An Analytical Framework for Measuring Network Security using Exploit Dependency Graph

Parantapa Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Email: parantapa@cse.iitkgp.ernet.in

Soumya. K. Ghosh
School of Information Technology
Indian Institute of Technology, Kharagpur
Email: skg@iitkgp.ac.in

***Abstract*—Attack graph is a popular tool for modeling multi-staged, correlated attacks on computer networks. Attack graphs have been widely used for measuring network security risks. A major portion of these works, have used host based or state based attack graphs. These attack graph models are either too restrictive or too resource consuming. Also, a significant portion of these works have used 'probability of successfully exploiting a network' as the metric. This approach requires that the 'probability of successfully exploiting individual vulnerabilities' be known a priori. Finding such probabilities is inherently difficult.**

This work uses exploit dependency graph, which is a space efficient and expressive attack graph model. It also associates an additive cost with executing individual exploits, and defines a security metric in terms of the 'minimum cost required to successfully exploit the network'. The problem of calculating the said metric is proved to be NP-Complete. A modified depth first branch and bound algorithm has been described for calculating it. This work also formulates, a linear time computable, security metric in terms of the 'expected cost required to successfully exploit the network' assuming a random attacker model and an uncorrelated attack graph.

***Index Terms*—Network Security, Attack Graph, Exploit Dependency Graph, Risk Quantification**

I. Introduction

Attack graph is a tool for modeling multi-staged, correlated attacks on computer networks. Attack graphs have great potential use in detecting hidden multi staged attack paths, as well as gaining deeper understanding of security risks. A number of works [1]–[8], have been published in recent times for measuring and analyzing network security using attack graphs.

Attack graph models can be broadly classified into three categories: *host based attack graphs* [9]–[13], *state based attack graphs* [11]–[14], and *exploit dependency graphs* [1]–[3], [15]. Much of the works on security metrics have been done using host based and state based attack graphs [9]–[14], [16], [17]. Works on security metrics using exploit dependency graphs have been mostly limited to using 'probability of successfully exploiting the network' as the security metric [1]–[3], [15].

Host based and state based attack graphs suffer some basic limitations. Host based attack graphs can only model exploits which give the attacker access to a new host from an old host. Many exploits are executed in stages, and may share these stages with other potential exploits. Such stages cannot be effectively represented using host based attack graphs. Moreover, many families of exploits, such as local privilege escalation, man in the middle attacks, etc. cannot be efficiently represented with host based attack graphs.

State based attack graphs represent the full state of the network within every vertex of the graph. Thus they suffer from an exponential space complexity. If a network's state can be encoded by n boolean security conditions, then the corresponding state based attack graph has 2^n vertices.

Security metrics which use 'probability of successfully exploiting the network' suffer from an inherent problem. This approach requires that the 'probability of successfully exploiting individual exploits' be known a priori. These probabilities are nearly impossible to obtain, as the result depends on numerous factors that are practically impossible to enumerate or evaluate. This has led authors in [2] to use the popular vulnerability metrics *Common Vulnerability Scoring System* or CVSS [18], [19] to generate values between zero and one, in lieu of the actual probability.

This work tries to develop an analytical framework of a security metric, which is devoid of the problems as discussed above. The major contributions of this work are as follows:

- Extending the concept of 'exploit dependency graphs' to 'regular attack graphs', which have additional constraints enforcing 'monotonicity' and removing possible *irregularities* allowed by the basic definition.

- Formulation of a theoretical framework for measuring network security using exploit cost measures which are additive in nature.
- Proof of NP-Completeness for the problem of finding the 'minimum cost to exploit the network'.
- Development of a *complete* and *optimal* depth first, branch and bound algorithm, using polynomial space, to find an exploit sequence with minimum cost.
- Development of a linear time algorithm to estimate the 'expected cost to exploit the network', required by a random attacker on an *uncorrelated* attack graph.

The rest of the paper is organized in four sections. Section II formally defines attack graphs and introduces the notion of 'regular attack graphs'. Section III defines the problem of calculating the 'minimum cost to exploit a network' and proves it to be NP-Complete. The section also presents a depth first branch and bound based algorithm for finding a 'minimum cost exploit sequence' required to exploit the network. Section IV presents a linear time algorithm to estimate the 'expected cost to exploit the network' by a random attacker on an uncorrelated attack graph. Section V concludes the paper with future direction of work.

II. Attack Graph

An attack graph is a representation of a network and the exploits that can be executed on its elements. This work uses exploit dependency model of attack graphs [1]–[3], [15]. In this model, the network's state is encoded in form of boolean security conditions. The vertices in the graph are of two types, representing the security conditions and the exploits that connect them [1]–[3], [15], [20], [21]. Every exploit requires a certain set of security conditions to be true. On execution of the exploit another disjoint set of security conditions become true. From an attacker's standpoint, security conditions represent privileges that are already acquired or can be later gained by executing the exploits.

The term *exploit* is used quite loosely in this context. Attacking a network may, and in most cases does, make use of systems and services in regular intended ways, such as accessing a remote system. For the purpose of attack graph terminology, this work uses the term exploit to mean any atomic action performed by the attacker.

A. Definition

Definition 1. Given a finite set of exploits E, a finite set of conditions C, a *requires* relation $R \subseteq C \times E$, and an *implies* relation $I \subseteq E \times C$, the attack graph A is defined as $G(E \cup C, R \cup I)$ [21].

Thus, attack graph is a directed bipartite graph. The condition c is a *pre-condition* or *post-condition* of the exploit e if $(c, e) \in R$ or $(e, c) \in I$ respectively.

For an exploit e, the set of pre-conditions and post-conditions of the exploit are denoted by $Pre(e)$ and $Post(e)$ respectively. Similarly, for a condition c, the set of exploits for which it is a pre-condition and post-condition are denoted by $Req(c)$ and $Imp(c)$ respectively.

$$\begin{aligned} Pre(e) &= \{c \mid (c, e) \in R\} \\ Post(e) &= \{c \mid (e, c) \in I\} \\ Req(c) &= \{e \mid (c, e) \in R\} \\ Imp(c) &= \{e \mid (e, c) \in I\} \end{aligned}$$

In an attack graph, the traditional notions of *walks* and *paths* do not apply. The concept of *executing* an exploit and *satisfaction* of a condition are used instead. An exploit is said to be executable if and only if all of its pre-conditions are satisfied. Execution of an exploit is an event which marks the escalation of an attacker's privilege in the network. If an exploit is executed then all of its post-conditions are satisfied.

B. Regular Attack Graph

Many works on attack graph [15], [20]–[22] have used the requirement of monotonicity. Monotonicity, in attack graph terminology, simply means that, an exploit can only depend on security conditions being true. In other words, this means execution of an exploit must not require the absence of any privilege or, an attacker never loses any privilege gained in course of his attacks to execute further exploits. Monotonicity assumption makes the problem of calculating reachability of security conditions much simpler. However this also means that certain exploits, such as denial of service, cannot be easily modeled by an attack graph using monotonicity.

Alone, the definition of attack graph presented above leaves room for a lot of *irregularities*. By irregularity one means, notions that are counter intuitive to the idea of attacking a network and the concept of monotonicity. Here the notion of *regular* attack graph has been formalized using the concepts of monotonicity.

$$\forall e \in E(Pre(e) \neq \emptyset \text{ and } Post(e) \neq \emptyset) \quad (1)$$

$$\forall c \in C(Req(c) \neq \emptyset \text{ or } Imp(c) \neq \emptyset) \quad (2)$$

$$\forall c \in C(\nexists e \in E \mid c \in Pre(e) \text{ and } c \in Post(e)) \quad (3)$$

Eqn. 1 ensures that every exploit has at least one pre-condition and one post-condition. As without any pre-condition the executable status of an exploit becomes ambiguous. The definition does not state the executable status of exploits without pre-conditions. The requirement of at least one post-condition ensures that the attack graph does not contain exploits which do not provide any privileges to the attacker. These are also the only relevant exploits from the point of view of system administrators and attackers.

Eqn. 2 asserts that for every condition there exists at least one exploit for which it is pre-condition or a post-condition. A security condition not related to any exploit does not have any significance from the point of execution of exploits. They can never be reached (unless they are also starting conditions as described later) and do not help in execution of exploits. Their redundant presence is irrelevant for purpose of security analysis.

Eqn. 3 makes sure that a condition can not simultaneously be both pre-condition and post-condition of the same exploit. An exploit cannot simultaneously require and then provide the same security condition. This situation violates this intuitive notion of gaining privileges by executing exploits. Since an already present privilege cannot again be gained.

This work only uses *regular* attack graphs. For the rest of the paper, attack graph is used to mean *regular* attack graphs.

C. Exploit Sequence

Associated with an attack graph and an attacker model, is a set of starting conditions $S \subseteq C$. All conditions in the set of starting conditions are always satisfied. The attacker starts with this initial set of conditions. Using these, the attacker tries to attain the goal conditions $G \subseteq C$. The two major points of analysis of attack graphs involve determining whether these goal conditions are attainable and, if so, what is the cost of doing it.

The same attack graph may have associated with it different starting and goal conditions. An attack graph represents the network's security conditions independent of any attacker. Starting and goal conditions add the attacker's perspective to the attack graph. Whatever be the capabilities and goal of an attacker, the attack graph of the network remains fixed.

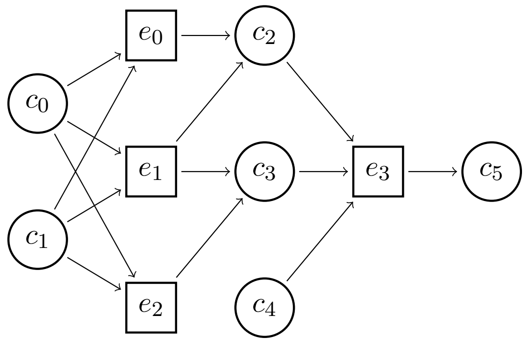

Fig. 1. Example attack graph with six security conditions and four exploits

An exploit sequence p with respect to its starting conditions S and goal conditions G is a sequence of exploits $\langle e_1, e_2, \ldots, e_n \rangle$ with the following conditions:

$$Pre(e_1) \subseteq S$$

$$Pre(e_i) \subseteq \left(\bigcup_{j=1}^{i-1} Post(e_j) \right) \cup S \qquad \text{for } i = 2, 3, \ldots, n$$

$$G \subseteq \left(\bigcup_{i=1}^{n} Post(e_i) \right) \cup S$$

Fig. 1 shows an example attack graph with six security conditions and four exploits. In this example, an attacker begins with the starting conditions c_0, c_1, c_4 and tries to acquire the goal condition c_5. The attacker can execute exploits e_0, e_2 or e_1 to gain the post-conditions c_2, c_3. The pre-conditions of exploit e_3 are c_2, c_3, c_4 which the attacker has now gained and can thus move to execute e_3. On executing e_3 the attacker achieves the goal condition c_5. In this example, the attacker can use the exploit sequence $\langle e_0, e_2, e_3 \rangle$ or $\langle e_1, e_3 \rangle$.

D. Cost of Executing an Exploit

For the purpose of network security quantification, it is essential to associate a notion of cost or difficulty to the act of executing an exploit. For an attacker, executing an exploit requires knowledge, time, access to specialized resources such as password crackers, sniffers, other software, and hardware tools. Depending on the type of the exploit, it may contain some probabilistic runtime components. The exploits may require some interaction from the user. It may also depend on some race conditions in the vulnerable application's execution logic. Many exploits are totally deterministic.

Works on security metrics using exploit dependency graphs [1]–[3], [15] have been mostly limited to using ‘probability of successfully executing an exploit’ as the cost function. The true probabilities for such events are nearly impossible to obtain, as the result depends on numerous factors that are practically impossible to enumerate or evaluate. The work in [2] uses the popular vulnerability metric, CVSS [18], [19] to generate values between zero and one, in lieu of the actual probability.

It must be noted that the indeterminacy in determining the ‘cost of executing an exploit’ comes mostly from lack of accurate information about the exploits and the attackers, and not from *randomness of the outcome of an attack*. Thus, the cost of executing an exploit can thus be much better represented using the phrase ‘difficulty of executing the exploit’ rather than the ‘probability of succeeding in an attempt’. This representation makes the indeterminacy explicit.

The design of a proper cost function is in itself a major research work. The focus of this paper is to develop a theoretical framework for security evaluation, rather than actual development of practical tools, which we consider the immediate future work. Thus we refrain ourselves from describing a concrete cost function in this work. However we introduce the basic ideas necessary for developing such a cost function by making judicious use of CVSS.

CVSS is a system for associating a score to a discovered and publicly disclosed vulnerabilities. The CVSS scoring system consists of three parts viz. base, temporal, and environmental.

The CVSS base metric consists of six components viz. access vector, access complexity, authentication scores, and the impact scores confidentiality, integrity and, availability. Access vector describes the type of access required to execute the vulnerability, such as, local access, access within a subnet, and remote access. Authentication describes the number of times the exploit has to authenticate itself to the vulnerable system. Access vector and authentication are two aspects of the CVSS scoring system that are already incorporated in the attack graph model by means of exploits and their pre-conditions. The impact scores of CVSS tries to describe the result of an exploit in *isolation* and associates a score with it. The impact scores essentially describe post-conditions of exploits in an attack graph. To summarize, five of the six components of CVSS base metric describes features that an attack graph already models. Thus it does not provide any information from the perspective of ’cost or difficulty of exploit execution’.

Access complexity is the only metric that truly tries to describe notion of cost or difficulty of executing an exploit. It must also be noted that it does so using fuzzy terms *easy*, *medium*, and *hard*.

The temporal metric group focuses on characteristics of a vulnerability that can change over time. The environmental metric group is used to modify the CVSS metric on the basis of requirement of the organization using it. One can map the temporal metric group component’s exploitability and report confidence to the concept difficulty of executing an exploit. The temporal metric component remediation level and the environmental metric group as a whole doesn’t map to the notion of attacking and the notion of difficulty of executing an exploit.

From the above discussion that very few of the components of CVSS are actually relevant for developing a ‘cost of exploitation’ score, and thus suitable for using with attack graphs. Moreover the scores thus developed are more appropriately expressed as fuzzy values that can be then *defuzzyfied* to obtain appropriate crisp scores usable as a cost function with attack graphs.

III. Minimum Cost Exploit Sequence

The main motivation of performing any form of network security analysis, is to determine how secure a network is. In this work we restate the problem as, to find the ‘minimum cost of successfully exploiting the network’. Successfully attacking the network means acquiring a defined set of goal privileges or conditions from a given set of starting privileges or conditions.

To analyze the cost of exploiting a network, one associates a cost parameter to every exploit $w : E \to \mathbb{R}^+$ that can be used in the network. The cost function is additive in nature. Given a cost function w the cost of an exploit sequence $p = \langle e_1, e_2, \ldots, e_n \rangle$ is defined as:

$$w(p) = \sum_{i=1}^{n} w(e_i)$$

Let P be the set of all exploit sequences with respect to the starting conditions S and goal conditions G. The minimum cost for reaching the goal conditions G from the starting conditions S is defined as:

$$\rho = \begin{cases} \min w(p) & \forall p \in P \quad \text{if } P \neq \emptyset \\ \infty & \text{otherwise} \end{cases}$$

A minimum cost exploit sequence p_m is defined to be an exploit sequence such that $w(p_m) = \rho$.

Lemma 1. The problem of finding a minimum cost exploit sequence p_m given an attack graph A, a set of starting conditions S, a set of goal conditions G, and a weight function w is NP-Complete.

To prove the above statement, we first state the decision version of the problem. Given an attack graph A, a set of starting condition S, a set of goal conditions G, a weight function w, and a cost limit x, the problem is to decide the existence of an exploit sequence p such that $w(p)$ is less than a given cost x.

Proof: The problem is in NP. The above problem is clearly in NP. The exploit sequence itself is a certificate for the solution. The problem of validating the exploit sequence and calculating its total cost can be performed in polynomial time. Algorithm 1 verifies an exploit sequence in polynomial time. $\square$

Algorithm 1 Verify an exploit sequence

```
1:  function VERIFY-EXP-SEQ($A, S, G, w, x, \langle e_i \rangle$)
2:      $Q \leftarrow S$
3:      $cost \leftarrow 0$
4:      for all $e \in \langle e_i \rangle$ do
5:          if $Pre(A, e) \subseteq Q$ then
6:              $Q \leftarrow Q \cup Post(A, e)$
7:              $cost \leftarrow cost + w[e]$
8:          else
9:              return FALSE
10:     if $G \subseteq Q$ and $cost \leq x$ then
11:         return TRUE
12:     else
13:         return FALSE
```

Algorithm 1 takes as input an attack graph A in form a pre-condition, post-condition list. For every exploit e in the attack graph, two lists are maintained, one for its pre-conditions, and one for its post-conditions. The set of starting conditions S and the set of goal conditions G are also maintained as lists. Cost of the individual exploits w is taken as input in form of an array of floating point numbers. The exploit sequence $\langle e_i \rangle$ is also taken in form of a list.

The algorithm internally maintains a set of currently achieved goal conditions Q which is implemented as a hash table. *cost*, the cost measure, keeps track of the cost incurred by the already executed exploits.

The algorithm takes one exploit at a time from the exploit sequence $\langle e_i \rangle$, checks whether the exploit's pre-conditions are satisfied by the currently available set of conditions Q, adds its cost to *cost*, and finally adds its post-conditions to the set Q. At the end of the exploit sequence, the algorithm checks whether all the requirements of the goal conditions are met and the cost of the exploits are within the bound x.

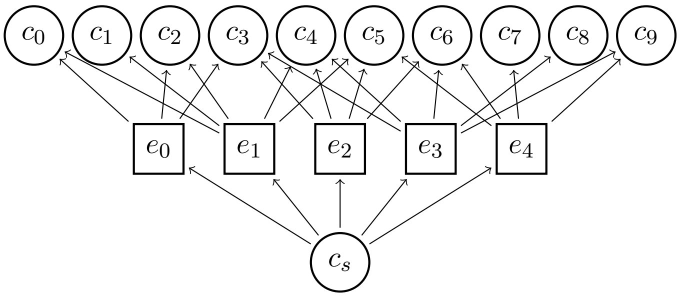

Fig. 2. Reduction of the set cover problem using minimum cost exploit sequence problem

Initialization of Q and *cost* takes $O(|S|)$ and $O(1)$ time respectively. The total time spent in checking the condition at line 5 and the union operation in line 6 are $O(|R|)$ and $O(|I|)$ respectively. Here R and I are the requires and implies relation set of the attack graph A. The total time spent in the assignment operation on line 7 is $O(|\langle e_i \rangle|)$. The condition on line 10 can be checked in $O(|G|)$ time. The whole algorithm runs in linear time in the size of the input.

Proof: The problem is NP-Hard. To prove the problem is NP-Hard one can reduce the set cover problem with it. The set cover problem is NP-Complete [23].

Given a finite family $\mathcal{F}$ of subsets of $\mathcal{S} = \langle \epsilon_1, \epsilon_2, \ldots, \epsilon_n \rangle$, and a positive integer k, the set cover problem is to decide the existence of a subfamily $\mathcal{C} \subseteq \mathcal{F}$ of size less than or equal to k which is a cover of $\mathcal{S}$.

To solve the set cover problem one can use the following reduction. For every element $\epsilon_i \in \mathcal{S}$ a condition c_i is created. For each set $S_j \in \mathcal{F}$ an exploit e_j is created. The condition $c_i \in Post(e_j)$ if and only if $\epsilon_i \in S_j$. Additionally, we create one more condition vertex c_s. $Pre(e_j) = \{c_s\}$ for all e_j. The starting condition set is defined as $S = \{c_s\}$ and the goal condition set is defined as $G = \{c_j\}$. The cost function $w(e) = 1$ for all e and $x = k$.

If $\mathcal{C}$ be a set cover with less than k sets and $\{e_n\}$ be the set of exploits in the generated attack graph corresponding to the set cover $\mathcal{C}$, then any sequence of exploits in $\{e_n\}$ is a valid exploit sequence with cost less than x. Conversely if $\langle e_n \rangle$ be a valid exploit sequence for the generated attack graph with cost less than x, then $\mathcal{C}$ is a valid set cover of the original problem with size less than k where $\mathcal{C}$ contains the subsets corresponding to exploits in the exploit sequence $\langle e_1, e_2, \ldots, e_n \rangle$. $\square$

To understand the reduction with an example, let $\mathcal{S} = \{\epsilon_0, \epsilon_1, \ldots, \epsilon_9\}$ be a set with ten members. Let the family of subsets $\mathcal{F} = \{S_i\}$ be as follows:

$$S_0 = \{\epsilon_0, \epsilon_2, \epsilon_3\},$$
$$S_1 = \{\epsilon_0, \epsilon_1, \epsilon_2, \epsilon_4, \epsilon_5\},$$
$$S_2 = \{\epsilon_3, \epsilon_4, \epsilon_5, \epsilon_6\},$$
$$S_3 = \{\epsilon_3, \epsilon_4, \epsilon_6, \epsilon_8, \epsilon_9\},$$
$$S_4 = \{\epsilon_5, \epsilon_6, \epsilon_7, \epsilon_9\}$$

Fig. 2 shows the corresponding attack graph. In the above problem $\langle e_0, e_1, e_3, e_4\rangle$ is a valid exploit sequence of cost *four*. Executing these four exploits the attacker has access to all the goal conditions. Correspondingly, a cover of the original set cover problem is given by $\mathcal{C} = \{S_0, S_1, S_3, S_4\}$ which is of size four.

Algorithm 2 Finding a minimum cost exploit sequence

```
1: function FIND-EXP-SEQ($A, S, G, w$)
2:     PUSH($Q$, NIL)
3:     $cost' \leftarrow \infty$
4:     $seq' \leftarrow$ NIL
5:     while $Q \neq \emptyset$ do
6:         $seq \leftarrow$ POP($Q$)
7:         $conds \leftarrow S$
8:         $cost \leftarrow 0$
9:         for all $e \in seq$ do
10:            $conds \leftarrow conds \cup Post(A, e)$
11:            $cost \leftarrow cost + w[e]$
12:        if $G \subseteq conds$ then
13:            if $cost < cost'$ then
14:                $cost' \leftarrow cost$
15:                $seq' \leftarrow seq$
16:        else
17:            $exps \leftarrow \emptyset$
18:            for all $e \in E - seq$ do
19:                if $Pre(A, e) \subseteq conds$ then
20:                    if $|Post(A, e) - conds| > 0$ then
21:                        $exps \leftarrow exps \cup e$
22:            for all $e \in exps$ do
23:                if $cost + w[e] < cost'$ then
24:                    PUSH($Q, seq + e$)
25:    return $seq'$
```

For the rest of this section we describe Algorithm 2 which can be used to find a minimum cost exploit sequence.

Algorithm 2 takes as input an attack graph A in form of pre-condition and post-condition lists. The start and goal conditions S and G are also taken as lists. The cost function for individual exploits w is taken in the form of an array of floating point numbers.

The algorithm internally uses a stack Q. The stack is used to store exploit sequences. The exploit sequences themselves are maintained as lists. Q is initialized by pushing an empty sequence into it. seq' stores the currently known best exploit sequence. seq' is initialized to NIL. Also maintained is the cost of best known exploit sequence $cost'$. $cost'$ is initialized to ∞

Algorithm 2 is essentially a depth first branch and bound search algorithm to search for the minimum cost exploit sequence. The algorithm stores partial exploit sequences in the stack Q. In each iteration of the algorithm (line 5), an exploit sequence seq is popped from the stack.

The algorithm checks if seq reaches the goal conditions (line 12). If so its cost is checked with that of the currently best known sequence seq'. If its cost is less that of the currently best known sequence, the new cost is stored in $cost'$ and the sequence itself is saved in seq'.

If the popped sequence is not sufficient to reach the goal, all the exploits which can be executed after executing the popped sequence, $exps$ is generated (lines 17–21). This list excludes any exploits that do not have any new post-conditions left to be obtained. For each such exploit $e \in exps$, a new sequence is generated by appending e to seq. If the cost of the newly generated sequence greater than $cost'$, then the generated sequence is ignored. Otherwise it is pushed on to the stack Q (line 24).

To understand the memory requirement of Algorithm 2 it must first be realized that the size of an exploit sequence is $O(|C| \log |E|)$. Here C and E are the condition and exploit set for the attack graph A. There can be no more than $|C|$ exploits in any exploit sequence since every exploit contributes at least one privilege to the attacker. Also each exploit can be uniquely identified using $\log |E|$ bits. The depth of the created search tree can also not be more than $|C|$. The maximum branching factor is the maximum possible number of exploits available at any level or $|E|$. Thus the worst case space requirement of the algorithm is $O(|C|^2|E| \log |E|)$. For this analysis we can safely ignore other space requirements of the algorithm as the space requirement of the stack Q dominates.

Algorithm 2 is *complete* and *optimal*. It always finds an exploit sequence if there exists one. Also, it finds the minimum cost exploit sequence. However in the worst case it checks through all possible exploit sequences which can be factorial in input size. A depth first approach is suitable for this problem as the generated tree has a limited depth which is linear in the input size.

IV. Network Resilience Against a Random Attacker

Finding the exact cost of the minimum cost exploit sequence has been proved to be NP-Complete. There is not much scope of finding an efficient algorithm absent of any domain specific information. This section takes a totally different approach towards the problem. Instead of looking for the 'minimum cost' that would be required by the most skilled attacker, we try to find the 'expected cost' that would be incurred by a very naive but extremely resourceful attacker. Here we present a linear time algorithm that tries to calculate the 'expected cost' of exploiting the network for a random attacker when presented with network with an *uncorrelated* attack graph. It must be noted that the approach presented in this section is especially suitable for large networks, which may have extremely huge attack graphs.

The assumptions made for this section are as follows:

- The goal conditions are reachable when starting with the starting conditions.
- Given a possible set of exploits to execute from, the attacker chooses one at random.
- The attacker never re-executes exploits. However he doesn't know a priori all the post conditions of the exploit. Hence he does not discard exploits whose, post conditions are already achieved.
- The attack graph is *uncorrelated.* This means that for any exploit, every security condition, other than its pre-conditions, are equally likely to be its post-condition.

Let, $c = |C|$ be the total number of conditions and $e = |E|$ be the total number of exploits in the attack graph. Also, σ and ρ be the average number of conditions required and implied by exploits respectively in the attack graph.

Let, after executing exploit number i, the attacker has obtained c_i conditions. Exploit number $i+1$ will serve the attacker with ρ post-conditions. Since the attack graph is *uncorrelated*, the fraction of these that are new is given by $\left(\frac{c-c_i}{c-\sigma}\right)$.

Thus we have:

$$\begin{aligned} c_{i+1} &= c_i + \rho \cdot \frac{c-c_i}{c-\sigma} \\ &= c_i \cdot \big(1 - \frac{\rho}{c-\sigma}\big) + \frac{c\rho}{c-\sigma} \end{aligned} \qquad (4)$$

Assuming $c_0 = s = |S|$ is the number of starting conditions, we have

$$c_i = (s-c) \cdot \big(1 - \frac{\rho}{c-\sigma}\big)^i + c \qquad (5)$$

Let p_i be the probability that $g = |G|$ goal conditions are met after executing i exploits. Since, c_i is the number of conditions obtained after executing i exploits, p_i can be calculated as the probability of having g goal conditions in the set of c_i conditions currently present. Thus

$$p_i = \binom{c-g}{c_i-g} \cdot \binom{c}{c_i}^{-1} = \binom{c_i}{g} \cdot \binom{c}{g}^{-1} \qquad (6)$$

Hence, expected number of exploits required to reach the goal conditions can thus be calculated as

$$v = \sum_{i=0}^{e} i \cdot p_i \qquad (7)$$

Obviously, the above expression can be calculated in linear time in the number of exploits.

Since we assume that the attacker ignores the cost of executing the exploits, we can just present the expected number of exploits as a security metric for the network. However we can approximate the expected cost by the product of the expected number of exploits and the average cost per exploit $\bar{w}$.

$$\text{expected cost} = v \cdot \bar{w} \qquad (8)$$

The approximate, expected cost obtained may be used as the initial bound for the branch and bound algorithm described in the previous section. However it must be noted that this bound is neither a strict over or under estimate of the minimum cost.

V. Conclusion

This work extends the notion of exploit dependency graphs to define regular attack graphs using the concept of monotonicity. A theoretical framework for development of security metrics which uses cost of executing individual exploit as basis has been proposed. The problem of finding the minimum cost for exploiting a network has been proved to be NP-Complete. A polynomial space depth first branch and bound based algorithm for searching for an optimal exploit sequence using polynomial memory has been described. A linear time computable expected cost measure has been proposed assuming random attacker model and uncorrelated attack graph.

The algorithm proposed in this work for finding an optimal exploit sequence has a factorial time requirement in the worst case. The expected cost measure described has been developed using assumptions of random attacker model and uncorrelated attack graph. Real life attackers obviously don't follow the said model. In future, a suitable approximation algorithm needs to be developed with strict approximation guarantee. Further, development of an acceptable cost measure is a major challenge and an immediate future work.

References

[1] L. Wang, T. Islam, T. Long, A. Singhal, and S. Jajodia, "An Attack Graph-Based Probabilistic Security Metric," in *Data and Applications Security XXII, LNCS*, V. Atluri, Ed. Springer Berlin / Heidelberg, 2008, vol. 5094, pp. 283–296.

[2] M. Frigault, L. Wang, A. Singhal, and S. Jajodia, "Measuring Network Security Using Dynamic Bayesian Network," in *4th ACM workshop on Quality of Protection, QoP '08*, 2008, pp. 23–30.

[3] S. Noel, L. Wang, A. Singhal, and S. Jajodia, "Measuring Security Risk of Networks Using Attack Graphs," *International Journal of Next-Generation Computing*, vol. 1, no. 1, 2010.

[4] N. Ghosh and S. K. Ghosh, "A planner-based approach to generate and analyze minimal attack graph," *Applied Intelligence*, pp. 1–22, 2010.

[5] ——, "An Intelligent Approach for Security Management of an Enterprise Network Using Planner," in *Intelligent Autonomous Systems*, D. Pratihar and L. Jain, Eds., 2010, vol. 275, pp. 187–214.

[6] N. Ghosh, S. Nanda, and S. K. Ghosh, "An ACO Based Approach for Detection of an Optimal Attack Path in a Dynamic Environment," in *Distributed Computing and Networking, LNCS*, K. Kant, S. Pemmaraju, K. Sivalingam, and J. Wu, Eds., 2010, vol. 5935, pp. 509–520.

[7] N. Ghosh and S. K. Ghosh, "An Approach for Security Assessment of Network Configurations Using Attack Graph," *International Conference on Networks & Communications, NetCom' 09*, vol. 0, pp. 283–288, 2009.

[8] ——, "An Intelligent Technique for Generating Minimal Attack Graph," in *Workshop on Intelligent Security SecArt '09 in 19th International Conference on Automated Planning and Scheduling, ICAPS '09*, 2009, pp. 42–51.

[9] X. Liu, C. Fang, D. Xiao, and H. Xu, "A Goal-Oriented Approach for Modeling and Analyzing Attack Graph," in *International Conference on Information Science and Applications, ICISA '10*, 2010, pp. 1–8.

[10] P. Ammann, J. Pamula, R. Ritchey, and J. Street, "A Host-Based Approach to Network Attack Chaining Analysis," in *21st Annual Computer Security Applications Conference, ACSAC '05*, 2005, pp. 72–84.

[11] J. Pamula, S. Jajodia, P. Ammann, and V. Swarup, "A Weakest-Adversary Security Metric for Network Configuration Security Analysis," in *2nd ACM workshop on Quality of Protection, QoP '06*, 2006, pp. 31–38.

[12] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated Generation and Analysis of Attack Graphs," in *IEEE Symposium on Security and Privacy*, 2002, pp. 273–284.

[13] O. Sheyner and J. Wing, "Tools for Generating and Analyzing Attack Graphs," in *Formal Methods for Components and Objects, LNCS*, F. de Boer, M. Bonsangue, S. Graf, and W. de Roever, Eds. Springer Berlin / Heidelberg, 2004, vol. 3188, pp. 344–371.

[14] S. Jha, O. Sheyner, and J. Wing, "Two Formal Analyses of Attack Graphs," in *15th IEEE Computer Security Foundations Workshop*, 2002, pp. 49–63.

[15] L. Wang, A. Singhal, and S. Jajodia, "Measuring the Overall Security of Network Configurations Using Attack Graphs," in *Data and Applications Security XXI, LNCS*, S. Barker and G. Ahn, Eds. Springer Berlin / Heidelberg, 2007, vol. 4602, pp. 98–112.

[16] L. P. Swiler, C. Phillips, D. Ellis, and S. Chakerian, "Computer-Attack Graph Generation Tool," in *DARPA Information Survivability Conference Exposition II, DISCEX '01*, vol. 2, 2001, pp. 307–321.

[17] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, Graph-Based Network Vulnerability Analysis," in *9th ACM conference on Computer and Communications Security, CCS '02*, 2002, pp. 217–224.

[18] P. Mell, K. Scarfone, and S. Romanosky, "Common vulnerability scoring system," *IEEE, Security Privacy*, vol. 4, no. 6, pp. 85–89, 2006.

[19] ——, "A Complete Guide to the Common Vulnerability Scoring System Version 2.0," 2007. [Online]. Available: http://www.first.org/cvss/cvss-guide.html

[20] S. Noel, S. Jajodia, B. O'Berry, and M. Jacobs, "Efficient Minimum-Cost Network Hardening Via Exploit Dependency Graphs," in *19th Annual Computer Security Applications Conference*, 2003, pp. 86–95.

[21] L. Wang, S. Noel, and S. Jajodia, "Minimum-cost network hardening using attack graphs," *Computer Communications*, vol. 29, no. 18, pp. 3812–3824, 2006.

[22] S. Noel and S. Jajodia, "Managing Attack Graph Complexity Through Visual Hierarchical Aggregation," in *ACM workshop on Visualization and Data Mining for Computer Security, VizSEC/DMSEC '04*, 2004, pp. 109–118.

[23] R. Karp, "Reducibility Among Combinatorial Problems," in *Complexity of Computer Computations*, R. Miller and J. Thatcher, Eds. Plenum Press, 1972, pp. 85–103.