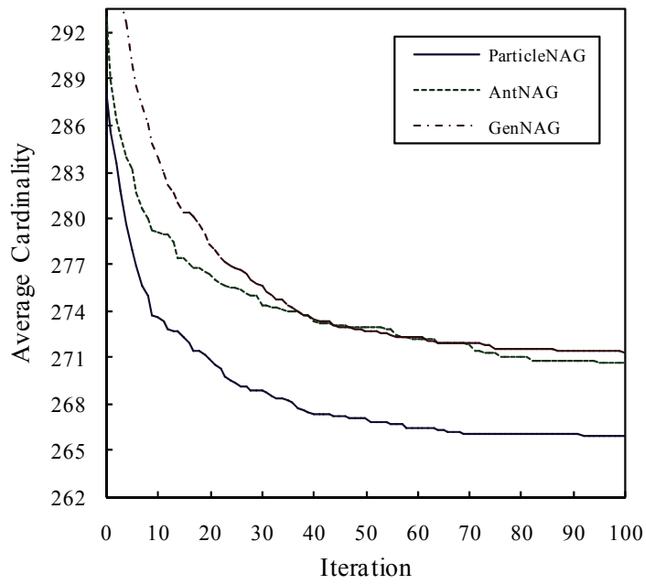


Fig. 7. Comparison of the performance of ParticleNAG, AntNAG, and GenNAG for minimization analysis of NAG_4

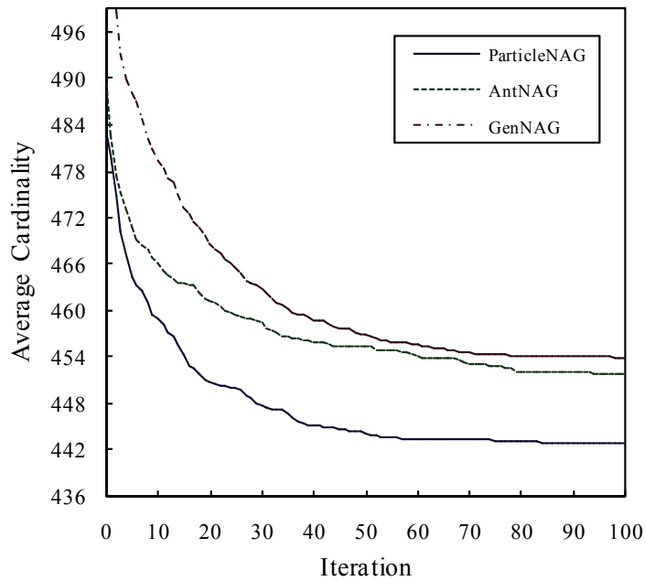


Fig. 8. Comparison of the performance of ParticleNAG, AntNAG, and GenNAG for minimization analysis of NAG_7

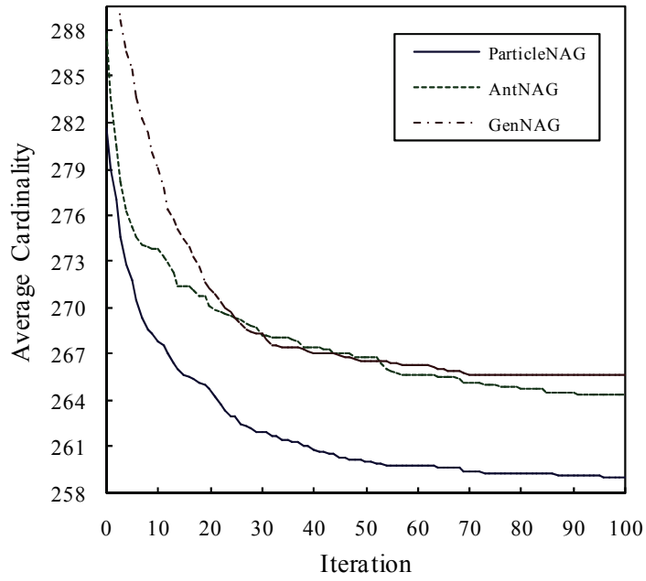


Fig. 9. Comparison of the performance of ParticleNAG, AntNAG, and GenNAG for minimization analysis of NAG_{12}

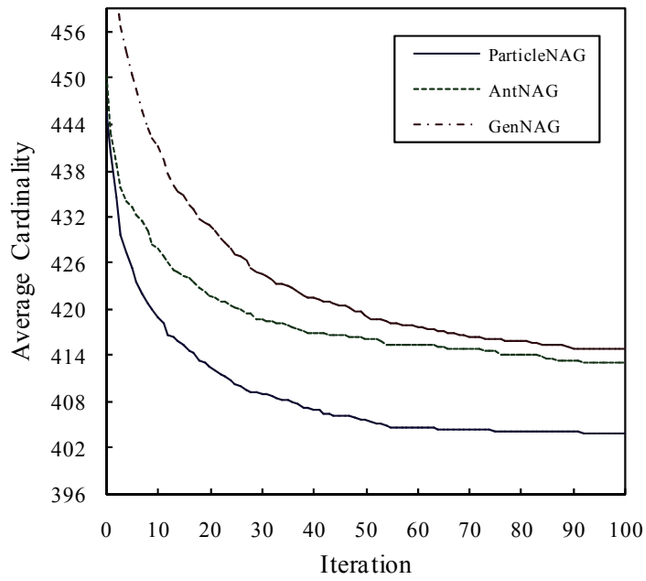


Fig. 10. Comparison of the performance of ParticleNAG, AntNAG, and GenNAG for minimization analysis of NAG_{14}

6.3 Algorithm parameters

We performed experiments to analyze the effect of different settings of parameters on the performance of ParticleNAG.

The effect of using the local search heuristic on the performance of ParticleNAG was analyzed by comparing the results of running the algorithm with and without the local search heuristic. Figures 11 and 12 show the progress of the average cardinality of the global

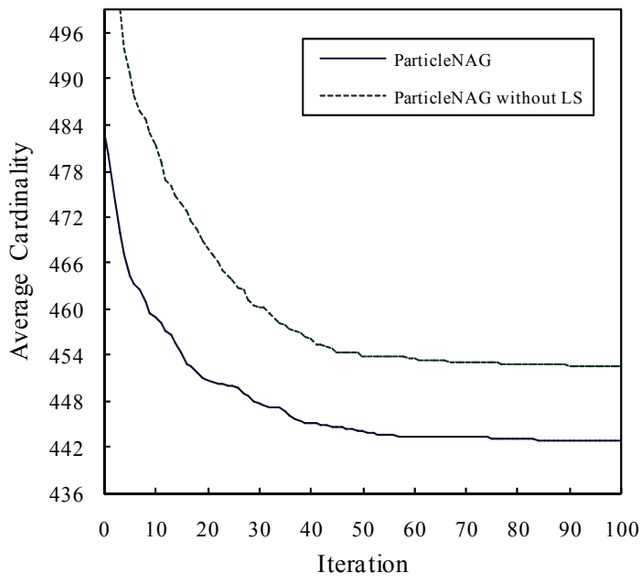


Fig. 11. Comparison of the performance of ParticleNAG and ParticleNAG without the local search heuristic for minimization analysis of NAG_7

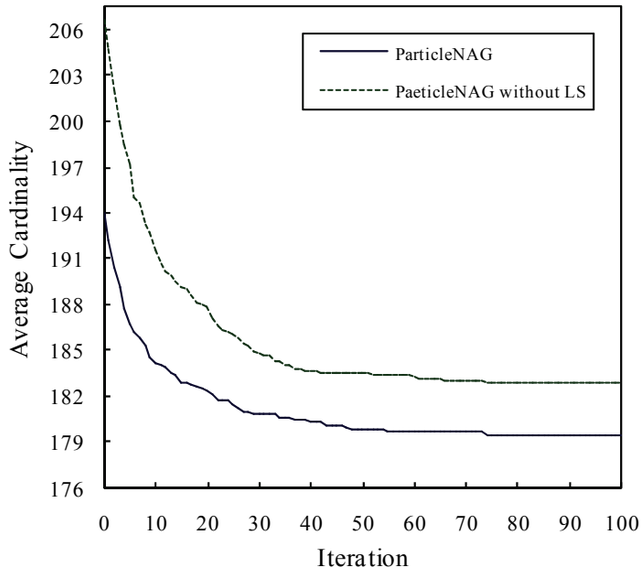


Fig. 12. Comparison of the performance of ParticleNAG and ParticleNAG without the local search heuristic for minimization analysis of NAG_{10}

best position, obtained from 10 runs of ParticleNAG and 10 runs of ParticleNAG without the local search heuristic in the experiments for minimization analysis of NAG_7 and NAG_{10}, respectively.

As the figures show, ParticleNAG significantly performs better than ParticleNAG without the local search heuristic and finds a critical set of exploits with less cardinality. This is because before updating the personal best position of a particle, its current position is improved by the local search heuristic. Hence, the personal best position of the particle shows a locally optimized solution.

To analyze the effect of the swarm size on the performance of ParticleNAG, the algorithm was run with the parameter settings from Section 6.2 but this time with the swarm size, m, set to 2, 5, 15, and 20, respectively.

As it can be seen in Table 6, when using a very small number of particles, ParticleNAG shows a poor performance. This is because the fewer the number of particles, the less the

Network Attack Graph	SwarmNAG			
	$m=2$	$m=5$	$m=15$	$m=20$
NAG_1	89.1	88.6	87.8	87.3
NAG_2	179.7	178.1	176.9	176.5
NAG_3	201.0	198.1	197.0	196.6
NAG_4	271.6	267.8	265.7	265.9
NAG_5	294.1	290.4	288.7	288.4
NAG_6	361.8	355.1	354.2	352.8
NAG_7	451.1	446.0	442.8	442.8
NAG_8	82.7	81.8	81.5	80.8
NAG_9	163.2	160.6	160.4	159.6
NAG_{10}	184.2	181.2	179.1	179.4
NAG_{11}	245.0	242.2	241.3	240.8
NAG_{12}	263.8	261.6	259.9	259.0
NAG_{13}	330.5	326.9	323.8	323.6
NAG_{14}	413.1	408.1	404.7	404.0

Table 6. Effect of the swarm size on the performance of ParticleNAG

exploration ability of the algorithm, and consequently the less information about the search space is available to all particles.

7. Conclusions

Each attack scenario is a sequence of exploits launched by an intruder for a particular goal. To prevent an exploit, the security analyst must implement a suitable countermeasure such as the firewall configuration or patch the vulnerabilities that made this exploit possible. The collection of possible attack scenarios in a computer network can be represented by a directed graph, called network attack graph. In this directed graph, each path from an initial node to a goal node corresponds to an attack scenario.

The aim of minimization analysis of network attack graphs is to find a minimum critical set of exploits/countermeasures so that by preventing/implementing them the intruder cannot reach his goal using any attack scenarios. This problem 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.

Several factors can make network attack graphs larger so that finding a minimum critical set of exploits/countermeasures becomes more difficult. An obvious factor is the size of the network under analysis. Our society has become increasingly dependent on networked computers and the trend towards larger networks will continue. Also, less secure networks clearly have larger network attack graphs. Each network host might have several exploitable vulnerabilities. When considered across an enterprise, especially given global internet connectivity, network attack graphs become potentially large.

Particle swarm optimization (PSO) is a population based stochastic optimization algorithm that was inspired by social behaviour of flocks of birds when they are searching for food.

While evolutionary algorithms take inspiration from biological evolution, memetic algorithms mimic cultural evolution. The term meme refers to a unit of cultural information that can be transmitted from one mind to another after reinterpretation and improvement that in the context of combinatorial optimization corresponds to local search.

In this paper, we presented a memetic particle swarm optimization algorithm, called ParticleNAG, for minimization analysis of network attack graphs. A greedy repair method was used to convert the constrained optimization problem into an unconstrained one. We reported the results of applying ParticleNAG for minimization analysis of 14 large-scale network attack graphs. We also applied an approximation algorithm, ApproxNAG (Sheyner et al., 2002); (Jha et al., 2002), an ant colony optimization algorithm, AntNAG (Abadi & Jalili, 2006), and a genetic algorithm, GenNAG (Abadi & Jalili, 2008), for minimization analysis of the above large-scale network attack graphs.

On average, the cardinality of critical sets of exploits found by ParticleNAG was 10.77 percent less than the cardinality of critical sets of exploits found by ApproxNAG. Also, ParticleNAG performed better than AntNAG and GenNAG in terms of convergence speed and accuracy.

We performed experiments to analyze the effect of swarm size and local search heuristic on the performance of ParticleNAG. The results of experiments showed that ParticleNAG significantly performs better than ParticleNAG without the local search heuristic.

8. References

Abadi, M. & Jalili, S. (2005). Automatic discovery of network attack scenarios using SPIN model checker, *Proceedings of the International Symposium on Telecommunications (IST 2005)*, pp. 81–86, Shiraz, Iran, September 2005

Abadi, M. & Jalili, S. (2006). An ant colony optimization algorithm for network vulnerability analysis. *Iranian Journal of Electrical & Electronic Engineering (IJEEE)*, Vol. 2, Nos. 3 & 4, pp. 106–120, July 2006

Abadi, M. & Jalili, S. (2008). Minimization analysis of network attack graphs using genetic algorithms. *International Journal of Computers and Their Applications (IJCA)*, Vol. 14, No. 4, pp. 263–273, December 2008

Ammann, P.; Wijesekera, D. & Kaushik, S. (2002). Scalable, graph-based network vulnerability analysis, *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pp. 217–224, Washington, DC, USA, November 2002

Ammann, P.; Pamula, J.; Ritchey, R. & Street, J. (2005). A host-based approach to network attack chaining analysis, *Proceedings of the 2005 Annual Computer Security Applications Conference (ACSAC 2005)*, pp. 72–84, Tucson, AZ, USA, December 2005

Braendler D. & Hendtlass T. (2002). The suitability of particle swarm optimisation for training neural hardware, *Proceedings of the 15th International Conference on Industrial and Engineering, Applications of Artificial Intelligence and Expert Systems*, pp. 190–199, Cairns, Australia, June 2002

Deraison, R. (2010). Nessus Vulnerability Scanner. http://www.nessus.org

Eberhart, R. C.; Simpson P. & Dobbins R. (1996). *Computational Intelligence PC Tools*, Academic Press Professional, San Diego, CA, USA

Eiben, A. E. & Smith, J. E. (2003). *Introduction to Evolutionary Computing*, Springer-Verlag, Berlin, Germany

Engelbrecht, A. P. (2005). *Fundamentals of Computational Swarm Intelligence*, John Wiley & Sons, Hoboken, NJ, USA

Hendtlass, T. & Randall, M. (2001). A survey of ant colony and particle swarm meta-heuristics and their application to discrete optimization problems, *Proceedings of the Inaugural Workshop on Artificial Life*, pp. 15–25, Adelaide, Australia, December 2001

Jha, S.; Sheyner, O. & Wing, J. M. (2002). Two formal analyses of attack graphs, *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pp. 49–63, Cape Breton, Nova Scotia, Canada, June 2002

Kennedy, J. & Eberhart, R. C. (1995). Particle swarm optimization, *Proceedings of the IEEE International Joint Conference on Neural Networks*, pp. 1942–1948, Perth, Australia, 1995

Kennedy, J. & Eberhart, R. C. (1997). A discrete binary version of the particle swarm algorithm, *Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics*, pp. 4104–4109, Orlando, FL, USA, October 1997

Kennedy, J.; Eberhart, R. C. & Shi Y. (2001). *Swarm Intelligence*, Morgan Kaufmann, San Mateo, CA, USA

Krasnogor, N.; Aragon, A. & Pacheco J. (2006). Memetic Algorithms, In: *Metaheuristic Procedures for Training Neural Networks*, Enrique Alba & Rafael Martí (Eds.), Springer-Verlag, Berlin, Germany

Mehta, V.; Bartzis, C.; Zhu, H.; Clarke, E. M. & Wing, J. M. (2006). Ranking attack graphs, *Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID 2006)*, pp. 127–144, Hamburg, Germany, September 2006

Noel, S.; Jacobs, M.; Kalapa, P. & Jajodia, S. (2005). Multiple coordinated views for network attack graphs, *Proceedings of the IEEE Workshop on Visualization for Computer Security (VizSEC 2005)*, pp. 99–106, Minneapolis, Minnesota, USA, October 2005

NuSMV. (2010). A New Symbolic Model Checker. http://afrodite.itc.it:1024/~nusmv/

NVD. (2010). National Vulnerability Database. http://nvd.nist.gov/

Ou, X.; Govindavajhala, S. & Appel, A. W. (2005). MulVAL: A logic-based network security analyzer, *Proceedings of the 14th USENIX Security Symposium*, pp. 8–8, Baltimore, MD, USA, August 2005

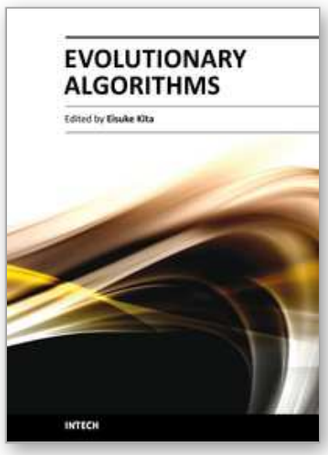
Ou, X.; Boyer, W. F. & McQueen, M. A. (2006). A scalable approach to attack graph generation, *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS 2006)*, pp. 336–345, Alexandria, VA, USA, October 2006

Phillips, C. & Swiler, L. P. (1998). A graph-based system for network-vulnerability, *Proceedings of the New Security Paradigms Workshop*, pp. 71–79, Charlottesville, VA, USA, September 1998

Sheyner, O.; Haines, J. W.; Jha, S.; Lippmann, R. & Wing, J. M. (2002). Automated generation and analysis of attack graphs, *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 273–284, Berkeley, CA, USA, May 2002

Shi, Y. & Eberhart, R. C. (1999). Empirical study of particle swarm optimization, *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1945–1950, Washington, DC, USA, July 1999

Shi, Y. (2004). Particle swarm optimization. *IEEE Connections*, Vol. 2, No. 1, pp. 8–13, February 2004



Evolutionary Algorithms

Edited by Prof. Eisuke Kita

ISBN 978-953-307-171-8
Hard cover, 584 pages
Publisher InTech
Published online 26, April, 2011
Published in print edition April, 2011

Evolutionary algorithms are successively applied to wide optimization problems in the engineering, marketing, operations research, and social science, such as include scheduling, genetics, material selection, structural design and so on. Apart from mathematical optimization problems, evolutionary algorithms have also been used as an experimental framework within biological evolution and natural selection in the field of artificial life.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen