A Population-Based Incremental Learning Approach to Network Hardening

Alexander Paulsen [a], Anis Yazidi [a], Boning Feng [a] and Xinming Ou [b]
[a] *Computer Science Department, Oslo Metropolitan University*
[b] *Computer Science and Engineering Department, University of South Florida*

Abstract.
Enterprise networks constantly face new security challenges. Obtaining complete network security is almost impossible, especially when usability requirements are taken into account. Previous research has provided ways to identify multi-stage attacks caused by network vulnerabilities and misconfigurations, but few have addressed ways to circumvent those multi-stage attacks, especially when usability requirements are taken into account. The latter problem is reckoned as Network Hardening problem [10] and is known to be an NP hard combinatorial problem. In this paper, we map the network hardening problem to a constrained optimization problem and resort to the theory of Population-Based Incremental Learning (PBIL) in order to solve it. We devise two approaches based on the PBIL, namely the Acceptance-Rejection approach, and the Penalty-based approach. Our aim is to tighten the security of the network by minimizing the number of privileges that an attacker can gain over network under some usability constraints measured in terms of the number of configurations in a network that can be activated or cannot be deactivated. The Acceptance-Rejection approach disqualifies configurations that violate the usability constraint while the Penalty-based approach relaxes the latter constraint by attempting to find a compromise between security and usability of the configuration. While the Acceptance-Rejection approach can be seen as a simple alternative to the state of the art MinCostSAT solution adopted in [10], the Penalty-based approach is, to the best of our knowledge, the first solution in the literature that tries to find such compromise. Experimental results show that the devised approaches are computationally efficient, scalable and reliable.

Keywords. Security Tightening, Population Based Incremental Learning, Acceptance-Rejection, Penalty-based Approach

1. Introduction

Targeted attacks usually exploit multiple vulnerabilities in a so-called multi-stage attack to elevate their privileges and access in a network, hopefully evading security mechanisms. Intrusion detection systems generate large amounts of alerts, and for a system administrator, it is easy to overlook an ongoing multi-stage attack. Vulnerability analysis implies that every vulnerability shall be removed, but this is not usually the case in practice. Removing vulnerabilities is complicated as a result of the reduced availability of patches and upgrades, costs, and demands regarding efficiency, usability and uptime. Vulnerability scanners are tools which can assist system administrators in identifying vulnerabilities in a system [9]. However, these scanners can only identify vulnerabilities in *isolation*. Attack graph tools can enhance the information from a vulnerability scan to give more contextual meaning about possible multi-stage attacks in a graphical manner. Previous studies have provided solutions to how to identify network vulnerabilities, but few of them have addressed how to correct them. Vulnerability scanning of large networks results in massive amounts of data, which can be used to generate complex attack graphs that are often unreadable and overwhelming. One of the intentions of the attack graph is to provide a basis for future security decisions. However, attacks graphs are not usually human-readable even for a small-size network and thus cannot provide an efficient and satisfactory basis for decisions. Few methods are available for tightening the security in a network by using the information from the attack graphs, in addition to taking usability and network requirements into account. In this paper, we map the network hardening problem to a constrained optimization problem and resort to the theory of Population-Based Incremental Learning (PBIL) for solving it [5].

The aim of this paper is to tighten the security of the network by minimizing the number of privileges that an attacker can gain over network under some usability constraints measured in terms of the number of configurations in a network that can be activated or cannot be deactivated. We devise two approaches based on the PBIL, namely an Acceptance-Rejection approach, and a Penalty-based approach. The Acceptance-Rejection approach disqualifies configurations that violate the usability constraint while the Penalty-based approach relaxes the latter constraint by attempting to find a compromise between security and usability of the configuration. The Acceptance-Rejection can be seen as a simple alternative to the state of the art MinCostSAT solution adopted in [10]. The Penalty-based approach is the first reported approach in the literature that allows to explore more secure solutions as the cost of some violation of the usability constraint, which is a major contribution in itself [10].

2. Background and related work

2.1. Attack Graphs

There are multiple attack graph tools available such as Topological Analysis of Network Attack Vulnerability (TVA) [12], Attack Graph Toolkit [19] and Cauldron [16]. Multi-host, multi-stage Vulnerability Analysis (MulVAL) is an open source project developed by Ou et al. [15] at Princeton University. In this paper, we resort to MulVAL as a attack graph tool.

MulVAL uses information from vulnerability databases, configuration information from each node as well as other relevant information to graph the pre and post conditions from each exploit and how they interact with each other in a network. Additionally, the tool makes available textual formats of attack paths. The reasoning engine also scales well (O(n^2)) with network size. It works with Nessus and OVAL vulnerability scanners.

The information MulVAL processes can be organized into three different entities shown in an attack graph: *configuration* entities, *exploit* entities and *privilege* entities.

- *System configurations* are represented as rectangular shapes. These include host access permissions, existing vulnerabilities, applications, etc.
- *Privileges* are represented as diamond-shapes. A privilege is what an attacker can gain through exploits.

- *Exploits* are represented as elliptical shapes. A potential exploit links the pre-conditional configuration entities, which enables the exploit, with the effect of the attack, the post-conditional privileges [10].

Arcs coming out from an exploit entity form logical AND relations, meaning all of the child relations must be true in order for the exploit can be used. Removing only one of the child relations would be sufficient for the exploit to be unavailable. In the MulVAL attack graph in Figure 1, we can express the exploits in the following boolean formulas:

$$
\begin{aligned}
e1 &= c1 \wedge c2 \\
e2 &= p1 \wedge c3 \\
e3 &= c4 \wedge c5
\end{aligned}
$$

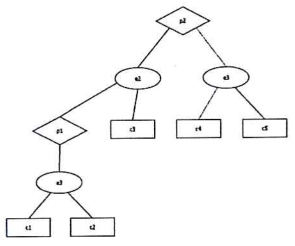

Figure 1. Example of MulVAL attack graph.

Arcs from privilege entities like *p1* and *p2* form logical OR relations. This indicates that only one of its child relations need to be true in order for a privilege to be achieved. Either *e2* or *e3* is required for obtaining privilege *p2*.

$$
p2 = e2 \vee e3
$$

2.2. *Network Hardening*

In security communities, network hardening has often been considered an art rather than a science [20]. Experienced security analysts perform tedious work to identify and prioritize different vulnerabilities and network weaknesses to be fixed and patched. To make network hardening more like a science than an art, systematic approaches to automatically computing potential hardening solutions are essential.

For network hardening, we need to measure the overall security of a network. Previous research [21] states that a crucial element of measuring network security lies in understanding the interplay between network components such as how vulnerabilities can be combined by attackers in an advancing multi-stage attack. This study framework focuses on computing overall security with respect to critical resources. Two dependency models are presented, one captured by attack graphs, and another by additional functions.

The latter affects the measure of network components but does not enable it to be reachable. Another approach measures the security strength of a network using the weakest attacker model, i.e. the weakest adversary who can successfully penetrate the network [17]. Moreover, some other studies measure the likelihood of a software being vulnerable to attacks using a metric called attack surface [11,13]. This is a security metric for comparing the relative security of similar software systems where the *attack surface* is an indicator of the software's security.

Network hardening with respect to initial conditions level was introduced in 2003 [14]. The article states that an approach using configuration elements is beneficial compared to exploit-level approaches as it resolves hardening irrelevancies and redundancies better. The aim of the research conducted was to find a set of initial conditions that can disable a goal condition at a minimum cost. The authors represent the resources in a network as logical propositions of initial conditions where vulnerabilities are viewed as Boolean variables. A false condition would mean the condition is suggested disabled for hardening. The presented product does not scale well as the number of terms in the equation can grow exponentially in the number of conditions in the network.

In [10], Homer and Ou propose to formulate network hardening as Boolean satisfiability problem. In this perspective, the authors proposes two SAT solving techniques, namely MinCostSAT and UNSAT Core Elimination. This approach reduces complex problems to manageable levels in addition to requiring minimal user interaction in order to rapidly fix misconfigurations that can lead to multi-stage attacks. Human interaction includes making decisions about the relative value of specific instances of security and usability enabling the research to take account of both security and usability requirements. Results show that this approach is scalable and effective.

In [14], Noel et al. introduce a framework for computing the minimum-cost solution, while guaranteeing the security of given critical resources. The proposed solution uses a graph-based representation of exploit dependencies. The representation in the study has low-order polynomial complexity in contrast to exponential complexity, which has been found in most studies according to the authors.

2.3. *Population-Based Incremental Learning*

Population-based incremental learning (PBIL) is a probabilistic model commonly used for optimization of large dynamic combinatorial search problems. It is a combination of evolutionary optimization [4] and hill climbing [6] according to Baluja [1]. The method incorporates genetic algorithms (GA) and competitive learning for function optimization, but rather than being based on population-genetics, PBIL is similar to learning automata in which each automaton chooses actions independently. The combination of these two methods constitutes a tool that is much simpler than a GA and outperforms GA on many optimization problems. The intention of the algorithm is to generate a real-valued probability vector which, when sampled, reveals high evaluation solution vectors with high probability [2].

The features of PBIL are as follows [5]:

1. PBIL has no crossover and fitness proportional operators.
2. For each iteration, a set of samples is generated according to the current probability vector. The set of samples is evaluated according to the problem-specific function.

3. Only the elite solution and the one being evaluated are stored.

Evolutionary Algorithms (EA), which utilize the principles of natural selection and population genetics (e.g. GAs), have become common in optimization and search techniques due to their powerful capabilities for finding solutions to difficult problems. Especially in static environments, where the landscape does not change during computation [7]. However, real-world environments are often prone to changes, making traditional EAs unsuitable as they cannot adapt properly to changed environments once converged [22]. In contrast, PBIL has proven itself to be very successful when compared to different standard genetic and hill climbing algorithms on various benchmarking [3] and real-world [8] problems.

For each iteration, a set of samples is generated according to the current probability vector. The set of samples is evaluated according to the problem-specific function. Finally, the probability vector is learnt and shifted towards a solution with the best result. The distance the probability vector is pushed each iteration depends on a parameter called the learning rate (LR) which weights the previous vector with the new vector. The learning rate is commonly set to be 0.05. For the next iteration, when the probability vector is updated, a new set of solutions is generated according to the updated probability vector. The algorithm ends when the termination conditions are satisfied, usually when all vectors have converged (normally towards 0 or 1) or the number of iterations has reached a maximum number.

As an example, the solution to a problem can be represented as a vector of 1's and 0's. The initial value of a probability vector is normally 0.5 and a converged final probability vector can be for example 0.99, 0.01, 0.01, 0.01, 0.99, 0.99. Sampling from an initial vector reveals random solution vectors since there is an equal probability of generating 0 or 1. As the search progresses, the values in the probability vectors will shift towards 1 or 0. The pseudo code for PBIL is exemplified in Algorithm 1.

The initial probability vectors are established. The new sample vectors are generated and then evaluated, and sorted from best to worst. The highest evaluation vectors are kept. The learning rate (LR) determines how fast the probability vectors shifts each iteration. The probability is defined as:

$probabilityVector_i = (probabilityVector_i \times (1.0 - LR)) + (LR \times vector_i)$ where $probabilityVector_i$ is the probability of generating a 1 in bit position $_i$ and $vector_i$ is the i position in the solution vector which the probability vector is shifted towards. *LR* is the learning rate.

3. Approach

In this section, we will show how network hardening can be mapped into a constrained optimization problem. A tool is developed with the goal of analyzing the current network security state, based on data from an attack graph, and to be able to compute a solution which will provide increased network security. More specifically, the tool should suggest what configurations should be fixed or patched. We will present two approaches both based on PBIL, one Acceptance-Rejection approach, and the other a Penalty-based approach. These approaches will be described later in this paper.

The attack graph tool used in this research is MulVAL. MulVAL is open source and uses external sources to enrich vulnerability data. In addition, MulVAL uses a command-

Algorithm 1 The basic version of the PBIL algorithm for a binary alphabet, adapted from [2]

```
PBIL CONSTANTS:
NUMBER_SAMPLES: the number of vectors generated before updating of the probability vector.
LR: the learning rate, how fast to exploit the search performed (0.05).
NUMER_OF_VECTORS_TO_UPDATE_FROM: the number of vectors in the current population which are used to update the probability vector.
LENGTH: number of bits in the solution (determined by the problem encoding).
for $i$ in $1 : LENGTH$ do
  $P[i] = 0.5$
end for
$S$ is an array of integer
while NOT termination condition do
  ***** Generate Samples *****
  for $i$ in $1 : NUMBER\_SAMPLES$ do
    solution_vectors[i] :=
    generate_sample_vector_from_probabilities(P)
    evaluations[i] :=
    evaluate_Solution (solution_vectors[i])
    solution_vectors :=
    sort_vectors_from_best_to_worst_according_to_evaluations()
  end for
  **** Update Probability Vector towards best solutions****
  for $j$ in $1 : NUMBER\_VECTORS\_TO\_UPDATE$ do
    for $i$ in $1 : LENGTH$ do
      P[i] := P[i] * (1 - LR) + solution_vectors[j][i]* LR
    end for
  end for
end while
```

line interface which is useful for fully automated graphing solutions. We aspire to maximize network security while maintaining a constraint based on the maximum number of configurations in a network that can be activated. The standard PBIL method itself does not have the ability to solve constrained optimization problems and is rather designed originally for solving unconstrained optimization problem. In order to mitigate this limitation, we rely on two algorithms, namely Acceptance-Rejection and Penalty based PBIL. Section 3.2 provides an introduction to these two constrained optimization methods.

3.1. Measuring security

A naive approach would evaluate security in terms of the number of vulnerabilities an attacker is able to exploit. However, when evaluating security, it is important to consider the consequences we wish to avoid. In terms of an attack graph, an exploit leads to a privilege, and a privilege can be considered a consequence (POST condition) of realizing an exploit. Multiple exploits may lead to the same privilege, they namely form an

OR relation. Fixing/patching a vulnerability may not be sufficient to avoid the consequence/privilege, as another vulnerability might incur the same result. In such case, an attacker would simply use another exploit in order to obtain a desired privilege. Therefore, we propose to evaluate security in terms of the total number of privileges an attacker from the Internet is able to gain.

3.2. *Constrained optimization*

We formulate the network hardening problem as a constrained optimization problem. The Acceptance-Rejection approach does not allow the violation of the constraints in the generation process of the possible configurations while the Penalty-based approach "relaxes" the constraint and permits to obtain a compromise between security and usability.

It is worth mentioning that Acceptance-Rejection approach and the Penalty-based approach are inspired by the Cross-Entropy method (CE) [18] due to Rubinstein and Kroese. The CE method is widely applied in discrete optimization tasks and has similarities to PBIL.

The Acceptance-Rejection approach does not allow the violation of the constraints in the generation process of the possible configurations

The Penalty-based approach is generally more applicable and quite easy to implement. It involves using a penalty parameter which penalizes solutions violating the constraints without totally discarding them. The Penalty-based approach is very sensitive to the choice of the penalty parameter.

In a nutshell, the main difference with [14,10] lies mainly in the fact that, by using the Penalty-based approach, we allow relaxation of the constraints compared to using hard constraints. For example, the study [10] formulates the network hardening problem as MinCostSAT problem and thus only configurations that satisfy the usability constraints are considered as feasible solutions. In our work, we allow a partial violation of the constraints so that we can still obtain a solution even if all possible configurations are unfeasible according to the MinCostSAT approach [10].

3.3. *Constraints*

Some human interaction is needed for hardening the network with some constraints on the feasible configuration.

3.3.1. *Maximum configuration constraint*

The user of the tool should decide a threshold for how many configurations the tool should suggest to fix or patch. In a network of 100 exploit-related configurations for example, a network and system administrator would for instance like to know which ten configurations should be fixed in order to provide the best security. In an optimal network, all vulnerable services should be patched at all times, but in real life, this is not the case. Bearing this in mind, a network and system administrator must do some prioritization when hardening the network. Thus, the administrator wants to know what he should fix immediately in order to increase security in the network.

As an example, in a network of 145 exploit-related configurations $c = (c_1, \ldots c_{145})$, we can envisage for example that the administrator can fix/patch approximately ten configurations within this time limit. Therefore, the network hardening tool should suggest a maximum of ten configurations that should be fixed or patched.

3.4. Notations

Given a list of exploit-related configurations:
$c = (c_1, c_2, c_3..., c_N)$.
PBIL generates samples each with a vectorial length N:
$x = (x_1, x_2, x_3...x_N)$ where $x_i \in \{0, 1\}$

$$x_i = \begin{cases} 1\ if\ c_i\ is\ enabled \\ 0\ if\ c_i\ is\ disabled \end{cases}$$

Enabled configurations are exploit-related configurations which are *turned on*, also regarded as *not suggested patched/fixed*, *TRUE configurations* or in the code read as 1. Disabled configurations are exploit-related configurations which are *turned off*, also regarded as *suggested patched/fixed*, *FALSE configurations* or in the code read as 0.

$f = \sum I(x_i = 0)$ represents the number of disabled exploits where $I(.)$ is the indicator function.
PBIL generates samples based on probability vectors:
$p = (p_1, p_2, p_3..., p_N)$ where p_i is the probability that $x_i = 1$.

3.5. Proposed approach

The input data will be retrieved from MulVAL's generated csv-files, namely *VERTICES.CSV* and *ARCS.CSV*. The initial configurations, exploits and privileges can be retrieved from *VERTICES.CSV*. The general structure of the program where one lap of the loop is one iteration is given below:

```
1: converged=False
2: count=0
3: while (converged == False) and (count < max_iterations) do
4:     genNum=generateVectors()
5:     avg=findBestLists(genNum)
6:     sample=updateProbVector(avg)
7:     converged=hasConverged(sample)
8:     count=count+1
9: end while
```

- Line 3: Continue to next iteration as long as one element of the probability vector has not converged and the highest number of iterations is not reached.
- Line 4: For each component of the probability vector, generate a random number between 0 and 1. If the generated number is less than the corresponding probability vector component value, then the configuration can be regarded as be TRUE (1), otherwise regard the configuration as FALSE (0). Check whether a disqualified configuration is FALSE, and change it to TRUE. When using the Acceptance-Rejection method, ignore samples that exceed a given number of FALSE configurations (threshold t). Stop generating samples when there are X number of accepted samples. X is recommended to be 100.

Notation	Description
m	Number of privileges an attacker can gain from a given configuration
f	Number of FALSE configurations in a sample. $f = \sum I(x_i = 0)$ described in 3.4.
t	Threshold. The threshold states a cut-off point for undesired FALSE configurations.
λ	The penalty parameter.
p	Penalty. Penalty $pe = \beta \times (f - t)$.
s	Calculated security *Acceptance-rejection* approach: $s = m$. *Penalty* approach: $s = m + pe$.

Table 1. Notations

Complexity	Description
	Nodes: 8
	Exploits: 17
Medium	Configurations: 16
	Privileges: 14

Table 2. Overview of networks used in experiments

- Line 5: Evaluate each sample according to the measured security described in section 3.1. Let measured security be m, where optimal security m is 0. When using the *penalty* approach, add the penalty pe to the number of privileges an attacker can realize (m). The suggested approach is to *only* add a penalty to samples that exceed a *threshold* (t) of FALSE configurations (f) in a sample. Let $pe = \beta \times (f - t)$. The penalty parameter β and threshold parameter t should be established based on the total number of configurations given in the attack graph to ensure scalability. In both approaches, *Acceptance-Rejection* and *Penalty*, security s is measured by $s = m$, meanwhile samples penalised in the *penalty* approach by exceeding $f > t$ are calculated as $s = m + \beta \times (f - t)$. The samples with the top ten fitness values from the total number of 100 samples are summarized by their mean vector, which we call mean of the elite samples.
- Line 6: Now the update probability vector is recomputed as a convex combination of the previous update probability vector and the mean of the elite sample. The learning rate (LR) indicates how much weight should be given to the mean of the elite sample. The suggested LR should be 0.05 or 0.1.
- Line 7: The probability vector is checked to determine whether it has converged or not. It has not converged if one of the component did not converge, namely, there exists an index i such as $p_i > 0.01$ *AND* $p_i < 0.99$.

4. Experiments

4.1. Networks Configurations

We consider a network firewalls, one perimeter firewall and one internal firewall. However, the number of configurations and exploits are lower than network 1.

4.1.1. *Illustrations of suggested network topology*

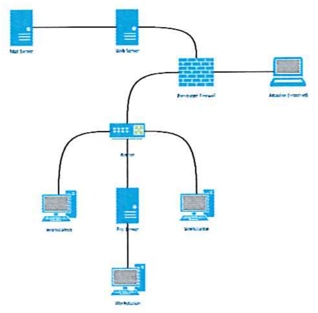

Figure 2. Network Topology.

4.2. *Experiment*

The main goal of this experiment is to observe how the two approaches, Penalty and Acceptance-Rejection, compare with an LR of 0.1 and an LR of 0.05. Another intention is to observe how Acceptance-Rejection sample generation progresses over iterations. This network has been given the following constraints:

- Maximum configuration constraint: 4
- Configuration disqualify constraint: '11','25','12','38','39','57'

4.2.1. *Acceptance-Rejection*

The maximum configuration constraint implies that all generated samples with more than four disabled values should be rejected. The results are found in Table 3 and Table 4 for different learning rates, 0.1 and 0.05 respectively.

Experiment LR=0.1

Suggested solution	Iterations	Seconds
('13', '26', '40', '58')	65	11.6
('14', '26', '32', '58')	64	11.8
('14', '26', '40', '58')	71	12.8
('14', '26', '40', '58')	75	13.4
('13', '26', '40', '58')	85	15.2
('13', '26', '40', '58')	102	18.1
('14', '26', '40', '58')	90	16.3
('13', '26', '32', '58')	70	12.5

Table 3. Results Network 2 Acceptance-Rejection LR=0.1

Experiment LR=0.05

Suggested solution	Iterations	Seconds
('13', '26', '40', '58')	173	30.6
('14', '26', '40', '58')	198	35.9
('13', '26', '32', '58')	175	30.7
('14', '26', '32', '58')	198	34.9
('14', '26', '32', '58')	186	32.9
('14', '26', '32', '58')	164	29.3
('14', '26', '40', '58')	148	26.1
('14', '26', '32', '58')	177	31.7

Table 4. Results Network 2 Acceptance-rejection LR=0.05

4.2.2. *Penalty*

$\lambda = 1$ and $t = 4$

The threshold value t represents how the Penalty-based approach annotates the maxi-

mum configuration constraint. The results are found in Table 5 and Table 6 for different learning rates, 0.1 and 0.05 respectively.

For Table 6, the mean number of iterations for convergence is 161 while 80 in Table 5.

Experiment LR=0.1

Suggested solution	Iterations	Seconds
('14', '26', '32', '58')	69	14.9
('13', '26', '40', '58')	73	16.8
('13', '26', '40', '58')	86	16.5
('14', '26', '32', '58')	75	13.7
('13', '26', '32', '58')	65	12.1
('14', '26', '32', '58')	73	13.5
('14', '26', '40', '58')	77	14.6
('13', '26', '40', '58')	83	15.3

Table 5. Results Network 2 Penalty LR=0.1.

Experiment LR=0.05

Suggested solution	Iterations	Seconds
('13', '26', '32', '58')	129	24.3
('14', '26', '40', '58')	179	32.4
('14', '26', '32', '58')	146	26.7
('14', '26', '32', '58')	213	39.4
('14', '26', '40', '58')	152	27.7
('13', '26', '32', '58')	177	32.2
('14', '26', '32', '58')	152	27.9
('13', '26', '32', '58')	156	28.3

Table 6. Results Network 2 Penalty LR=0.05.

We can observe the fact that the suggested solution results in each experiment vary to some degree. However, the experiments are more or less similar. However, regardless of the computation output, the reduced attack graphs are identical. The hardening reduces the number of accessible privileges from 18 to 2.

Figure 3 illustrates how the *Acceptance-Rejection* approach generates samples per iteration in this experiment.

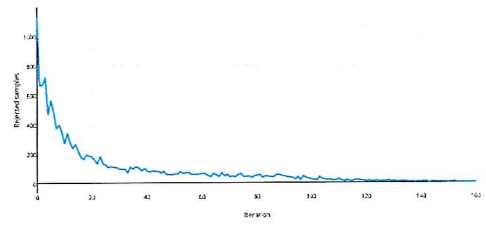

Figure 3. Rejected samples per iteration.

4.3. Analysis

The results prove that a network can be hardened with simple steps. All experiments show that the initial attack graph is reduced dramatically after running the presented tool. The results also show that the given constraints also are maintained.

We can observe in terms of iterations that doubling the learning rate approximately halves the number of iterations needed for all probability vectors to converge. We can observe that the methods *Acceptance-Rejection* and *Penalty* have little variation in number of iterations when solving the same task. What also is worth noting is that the *Acceptance-Rejection* method generates a noticeable greater amount of samples during a computation as illustrated in Figure 3. Figure 3 tells us that the method generates an enormous amount of samples the first iterations compared with the *Penalty* method, and the number decreases exponentially with the number of iterations. In the given network environments, the time of generating more samples is negligible when comparing the two methods. However, using the *Acceptance-Rejection* method with a very low threshold and in more complex environments might result in notably increased computation time. What also is important is that a very low threshold can result in a solution which will not harden the network significantly. The user of the tool will need to have a realistic approximation when determining the threshold value, because one cannot necessarily expect the tool to provide a good solution with only being able to suggest just a few fixes/patches in a large network with numerous constraints.

However, the effectiveness of two methods prove themselves when changing the learning rate. Table 4 proves that the *penalty* method spends less time per iteration than the *Acceptance-Rejection* method when having a low learning rate of 0.05. The tables are turned when doubling the learning rate (0.10), then the *Acceptance-Rejection* method proves itself more effective per iteration in a computation as Table 5 proves.

When increasing the learning rate dramatically, even up to 0.1, the results remain unchanged and the computation time gets reduced enormously. Even though the results in our experiments showed the most optimal solution with the given constraints, increasing the learning rate decreases the reliability of the solution considerably. To consider the results reliable, the learning rate should not exceed 0.1, and lower rates are even more reliable.

5. Conclusion

There is a significant research attention on generating attack graphs that give better insights into the network security, but little attention was given to network hardening. In this paper, we devise a novel solution to the network hardening problem that is motivated by the need to balance network security and usability. In this perspective, we extend the PBIL algorithm to handle usability constraints by proposing the Acceptance-Rejection and Penalty methods. The conducted experiments show that complex initial attack graphs can be simplified dramatically resulting into a more secure network.

References

[1] S. Baluja. Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning. Technical report, DTIC Document, 1994.

[2] S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. In *Machine Learning: Proceedings of the Twelfth International Conference*, pages 38–46, 1995.

[3] S. Baluja and S. Davies. Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. Technical report, DTIC Document, 1997.

[4] D. B. Fogel. An introduction to simulated evolutionary optimization. *Neural Networks, IEEE Transactions on*, 5(1):3–14, 1994.

[5] K. Folly. Multimachine power system stabilizer design based on a simplified version of genetic algorithms combined with learning. In *Intelligent Systems Application to Power Systems, 2005. Proceedings of the 13th International Conference on*, pages 7–pp. IEEE, 2005.

[6] I. P. Gent and T. Walsh. Towards an understanding of hill-climbing procedures for sat. In *AAAI*, volume 93, pages 28–33, 1993.

[7] D. E. Golberg. Genetic algorithms in search, optimization, and machine learning. *Addion wesley*, 1989, 1989.

[8] J. Greene. Population-based incremental learning as a simple, versatile tool for engineering optimization. In *Proceedings of the First International Conf. on EC and Applications*, pages 258–269, 1996.

[9] H. Holm, T. Sommestad, J. Almroth, and M. Persson. A quantitative evaluation of vulnerability scanning. *Information Management & Computer Security*, 19(4):231–247, 2011.

[10] J. Homer and X. Ou. Sat-solving approaches to context-aware enterprise network security management. *Selected Areas in Communications, IEEE Journal on*, 27(3):315–322, 2009.

[11] M. Howard, J. Pincus, and J. M. Wing. *Measuring relative attack surfaces*. Springer, 2005.

[12] S. Jajodia, S. Noel, and B. O. Berry. Topological analysis of network attack vulnerability. In *Managing Cyber Threats*, pages 247–266. Springer, 2005.

[13] P. Manadhata, J. Wing, M. Flynn, and M. McQueen. Measuring the attack surfaces of two ftp daemons. In *Proceedings of the 2nd ACM workshop on Quality of protection*, pages 3–10. ACM, 2006.

[14] S. Noel, S. Jajodia, B. O'Berry, and M. Jacobs. Efficient minimum-cost network hardening via exploit dependency graphs. In *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*, pages 86–95. IEEE, 2003.

[15] X. Ou, S. Govindavajhala, and A. W. Appel. Mulval: A logic-based network security analyzer. In *USENIX security*, 2005.

[16] S. OHare, S. Noel, and K. Prole. A graph-theoretic visualization approach to network risk analysis. In *Visualization for Computer Security*, pages 60–67. Springer, 2008.

[17] J. Pamula, S. Jajodia, P. Ammann, and V. Swarup. A weakest-adversary security metric for network configuration security analysis. In *Proceedings of the 2nd ACM workshop on Quality of protection*, pages 31–38. ACM, 2006.

[18] R. Y. Rubinstein and D. P. Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer Science & Business Media, 2004.

[19] O. Sheyner and J. Wing. Tools for generating and analyzing attack graphs. In *Formal methods for components and objects*, pages 344–371. Springer, 2004.

[20] L. Wang, M. Albanese, and S. Jajodia. *Network Hardening - An Automated Approach to Improving Network Security*. Springer Briefs in Computer Science. Springer, 2014.

[21] L. Wang, A. Singhal, and S. Jajodia. Toward measuring network security using attack graphs. In *Proceedings of the 2007 ACM workshop on Quality of protection*, pages 49–54. ACM, 2007.

[22] S. Yang and X. Yao. Experimental study on population-based incremental learning algorithms for dynamic optimization problems. *Soft Computing*, 9(11):815–834, 2005.