Security Countermeasures Selection Using Attack Graphs

Mohammed Alhomidi, Martin Reed
School of Computer Science and Electronic Engineering
University of Essex
Wivenhoe Park, Colchester
UK
{malhom, mjreed}@essex.ac.uk

ABSTRACT: *Enterprise organizations expend significant resources on security countermeasures to make sure that their networks are protected. Risk analysis is one area of information security research that aims to help organizations in making quick decisions and preventing attackers from compromising their networks. Here we use attack graphs to both display possible attack vectors in simple systems and as an analysis tool for more complex systems. This can be used within a risk analysis strategy. System administrators face constant challenges when they have to decide what countermeasures they must deploy taking into account the minimum budget to deploy a set of countermeasures. The attack graph approach used here aims to minimize the cost of deploying countermeasures. Specifically we develop an approach to find the minimum cut set in dependency attack graphs using a genetic algorithm (GA). We also combine the GA with a local search algorithm to improve the performance of the GA. The minimum cut set is a natural graph representation describing a set of security countermeasures that prevent attackers reaching their targets. More importantly, this work considers shared security countermeasures that are deployed in more than one place in the attack graph. Therefore, there may be one security countermeasure that can fix multiple vulnerabilities. Alternatively, there may be a vulnerability or an exploit in the attack graph that can be stopped by one of multiple countermeasures. The work shows that the problem maps naturally to a binary encoded GA and gives good results without the need to deploy problem specific GA operators.*

Keywords: Attack Graph, Countermeasure Problem, Genetic Algorithm, Secrutiy Countermeasers, Genetic Algorithm

Received: 28 November 2012, Revised 26 December 2012, Accepted 5 January 2013

1. Introduction

Network systems need to be protected against weaknesses that enable attackers to violate security polices of organizations. The process of keeping network systems safe never ends because networked computer systems always bring new technologies and software into deployment and attackers bring new exploits against these systems. Flaws that lead to security vulnerabilities are many faceted and include design or architecture weaknesses, implementation flaws, configuration errors and untimely application of updates or patches. The combination of heterogeneous systems and many types of security weaknesses leads to a security solution becoming a complex system problem.

Despite information security regulations, laws and awareness, security breaches continue to increase. According to an information

security survey report conducted by Infosecurity in 2010, around 92% of participated organizations had a recent security incident [1]. The survey shows that organizations' decision makers need to continually invest and update security countermeasures and they must find out which are the most appropriate set of countermeasures from a possibly large selection. The combination of a complex system problem and the requirement to update systems quickly and repeatedly means that automated systems are required to aid organisations plan their security decision making process.

There has been much research that discusses the use of attack graphs to build security models [2] [3] [4] [5]. Attack graphs are important tools to analyse network security and provide a framework for a security analyst to make decisions about countermeasures. Attack Graphs have been considered a tool to analyse potential threats, particularly in terms of identifying attack paths [6] [7] [8]. Although, it is useful for the network administrators to identify attack paths, it is still a challenging task to determine the set of countermeasures with the minimum cost of deployment especially when the attack graph becomes large and complex. Therefore, this work gives a network administrator the set of countermeasures that should be deployed when considering certain optimisation targets such as minimising cost.

This work discusses security countermeasure selection as a single-objective optimisation problem and develops a genetic algorithm (GA) approach to solve the problem. In particular, we first present the use of an attack graph to show possible attack paths and then determine the minimum cut-set in the attack graph to meet certain objectives. Second, we develop a framework for security countermeasure selection that illustrates how the selection process is carried out and define the role of the system administrator in this framework. Last, we incorporate the attack graph with shared countermeasure optimisation to reflect a real-world scenario in which there can be one security countermeasure that prevents the exploit of multiple vulnerabilities or multiple security countermeasures required to prevent exploitation of one or multiple vulnerabilities.

1.1 Related Works

Attack graphs have been incorporated in many techniques for network hardening: first representing a system and vulnerabilities as an attack graph and then performing some analysis to remove vulnerabilities represented as some features in the attack graph. For example, Jha *et al.* and Sheyner *et al.* [3, 5] provided a minimum critical set which is the minimum set of exploits. If this set of exploits can be prevented, then the network is protected. They developed a model to generate and analyse attack graphs based on model checking techniques. These techniques generate attack graphs with a large number of states, including many redundant states. They reported that the minimum set of attack is equivalent to the Hitting-Set problem [3, 5] and can be solved using a greedy algorithm to find the minimal attack set. The main problem with these techniques is that they generate attack graphs with a large number of states. It is thus computationally expensive to find the minimum set of exploits. The main purpose of these works is to obtain a minimal graph rather than determine the minimum countermeasure cost. Noel *et al.* and Wang *et al.* [7, 9] introduced a model that guarantees the safety of a given set of critical resources by computing attack paths to describe a set of exploits. The model represents the existence of a security condition as a boolean variable. Then, the model represents successful exploits as a boolean function of some set of conditions. The model aims to compute a set of initial security conditions that have the minimum cost and at the same time assure the security of the network. This model requires the security administrator to assign a cost for each individual hardening measure. At the end, the approach selects the configurations with the lowest total cost. The model in those works only considers initial conditions as it is claimed that the graphs may be too large to analyze fully. The work in this paper aims to provide a solution that considers all edges in the graph, not only those related to the initial conditions. This gives rise to the possibility of lower countermeasure cost, although some initial states may not be protected.

Islam and Ligyu introduced a heuristic approach to minimum cost network hardening [8]. They explored disabling some initial conditions based on the decision of network administrators. The authors developed a heuristic algorithm to reduce the sum of costs of set of initial conditions that need to be disabled. Their work is based on a directed attack graph that has two types of vertices as exploits and security conditions. This approach is supposed to help network administrators find a solution when it comes to preventing exploits.

An interesting development in attack graphs is to represent the dependency between states so that the size of the graph is significantly reduced compared to the earlier works reviewed. This type of attack graph is termed a dependency attack graph [6, 10]. Sawilla *et al.* addressed partial cuts in AND/OR directed dependency attack graphs to find the critical paths in the graph. The aim is to help a system administrator determine which are the likely attack paths rather than providing the lowest countermeasure cost. The models show that a k-connected graph is a graph where k is the least number of vertices whose removal disrupts the graph. This can be used to help to determine a critical path in the graph. In particular, their approach used a ranking for each vertex based on the Google's PageRank algorithm to indicate the importance of each asset in the network to

a potential attacker. Then, a loss function is used to provide a cost metric related to each removed vertex. Therefore, the approach considers the problem of reducing the connectivity in AND/OR directed graphs. In fact, this paper aimed to remove vertices in order to maximally decrease the connectivity between the goal vertices and the attacker's vertices. Again, these works do not address countermeasure costs directly but do provide a more compact graph representation.

A recent work presented an approach to generate and analyze minimal attack graph using *planner* search algorithms [11]. The work proposed a planner-based algorithm to overcome the problem of having redundant vertices in enumeration attack graph in [3, 5] . The inputs of the planner search algorithm are initial network configurations, attacker's goals, and exploits. Then, the algorithm generates paths as (i.e. exploit, source host and destination host). It is important to distinguish between *minimal attack graphs* presented in [11] and the *minimum cut set* in attack graphs presented in this paper. While, minimal attack graphs consist of only successful attack paths between the initial state and the target state, the minimum cut set is a set of edges whose removal disrupts the attack graphs and it is described in more detail in later sections.

2. Theoretical Framework

2.1 Dependency Attack Graph

Definition 1 Let $G = (V, E)$ be an AND/OR dependency attack graph that is a directed acyclic graph defined by a set of V of n vertices and a set of E of e directed edges. Each edge (i, j) E has an associated *cost* c_{ij} that represents the cost of an edge.

Each vertex represents a condition state, and each edge represents the causality relations between the system conditions. The graph G has three types of vertices: an AND vertex, (represented graphically as an ellipse), that requires all pre-conditions to be satisfied; an OR vertex (represented as a diamond) that requires only one of the pre-conditions to be satisfied; and a Leaf vertex (represented as a rectangle), that represents a vulnerability or condition. A simple example graph is shown in Figure 1.

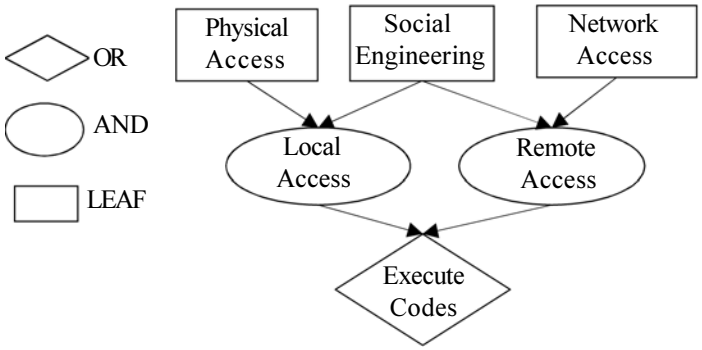

Figure 1. Example of AND/OR attack graph [1]

Figure 1 shows an example of AND/OR dependency attack graph. It describes how an attacker may execute arbitrary codes through either a local access or remote access because it is an OR vertex. We assume that there are three vulnerabilities, two AND vertices and one OR vertex. To reach the final state (execute codes), the attacker must either have a local access *or* remote access. Both, the local and remote access, require two vulnerabilities to be satisfied.

2.2 Graph Cut Set

Definition 2: a cut is a partition of the vertex set V into two parts, S and S'. Each cut represents a set of edges consisting of those edges that have one endpoint in S and another endpoint in S' [12]. An edge cut set is a set of edges whose removal from the graph will disrupt the vertex connectivity. A minimum edge cut set is a cut set of edges the sum of whose costs has the minimum possible value for such a set.

Definition 3: an *s-t cut* is defined with respect to two specific vertices s and t, and is a cut $[S, S']$ satisfying the property that $s \in S$ and $t \in S$ [12].

Figure 2 illustrates a cut with $S = \{1, 2, 3\}$ and $S' = \{4, 5, 6, 7\}$ and shows that the set of edges in this cut are {(2, 5), (2, 4), (3, 4), (3, 6)}. It also can be seen as an *s-t* cut in which $s = 1$ and $t = 7$. This concept of a cut set in graphs has been widely used in attack graphs.

For instance, let us assume that an attacker starts with an initial state vertex number 1 and wants to reach the goal vertex number 7. The attack graph in Figure 2 shows that the attacker has four attack paths that lead to the goal state. In this case, a security administrator can harden the network by disconnecting the graph in which the initial state and goal state are not connected in the graph.

2.3 Problem Statement

In most real world scenarios, a problem is often formulated to address some objective criteria and a decision choice to optimize these objectives is searched for. A problem is solved according to a single or multi-objective optimization. Here we consider security countermeasure selection as a single-objective optimizing problem because we tend to provide the system or network administrators with a set of optimal solutions using a single measure.

This work presents a security countermeasure selection framework in Figure 3 and this demonstrates the entire process of the most appropriate security countermeasure. First, several systems are scanned using Nessus Vulnerability Scanner [13]. Then, the Nessus scanner produces a report which is used as input for Multi-host, Multi-stage Vulnerability Analysis Language tool (MulVal) [14]. MulVAL constructs and generates the attack graph using the Nessus report. Next, the security analyst decides a multiplicity of appropriate countermeasures as well as their costs. Lastly, attack graph information and countermeasure information are included in an optimization algorithm to determine the lowest cost of deploying a set of countermeasures from the range supplied by the analyst.

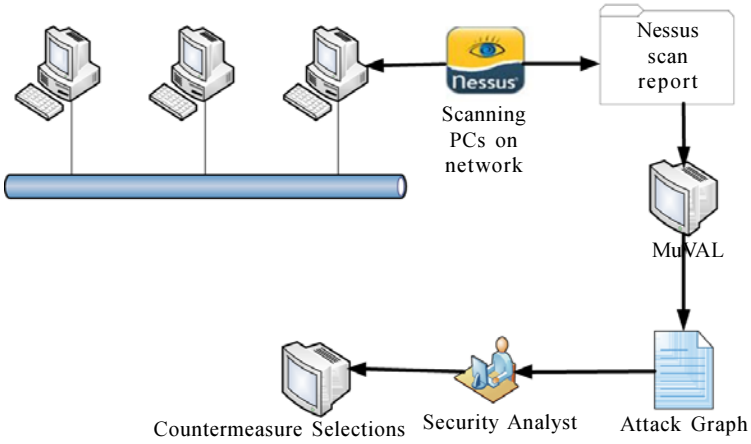

Figure 3. Security countermeasure selection framework

Finding the minimal cut set in dependency attack graphs is not trivial as it is an NP-complete problem [15]. Here we develop a genetic algorithm (GA) that provides a good solution to this problem.

Define a cut set $s \subset E$ which provides a cut with s and t in different sub-graphs and where each $e \in E$ has an associated cost $c(e)$ which defines the cost of deploying a control on e. The aim is to find s such that:

$$min \sum_{e \in s} c_e x_e \tag{1}$$

subject to

$$c(e) \geq 0 \quad \forall e \in E \tag{2}$$

If we define:

$$x_e = \begin{cases} 1, \textit{ if edge e is selected in the cut set} \\ 0, \quad \text{otherwise} \end{cases} \tag{3}$$

Algorithm 1 shows a high level view of the GA incorporating a generic local search method. The inputs are the attack graph which consists of vertices, edges and costs. The output is a minimized cut set which includes a set of edges. These sets of edges are the places where a set of countermeasures must be deployed to prevent attacks against the final protected goal. The costs associated with each edge in the cutset are the actual countermeasure costs that would be incurred to remove a particular exploit step in the graph.

Algorithm: *GA minimal cutset* (*G*)
Input: $G(V, E, C)$
Output: The feasible minimal cutset

1. **Generate initial population** $\mathbb{P}$ of size $\mathcal{N}$ randomly
2. **While termination condition is not met**
3. $\mathbb{P}' \leftarrow$ **Genetic Search** $\mathbb{P}'$
4. **Evaluate each member of** $\mathbb{P}'$ /* **removing invalid solutions***/
5. $\mathbb{P}'' \leftarrow$ **Local Search** $\mathbb{P}'$
6. $\mathbb{P} \leftarrow$ **Generation update** $(\mathbb{P} \cup \mathbb{P}' \cup \mathbb{P}'')$
7. **Keep the Best individual and best** $\mathcal{N}$ **solutions in** $\mathbb{P}$
8. **end while**

Algorithm 1. The GA that finds the minimum cut set in attack graphs

The GA works as the following: First, the GA randomly generates an initial population with a given size $\mathcal{N}$. Each individual member is a binary string where each bit represents an edge and whether it is cut or not cut, *i.e.* representing whether a control is deployed or not deployed. Then, an offspring population $\mathbb{P}'$ is generated by performing genetic operators with certain probabilities. The operators used here are uniform crossover and mutation. Last, the best solutions are selected as the next population $\mathbb{P}$ from the merged population $(\mathbb{P} \cup \mathbb{P}')$. This is repeated (Step 2) until the termination condition is met. Step 3 randomly selects individuals to be updated using the standard GA operators of uniform crossover and mutation. These operators may modify a particular solution such that it is not a valid cut set. Step 4 checks whether this selected individual is a valid cut set or not by testing for a path. The question remains as to what to do with invalid solutions: they may either be removed or "*repaired* " to produce a valid solution. In practice, the authors have found that a repair function considerably slows the iterations and simply removing invalid solutions gives faster convergence results in terms of real-time performance (although in more iterations). Step 4 evaluates each valid solution by computing its fitness according to (1). Step 5 performs a *local search* which aims to improve the performance of individuals in the population. Algorithm 2 presents the local search algorithm. The local search algorithm is used as an iterative process of examining the set of individuals in the neighborhood of the current solution, and replacing it with a better neighbor if one exists. Here a generic local search technique of random *bit-flipping* is utilized. The *bit-flipping* technique examines a set of neighbors of a current solution looking for a better solution than the current one.

Algorithm: *local_search* ($\boldsymbol{y}$)
Input: A valid solution $\boldsymbol{y}$ maxLocalMoves $\mathcal{N}$
Output: Improved Solution $\boldsymbol{y}'$
for $i = 0$ to $\mathcal{N}$ **do**
    $\boldsymbol{y}' \leftarrow \boldsymbol{y}$ {/* bit flipping */}
    $\mathcal{F}\mathbf{1} \leftarrow \boldsymbol{y}'$ {/*fitness of current solution*/}
    $\mathcal{F}\mathbf{2} \leftarrow \boldsymbol{y}'$ {/*fitness of neighbor of current solution*/}
    If $\mathcal{F}\mathbf{2} \leftarrow \mathcal{F}\mathbf{1}$
        $\boldsymbol{y}' \leftarrow \boldsymbol{y}$
    end if
end for

Algorithm 2. The local search algorithm

3. Attack Graph Generation

Figure 4 shows an actual system that is used generate an example attack graph. We provide information about the network and

host including running services, open ports, vulnerabilities and the CVE[1]-IDs of vulnerability on the host in Table 1. The network, shown in Figure 4, is scanned using Nesses. Then, MulVAL [14] is used to produce the attack graph that is presented in Figure 5. The attack graph consists of 44 vertices and 58 edges in which the attacker starts at vertex number 10 and exploits the machine until the target vertex number 1 is reached.

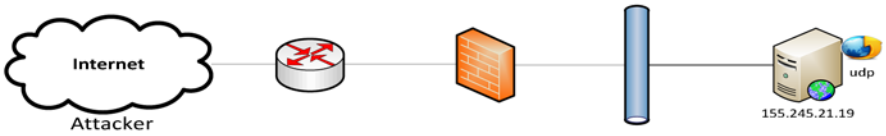

Figure 4. Network Configuration

Host	Services	Open Ports	Vulnerability	CVE-IDs [16]	Operating System
155.245.21.19	ssh	22	SSH server SYN-port scanner	none	Linux Fedora14
	http mysql	80 139 443 587 3306	HTTP Only cookies TCP port scanner Track and trace method	CVE-2012-0053 CVE-2010-0386 CVE-2004-2320 CVE-2003-1567	
	rpc	111	RPC Services Enumeration	CVE-1999-0632	
	snmp udp	161 754 5353	SNMP Agent Default Community Name (public)	CVE-1999-0517	

Table 1. Information about host and vulnerability

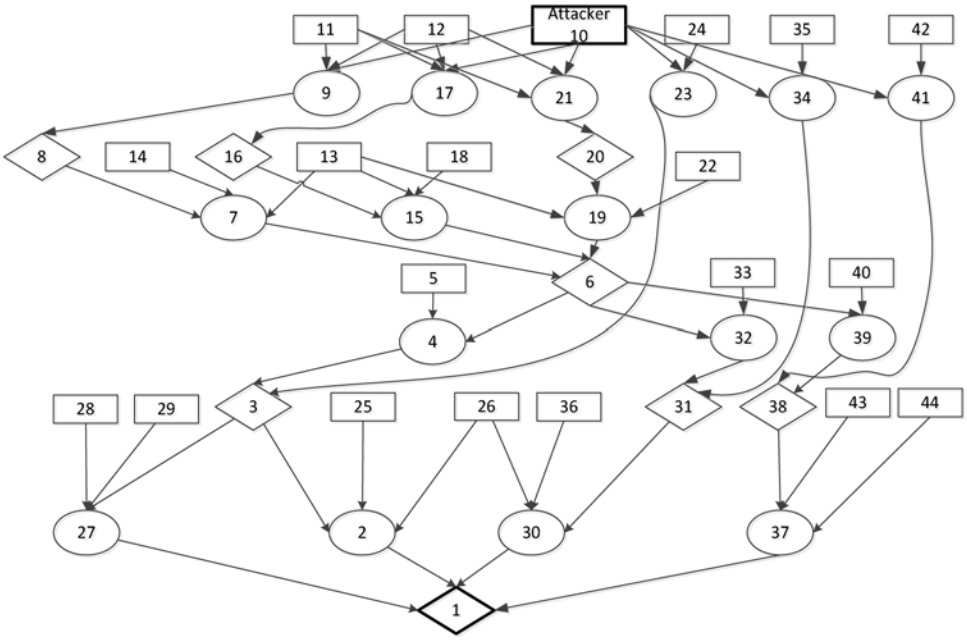

Figure 5. Attack graph generated using MulVAL

4. Selecting Security Countermeasures:

Recalling the security countermeasure selection framework in Figure 3, we have the attack graph and now the task is to analyze the attack graph. The attack graph generated by MulVAL represents the attacker as the initial vertex and then the target vertex

[1]*CVE refers to Common Exposure and Vulnerability. It is a widely known database for security vulnerability.*

which the attacker needs to reach. The attack graph describes exploits and conditions along with vulnerabilities and when certain preconditions are true, the attacker can go further in compromising the network. We assume that a security analyst will first determine a list of countermeasures along with their costs for all exploits and conditions as presented in Table 2. Then, the security analyst assigns the countermeasures to the corresponding edges in attack graph as shown in Figure 6. Sometimes, the security analyst can assign multiple countermeasures on a single edge where in this case the genetic engine selects the countermeasure with the lowest cost. Next, the GA takes this information as inputs and run through the graph to come up with the best solution. Last, the security analyst reviews the solution found by the GA and make required decisions to deploy the selected set of countermeasure.

Security Countermeasure	Cost $
Firewall	500
Packet filter udp 161	200
Packet filter tcp 80	200
Packet filter tcp 443	200
Anti-Malware	300
Packet filter http port	200
Disable track and trace method	40
Patch	50
Antivirus	400
Disable SNMP service	40
Remove MD5	80

Table 2. A List of security countermeasures with their costs

It is seen in Table 2, a list of countermeasure with their costs. In this research, we assign hypothetical costs as these costs are split between purchasing commercial products especially antivirus or just having the security administrators manually creating tables of rules inside a current firewall or router. For example, if a network administrator is going to create rules inside a firewall or enable or disable services, there is no need to purchase products to do such tasks. For instance, the cost of patching web servers is actually the cost of time spent in this task. Let us assume that such task as patching a web server takes an hour from the network administrator, therefore the cost of this countermeasure is equal to the cost of an hour work from the network administrator.

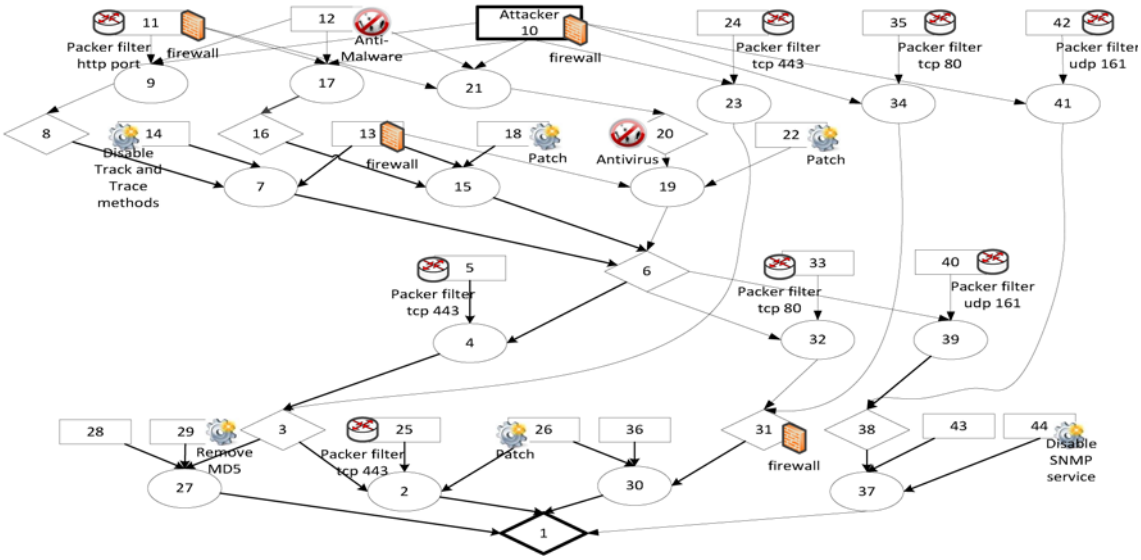

Figure 6. Attack graph generated by MULVAL. Several security countermeasures are applied to stop the attacker reaching the final goal

4.1 Experimental Evaulations of GA appraoch

In this section, we present all results of running the GA to select the best set of countermeasures. We want to show the results of the GA using different parameters such as different sets of crossover and mutation probabilities, population sizes and combining a local search generic method with the GA. In this experiment, we first run the GA with several crossover probabilities (0.1, 0.2, 0.3, 0.5, 0.8 and 1.0). Second, we run the GA with different mutation probability (0.01, 0.2, 0.4, 0.5, 0.7 and 1.0). Then, we run the GA with different population sizes (100, 250, 750, 1000, 2500, and 5000). Last, we run the GA with a local search method with pre-defined local moves (2, 3, 5, 6, 8, 10, 15, 25).

4.2 Running GA with several crossover probabilities:

In this section, we show the results of running the GA with different values of crossover probabilities. We set the other parameters as (mutation probability = 0.001, population size = 500, and maximum no. of generation = 500).

Figure 7 and Table 3 present the results of the GA. They both show that different crossover probabilities produce different sets of countermeasure. At this moment, the GA significantly helps the security administrators to make their decisions based on the results. According to the attack graph in Figure 6, the results in Figure 7, and Table 3, it is easy to determine what set of countermeasure will be deployed to stop the attacker reaching the target vertex. Table 3 describes the best set of countermeasure as well as their costs. On the other hand, the GA selects only one countermeasure as the minimum solution when the crossover probabilities are 0.1, 0.5, 0.8 and 1. It is possible to stop the attacker reaching the target by deploying a firewall which costs 500. However, the network administrator would certainly cost more than the total of the set of countermeasure in Table 3. Thus, it is the best solution.

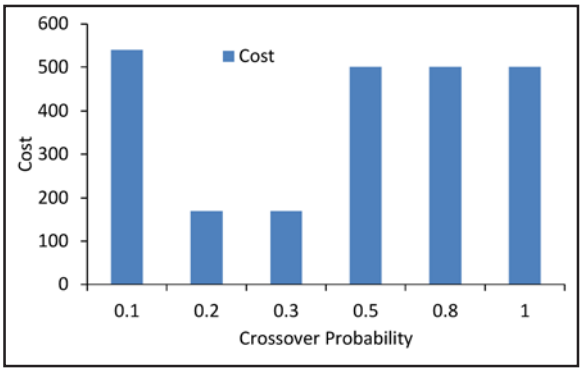

Figure 7. Cost when GA considers several crossover probabilities

Seleted Countermeasures	Cost
Disable track and trace method	40
Patch	50
Remove MD5	80
	Total Cost = 170

Table 3. A List of selected countermeasure and their costs when crossover probabilities 0.2 and 0.3

Figure 8 presents the results of the GA when considering several crossover probabilities. The aim of this is to find out if the GA shows a better conversion toward the best minimum cost in this case 170. It is clear that the GA finds the best cost 170 when the crossover probability is 0.2 and 0.3. The reason that the GA with crossover probability (0.1, 0.5, 0.8, and 1.0) produces 500 as the minimum cost is that the GA got stuck in local optima. It's shown in Table 4 that the number of generations, the time, and the cost of each solution. It is clear that the GA with crossover probability of 0.1 took 1400 ms (1.4 sec) to find the solution 500 in the 10th generation.

4.3 Running GA with several mutation probabilities

This section demonstrates running the GA with different values of mutation probabilities. Note, we kept other GA parameters as the same (crossover probability = 0.8, population size =500, maximum number of generation = 500 and no local search). Typically, the mutation probability is 0.001 (i.e. one bit in every thousand is mutated) [17].

Figure 9 shows the costs for selected countermeasures when using mutation probabilities 0.2, 0.4, 0.5, 0.7, and 1.0. All mutation probabilities produce the cost of 170 except one. It appears that the GA in this particular small attack graph hits the minimum cost 170 when it has mutation probabilities of 0.2, 0.4, 0.5, 0.7 and 1.0 although the speed at which it reaches the minimum cost varies between values of mutation probability.

We can see the conversion of the GA towards the minimum cost in Figure 10 when having several mutation probabilities (0.01, 0.2). The worst GA conversion is when the mutation is 0.01, because the plot line starts at the cost of 590, drops until 500 and continues on 500 as minimum cost. The best GA conversion is when the mutation is 0.4. The plot line starts at a cost of 500 and vary in dropping until it hits the minimum cost 170. Other results are presented Table 5. The only difference in the other mutation probabilities is that each one hits the minimum cost after a different number of generations. For instance, the GA with mutation probability of 1.0 finds the minimum solution in the 8th generation. Table 5 presents the execution time in milliseconds for each running of the GA.

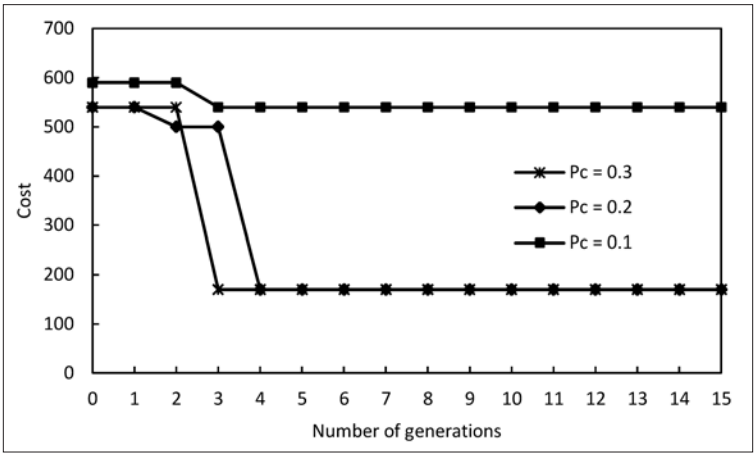

Figure 8. Cost vs. number of generation when running GA with different values of crossover probability, only three crossover values (including best and worst) are shown

Crossover Probability	Number of generation	Time (milliseconds)	Cost
0.1	10	1364	500
0.2	4	602	**170**
0.3	3	**540**	**170**
0.5	2	442	500
0.8	4	415	500
1.0	1	211	500

Table 4. Best cost with running times for each crossover probability

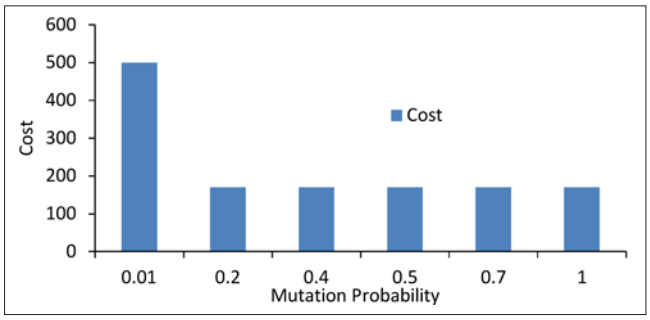

Figure 9. Cost when GA considers several mutation probabilities

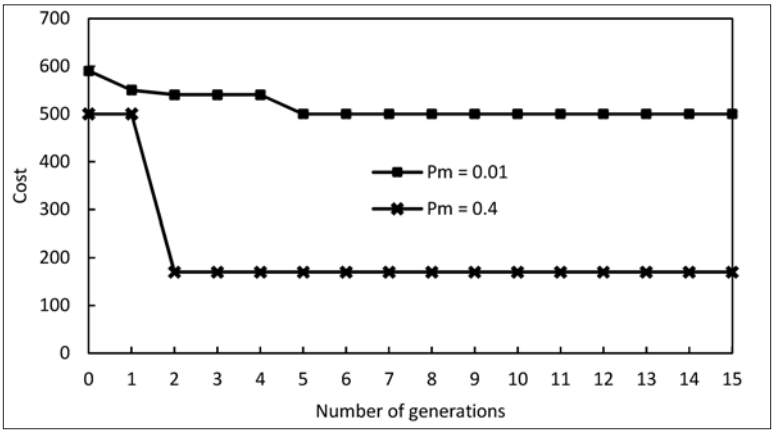

Figure 10. Cost vs. number of generation when running GA with different values of mutation probability

Mutation Probability	Number of Generation	Time (milliseconds)	Cost
0.01	5	804	500
0.2	4	832	**170**
0.4	2	**769**	**170**
0.5	6	1660	**170**
0.7	4	1231	**170**
1.0	8	2257	**170**

Table 5. Best cost with running times for each mutation probability

4.4 Running GA with several population sizes

Determining the right population size for a problem is not straightforward: problems that are solved by GAs are different and there is no obvious *a priori* right population size. Usually too small population sizes tend to converge too quickly to poor solutions and too large population sizes may be computationally intensive, and thus slower in real-time [18]. This section demonstrates the results of running the GA with different population sizes (100, 250, 750, 1000, 2500, and 5000). Other parameters are set as crossover probability = 0.8, mutation probability = 0.001. There are two reasons why the crossover probability 0.8 and the mutation probability 0.001 are still used even though they did not produce the best solution in earlier experiments in Sections 4.2 and 4.3. First, a typical crossover probability is usually high such as 0.8 and a typical mutation probability is usually low like 0.001 [17] [19] [20]. The other reason is that the GA still finds the best solution using the crossover probability 0.8 and the mutation probability 0.001 in the experiment of the current section and next section.

Figure 11 presents all solutions when having several population sizes. The best solution when the population size (750, 1000, 2500 and 5000). The worst solution costs nearly 800 and this is founded when the GA has 10 and 20 as the population size. It is obvious that small population sizes do not produce high quality solutions. As the population size increases, the better solutions are founded. However, the GA usually takes longer times to find high quality solutions in large population sizes.

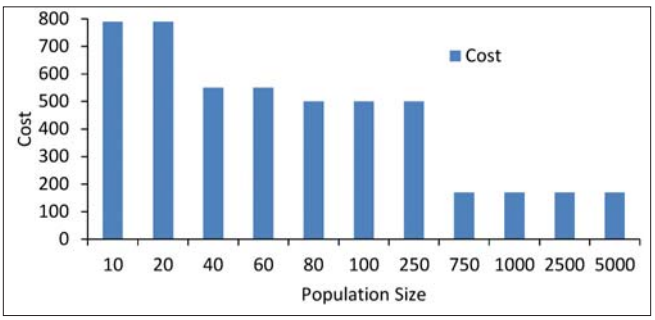

Figure 11. Cost when GA running with different population sizes

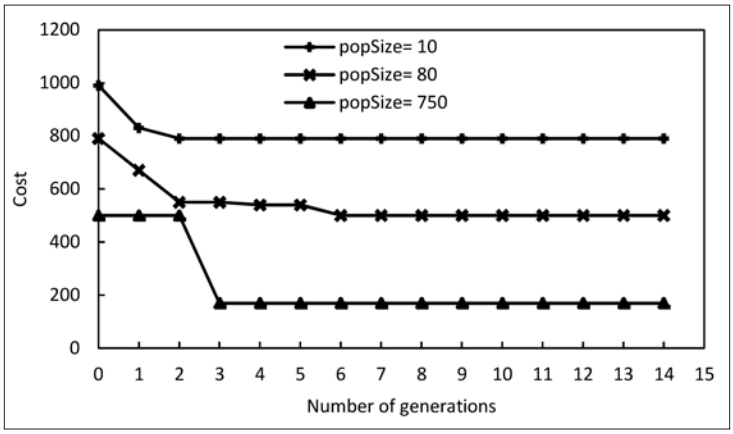

Figure 12. Cost vs. number of generation when running GA with different population sizes

In Figure 12, we notice that the best convergence towards the optimum solution 170 is when population sizes is 750. The GA with population sizes 10 starts with a very high cost. Then, the plot line slightly drops until it reaches the cost of 790. The GA convergence is slow because there is only a few individuals in the population. The GA convergence becomes better when the population size increases to 80 but still not enough to find the best solution. Hence, we are looking for the minimum cost of a set of countermeasure. Therefore, the GA actually finds the best solution with the population sizes of 750. The plot line starts at a cost of 500 and rapidly decreases until it hits the minimum cost. Now, we show that large population sizes in this experiment produce the optimum solution, but they take longer in real-time terms when the population is too large. Table 6 presents the number of generation, the time and cost for all experiments using different population sizes.

Population Size	Number of Generation	Time (milliseconds)	Cost
10	2	6	790
20	1	6	790
40	3	19	550
60	4	215	550
80	6	77	500
100	1	92	500
250	4	345	500
750	3	**1093**	**170**
1000	3	1614	**170**
2500	2	5598	**170**
5000	2	23802	**170**

Table 6. Best cost with running times for each population size

4.5 Running GA with a local search method

This research develops a local search method for two reasons: to escape local optima and to speed up the convergence of the GA toward the global optima. The local search, which is explained before in Problem Statement section, uses the bit-flipping technique to move to neighbors of current solution. The local move number indicates how many neighbors will be explored. Figure 13 shows that all experiments using local search produce the best solution 170, unlike previous experiments. We generally find that the local search greatly improves the performance of the GA to escape local minima. In Figure 14, two plot lines are shown and both start at lower costs and converge toward the best solution. However, the local search with 8 moves finds the best solution from the first generation. This is the first time the best solution is founded from the first generation.

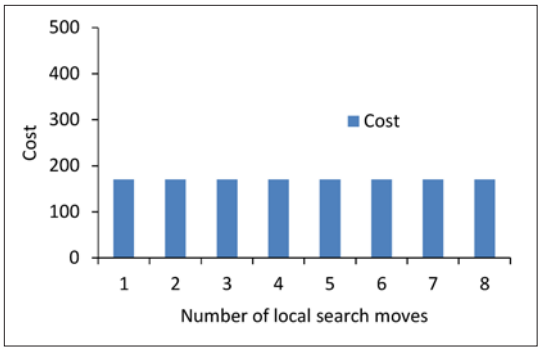

Figure 13. Cost when GA is combined with Local Search

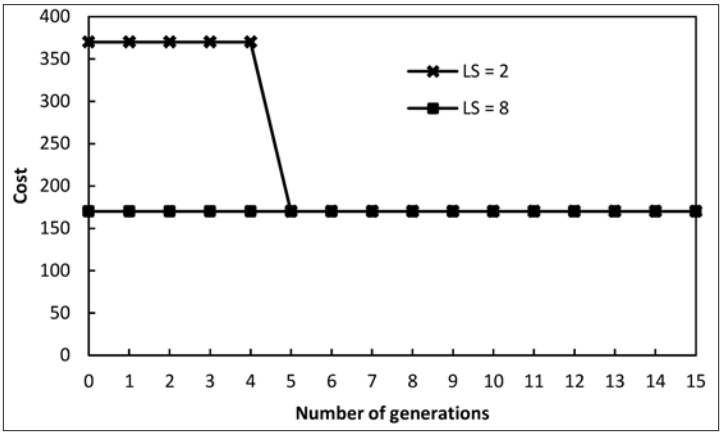

Figure 14. Cost vs. number of generation when running the GA with the local search

Table 7 presents the number of generation, running times and costs for different experiments with different local search moves. The experiments includes local search move of (2, 3, 5, 6, 8, 10, 15, 25). The local search moves 8, 10, 15 and 25 produce the best solution from the first generation. As the number of local search moves decreases, the more generations are needed to find the best solution.

Local Search Moves	Number of Generation	Time (milliseconds)	Cost
2	5	1353	**170**
3	3	961	**170**
5	3	996	**170**
6	2	806	**170**
8	1	**423**	**170**
10	1	437	**170**
15	1	468	**170**
25	1	474	**170**

Table 7. Best cost with running times for each local search move value

5. Conclusion

This work shows how attack graphs can be used to aid the decision on which controls to deploy and thus harden networks with a reduced cost. While attack graphs have been widely used, this work shows how a particular representation, the dependency

attack graph, can be used to solve this problem. It is particularly attractive as it has much lower graph size than other representations such as a full state graph.

This paper demonstrates that attack graphs are very useful in terms of analysing security of networked systems. However, it is important to understand that when the attack graph is very large due to the increase in number of vertices and edges (i.e. increase in the number of systems, conditions and vulnerabilities) it becomes a very complex problem for the human to analyse or even understand the graph. In this work, we aim to take advantages of attack graph and automate the problem of security countermeasure selections.

This work develops a GA approach that provides a solution for the problem of selecting a set of countermeasure by representing the problem as the minimum cut set problem. In particular, it shows that a generic GA can give satisfactory performance without the need to deploy problem specific operators such as a repair function or problem specific local searches. Furthermore, it shows that the use of generic GA local search can significantly improve the optimization performance.

References

[1] Poter, C., Beard, A. (2010). Information Security Breaches Survey, Infosecurity Europ, London, UK.

[2] Ammann, P., Wijesekera, D., Kaushik, S. (2002). Scalable, graph-based network vulnerability analysis, *In*: Proceedings of the 9th ACM conference on Computer and communications security, Washington, DC, USA, p. 217-224.

[3] Jha, S., Sheyner, O., Wing, J. (2002). Two formal analyses of attack graphs, *In*: Computer Security Foundations Workshop. Proceedings. 15th IEEE, p. 49-63.

[4] Jajodia, S., Noel, S. (2007). Topological vulnerability analysis: A powerful new approach for network attack prevention, detection, and response: World Scientific Press.

[5] Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J. M. (2002). Automated generation and analysis of attack graphs, *In*: Proceedings of the 2002 IEEE Symposium on Security and Privacy (Oakland 2002), Oakland, CA.

[6] Sawilla, R., Ou, X. (2008). Identifying critical attack assets in dependency attack graphs, *Computer Security-ESORICS* 2008, p. 18-34.

[7] Noel, S., Jajodia, S., O'Berry, B., Jacobs, M. (2003). Efficient Minimum-Cost Network Hardening Via Exploit Dependency Graphs, *In*: Proceedings of the 19th Annual Computer Security Applications Conference, Las Vegas, USA, p. 86.

[8] Islam, T., Lingyu, W. (2008). A Heuristic Approach to Minimum-Cost Network Hardening Using Attack Graph, *In*: New Technologies, Mobility and Security. NTMS '08, p. 1-5.

[9] Wang, L., Noel, S., Jajodia, S. (2006). Minimum-cost network hardening using attack graphs, *Computer Communications,* 29, p. 3812-3824.

[10] Sawilla, R., Burrell, C. (2009). Course of action recommendations for practical network defence, *Defence R&D Canada Technical Memorandum*, 130, 1–31.

[11] Ghosh, N., Ghosh, S. (2012). A planner-based approach to generate and analyze minimal attack graph, *Applied Intelligence*, 36, 369-390.

[12] Ahuja, R. K., Magnanti, T. L., Orlin, J. B. (1993). *Network flows: theory, algorithms, and applications*: Prentice Hall.

[13] Security. T. N. (2012, 11 October). *Nessus vulnerability scanner*. Available: http://www.tenable.com/products/nessus/nessus-product-overview

[14] Ou, X., Govindavajhala, S., Appel, A. W. (2005). MulVAL: A logic-based network security analyzer, *In*: 14th USENIX Security Symposium, p. 8-8.

[15] Noel, S., Jajodia, S. (2004). Managing attack graph complexity through visual hierarchical aggregation, *In*: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security, Washington DC, USA, p. 109-118.

[16] Corporation,T. M. *Common Vulnerabilities and Exposures*. Available: http://cve.mitre.org/

[17] Coley, D. A. (1999). *An introduction to genetic algorithms for scientists and engineers*: World Scientific Pub Co Inc.

[18] Michalewicz, Z. (1996). *Genetic algorithms+ data structures*: Springer.

[19] Eiben, A. E., Smith, J. E. (2003). *Introduction to Evolutionary Computing*: SpringerVerlag.

[20] Back, T. (1996). *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*: Oxford University Press, USA.