IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received March 26, 2021, accepted April 9, 2021. Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2021.3073089

On Parallel Real-Time Security Improvement Using Mixed-Integer Programming

HUSSAIN M. J. ALMOHRI[1], (Member, IEEE), LAYNE T. WATSON[2,3,4], (Life Fellow, IEEE), HOMA ALEMZADEH[5], (Member, IEEE), AND MOHAMMAD ALMUTAWA[1]

[1]Department of Computer Science, Kuwait University, Kuwait
[2]Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061, USA
[3]Department of Mathematics, Virginia Polytechnic Institute and State University, Blacksburg, VA 24060, USA
[4]Department of Aerospace and Ocean Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA 24060, USA
[5]Department of Electrical and Computer Engineering, University of Virginia, Charlottesville, VA 22904, USA

Corresponding author: Hussain M. J. Almohri (almohri@ieee.org)

ABSTRACT Network security defenses evolve, responding to real-time attack incidents, modifying the underlying topology, or reallocating defense systems across the network. The present work emphasizes reducing the time to compute new optimal reallocations of defense systems, responding to emerging real-time remote attacks. The proposed heuristic method utilizes parallel processing by slicing the underlying graphical model representing the network topology, solving in parallel multiple mixed-integer programming problems corresponding to the created subgraphs, and producing an estimate of the optimal defense. The parallelized method to compute a new defense enables producing a response, in real-time, before remote attackers compromise a target machine in the network. Our prototype tool to compute a new defense, the high-performance security analyzer, has a speedup of at least 20 over solving the original problem using a serial algorithm, and with an insignificant difference between the performance of the (computed in parallel) approximately optimal defense and the (serially computed) optimal defense. A major conclusion is that further speedups will come from parallel integer programming algorithms rather than from graph partitioning.

INDEX TERMS Security management, parallel processing, tree graphs, mathematical programming.

I. INTRODUCTION

System administrators can analyze a network's security by constructing a graph-theoretic model, referred to as the attack graph, showing access control rules across the network nodes. Using graphs to represent networks is a common practice and can benefit from a wealth of graph algorithms. A specific advantage is to model the flow of attacks in a network using a digraph (defined as an attack graph or an attack tree [1]–[4]). Particularly, as shown in some of the previous works (such as [5]–[7], and [8]), graph-theoretic models can be used to optimize network security goals against powerful attackers. Given prior attack data, attack success probabilities can be propagated through an attack graph, capturing the attacker's probability of success in compromising a specific target machine (represented as a sink node in the graph). Using the success probability as a measurable quantity, improving the defense can be realized as an integer optimization problem to minimize the attacker's success, considering several defense opportunities subject to access control rules. In a volatile network environment, remote attacks emerge supported by automated tools and prior planning, requiring only a few seconds to penetrate the network. Groups of coordinating or independent attackers may attack simultaneously or within short time intervals, stressing the defense to develop new plans for combating the incoming offenses. Methods such as patching vulnerable software [9] and moving target defenses [10] are orthogonal to graph-theoretic analysis of security in the network. However, we only focus on using optimization for improving the security of networks against external attacks.

This work emphasizes the need to reconsider the efficacy of existing optimization methods to react in real time to attacks, minimizing the progress of the attack through the network. The immediacy of response necessitates generating a new defense plan in real-time, utilizing intrusion prevention systems, modifying the defense parameters, and

The associate editor coordinating the review of this manuscript and approving it for publication was Chunsheng Zhu.

reorganizing the network topology. A swift response must therefore produce a new defense plan before an attack completes. We observe that the time required for solving optimization problems can preclude swift responses to attacks in real time, creating the need for improving the computation time.

The efficiency of graph-based optimization is mainly dependent on the approach to model the optimization problem and the graphical structure of the network. The main challenge is designing realistic optimization methods against external attacks while minimizing the computational hurdles. The graphical structure of the network would impose further requirements on building computationally efficient algorithms to aid in real-time network attack strategy assessment. Thus, we elaborate on the need to exploit parallelism for improving the efficiency of optimization methods for reacting to emerging external attacks.

There are several ways for implementing a parallel security optimization algorithm. Shared memory and distributed memory methods deal primarily with hardware than structural parallelism. An analysis of the problem's potential parallelism, and parallel computational results, show that existing optimization software only allows coarse grained parallelism, whereas better speedups could be attained if only fine grained parallel optimization algorithms were available. It is beyond the scope of this work to develop from scratch a fine-grained parallel mixed integer optimization code. Thus, the main contribution of this work is to present the methods for an improved computational efficiency by parallelizing the optimization effort across several subgraphs.

There is a large body of work concerning graph partitioning (for example [11], [12]) and parallel graph algorithms such as Pregel [13]. The literature on graph partitioning and parallel graph processing mainly focus on processing social networks or graphs used in machine learning algorithms. Kaynar and Sivrikaya proposed a distributed processing model to improve the efficiency of attack graph generation [14]. Similarly, Hong *et al.* proposed a scalable attack graph processing method [15]. The previous works share our concern about the scalability problem when processing intrusion using graph-theoretic models. However, the efficiency of processing graph optimization remains an open problem.

We address the problem by using a coarse-grained decomposition method of the original problem to benefit from parallelism, and a parallel heuristic that only computes a local optimum. We have designed and experimented with two partitioning methods to ascertain the benefits of our approach: a tree partitioning method and a method we call improvement target partitioning. We first present the tree partitioning method. This method receives an attack tree and decomposes it into several complementary subtrees. The resulting subtrees are used to compute probability propagation across the tree, reduced to compute a final attack success probability. A similar technique is used as a heuristic to decompose the security improvement problem into multiple subproblems, each testing the security improvement of one or several of the available security improvement instruments. The subproblems results are integrated in a final stage to find the best achieved objective among the competing subproblems. The second method, improvement target partitioning, is also presented, evaluated, and compared against the tree partitioning method. This method aims to reduce the number of nodes to which a specific improvement instrument is applied. Our experiments show the effectiveness of tree partitioning for parallel probability propagation, which achieves a performance gain of up to 20 times faster than a serial security improvement computation method. The parallel security improvement using tree partitioning was also tested for both cases of an exact and estimated solution, achieving an average improvement of 30 times faster computation time over the serial algorithm. Estimating the global optimum has negligible relative errors when evaluated on synthetic attack graphs mimicking real organizational networks.

In summary, this article presents an overview of probabilistic propagation and security improvement using attack graphs (Section III), the design of the architecture for a real-time security response system (Section IV), and the description of a parallelized security improvement using tree partitioning (Section V-B) and improvement target partitioning (Section V-C). The presented methods are evaluated (Section VI) using the developed tool on large synthetic attack graphs.

II. RELATED WORKS

This work addresses the efficiency problem with computing optimized security defenses based on the graph-theoretic representation of enterprise networks. We hypothesize that system administrators can utilize coarse-grained parallelism to produce efficient security improvement while using existing graph-theoretic and optimization algorithms to represent vulnerabilities. Specifically, we assume a probabilistic attack graph (Section III-A) is generated that represents the depth of the vulnerability of networks to remote intrusions. Attack graphs (developed through several years, notably in [16]–[18]) are merely used to represent the flow of an attack. The benefit of using attack graphs to describe an attack flow is to exploit the existing literature on digraphs (see [2]–[4], [19], [20]). Attack graphs can quantitatively model a probabilistic view of attacker capabilities (defined as Bayesian attack graphs [6]).

As proposed in previous works, such as [5], [8], a probabilistic attack graph showing the probability of success for a remote intruder can further be used in optimization problems to compute an improved network structure. In [8], the authors used MulVAL attack graphs (as discussed in Section III-A) and provided a novel computational model using mixed-integer programming. This model enables network security administrators to utilize state-of-the-art optimization models and systems to enhance network security analysis further. The present work explores the use of parallelization technology for enhancing the efficiency of security improvement using attack graphs. The work by Yin *et al.* discussed real-time

attack analysis using attack graphs in which multiple vulnerabilities were analyzed [21]. Real-time intrusion alert correlation was also addressed by Ramaki *et al.* [22].

In the remainder of this section, we briefly discuss the related work on security planning followed by a discussion of parallelized belief propagation and graph analysis.

A. SECURITY PLANNING

Security planning requires a complete analysis of the security of the network, an examination of the plausible security strategies, and the production of the optimal security model. Towards the goal of producing plans for *improving* the security of a network, a wealth of previous works have presented various approaches. For example, genetic algorithms were used in [23] and [5] for improving the security of networks given multiple potentially conflicting objectives. The problem formulated by Dewri *et al.* is to find a network configuration that minimizes the total security control cost and the residual damage by the produced policy. Minimum-cost SAT-solvers were used by Huang *et al.* to distill critical parts of attack graphs, which are used for preparing defenses [24]. Other approaches, such as the use of diffusion games [25], have also been proposed for hardening the security of networks. In this work, our focus is on producing real-time security plans using methods to parallelize attack graph analysis and security improvement planning.

B. PARALLEL BELIEF PROPAGATION IN BAYESIAN NETWORKS

Several previous works have investigated belief propagation in Bayesian networks with an emphasis on junction trees (details in [26]). Darwiche demonstrated a differential approach to evaluate probabilistic queries in Bayesian Networks by constructing junction trees [27]. Later Vasimuddin *et al.* developed an algorithm to parallelize Bayesian Network inferences using arithmetic circuits where operations such as sum and product [28]. Bayesian networks share similarities with attack graphs. Posterior probabilities are computed in Bayesian networks, allowing efficient computation of event probabilities. A probabilistic attack graph facilitates probability propagation using structural properties and logical connections that are constructed based on the underlying computer network. Further, junction trees allow for answering inference queries based on Bayesian networks. Although junction trees have useful applications, attack graphs have simpler requirements. Accordingly, equivalent attack trees are constructed in this work that prepare the foundation of data parallelization on parallel shared-memory machines.

C. PARALLELIZED GRAPH ANALYSIS

The only work concerning parallelizing attack graphs is by Kaynar and Sivrikaya, which introduced a model for distributed attack graph generation [14]. In that work, a hyper-graph is partitioned according to various reachability subgraphs. The proposal is to use the distributed memory manager algorithm by Li and Hudak [29] to distribute the task of *generating* an attack graph among independent distributed agents. The authors developed a parallel, shared-memory depth-first search to produce the attack graph with a complexity of $O(N^2/log(N))$. The experiments demonstrated a speedup of about 40%. In contrast, the focus of our work is on generating parallel security improvements.

Although parallelizing attack graph computations has not been the subject of previous works, numerous parallelization approaches for various graph models have been studied. For example, PowerGraph [30] and GraphChi [31] propose system-level approaches for massively scalable graph computations. GraphChi introduces the parallel sliding window model, which uses efficient disk access to parallelize graph computations on a consumer PC. GraphChi leverages standard graph partitioning methods to enhance the computation efficiency. The graph partitioning method introduced in Section V is inspired by GraphChi [31], classical tree partitioning [32], and MapReduce [33]. A work by Li *et al.* also described attack graph partitioning using forward search [11]. A partition and merge approach was previously proposed by Hong *et al.* [15] for infrastructure as a service clouds. First, the work describes the architecture of partition and merge to be used as part of Infrastructure as a Service (IaaS). Second, the work does not concern with the optimization of defense against a probabilistic model of the attack. Third, the experimental results describe an analysis of a search method, which further puts their work apart from others.

III. BACKGROUND

This section defines attack graphs, describes the success probability measurement in terms of a Bernoulli distribution, a probability propagation model, and an optimization model for improving the security of a network. These preliminary models are introduced in a previous work [8]. The contribution of this work is to improve the security evaluation of a network using parallel computation (Section V). The approach can be used for efficient online computation of security improvement strategies for real-time systems.

A. PROBABILISTIC ATTACK GRAPHS

A probabilistic attack graph is a directed graph with nodes that participate in weighted causal dependency relationships. The weights of these relationships are probabilities that indicate the expected chance that an attack step, represented by a node, is exploited by an attacker.

A *probabilistic logical attack graph* $G = (V, A)$ is an acyclic digraph where $V = N_f \cup N_g \cup N_r$ and N_f, N_g, N_r are disjoint sets of nodes containing fact nodes, goal nodes (logical disjunctions), and rule nodes (logical conjunctions), respectively. A is the set of arcs, and $\mathcal{G} \in N_g$ is the attacker's goal. In a logical attack graph, nodes are of three types and are defined as tuples. Each attack graph node u is a tuple $(d_u, E[X_u])$ where d_u is the description of a network configuration item (when $u \in N_f$), an attack rule (when $u \in N_r$), or an attack goal (when $u \in N_g$), and $E[X_u] \in [0, 1]$

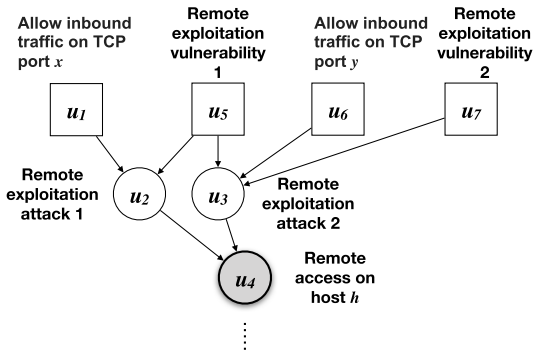


FIGURE 1. An example subgraph (of the attack graph in Figure 3) showing two possible ways to gain remote access on the host *h*. Boxes show fact nodes, plain circles are rule nodes, and gray circles are goal nodes. Nodes are labeled with the information they represent about the network.

is the corresponding expected chance that the node u is exploited by an attacker.

For example, suppose that an attacker wants to exploit a host h in a target network. The attacker should use a remote exploitation method (rule) to gain remote access on h. There could be more than one possible method, depending on the vulnerabilities and configurations of the target network. The attacker performs reconnaissance to develop an understanding of the vulnerable points in the network. This reconnaissance effort results in the picture of Figure 1. The attacker knows that there are two possible methods for exploiting the vulnerable host h, represented by the rule nodes u_2 and u_3. Using rule node u_2, the attacker should be capable of using the remote exploitation vulnerability of the fact node u_5. Also, the host should allow inbound TCP traffic on some port x. The alternative is to use rule node u_3, which requires inbound TCP traffic on port y, and the capability to exploit the remote exploitation vulnerability of the fact node u_7. The goal node u_4 indicates the possibility of gaining remote access on h using either of the two methods. After constructing the subgraph, the attacker executes the remote exploitation attack, provided that the preconditions of one of the two rule nodes are satisfied.

B. PROBABILISTIC ATTACK MODEL

The probabilistic component of an attack graph is represented using a Bernoulli random variable X_u defined on a corresponding sample space $\Omega(u)$ for a node $u \in V$. The outcome $\omega \in \Omega(u)$ of an attack exploiting a node u can either be a success ($X_u(\omega) = 1$) or a failure ($X_u(\omega) = 0$). For any node $u \in V$ of an attack graph, *the expected chance of a successful exploitation*, simply the expected success, at a node u is given as $E[X_u] = P(X_u = 1)$, that is, the probability of success for the random variable X_u.

The security of the network can be improved by analyzing a set of security improvement strategies that could be deployed in the network. As the probabilistic attack graph captures the internal structure of security vulnerabilities, security improvement options can be directly applied to attack graphs. The success probability of an attack is measured according to specific network topology, anticipated vulnerabilities, and a set of initial belief values about the feasibility of attacks. An *initial belief value*, associated with a fact node, is the estimated probability that an attack step precondition is satisfied by an arbitrary attacker. In the subdigraph of Figure 1, The initial belief value $E[u_1] = 1$ encodes the belief obtained from historical data that h is expected to be responding on port x continuously with no downtime, and $E[u_5] = 0.8$ indicates that the remote exploitation vulnerability is expected to be exploited with probability 0.8. The initial belief vulnerability values could be determined from vulnerability databases [34], expert knowledge, and using vulnerability assessment metrics such as total vulnerability exposure [35], the Common Vulnerability Scoring System (CVSS) [36] and the Common Weakness Scoring System (CWSS^{TM}) [37].

1) PROBABILITY PROPAGATION MODEL

An attack graph shows the intermediate attack steps required to exploit a target node in the network. Thus, a typical attack graph has a single leaf node $\mathcal{G}$ (digraph sink) that captures the ultimate attack goal. To measure the success of the attacker in exploiting the target, $E[X_{\mathcal{G}}]$ is computed. This computation requires propagating an initial set of belief values throughout the attack graph. The initial set of belief values are the expected chances of exploiting the fact nodes in the graph. That is, for each fact node $u \in N_f$, system administrators compute an initial belief value $E[X_u]$. These values represent historical attack success data (note that this work does not concern the computation of the initial belief values). An initial belief value is propagated from a node u to a node v, when $(u, v) \in A$. The propagation occurs using the model below. The propagation model is defined based on the type of the receiving node v.

Let $\phi(u)$ denote the set of predecessor nodes to node u. The propagation model combines the initial belief values to form the expected success at a rule node u:

$$E[X_u] = \prod_{v \in \phi(u)} E[X_v], \tag{1}$$

and multiple rule node values can be combined to form the expected success at a goal node u:

$$E[X_u] = \sum_{v \in \phi(u)} w_{vu} E[X_v], \tag{2}$$

where $w_{vu} \in [0, 1]$ is the weight of the predecessor v for propagating $E[X_v]$ to u. This weight represents the strength of the attack path that passes through v. Note that w_{vu} can represent a binary integer variable to select a maximally successful attack path in the graph. System administrators do not necessarily provide the weights. One advantage of the presented model is that the weights can be estimated using the optimization problem that is described later in this section. For the fact nodes, the expected success is simply

$$E[X_u] = y_u, \tag{3}$$

where y_u is an estimated initial belief value for the node u. Given the equations above to propagate the expected chance of a successful attack, the overall probability that a network target is compromised is given by $E[X_G]$.

To compute the optimal propagation, an optimization problem is defined. Here, the goal is to maximize $E[X_G]$ subject to the propagation equations above as constraints. The expected success $E[X_u]$ for each node u corresponds to a real-valued decision variable in the optimization problem. The decision variables are initially set to zero, except those for the fact nodes where the initial belief values are used. Let $x_i = E[X_i]$ denote the decision variable for node i in the graph whose nodes are indexed by 1, 2, . . ., $|V|$, with $x_{|V|} = E[X_G]$ being the final goal node. Let

$$\left(y_1, y_2, \ldots, y_{|V|}\right)$$

be the vector of initial values for the decision variables x_i. The optimization problem is formulated as

$$\begin{aligned} &\max\ x_{|V|} \\ &\text{subject to} \\ &\qquad x_i = g_i\left(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{|V|-1}\right), \\ &\qquad 1 \le i \le |V|. \end{aligned}$$

The function g_i is defined by Equations (1), (2), and (3) for the variable x_i depending on the type of the corresponding node in the graph. The weights in Equation (2) can also be modeled as decision variables when supporting data for preferred attack paths does not exist. For each predecessor v of a goal node u an additional decision variable $w_{vu} \in [0, 1]$ is added to the optimization problem. An additional constraint is added for each goal node u:

$$\sum_{v \in \phi(u)} w_{vu} = 1.$$

This constraint ensures that the weights act as selectors for the best possible path for the attacker among the available predecessors of the goal node u. The only inputs to the optimization problem are the structure of the graph and the initial belief values that are estimated based on metrics computed from vulnerability databases (such as CVSS [36]).

The optimization problem is a mixed-integer programming problem that computes the probability of success for the attacker at the graph's sink (the ultimate attack goal). In addition, the computed values of the variables, representing graph nodes, demonstrate the maximum vulnerability paths in the graph. This is because Equation 2 guides the maximization problem to select, among the various available attack choices, the one with the highest incoming probability value. The computed expected success values are essential to the improvement planning problem. Intuitively, the expected success values must be minimized to reduce the attack risk on the target network. Security improvement in the context of real-time security planning is explored next.

IV. REAL-TIME SECURITY IMPROVEMENT

In this work, we consider a setting in which the network is under time constraints, and thus a *security response* subject to a definite deadline is required. Real-time security systems are similar to general real-time systems [38] in which tasks are scheduled and processed. In real-time security systems, attacks impose tasks on the defense system for which counterattack measures forming a security response must be scheduled. Security response is a set of actions taken by a trustworthy subsystem that manages the security of the overall network. This trustworthy subsystem, referred to as the controller, is responsible for reacting to incidents that are security-critical. For example, new nodes joining the system require a reconsideration of the security of the overall system. In an environment that nodes are frequently added and removed from the system, the controller must provide *real-time security* by first analyzing the security status of the system and then generating the appropriate response.

The focus here is to investigate the possibility of reducing the computation time for expected success probabilities in an attack graph to aid the controller in generating the appropriate response given tight time deadlines. The deadlines are determined by the duration of sudden and unexpected attack events. Attack events are assumed to endure for a specific time interval. For example, a data leak attack event starts by unauthorized transferring of a large amount of data and ends when the transfer is complete. During the required time to complete a successful attack, the defense must be capable of detecting the intrusion, analyzing the event, and producing a response. Similarly, a denial of service attack on a subset of nodes in the network downgrades the quality of service and should be detected and mitigated immediately to ensure service availability.

The architecture of a real-time security response system based on attack graphs is given in Figure 2. The Attack Graph Repository is a component of the architecture. As the target network is modified by configuration events (new users, new network nodes, modified firewall rules) and attacks, network configuration changes are reported to the Real-time Security Improvement Planner (simply called the planner). The planner ideally produces a real-time response (a response that is generated and executed before a deadline), which results in a modified attack graph. The modified attack graph is cached in the Attack Graph Repository and is communicated to the Security Response Engine. Finally, the security response engine applies the modifications to the target network and acknowledges the changes with the Attack Graph Repository.

In the subsequent sections, the core functionality of the Real-time Security Improvement Planner is presented. A security improvement model based on an attack graph is developed to realize various security improvement scenarios. Later in Section V-B the parallelized versions of the probability propagation model and the security improvement model are presented and discussed. Note that this work does not concern the underlying mechanisms for generating attack

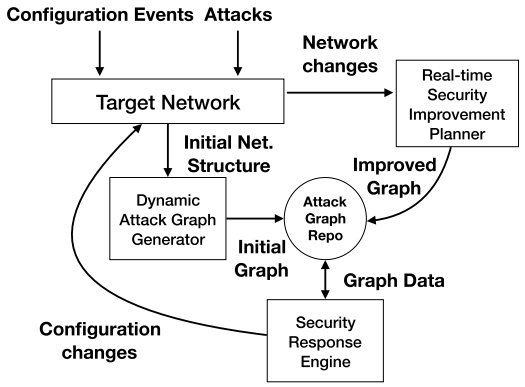


FIGURE 2. The architecture of real-time security response system using attack graphs.

graphs (by the Dynamic Attack Graph Generator) or the techniques for applying security policies (executed by the Security Response Engine).

A. MODELING SECURITY IMPROVEMENT

Security improvement refers to the problem of developing a security plan, given a set of alternative strategies, such that the expected success of a prospective attacker is minimized. In this context, security improvement instruments (or simply called instruments) are hardware or software resources (such as network intrusion prevention systems) that could be purchased and deployed in quantities, depending on the organizational and technical constraints. Thus, a specific instrument can be repeated within a network, or multiple instrument types could be considered.

There are two fundamentally different approaches for security improvements: (1) the sensor distribution strategy is to place multiple sensors across the network, and (2) the network restructuring strategy is to modify the network topology to achieve improved security. The challenge with the first approach is to find a non-conflicting subset of improvement instruments that minimize $E[X_G]$. In the second approach, the challenge is to find a variation of the attack graph, subject to constraints on the structural changes that can lower the value of $E[X_G]$. In this work, the first approach is considered. That is, given a set of possible security instruments (such as intrusion detection and prevention systems and network traffic monitors), the problem is to find the best placement of these instruments, given the security structure captured in an attack graph. The main contribution is to compute an improved network security plan by using a shared-memory parallel architecture. This parallelism helps the overall objective of aiding real-time security decisions in large network environments.

This section develops the optimization problems for computing a security improved network, given a set of security improvement instruments and strategies. Complex networks often face several network security improvement instruments and strategies. Both modeling single (e.g., [8]) and multiple security improvement (e.g., [5]) options have been studied in previous works. However, in this work, existing models are extended to leverage mixed-integer programming for developing network security improvement strategies. A single security improvement refers to computing the placement of a single security improvement in the network such that the expected attack success is minimized. Multiple security improvements refers to computing the placement of multiple security instruments across the network, minimizing the expected attack success.

Following the model in [8], improvement instruments are formalized as additional synthetic nodes. Here, we further expand the model and define an *improvement node* as a fact node that has an arc to exactly one rule node. An *improvement target* is the unique rule node successor of an improvement node. In a computer network, the improvement target represents the exploitation of a service on a machine in the network. A *placement* of an improvement instrument is the installation of the instrument in a particular location in the network. Within an attack graph, a placement is an arc from an improvement node to an improvement target. Thus each improvement instrument (such as an intrusion prevention system) is represented by several improvement nodes. When an improvement instrument can be potentially placed in several machines in the network, the possibility of placing the instrument for each location is referred to as a *placement option* (or *candidate*). All the placement options for an improvement instrument are represented as a set of arcs, as defined in Section IV-B.

The rationale for this model is to examine the effect of the improvement instrument on the expected success chance at a particular rule node. For example, in the attack graph of Figure 3, one can consider *attaching* an improvement node to the improvement target rule node u_2 along with the existing predecessors u_1 and u_5. The expected success chance of the improvement node represents the success of the improvement instrument in reducing the security risk, as opposed to the measured attack success at other nodes in the graph.

B. SECURITY IMPROVEMENT CONSTRAINTS

Security improvement is computed by solving a mixed-integer programming problem using a local gradient-based algorithm [39]. The improvement nodes are added with the constraints that either the improvement node is included or excluded. The models below can be used for single improvement (mutual exclusion of improvement placement options), multiple placement improvement, and multiple conflicting placements of improvement nodes.

For an improvement instrument with multiple placement candidates, a fact node is added for each placement candidate as a predecessor of a rule node. Let $N_h \subset N_f \subset V$ be the set of improvement (fact) nodes corresponding to a single improvement instrument. Let $A_h \subset A$ be the set of arcs connecting improvement nodes to improvement targets. Each $u \in N_h$ has a rule node successor $x \in N_r$. If $u, v \in N_h$, $(u, x) \in A_h$ and $(v, x) \in A_h$, then $u = v$. Each arc in A_h designates a possible *placement* of the improvement instrument. The nodes in N_h

are identical in nature but differ in their placements across the attack graph.

The expected success probability for a rule node with an improvement instrument is computed as

$$E[X_u] = \big((1 - t_w) + t_w(1 - E[X_w])\big) \prod_{v \in \phi(u) \setminus \{w\}} E[X_v], \quad (4)$$

where t_w is a binary variable, with $t_w = 1$ indicating that the improvement node w is placed for the improvement target u. Note that $E[X_w]$ is the probability of success of the corresponding instrument, so $(1 - E[X_w])$ is the corresponding probability of attack success. An extra constraint is added to the optimization problem in Section III-B1 for limiting the number of possible placements:

$$t_1 + t_2 + \cdots + t_{|N_h|} \leq m,$$

where m is the maximum number of placements for the considered improvement instrument.

For multiple nonconflicting improvements, consider k types of possible improvement instruments leading to the set

$$\mathcal{N}_h = N_h^1 \cup N_h^2 \cup \cdots \cup N_h^k$$

of improvement nodes, where the N_h^i are disjoint. Further, define the corresponding set

$$\mathcal{A}_h = A_h^1 \cup A_h^2 \cup \cdots \cup A_h^k$$

of all possible placements (arcs) for all improvement instruments.

Modeling the security improvement follows Equation (4) as the improvement nodes are added according to the improvement node sets in $\mathcal{N}_h$, with the single improvement factor becoming a product of such factors. The maximum number of placement options also follows the constraint above, with $|N_h|$ replaced by $|\mathcal{N}_h|$. Note that if the number of possible placements from each improvement set is not bounded, the resulting combinatorial problem is computationally intractable. Finally, the mutual exclusion between two improvement instruments can be defined as a complementary constraint on each possible placement for each of the improvement instruments. Conflicting options may require such constraints. For example, the installation of a network monitor conflicts with limiting connecting IP addresses on the same host. Thus, system administrators must choose the most promising option accordingly.

Suppose that N_h^i and N_h^j represent conflicting options, with an identical rule node successor for node $u \in N_h^i$ and node $v \in N_h^j$. Then $t_u + t_v \leq 1$ ensures mutual exclusion of the two options. Section V-B provides the model for parallel computation of the solution to the optimization problem for all three cases of the security improvement problem.

V. PARALLEL SECURITY IMPROVEMENT

This section describes a tree partitioning method for (1) parallelizing the propagation of expected success values throughout the attack digraph (Section V-A), and (2) parallelizing security improvement computations (Section V-B).

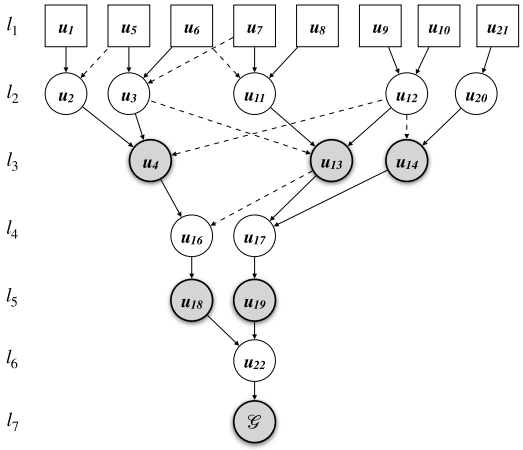


FIGURE 3. An attack graph *G* showing attack paths to $\mathcal{G}$. Dashed arcs are eliminated to prepare a tree $T \prec G$ for partitioning. Boxes are fact nodes, plain circles are rule nodes, and gray circles are goal nodes. The tree *T* can be generated using attack graph generators by requiring that fact and rule nodes only have forward edges to a single successor, avoiding the need to transform *G* to *T*. The extra edges can be connected by extra auxiliary nodes with the exact specification of the original nodes.

A. PARALLELIZED PROPAGATION

Enhancing the efficiency of computing expected success values is by using a gradient-based method to solve the optimization problem formulated in Section III-B1. In this section, the focus is not on parallelizing the gradient-based algorithm for approximating the solution for the nonlinear optimization problem. Instead, a divide-and-conquer approach (inspired by the MapReduce [33] programming model) is used to efficiently compute the expected success for all nodes in an attack graph by solving multiple nonlinear optimization problems.

The computation can be parallelized by first eliminating semicycles in the attack digraph G, resulting in an in-tree attack digraph $T = (V_T, A_T)$ logically equivalent to G. The notation $T \prec G$ is used to refer to an in-tree thusly created from G (Figure 3). The tree structure (the attack in-tree) is then partitioned into k in-trees, whose sinks are the 0-indegree nodes for a *final* in-tree, and this final in-tree. Classical tree partitioning methods (such as [32]) can be extended to a balanced partitioning scheme where each partition is constrained to a maximum number of nodes. k-BALANCED partitioning is useful but is computationally infeasible [40]. In the example attack digraph of Figure 3, eliminating the dashed arcs and adding other nodes and arcs results in an attack in-tree. The eliminated arcs are in the attack digraph because attack digraph generators can reduce the number of nodes in the digraph by generating extra arcs. For example, the arc (u_6, u_{11}) can be eliminated by introducing a new node u_{23} that is identical to u_6 with an arc (u_{23}, u_{11}).

The attack in-tree (logically equivalent to an attack digraph) is partitioned based on the characteristics of an in-tree and with a simple heuristic. Consider an attack in-tree T with $L = (l_1, l_2, \ldots, l_n)$ levels, where the first level only has fact nodes, and the last level only has the ultimate goal node $\mathcal{G}$. Recall that a level is a subset of nodes with an equal

distance from the leaf of the in-tree. A level l in an attack in-tree has an arbitrary number of nodes (as opposed to the case with k-ary trees) as the arcs in the in-tree correspond to the underlying computer network structure with no specific topological constraints. Suppose that each level in the in-tree has nodes of a single type. Define the distance of a level l to the leaf node $\mathcal{G}$ as the number of levels beyond l in T. The in-tree T is partitioned by finding the level l^* closest to node $\mathcal{G}$, such that l^* contains only goal nodes, is not the last level, and has more than a single node. The in-tree in Figure 3 is partitioned on l_5, which is the closest to l_7 (the last level). In this case, two subin-trees are created with the goal nodes u_{18} and u_{19} as their leaves.

The expected success values are propagated independently in each subin-tree created as part of attack in-tree partitioning. The computed expected success values for the leaves of the subin-trees are then passed over to a final in-tree F. The final in-tree includes all nodes of the original in-tree T, excluding the nodes from the subtrees $T_1, \ldots, T_k$ resulting from partitioning the in-tree T. k fact nodes are added to the in-tree, each representing the goal leaf node in a subin-tree T_i. The arcs in F include all arcs of T excluding arcs connecting the nodes in the subin-trees. Finally, each new fact node in F is connected to a rule node using the arcs that connected the corresponding goal nodes in the original in-tree T. For example, in Figure 3, the final in-tree consists of $\mathcal{G}$, u_{22}, and two fact nodes u_{18}, and u_{19}. The two fact nodes are the predecessors of u_{22}. Lastly, the propagation is computed on the final in-tree with the new fact nodes assigned their computed expected success values as the initial values.

Algorithm 1 summarizes the steps for partitioning an attack in-tree. Assume that `pred(l, G)` is a function that returns the immediate predecessors of the nodes in l from the graph G. An immediate predecessor of a node u is one that has a direct arc to u. Assume that `len(x)` is a function to compute the number of elements in a list x, `type(x)` is a function to return the type of the nodes in a given tree level x, `traverse(u)` is a function that performs a subtree (ending with u) traversal and returns the traversed subtree, and `append(x, y)` is a function to append an element x to the end of a list y. The graph G given to Algorithm 1 is assumed to have been converted to an in-tree, as described above, before input.

Once the attack tree is partitioned, the initial expected success values in the first layer of the tree must be propagated throughout the tree such that the propagation algorithm determines an expected success probability for every goal node in the graph. This propagation is performed by parallel calls to the optimization solver given independent subin-trees produced by Algorithm 1. Algorithm 2 summarizes the steps to propagate the expected success in the attack tree G using the list of attack in-trees T^* from Algorithm 1 and the final attack in-tree F. Here, assume that `success(T)` is a function that receives an attack tree (in-tree) T and computes the expected success using the optimization problem in Section III-B1; `leaf(T)` returns the leaf (goal) node of a given attack in-tree T; and `initialize(u, A)` sets the initial expected success value for u by matching it with a pair of node and expected success value in the list A.

Algorithm 1 Partition Tree

Require: G
  $l \leftarrow (\mathcal{G})$
  $T^* \leftarrow ()$
  loop
    $l \leftarrow$ `pred(l, G)`
    if `len(l)` ≥ 2 and `type(l)` is `goal` **then**
      for $u \in l$ **do**
        $T \leftarrow$ `traverse(u)`
        `append(T, T*)`
      end for
      break
    end if
  end loop
  return T^*

Algorithm 2 Propagate the Expected Success Values

Require: T^*, F
  $A \leftarrow ()$
  for each in-tree T from list T^* **do**
    $E \leftarrow$`success(T)`
    $u \leftarrow$`leaf(T)`
    `append((u, E[X_u]), A)`
  end for
  for each node u in tree F **do**
    if `type(u)` is `fact` **then**
      `initialize(u, A)`
    end if
  end for
  $E \leftarrow$`success(F)`
  return E

Algorithm 2 substantially reduces the size of the optimization problem and tackles two fundamental problems. First, the number of variables in each subproblem is reduced relative to the number of subin-trees created by Algorithm 1. Second, the combinatorial choices for finding the most vulnerable path are reduced, which substantially impacts the execution time of Algorithm 2 (Section VI-A). This algorithm provides the basis for enhancing the efficiency of the security improvement problem, as described next.

B. TREE PARTITIONING FOR SECURITY IMPROVEMENT

The goal is to parallelize the computation of security improvements using an attack digraph and a set of possible security improvement instruments. Recall that security improvement is computed by placing improvement nodes across an attack digraph, aiming to minimize the expected success values (Section IV-B). The optimization problem to compute and *select* the best security instruments and the best placement of the improvements requires a significant number

of additional decision variables and constraints. The plan here is to logically decompose the optimization problem and solve multiple optimization problems in parallel. The approach is to extend the model used in Section V-A and apply it to various cases of security improvement.

As described earlier, two major security improvement cases are considered: (1) selecting a *single* placement of an improvement instrument type, and (2) selecting *multiple* placements of improvement instruments. Let m be the number of acceptable placements and k be the number of types of available improvement instruments. Depending on the values of m and k, the resulting optimization problem (for security improvement) may be decomposed and solved in parallel by finding either an *optimal* solution or an *estimated* solution. The latter case is addressed using a heuristic. Recall that security improvement could be made with multiple instrument types, each represented by a set of improvement nodes N_h^i and a set of placements (arcs) A_h^i.

For selecting a single improvement instrument placement ($m = 1$ and $k \geq 1$), a parallelized security improvement computation is possible using the tree partitioning method outlined below. For selecting multiple improvement placements ($m > 1$), two cases are considered. In the first case when $1 < m < k$ (that is, selecting m placements with a larger number k of available instrument types), $z = \binom{k}{m}$ independent subproblems must be solved to find the optimal selection of security improvements. When $m \geq k$, the problem cannot be separated and thus no parallelization is possible. This is because all combinations of available instruments and their candidate placements must be considered to find the optimal solution. In this case, a parallelized heuristic estimates the solution, which does not guarantee the selection of optimal placements. The performance of the heuristic is evaluated in Section VI.

The tree partitioning method is used to parallelize two instances of the security improvement problem ($m = 1$ and $m \geq k \geq 1$). In this method, the attack digraph is partitioned into smaller in-trees (as in Section V-A). Using the in-tree partitioning method:

1) the original attack digraph is partitioned by executing Algorithm 1.
2) Then for each of the k improvement instrument types, $1 \leq i \leq k$, and for each $v \in N_h^i$ whose improvement target $x \in V_T$, v is added to V_T in the subin-tree $T = (V_T, A_T)$, and the placement option arc $(v, x) \in A_h^i$ is added to A_T.
3) The optimization problems (described below for the two cases) for all the subin-trees T are solved in parallel.
4) For each T, an intermediate objective value x_0^T is computed. The best m placements are selected as the final solution.

1) SINGLE INSTRUMENT PLACEMENT ($m = 1$)

In the simplest case where placing a single improvement instrument is considered, the improvement, such as placing a single sensor across the network, is represented as a set of improvement nodes. Given that there are several placement options for the improvement captured by the optimization problem in Section IV-B, the solution to the problem can be parallelized by either the tree or target partitioning (Section V-C) methods, both returning the global optimum. Each problem is solved using a nonlinear integer programming algorithm on a shared memory parallel machine. For improved efficiency utilizing multiple cores on a shared memory machine, several problems may be combined on a single core.

The tree partitioning method and the serial computation of security improvement with $m = 1$ produce identical objective function values. This is because an attack digraph, after conversion to an equivalent in-tree, has a unique sink (the goal node). Partitioning this in-tree into subin-trees, observe that the optimal path to the goal node for an attacker must lie wholly within one subin-tree. Therefore solving the placement problem in each subin-tree with $m = 1$ must find the optimal placement for the entire attack digraph after assembling and comparing the results for all the subin-trees.

2) MULTIPLE IMPROVEMENT PLACEMENTS ($m > 1$)

In applications with a small number of instrument types $k > m$, solving $z = \binom{k}{m}$ mixed-integer optimization problems in parallel on a shared memory machine can be feasible. However, with a large z, the parallel performance (defined as speedup on p processors divided by p) would be poor. A total of z independent problems (one for each size m subset of instrument types $\{1, \ldots, k\}$) should be solved in parallel (no subin-trees or target partitions are involved) to return the global solution. In this work, we do not address this case as an efficient solution requires parallelizing the underlying branch-and-reduce algorithm. This fine-grained parallelism is not a new problem and is beyond the scope of this work. Further, $k > m$ means that there are more options available than the required maximum number of placements. This problem is less likely to appear in large networks where the number of possible placements exceeds far beyond the number of instrument types k. Further, in cases where the instrument types are only software solutions, then practically $k \leq m$ as the possible copies of the solution is usually available in unlimited quantities.

For $m \geq k$, both the tree partitioning and the instrument target partitioning methods could be used. Both methods are used as heuristics to estimate the global optimum. In the tree partitioning method, p subin-trees are produced for solving p optimization problems in parallel. The goal is to find the instrument placements that minimize the expected success of the attacker. The proposed method is that for each subin-tree, an independent optimization problem is created with $\lceil m/p \rceil$ or $\lfloor m/p \rfloor$ (which one is randomly assigned) possible placements of any instrument types, summing to m. The optimization problems (assumed feasible) for all subin-trees are solved in parallel. The solution produced by this heuristic is not guaranteed to be optimal, because the globally

optimal solution might have more than $\lceil m/p \rceil$ placements in one subin-tree, and the interactions between placements in different subin-trees are being ignored. The same is true of the target partitioning.

C. INSTRUMENT TARGET PARTITIONING FOR SECURITY IMPROVEMENT

An alternative to the tree partitioning method is the *instrument target set partitioning method* that is used to parallelize the computation of security improvement for a single instrument placement ($m = 1$). This method works when $k \geq 1$ and can substantially reduce the computation time as k grows (Section VI-C). Suppose that for an attack in-tree T there is a single security improvement instrument placement ($m = 1$) from one instrument type ($k = 1$) with $\tau = |N_h^1| > 1$ improvement targets, and ℓ threads (cores) to use. Here, the approach is to partition the τ improvement targets into subsets of sizes $\lceil \tau/\ell \rceil$ or $\lfloor \tau/\ell \rfloor$, formulate an optimization problem for each subset and then solve these problems in parallel on ℓ cores. In this case, each optimization problem includes the entire attack in-tree, and the optimization problems differ in their possible improvement targets.

This method can be generalized to be used with multiple improvement instrument types $k > 1$. When multiple improvement instrument types are available ($k > 1$), the improvement targets are assigned to target sets by the instrument type, creating k target subsets (not a partition, since the same node can be a target of two different improvement type nodes). For each target subset, an optimization problem is formulated with the original entire attack in-tree T. Each optimization problem considers one of the k instrument types with that instrument type's targets. All optimization problems are then solved in parallel. Note that using the instrument target partitioning with $k > 2$, one can reduce the number of target subsets from k to $\lfloor k/n \rfloor$ for some $n < k$. This improves the efficiency by allowing one thread to have more work, when the number of threads $\ell < k$ is small. For $m = 1$ and $k \geq 1$ instrument target partitioning finds the same global optimum as the serial algorithm.

VI. EXPERIMENTS

The focus of the experiments is on evaluating the efficiency of the parallelization approaches discussed for the various instances of the security improvement problem using attack digraphs. The first experiment tests the parallelized propagation method discussed in Section V-A. The second and third experiments evaluate the instrument target partitioning (Section VI-C) and the tree partitioning (Section VI-B) methods. The case where more instrument types are available than acceptable placements ($k > m > 1$) is not evaluated as it is not expected to occur in reality.

Our experiments were conducted on a system with two 12-core Intel Xeon (E5-2690 V3 with a speed of 2.60GHz) CPUs running Ubuntu 18.04. The 24 cores were hyperthreaded for a nominal 48 threads, but there were only effectively 24 threads available for computation. During the experiments, no GUI utility was actively used, and the number of running processes was minimized.

The digraphs used in the experiments below follow the structure of Figure 4. The optimization problems are solved using BARON [41], which uses a branch-and-reduce approach to global mixed-integer programming problems [39]. The software tool developed for the experiments generates synthetic attack trees that are used to test the expected success propagation and the security improvement methods. The software tool is referred to as the High-Performance Security Analyzer (HPSA). The source code for HPSA is available at https://github.com/kussl/HPSA. When a synthetic attack tree is generated, initial belief values for the fact nodes are assigned using a uniform distribution. HPSA either propagates probabilities sequentially or using the parallelization described in Section V-A. The sequential method uses the partially quantified attack tree (only containing initial belief values for the fact nodes) to generate the corresponding optimization problem for computing probability propagations. BARON is fed the resulting problem using a file interface. BARON is then loaded into a separate process and computes the solution for the problem. For the parallel method, the tree partitioning component in HPSA produces the independent subin-trees. Each subin-tree is given to the sequential probability propagation for computation.

Note that for each subin-tree, a separate BARON process is created since, as of this writing, BARON does not provide loadable shared libraries. The scheduling of BARON processes depends on the scheduling algorithm used by the underlying operating system. Also, the abundance of memory can assist in improving the efficiency of BARON in solving some of the problems. The use of OpenMP [42] was also explored in the experiments. The scheduling of subin-trees was set to be managed by OpenMP [42]'s guided parallel for-loop. OpenMP parameters were set to defaults. OpenMP is expected to create p threads. In a guided schedule, OpenMP would separate the loop into approximately $\lceil \ell/p \rceil$ chunks, where ℓ is the number of optimization problems to be solved. However, no benefit of using OpenMP was observed as the BARON processes were managed efficiently by the operating system.

A. PROBABILITY PROPAGATION

In this experiment (Figure 5), Algorithms 1 and 2 were tested against the serial computation of probability propagation. The serial version takes as input the full attack digraph and a set of initial belief values and populates all other digraph nodes with expected success probabilities based on Equations (1)–(3). The main assumption of parallel probability propagation is the possibility of reducing an attack digraph to an attack in-tree by eliminating the unnecessary semicycles in the digraph (Section V-A). This is a reasonable assumption as all unnecessary semicycles can be eliminated by introducing extra nodes in the digraph. The parallel version produced by Algorithms 1 and 2 splits the attack in-tree into multiple smaller subin-trees and computes the expected success values

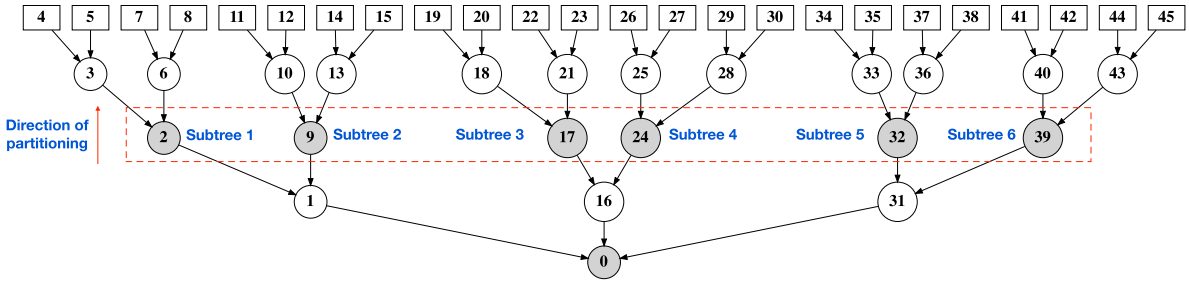


FIGURE 4. A digraph with 46 nodes that includes six subtrees, representing six goals (compromising six different goals in the network) that could aid the attacker to achieve the ultimate goal in node 0. In the experiments, this digraph is scaled horizontally (adding more subgoals as preconditions of the sink) and vertically (adding more preconditions to the subgoals).

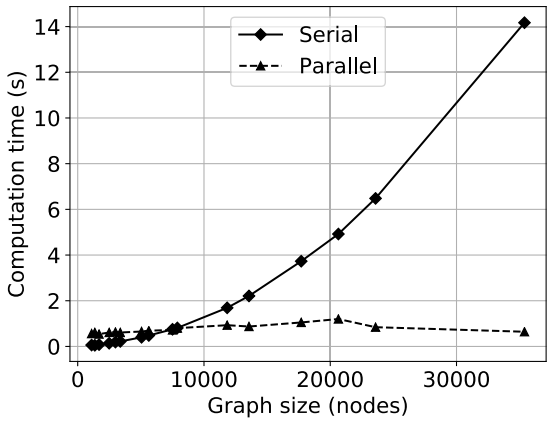


FIGURE 5. Execution time for probability propagation with serial and parallel algorithms. The parallelized computation was performed according to Algorithms 1 and 2.

for each node in each subin-tree in parallel. Figure 5 shows a comparison between the serial and parallel computation times of probabilities in an attack in-tree. Let $S = \frac{D_s}{D_p}$ be the speedup ratio, where D_p is the parallel computation time, and D_s is the serial computation time.

In this experiment, the maximum speedup ratio is $S = 22$ for 24 cores. Note that the total time D_p spent for the parallel computation includes the extra effort for partitioning the digraph and combining and recomputing the final results. Thus, the extra effort results in poor parallel performance for digraphs smaller than about 7000 nodes. Another component of the total time is preprocessing of the attack digraph to produce an equivalent attack in-tree, which may take a noticeable time. However, this could be done once when the attack digraph is generated, and consequent changes to the attack digraph and equivalent attack in-tree could be done by maintaining the invariant that any new node should not create a semicycle in the digraph.

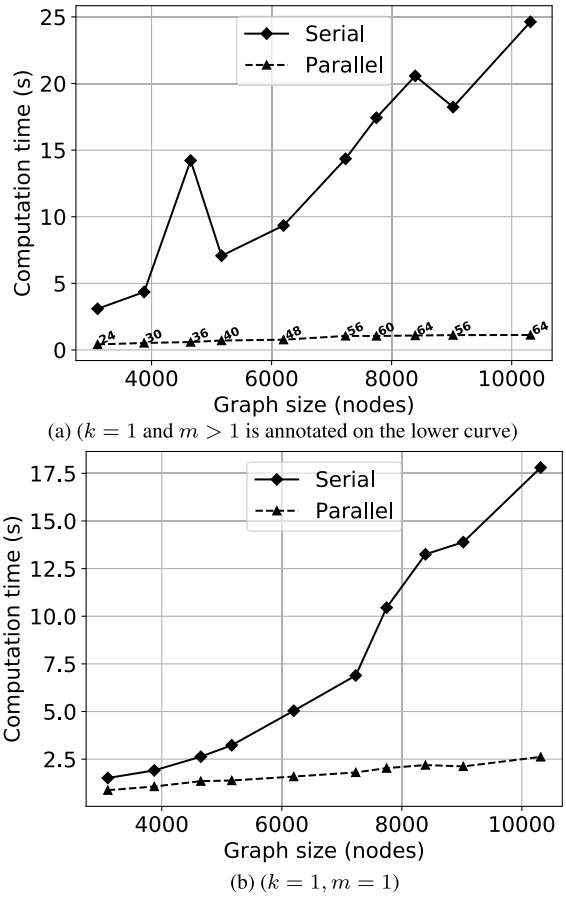
B. ATTACK TREE PARTITIONING

Here, the focus of the experiments is on the heuristic method provided in Section V-B for $m > 1$, $m \geq k$. To conduct this experiment, the original attack in-tree T is partitioned using Algorithm 1, producing several subin-trees $T_1, T_2, \ldots, T_\ell$, where ℓ is the number of goal nodes in the layer with the least distance to the ultimate goal node in T. For each improvement instrument of the available k instrument types, the tree with the improvement targets is identified. Then the improvement nodes are added to the identified trees, subject to $\lfloor m/\ell \rfloor$ acceptable placements. Each tree T_i is sent to a BARON process for optimization of the objective function, doing all ℓ in-trees in parallel using 24 cores. When the solutions for all trees from BARON are ready, the solutions are read from secondary storage into memory. The final security instrument placement is computed by combining the trees' optimal placements. The time spent for the BARON computation and combining the solutions is measured and compared against the serial computation in Figure 6, with the values of ℓ annotated on the dashed line. The values of k and m were taken as $k = \lfloor log_2(m) \rfloor$ and $m = \ell$. The serial computation involves security improvement with the full attack tree.

The results of Figure 6 indicate a large difference between the serial and the parallel tree partitioning heuristic times for solving the multiple improvement placement problem. Notice that the two computations have different mathematical interpretations with the optimal solution only guaranteed by the serial version. The tree partitioning method can yield an optimal solution for some problems. In our experiments, the achieved objective function of the heuristic method for all graphs had negligible relative error when compared to the exact serial version. This is an inherent capability of the model developed earlier in [8] as the optimization problem always selects the best route towards the target. Thus, each tree partition participates in a race to provide the best solution for the final objective. In this case, all sub-intrees attempt to select m best placements. The final optimization problem combines and selects the best value among all sub-intrees, thereby resulting in the best m placements being selected. One may argue that the selection may miss some of the opportunities in combining a subset of selections from several subtrees instead of distributing all the m selections in an entire sub-intree. However, per our experiments the difference in the final objective function is extremely small.

C. INSTRUMENT TARGET PARTITIONING

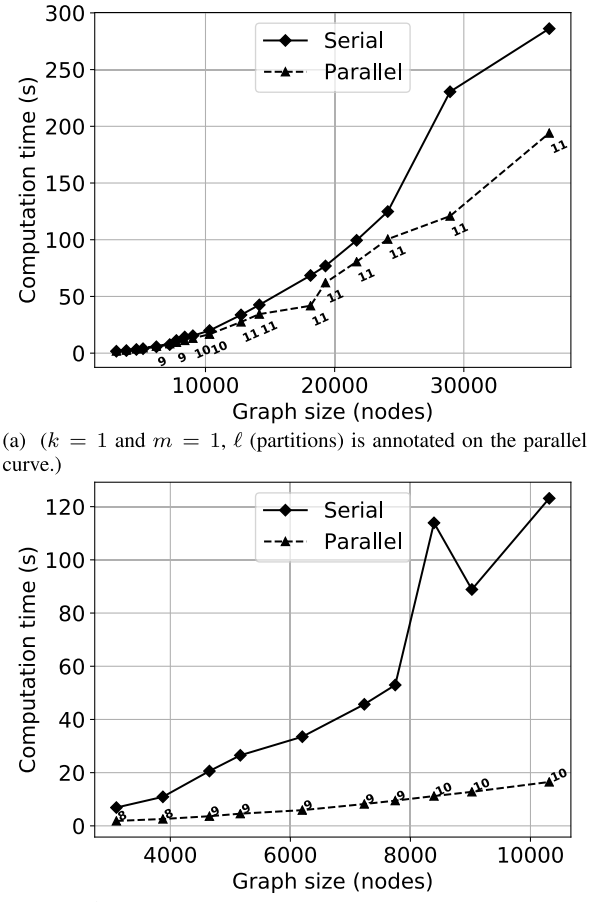
The performance of this approach is depicted in Figure 7. In Figure 7a, the case with $k = 1$ and $m = 1$ is considered



(a) ($k = 1$ and $m > 1$ is annotated on the lower curve)

(b) ($k = 1, m = 1$)

FIGURE 6. A comparison of the serial and parallel times for security improvements using the tree partitioning method.



(a) ($k = 1$ and $m = 1$, ℓ (partitions) is annotated on the parallel curve.)

(b) ($k > 1$ is annotated on the parallel curve, and $m = 1$)

FIGURE 7. A comparison of the serial and parallel times for security improvement using improvement instrument target partitioning.

whereas in Figure 7b, the case with $k > 1$ and $m = 1$ is considered.

In Figure 7a, the performance of the parallel algorithm is poor in smaller graphs of less than 8000 nodes. The performance improvement generally increases as the graph size increases, widening the gap between the serial and parallel versions, as the attack digraph increases in size. In this particular instance, the parallel algorithm has an average speedup of $S \approx 0.18$ with a maximum speedup of $S \approx 0.9$. The reason for poor parallel performance is due to the small number of binary variables requiring deep branch-and-reduce iterations. Note that here $k = 1$, which requires a single binary variable for each improvement target.

The performance in Figure 7b clearly shows the advantage of the parallel algorithm, showing an average speedup of $S \approx 6$ and a maximum speedup of $S \approx 10$. The graph shows the performance of both serial and parallel algorithms with the value of k (the number of available instrument types) shown as annotations on the parallel performance curve. In this experiment, as the value of k grows, the serial algorithm time increases rapidly. The parallel algorithm time has a slower growth as the value of k and the size of the graph are increased.

The number of threads used in this example is equal to the number k of target subsets used. Since the parallel version of the problem is mathematically equivalent to the serial version of the problem, the final objective function values for the two versions should be equal, as BARON was allowed to find the globally optimal solutions.

Table 1 shows a comparison of the serial (branch and reduce) optimization algorithm with the introduced tree partitioning method. In this experiment, the maximum wall time for BARON is set to 500 seconds. The results clearly indicate that the serial algorithm quickly becomes infeasible as the digraph size increases. In contrast, our proposed method can aid in producing real-time responses as the computation for digraphs as large as 80,000 nodes only requires less than 10 seconds of wall time. This indicates the efficacy of our proposed methods as described in Section V.

D. COMPARISON OF THE PARTITIONING METHODS

The two partitioning methods present alternative ways to decompose the security improvement problem and reduce computation time. Both models provide the same guarantees (or lack thereof, for the $m > 1$ heuristics) in terms of finding the optimal solution. While improvement target partitioning is simpler to implement, the tree partitioning method

TABLE 1. A comparison of computation time (wall time) between the serial and the tree partitioning methods for large digraphs. The computation time is in seconds. To avoid lengthy computation times, BARON is bound to 500 seconds of wall time for the optimization algorithm (the extra seconds are to finalize the results).

Number of nodes	Number of subproblems	Time (serial)	Time (Tree Partitioning)
28933	120	109.70s	2.72s
36613	120	167.69s	3.52s
39664	130	209.31s	3.67s
65144	130	506.42s	6.34s
70155	140	500.26s	7.14s
80177	160	504.02s	8.24s

outperforms the improvement target partitioning method in all experiments. Also, the improvement target partitioning method does not work with the case $m > 1$. Thus, comparing the relative objective function error between the two methods is not possible. The tree partitioning method horizontally divides the tree while the improvement target partitioning method maintains the same tree structure in each subproblem. The two models can be potentially combined to improve the results.

VII. CONCLUSION

This work presented a parallel method for probability propagation in attack digraphs and for computing security improvement plans based on attack digraphs. The performance achievement of the probability propagation is mainly due to solving significantly smaller subproblems and the nature of the attack trees. For large attack digraphs and $m = 1$ (e.g., to optimally repair a particular security hole), the parallel tree partitioning method for security improvement produces a solution comparable to that from the serial optimization, and takes only a second or so, making real-time security repairs practical. This work does not address the problem when $k > m$ as the solution cannot use coarse-grained parallelism. Better parallel performance gains can be achieved by fine-grained parallelism in which a new parallel algorithm (e.g., a parallel version of BARON) is designed to specifically parallelize the optimization of security improvements for attack digraphs. This is a significant and challenging problem, which is left for future work.

REFERENCES

[1] C. Phillips and L. P. Swiler, "A graph-based system for network-vulnerability analysis," in *Proc. Workshop New Secur. Paradigms (NSPW)*, 1998, pp. 71–79.

[2] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, graph-based network vulnerability analysis," in *Proc. 9th ACM Conf. Comput. Commun. Secur. (CCS)*, Nov. 2002, pp. 217–224.

[3] S. Jajodia, S. Noel, and B. O'Berry, "Topological analysis of network attack vulnerability," in *Managing Cyber Threats: Issues, Approaches and Challanges*, V. Kumar, J. Srivastava, and A. Lazarevic, Eds. Norwell, MA, USA: Kluwer 2003, ch. 5.

[4] X. Ou, S. Govindavajhala, and A. W. Appel, "MulVAL: A logic-based network security analyzer," in *Proc. USENIX Secur. Symp.* Baltimore, MD, USA, vol. 8, 2005, pp. 113–128.

[5] R. Dewri, N. Poolsappasit, I. Ray, and D. Whitley, "Optimal security hardening using multi-objective optimization on attack tree models of networks," in *Proc. 14th ACM Conf. Comput. Commun. Secur. (CCS)*, 2007, pp. 204–213.

[6] N. Poolsappasit, R. Dewri, and I. Ray, "Dynamic security risk management using Bayesian attack graphs," *IEEE Trans. Dependable Secure Comput.*, vol. 9, no. 1, pp. 61–74, Jan. 2012.

[7] M. Khouzani, Z. Liu, and P. Malacaria, "Scalable min-max multi-objective cyber-security optimisation over probabilistic attack graphs," *Eur. J. Oper. Res.*, vol. 278, no. 3, pp. 894–903, Nov. 2019.

[8] H. M. J. Almohri, L. T. Watson, D. Yao, and X. Ou, "Security optimization of dynamic networks with probabilistic graph modeling and linear programming," *IEEE Trans. Depend. Sec. Comput.*, vol. 13, no. 4, pp. 474–487, Jul. 2016.

[9] Y. Xu, Z. Xu, B. Chen, F. Song, Y. Liu, and T. Liu, "Patch based vulnerability matching for binary programs," in *Proc. 29th ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, Jul. 2020, pp. 376–387.

[10] J. Xu, P. Guo, M. Zhao, R. F. Erbacher, M. Zhu, and P. Liu, "Comparing different moving target defense techniques," in *Proc. 1st ACM Workshop Moving Target Defense (MTD)*, 2014, pp. 97–107.

[11] H. Li, Y. Wang, and Y. Cao, "Searching forward complete attack graph generation algorithm based on hypergraph partitioning," *Procedia Comput. Sci.*, vol. 107, pp. 27–38, Jan. 2017.

[12] H. Meyerhenke, P. Sanders, and C. Schulz, "Parallel graph partitioning for complex networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 9, pp. 2625–2638, Sep. 2017.

[13] G. Malewicz, H. M. Austern, J. C. A. Bik, C. J. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, New York, NY, USA: Association for Computing Machinery, 2010, pp. 135–146.

[14] K. Kaynar and F. Sivrikaya, "Distributed attack graph generation," *IEEE Trans. Depend. Sec. Comput.*, vol. 13, no. 5, pp. 519–532, Sep. 2016.

[15] J. B. Hong, T. Eom, J. S. Park, and D. S. Kim, "Scalable security analysis using a partition and merge approach in an infrastructure as a service cloud," in *Proc. IEEE 11th Intl Conf Ubiquitous Intell. Comput. IEEE 11th Intl Conf. Autonomic Trusted Comput. IEEE 14th Intl Conf Scalable Comput. Commun. Associated Workshops*, Dec. 2014, pp. 50–57.

[16] C. A. Phillips, "The network inhibition problem," in *Proc. 25th Annu. ACM Symp. Theory Comput. (STOC)*, 1993, pp. 776–785.

[17] M. Dacier, Y. Deswarte, and M. Kaâniche, "Quantitative assessment of operational security: Models and tools," in *Information Systems Security*, S. Katsikas and D. Gritzalis. London, U.K.:, Chapman & Hall, 1996, pp. 86–179.

[18] I. S. Moskowitz and M. H. Kang, "An insecurity flow model," in *Proc. workshop New Secur. Paradigms (NSPW)*, 1997, pp. 61–74.

[19] X. Ou, W. F. Boyer, and M. A. McQueen, "A scalable approach to attack graph generation," in *Proc. 13th ACM Conf. Comput. Commun. Secur. (CCS)*, 2006, pp. 336–345.

[20] N. Cao, K. Lv, and C. Hu, "An attack graph generation method based on parallel computing," in *Proc. Int. Conf. Sci. Cyber Secur.* Cham, Switzerland: Springer, 2018, pp. 34–48.

[21] X. Yin, Y. Fang, and Y. Liu, "Real-time risk assessment of network security based on attack graphs," in *Proc. Int. Conf. Inf. Sci. Comput. Appl. (ISCA)*, 2013, pp. 75–80.

[22] A. A. Ramaki, M. Khosravi-Farmad, and A. G. Bafghi, "Real time alert correlation and prediction using Bayesian networks," in *Proc. 12th Int. Iranian Soc. Cryptol. Conf. Inf. Secur. Cryptol. (ISCISC)*, Sep. 2015, pp. 98–103.

[23] M. Gupta, J. Rees, A. Chaturvedi, and J. Chi, "Matching information security vulnerabilities to organizational security profiles: A genetic algorithm approach," *Decis. Support Syst.*, vol. 41, no. 3, pp. 592–603, Mar. 2006.

[24] H. Huang, S. Zhang, X. Ou, A. Prakash, and K. Sakallah, "Distilling critical attack graph surface iteratively through minimum-cost SAT solving," in *Proc. 27th Annu. Comput. Secur. Appl. Conf. (ACSAC)*, 2011, pp. 31–40.

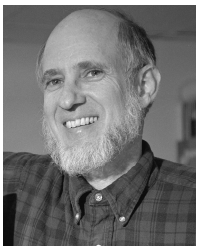
[25] M. Waniek, T. P. Michalak, and A. Alshamsi, "Strategic attack & defense in security diffusion games," *ACM Trans. Intell. Syst. Technol.*, vol. 11, no. 1, pp. 1–35, Feb. 2020.

[26] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques—Adaptive Computation and Machine Learning*. Cambridge, MA, USA: MIT Press, 2009.

[27] A. Darwiche, "A differential approach to inference in Bayesian networks," *J. ACM*, vol. 50, no. 3, pp. 280–305, May 2003.

[28] M. Vasimuddin, S. P. Chockalingam, and S. Aluru, "A parallel algorithm for Bayesian network inference using arithmetic circuits," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2018, pp. 34–43.
[29] K. Li and P. Hudak, "Memory coherence in shared virtual memory systems," *ACM Trans. Comput. Syst.*, vol. 7, no. 4, pp. 321–359, Nov. 1989.
[30] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," in *Proc. 10th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, 2012, pp. 17–30.
[31] A. Kyrola, G. Blelloch, and C. Guestrin, "Graphchi: Large-scale graph computation on just a PC," in *Proc. 10th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, Hollywood, CA, USA, 2012, pp. 31–46.
[32] J. A. Lukes, "Efficient algorithm for the partitioning of trees," *IBM J. Res. Develop.*, vol. 18, no. 3, pp. 217–224, May 1974.
[33] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Proc. 6th Symp. Operating Syst. Design & Implement. (OSDI)*, 2004, p. 10.
[34] S. Zhang, D. Caragea, and X. Ou, "An empirical study on using the national vulnerability database to predict software vulnerabilities," in *Proc. Int. Conf. database expert Syst. Appl.* Berlin, Germany: Springer, 2011, pp. 217–231.
[35] K. A. Farris, A. Shah, G. Cybenko, R. Ganesan, and S. Jajodia, "VULCON: A system for vulnerability prioritization, mitigation, and management," *ACM Trans. Privacy Secur.*, vol. 21, no. 4, pp. 1–28, Oct. 2018.
[36] P. Mell, K. Scarfone, and S. Romanosky, "Common vulnerability scoring system," *IEEE Secur. Privacy Mag.*, vol. 4, no. 6, pp. 85–89, Nov. 2006.
[37] S. C. Coley. (Sep. 2014). *Common Weakness Scoring System (CWSS)*. Accessed: Jan. 12, 2020. [Online]. Available: https://cwe.mitre.org/cwss/cwss_v1.0.1.html
[38] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*. New York, NY, USA: Springer, 2011.
[39] H. S. Ryoo and N. V. Sahinidis, "A branch-and-reduce approach to global optimization," *J. Global Optim.*, vol. 8, no. 2, pp. 107–138, 1996.
[40] A. E. Feldmann and L. Foschini, "Balanced partitions of trees and applications," *Algorithmica*, vol. 71, no. 2, pp. 354–376, 2015.
[41] N. V. Sahinidis, "BARON: A general purpose global optimization software package," *J. Global Optim.*, vol. 8, no. 2, pp. 201–205, 1996.
[42] R. Chandra, L. Dagum, D. Kohr, R. Menon, D. Maydan, and J. McDonald, *Parallel programming in OpenMP*. San Mateo, CA, USA: Morgan Kaufmann, 2001.

HUSSAIN M. J. ALMOHRI (Member, IEEE) received the B.S. degree in computer science from Kuwait University and the Ph.D. degree in computer science from Virginia Tech, in 2013. He is currently with the Computer Science Department, Kuwait University. He has co-founded a mobile payment startup. He has also advised a number of software startups. His research interests include systems and network security, intrusion detection in the IoT, and application of optimization in computer security. He served as a reviewer for several IEEE journals and security conferences.



LAYNE T. WATSON (Life Fellow, IEEE) received the B.A. degree *(magna cum laude)* in psychology and mathematics from the University of Evansville, Indiana, in 1969, and the Ph.D. degree in mathematics from the University of Michigan, Ann Arbor, MI, USA, in 1974.

He has worked with USNAD Crane, Sandia National Laboratories, and General Motors Research Laboratories. He served on the faculties of the University of Michigan, Michigan State University, and the University of Notre Dame. He is currently a Professor of Computer Science, Mathematics, and Aerospace and Ocean Engineering with Virginia Polytechnic Institute and State University. He has published well over 300 refereed journal articles and 200 refereed conference papers. His research interests include fluid dynamics, solid mechanics, numerical analysis, optimization, parallel computation, mathematical software, image processing, and bioinformatics. He is a fellow of the National Institute of Aerospace and the International Society of Intelligent Biological Medicine. He serves as a Senior Editor for *Applied Mathematics and Computation*, and an Associate Editor of *Computational Optimization and Applications*, *Evolutionary Optimization*, *Engineering Computations*, and the *International Journal of High Performance Computing Applications*.

HOMA ALEMZADEH (Member, IEEE) received the B.Sc. and M.Sc. degrees in computer engineering from the University of Tehran and the Ph.D. degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign. She worked with the DEPEND Group within the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign. Before joining the University of Virginia (UVA), she was a Research Staff Member with the IBM T. J. Watson Research Center. She is currently an Assistant Professor with the Department of Electrical and Computer Engineering, University of Virginia (UVA). She is also affiliated with the LinkLab, Multi-Disciplinary Research Center for Cyber-Physical Systems (CPS). She is particularly interested in data-driven resilience assessment and design of embedded and cyber-physical systems, with a focus on safety and security validation and monitoring in medical devices and systems, surgical robots, and autonomous systems. Her research interests include intersection of computer systems dependability and data science.

MOHAMMAD ALMUTAWA received the B.S. degree in computer science from Oregon State University, and the master's and Ph.D. degrees in computer science from the University of Colorado at Boulder, Boulder, CO, USA. He is currently an Assistant Professor of Computer Science with Kuwait University. His research interests include security and privacy, pervasive computing, and the Internet of Things.