Metrics-based Computer Network Defence Decision Support

Reginald Sawilla
Defence R&D Canada – Ottawa
3701 Carling Avenue
Ottawa, Ontario K1A 0Z4
Canada
reg.sawilla@drdc-rddc.gc.ca

Craig Burrell
Defence R&D Canada – Toronto
P.O. Box 2000
Toronto, Ontario M3M 3B9
Canada
craig.burrell@drdc-rddc.gc.ca

ABSTRACT

Recent advances in the construction and analysis of attack graphs have provided new tools to network defenders. Even so, improving the security of networks remains an incredibly complex task. With increasing numbers of vulnerabilities, maturing attacker tools, and organizations becoming ever more reliant on computer network infrastructure, automation and recommendation tools are essential. Much course of action recommendation research to date has worked with the assumption that perfect network security is possible. In reality, network administrators balance security with usability and so they tolerate vulnerabilities and imperfect security. In this paper we present course of action recommendation algorithms that compute efficient and effective solutions which improve the security of networks within real-world constraints including patch availability, resource costs, and usability costs. Our solution builds upon existing metric research in order to give courses of action that maximally disrupt an attacker's ability to reach critical targets of the administrator's choosing. A polynomial time algorithm makes greedy choices to produce courses of action that are almost always the optimal choices computed by an exponential algorithm, making our solution especially useful in practice. We demonstrate the value of our solution through several experiments.

1.0 INTRODUCTION

Modern network environments are extremely complex, as is the job of ensuring their security. Network administrators must defend assets of varying priority against attacks of diverse complexity and maturity. Additionally, they have only finite resources at their disposal and they face the difficult task of balancing security, usability, and resources. In this paper we present a solution that supports network administrators in making best-case security decisions that optimize resource utilization and maximally defend their network, within a specified budget.

Considerable work [1–8] has been done in recent years to develop automated security analysis tools that compute multi-hop network attacks based upon network configuration data such as host connectivity, installed software, services offered, and open source vulnerability databases. A computationally practicle method describes complex network attacks by the use of *dependency attack graphs*. Dependency attack graphs illustrate how low-level details of the network configuration can be chained together by an attacker to compromise network assets. Additionally, representing attacks as a graph allows problems to be solved using graph theory techniques.

Attack graphs may be generated with the attacker starting at any node (including internal nodes) and targetting any number of nodes. Since the information describing a successful attack is contained in the graph, a network administrator could, in theory, use an attack graph to decide on an ideal course of action (COA) to strategically or tactically improve the security of the network. However, while it is simple to consider each individual attack step in the graph, analyzing the global graph is overwhelmingly complex. Hence, the network defender is inundated with information and unclear about how best to proceed. Furthermore, zero-day vulnerabilities, resource limitations, and operational constraints make it very difficult to completely secure a network [9]. Network defenders require a solution that maximally (but not necessarily completely) improves security within real-world constraints.

1.1 Problem

Defenders of enterprise networks have finite resources. For instance, there are a fixed amount of staff resources available to test and roll-out configuration changes and patches. Defenders cannot assume that all attacks will be completely neutralized so they must determine the COA that maximally defends the network. We build on previous research on the defender's prioritization of services, computation of attack paths, and the attackers' relative dependence on intermediate attack steps [10, 11]. The defender wishes to spend his/her resources in the manner that maximally disrupts the attackers' capability to control the defender's services, according to the defender's priorities. The attackers' priorities might not be the same as the defender's priorities; however, our prime concern is the defender's priorities. The attackers' priorities will likely be diverse based on their differing motivations and they need not distract the defender, whose responsibility is to the organization's priorities. The degree to which an attacker depends upon each attack asset (e.g.: user credentials, network connectivity, vulnerability) in the attack graph is represented by rank values assigned to each attack asset. The cost to change a network configuration is provided by the defender. We do not assume the attack can be completely removed within the budget. The problem is computing a COA, in time polynomial with the network size, which removes the greatest amount of attacker capability to reach prioritized services, while staying within the defender's budget (for example, staff hours).

1.2 Contribution

In this paper we define the closure of a COA in the context of attack graphs, and present a polynomial-time best first search greedy algorithm for generating COA recommendations. Our approach combines knowledge of the logical structure of network attacks, configuration data, vulnerability attributes, exploit maturity, device prioritization, remediation costs, and budget limitations. Given an attack graph that describes how attacks against a specified set of network assets could occur, our method maximally removes attacker capabilities for reaching critical network services. Previous approaches took a goal-centric total defence strategy and concentrated on the full combination of privileges required to attack a goal. They compute the cost-optimal solution using exponential algorithms. Our approach is COA-centric and enables the polynomial computation of partial solutions which maximize the return on investment.

2.0 BACKGROUND

A network attack graph is a mathematical abstraction, consisting of vertices and arcs (directed edges), which describes how an asset, or set of assets, in a network could be attacked. Two main types of attack graphs have been described in the literature. Early work in the area focused on state enumeration attack graphs [1, 12] in

which each vertex represents a particular state of the entire system and arcs represent state transitions. These graphs suffered from scalability issues which were resolved by using dependency attack graphs [2–5], in which each vertex represents a particular fact about the system and arcs represent the logical dependencies between the facts. Because they do not attempt to encode all possible states of the system, dependency attack graph are more concise and efficient than state enumeration attack graphs, and their vertices directly suggest COAs. In this paper we deal with dependency attack graphs.

Although our approach to COA optimization could be applied to dependency attack graphs acquired from any source, we generate our graphs using the open source MulVAL security analysis tool suite [10]. MulVAL accepts as input a description of the network configuration (network routes, installed software, file privileges, active services, *etc.*), and software vulnerability data from the National Vulnerability Database (NVD)[1]. MulVAL employs Datalog [13] and a set of deductive logic rules to produce an attack graph describing all possible attack paths leading to a specified set of assets. The attack graph is a directed graph composed of three types of vertices: AND, OR, and SINK. The SINK vertices are the network configuration facts, the AND vertices correspond to attack methods, and the OR vertices correspond to new capabilities the attacker can derive. AND vertices are valid if all of their out-neighbours are valid while OR vertices are valid if any one of their out-neighbours are valid [14]. A network defender may examine the graph to see which combinations of SINK vertices allow an attacker to obtain a privilege (*e.g.*: the ability to execute arbitrary code on a particular server). Invalidating (removing) a SINK vertex corresponds with removing a network fact (*e.g.*: patching vulnerable software). These directed, AND/OR dependency graphs are a species of directed hypergraph called F-hypergraphs [15].

Attack graphs are valuable aids for identifying and analyzing security problems in a network, but even for relatively small networks, the attack graph can be large, complex, and difficult to interpret. Additional analysis is required to extrapolate intelligent COA recommendations from the graph to support the network administrator.

To be useful in practice, COA recommendations should take into account four factors. First, some network configuration changes are more costly than others. We use the concept of a *loss function* [16] to provide a *cost* corresponding to the removal of each SINK vertex. A loss function is a non-negative function mapping an event (a configuration change) to a real number (the cost incurred in making the configuration change). The use of a loss function scopes the complicated task of determining financial cost, time investment cost, and denial-of-use cost[2] to decision theory research. In our model, the user can iteratively increase or decrease the cost in response to COA recommendations. If a network fact cannot be changed (*e.g.*: vulnerable critical software without a patch available), its cost is considered infinite.

Second, the network administrator has a limited budget to expend on these costs. Except for very simple networks, it is unlikely that all of the security problems can be fixed; the administrator aims to maximally improve security, given the budget available.

Third, some vulnerabilities and privileges are more useful to attackers than others due to the ease of exploitation, stealthiness of the attack, capabilities they give attackers in furthering the attack, or other factors, and one ought to preferentially obstruct the attackers' ability to exploit or acquire the most useful privileges. We model the dependence that attackers are likely to have on each attack asset by assigning a rank weight using AssetRank [11] to every vertex in the attack graph.

AssetRank is an algorithm that assigns a rank weight to each vertex in a dependency graph based upon the global structure of the graph, the likelihood of attacker success in using or exploiting a network fact, the maturity of attack tools, the likelihood of a successful social engineering attack, and the identification of critical

[1] http://nvd.nist.gov/cvss.cfm

[2] Insofar as certain changes will disrupt network function and impair the organization's business activity.

assets. It is an eigenvector ranking algorithm that can handle weighted directed graphs with both AND and OR vertices, and vertex-specific damping factors. In the context of network attack graphs, the AssetRank value assigned to each vertex represents the relative dependence an attacker places on that vertex in furthering his attack against the network. As such, the defender's objective should be to preferentially eliminate vertices with high AssetRank values. All rank weights sum to 1 and the ranks reflect the relative value of the graph vertex to the attacker. Any rank function whose ranks can be linearly aggregated may be used as an alternative to AssetRank.

Fourth, some network assets are more valuable to the defender than other assets, and their protection should be prioritized accordingly. In our experiments we manually assign a priority to services but complex enterprise networks could again use AssetRank to compute a criticality metric for each service that reflects the system dependence upon that service.

3.0 COMPUTING COURSES OF ACTION

Given a dually weighted[3] dependency attack graph, we consider how to generate optimal COA recommendations under a budgetary constraint. In general, we want to maximally prevent the attacker from reaching the target vertices the defender has prioritized for protection. We define two sets of vertices in the attack graph: the goal set contains the goal vertices and the attacker set contains the attacker starting location(s). Our objective is then to maximally decrease the connectivity between the goal set and the attacker set. AssetRank values reflect the degree to which the attacker depends upon each vertex in reaching the goal set.

AND/OR directed graphs correspond to propositional logic formulas. SINK vertices are logic variables that may be true or false. When the attack graph is built, all SINK vertices are set to true (representing the fact that vulnerabilities exist, certain software is installed, network paths exist, *etc.*). The remaining vertices in the graph build conjunctive (AND vertices) and disjunctive (OR vertices) clauses. In addition, each vertex corresponds to a logical statement that can be simplified to consist entirely of SINK vertices. The statement will be true or false, depending upon the value of the variables (SINK vertices). Wang *et al.* [17] compute optimal COAs by building logic formulas for the goal vertices. However, they point out that building the formulas requires exponential time, and so a new approach is required to be useful in practice. Furthermore, if setting one or more SINK vertices to false does not falsify the goal, the approaches in [17, 18] will not indicate the degree to which the security of the network has changed.

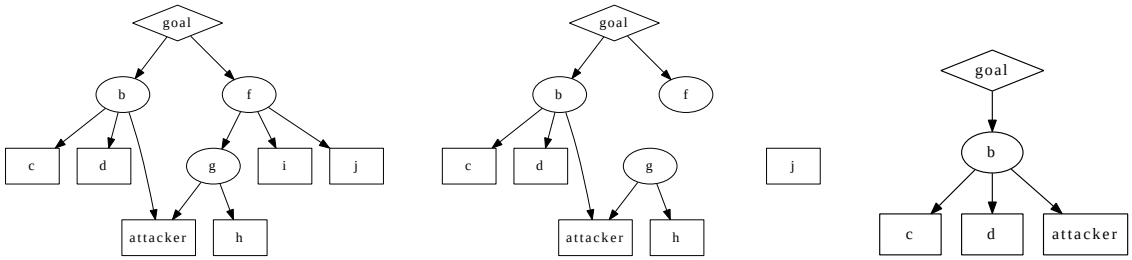
When a SINK vertex is made invalid, the portion of the graph that depended upon it (representing derived privileges) will become invalid, meaning that it is of no use to the attacker in reaching the goal vertices. Summing the AssetRanks of all of the invalid vertices gives a metric indicating the effect on the attacker of removing the SINK vertex (*i.e.*: the proportion of rank that becomes inaccessible to the attacker). However, the determination of invalid vertices is a cascading problem.

Initially, all vertices in the attack are on a path from a goal vertex to the attacker. If we delete the vertices which are invalid (and the incident edges) vertices remaining in the graph could become unreachable from the goal vertex, or unable to reach the attacker vertex. Each vertex that is not in a path from the goal set to the attacker does not aid an attacker to reach the goal. A disconnected subgraph is an example of both unreachability cases.

As an example, consider the attack graph shown in Figure 1(a). If vertex i is deleted, then the compound arc[4] incident to it is also removed, resulting in the graph shown in Figure 1(b). Vertex j is still present but it

[3] Rank weights for all vertices and cost weights for SINK vertices.

[4] Note that every AND vertex has a *single* outgoing compound arc. Invalidating any one of an AND vertex's out-neighbours deletes the arc to every out-neighbour.



(a) Example attack graph

(b) Figure 1(a) attack graph with vertex i and its incident compound arc deleted.

(c) Full effect of removing vertex i and its closure from the Figure 1(a) attack graph.

Figure 1: Determining the effects of vertex removal. AND vertices are denoted by ellipses and correspond to attack methods; OR vertices are denoted by diamonds and correspond to new capabilities the attacker can derive; SINK vertices are denoted by boxes and correspond to network configuration facts.

is not useful to the attacker because, being disconnected, it is no longer reachable. We see that g can reach the attacker but is not reachable by the goal, and so it too is no longer useful. Removing g isolates h so it should also be removed. Vertex f is reachable from the goal but cannot reach the attacker so it is not useful. Figure 1(c) shows the final effect of deleting vertex i. In this example, the COA set is $\{i\}$. We term the set of vertices that are deleted as a result of deleting the COA set the *closure* of the set. In this example the closure of $\{i\}$, denoted $\overline{\{i\}}$, is $\{f, g, h, i, j\}$.

It is important to note that removing a set of COA vertices from the graph can affect legitimate network users because their removal signifies changes to the network configuration; whereas removing the remaining vertices in the closure of the COA set only affects attackers. In the previous example, the removal of SINK vertex i can affect both attackers and users, whereas the SINK vertices j and h are removed from the attack graph, but do not cause configuration changes in the network.

The closure algorithm is used by both the polynomial and exponential algorithms; the full details are presented in Algorithm 1 (VertexSetClosure). Briefly, the algorithm computes the closure of a set of vertices by:

Line 2 Remove the COA vertices from the graph.

Line 3 Compute the AND and OR vertices unable to reach the attacker set in the directed graph. This is efficiently computed by finding the vertices reachable from the attacker set in the graph transpose.[5]

Line 5 Compute the vertices unreachable from the goal set in the directed graph.

Computing the vertex set closure is very efficient (see Section 3.1). Approaching the problem from the goal-informed but COA-centric point-of-view ("What are the network effects of removing this privilege (SINK)?") instead of a goal-centric approach ("What combination of privileges are required to attack the goal?") enables a polynomial complexity algorithm[6] and an efficient way to measure partial security improvements.

Our aim is to compute the COA vertex set with the least cost that removes the maximum rank from the graph. Each SINK vertex has an associated cost given by a loss function (recall Section 2.0), so the cost for the COA set is defined as the sum of the costs of its member vertices. Likewise, each vertex in the graph has a rank value, and we define the rank of the COA set closure[7] as the sum of the ranks of its members. Formally, our

[5]The graph transpose is the graph with all arcs reversed, and is denoted G^T for a graph G.

[6]Polynomial in terms of the number of vertices and arcs.

[7]Note that we are interested in the cost of the COA set but rank of its closure.

Algorithm 1 VertexSetClosure(G,C,S,T) — Closure of a vertex set

Input: AND/OR directed graph $G = (V, A)$, set of vertices C whose closure is to be computed, goal vertex set S that vertices must be reachable from, attacker vertex set T that vertices must be able to reach. Input variables are assumed to have been passed by value (they are modifiable copies).
Output: Graph $G = (V, A)$ with closure $\overline{C}$ deleted, closure $\overline{C}$ of C.

1: $\overline{C} \leftarrow C$
2: $G_V \leftarrow G_V - C$ {Arcs also deleted as necessary}
3: $C \leftarrow \{v | v \in G_V, type(v) =$ AND or OR, $\nexists t \in T(t \rightarrow_{G^T} v\}$
4: $G_V \leftarrow G_V - C$, $\overline{C} \leftarrow \overline{C} \cup C$
5: $C \leftarrow \{v | v \in G_V, \nexists s \in S(s \rightarrow_G v)\}$
6: $G_V \leftarrow G_V - C$, $\overline{C} \leftarrow \overline{C} \cup C$
7: **return** $G, \overline{C}$

optimization problem is to find a subset C of the SINK graph vertices S such that:

$$\max_{C \subseteq S} \left(\sum_{c \in \overline{C}} \text{rank}(c) \right) \tag{1}$$

under the constraint

$$\sum_{c \in C} \text{cost}(c) \leq \text{Budget} \tag{2}$$

Solving this exactly is a binary integer programming problem, which is NP-complete [19]. The optimal solution, Algorithm 2, is computed by brute force by considering all combinations of SINK vertices that are within budget. Since Algorithm 2 has exponential complexity (see Section 3.1), it is not practical for finding complete solutions in large graphs. However, we take advantage of it for subproblems, in particular to improve the results obtained by Algorithm 3 (BFSCOA).

Our greedy algorithm uses the "best first search" (BFS) heuristic with the following evaluation function:

$$f(v, G) = \sum_{u \in \overline{v}} \text{rank}(u) \Big/ \text{cost}(v) \tag{3}$$

This evaluation function gives a return on investment (ROI) by computing the amount of rank that would be removed from the graph with the investment of removing a particular vertex. That is, for each SINK vertex v whose cost is within the budget, we compute the rank of the COA set closure in the context of the graph G and divide by the cost of the vertex. The vertex with the highest ROI is removed from the graph (along with its closure) and the cost is deducted from the available budget. The process is repeated with the resulting subgraph and terminates when the budget is exhausted, or no further improvements are possible within the unspent budget. The full details are presented in Algorithm 3 (BFSCOA).

The closure of candidate COA sets is supermodular, meaning that the closure of the union of sets A and B contains at least the union of the closures of A and B (and often much more). Thus, in contrast to diminishing

Algorithm 2 OptimumCOA(G,S,T,Budget) — Optimum course of action

Input: AND/OR directed graph $G = (V, A)$, goal vertex set S, attacker vertex set T, Budget.
Output: OptimumCOASet, RankEliminated

1: CandidateCOAs $\leftarrow \emptyset$, RankEliminated $\leftarrow -\infty$
2: CandidateCOAs $\leftarrow [C \in$ PowerSet(Sinks$(G))$, cost$(C) \leq$ Budget$]$
3: **for** $(COA_i, Cost_i) \in$ CandidateCOAs **do**
4: $(G_i, \overline{COA_i}) \leftarrow VertexSetClosure(G, COA_i, S, T)$
5: $COARank_i \leftarrow \sum_{v \in \overline{COA_i}} Rank(v)$
6: **if** $COARank_i >$ RankEliminated **then**
7: RankEliminated $\leftarrow COARank_i$
8: CandidateCOAs $\leftarrow \{COA_i\}$ {Replace previous solutions}
9: **else if** $COARank_i =$ RankEliminated **then**
10: CandidateCOAs $\leftarrow$ CandidateCOAs $\cup \{COA_i\}$ {Set of sets}
11: Keep only least expensive solutions in CandidateCOAs
12: **return** CandidateCOAs, RankEliminated

returns, we have increasing returns as we add more vertices to the COA set. As a result, Algorithm 3 can compute a COA set with redundant removals. If the COA set is found in the order a, b, c, and if a is not in $\overline{b}$ or $\overline{c}$ (where overlines denote closure) but is in $a \in \overline{\{b, c\}}$, then $\{a, b, c\}$ is a sub-optimal set that could be returned by the BFSCOA algorithm. Although Algorithm 2 is exponential, it is useful on the subproblem of optimizing the BFS solutions. We can use the COA set found by the BFS algorithm to limit the number of possible COA sets considered by the brute force algorithm. Although the details are omitted, we created a hybrid algorithm that calls Algorithm 2 each time a new vertex is added to the COA set in Algorithm 3. The OptimumCOA algorithm only considers the vertices in the COA set (rather than all SINK vertices), removes the redundant vertices (such as a in the previous example), and returns their cost to the available budget.

3.1 Complexity

The main work in our approach is done in Algorithm 1 (VertexSetClosure). The most complex lines in that algorithm, 3 and 5, compute the reachable vertices from a specific set. The algorithm used is a depth-first-search implementation provided by the Python NetworkX module [20]. The implementation was modified slightly to allow the reachability of a set of vertices to be computed simultaneously. The complexity of the NetworkX reachability algorithm is $\mathcal{O}(v + a)$ where v is the number of vertices in the graph and a is the number of arcs. Since there are no loops in the algorithm, the complexity is $\mathcal{O}(v + a)$.

Algorithm 1 (VertexSetClosure) is called by both Algorithm 2 (OptimumCOA) and Algorithm 3 (BFS-COA). Algorithm 2 (OptimumCOA) computes the optimal course of action for the specified budget by a brute force comparison of all candidate combinations. While the power set of the sink set is of size 2^s where s is the number of sinks, the power set contains many combinations of sinks whose total cost exceeds the budget, so those are eliminated. The size of the remaining set can be much smaller than 2^s. For example, if every SINK has a cost of 1, the budget is b, and every vertex has a positive (non-zero) rank, then the number of candidate courses of action is given by $\mathcal{C}_b^s$ where $\mathcal{C}_b^s = s!/(b!(s - b!)$. As the attack graph grows in size, the discriminator of a limited budget significantly reduces the candidate space. We have implemented (the details are omitted) an

Algorithm 3 BFSCOA(G,S,T,Budget) — Best first search greedy course of action

Input: AND/OR directed graph $G = (V, A)$, goal vertex set S, attacker vertex set T, Budget.
Output: Graph $G' = (V', A')$ with $\overline{COA}$ deleted, COA, $\overline{COA}$ (closure), TotalCost, TotalRank.

TotalCost $\leftarrow 0$, TotalRank $\leftarrow 0$, COA $\leftarrow \emptyset$, $\overline{COA} \leftarrow \emptyset$, SinkClosureComputed $\leftarrow$ **false**, $G' \leftarrow G$
Sinks $\leftarrow \{v | v \in G'_V, Type(v) = \text{SINK}, Cost(v) \leq \text{Budget}\}$
while Sinks $<>$ $\emptyset$ **do**
  $MaxROI \leftarrow -\infty, m \leftarrow 0$
  for $v_i \in$ Sinks **do**
    if SinkClosureComputed **then** {Ensure vertex is within budget, adjust cost}
      COAInClosure$_i \leftarrow$ COA $\cap \overline{v_i}$
      VertexCost $\leftarrow$ VertexCost $- \sum_{u \in \overline{v_i}}$ Cost(u)
      if VertexCost + TotalCost $>$ Budget **then**
        Sinks $\leftarrow$ Sinks $-\{v_i\}$
        next
    $(G_i, \overline{COA_i}) \leftarrow VertexSetClosure(G', \{v_i\}, S, T)$
    $COARank_i \leftarrow \sum_{u \in \overline{COA_i}} Rank(u)$
    $ROI_i \leftarrow COARank_i$/Cost$(v_i)$
    if $ROI_i > MaxROI$ **then**
      $m \leftarrow i$
  if $m = 0$ **then** {No COA found within budget}
    Sinks $\leftarrow \emptyset$
  else
    TotalRank $\leftarrow$ TotalRank $+COARank_m$
    TotalCost $\leftarrow$ TotalCost + Cost(v_m)
    if SinkClosureComputed **then**
      $COA \leftarrow COA-$ COAInClosure$_m$
    $COA \leftarrow COA \cup \{v_m\}$
    $\overline{COA} \leftarrow \overline{COA} \cup \overline{COA_m}$
    $G' \leftarrow G_m$
    Sinks $\leftarrow$ Sinks $- \overline{COA_m}$
    SinkClosureComputed $\leftarrow$ **true**
return G', COA, $\overline{COA}$, TotalCost, TotalRank

efficient dynamic programming method of generating only the candidate courses of action within budget.

The most complex line in the **for** loop is line 4, which has a complexity of $\mathcal{O}(v + a)$. Given that line 4 is executed up to 2^s times in the worst case of an unlimited budget, the complexity of the algorithm is $\mathcal{O}(2^s(v+a))$, but runs within $\sum_{k=0}^{b} \mathcal{C}_k^s$ iterations when a limited budget is specified. Specifying a budget is usually practical because Algorithm 3 (BFSCOA) will compute a solution in polynomial time, thus giving a budget upperbound.

In Algorithm 3 (BFSCOA), the highest time complexity is at line 12 and it is part of a doubly-nested loop. The **for** loop at line 5 is performed over the sinks computed in lines 2 and 27. Recall that the sinks represent facts about the network and they are the only items which can be changed to alter the graph (improve the security of the network). The number of sinks whose cost is within budget is denoted s in the following discussion and is clearly less than the number of vertices in the graph. The **while** loop at line 3 is executed until no sinks remain within budget. The number of sinks decreases by at least 1 at each iteration. Lines 8 and 13 require a summation where the number of operands is bounded by v.

In summary, lines 3 and 5 each loop up to s times, the complexity of line 12 is $\mathcal{O}(v + a)$, giving an overall complexity of $\mathcal{O}(s^2(v+a))$. Note that the graph itself is shrinking (line 26), as are the remaining sinks to check (lines 10 and 27) so using the initial s, v, and a gives a conservative estimate.

4.0 EXPERIMENTS

We illustrate our method using three examples. The first is a network with an attack graph that is small enough to enable a full understanding of the concepts. The second is a network representative of an organization with several servers and remote workers.

4.1 Example 1

The network in Figure 2 has three hosts plus the attacker's PC. Each computer has a remotely exploitable vulnerability. Public databases give information regarding the nature of particular vulnerabilities, and the maturity of attacks against them. Given this information, we set the likelihood of an attacker successfully exploiting the vulnerability on the web server to 80%, the mail server to 40%, and the file server to 100%. The defender's primary concern is safeguarding the file server. The attacker does not have direct access to the file server but can reach it by a multistep attack through either the web server or the mail server.

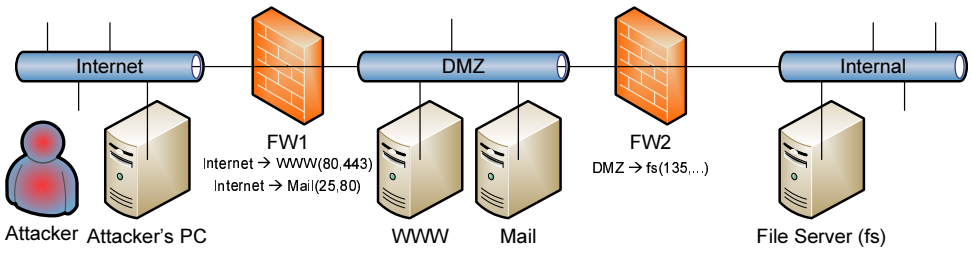


Figure 2: Example 1 network: The attacker can reach the file server by a multistep attack through the web server or mail server.

Figure 3 is the attack graph containing the possible attacks for this scenario. Vertices represent assets (or capabilities) that the attacker can use to further his attack, and arcs indicate dependencies that each derived asset has on other assets. The graph gives a proof tree for how the asset `execCode(fs,system)`[8] is derived from

[8] `execCode(fs,system)` states that the attacker can execute arbitrary code on the file server (fs) at the privilege of the system account.

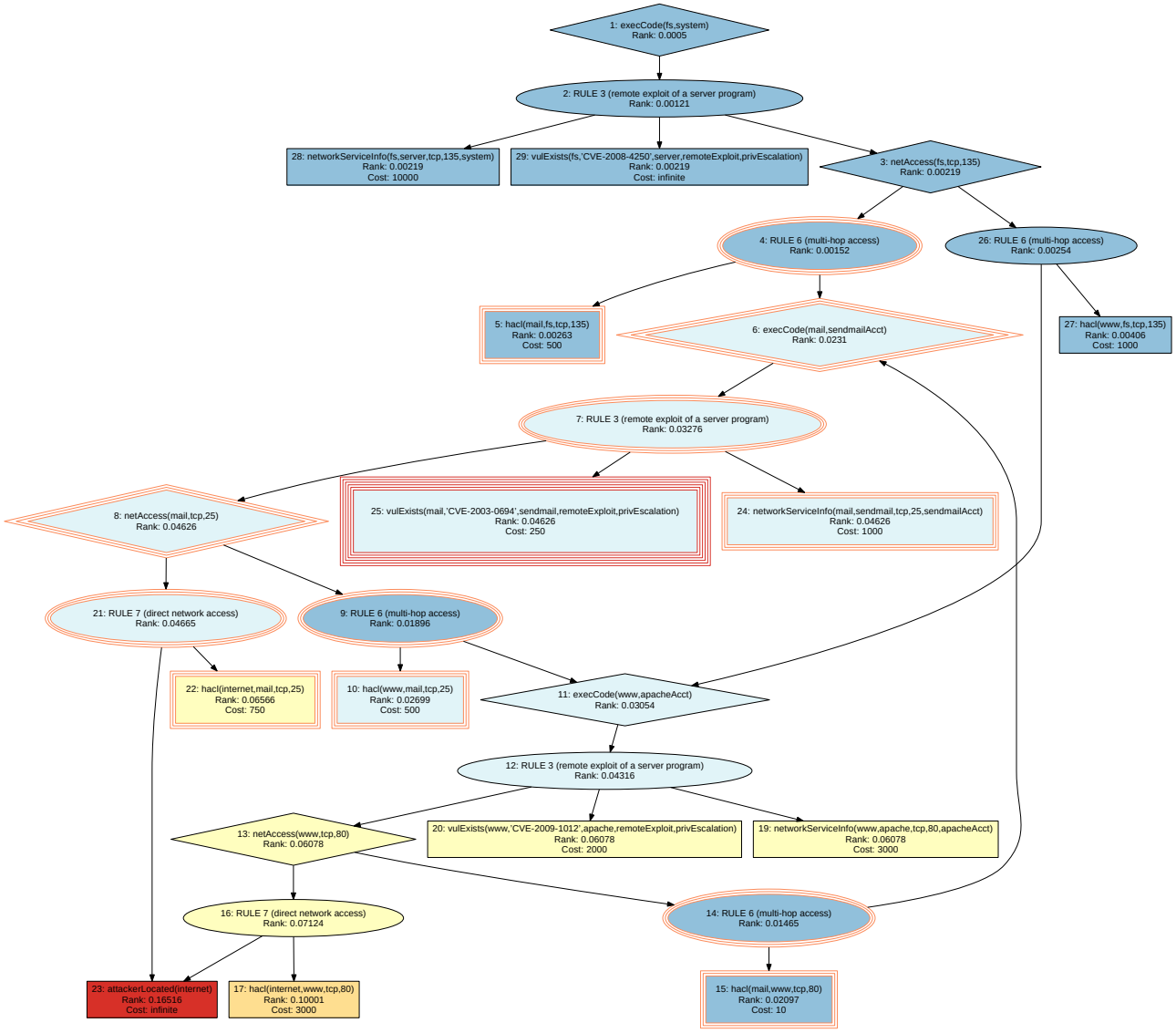


Figure 3: Attack graph for Example 1 network: The graph gives a proof tree of the attacker's ability to obtain code execution privileges on the file server. Vertex 25 is the recommended COA under loss function "Cost 2" with a budget of 1000; the remaining decorated vertices are the closure of vertex 25.

other facts. Vertices are coloured from red (high) to blue (low) to reflect their AssetRank values. The vertices related to the web server have higher rank than those related to the mail server because the exploit on the web server has a higher likelihood of success.

In a network as small as this example, the defender will likely have the resources to patch all software; however, we assume enterprise-level resource restrictions to readily demonstrate the benefits of COA analysis in a network small enough to evaluate. Table 1 gives costs under two different loss functions. The column "Rank Sum of Closure" gives the sum of the rank weights for the closure of each vertex. The goal is to deny the attacker the greatest amount of rank; a rank sum of 1 corresponds with completely eliminating the attack. The full graph contains 28 vertices and 30 arcs. The residual vertices and arcs columns give the number of vertices

and arcs remaining in the graph after the vertex closure is removed. One immediately notices that the number of residual vertices or arcs is a poor indicator of the solution quality.

Cost 1 simply assigns a cost of 250 for every vulnerability patch, 500 for every network route removal, and 1000 for every service shutdown. Using column Cost 1 and a budget of 1000, we immediately notice two solutions that remove the entire attack: vertex 28 at a cost of 1000, and vertex 29 at a cost of 250.

Column Cost 2 presents loss function costs from a manual process. Loss functions are an active and difficult area of decision theory research. Our scope is limited to the optimization of return on investment for courses of action. The user can use any rank or loss functions from decision theory research. The following assumptions are examples of information used by the loss function: an inconvenient workaround is available for the web server vulnerability (vertex 20); a well-tested patch is available for the mail server vulnerability (vertex 25); the file sharing service (vertex 28) is critical to operations and its availability is the highest priority; a patch does not exist for the file server vulnerability (vertex 29). Using Cost 2 and a budget of 1000, it is not at all obvious what the best solution is from an examination of the graph or table. Figure 3 illustrates the solution found by both the greedy and optimal algorithms; namely, the removal of vertex 25 at a cost of 250. There are several other combinations (*e.g.*: {5,10}, {10, 15, 25}, {22}, *etc.*), but every vertex within the budget is contained in the closure of vertex 25, so it is the optimal COA, removing the greatest amount of rank (39%) from the graph. The least cost COA that fully removes the attack on the file server is {25,27} for a cost of 1250.

Table 1: Vertex costs for Figure 3 graph

Vertex Number	Description	Rank Sum of Closure	Residual Vertices	Residual Arcs	Cost 1	Cost 2
5	route: mail to fs	0.00415	26	27	500	500
10	route: www to mail	0.04595	26	27	500	500
15	route: mail to www	0.03562	26	27	500	10
17	route: internet to www	0.17125	26	27	500	3000
19	apache service on www	0.51546	15	14	1000	3000
20	vulnerability on www	0.51546	15	14	250	2000
22	route: internet to mail	0.11231	26	27	500	1000
23	attacker exists	1	0	0	∞	∞
24	sendmail service on mail	0.39267	15	14	1000	1000
25	vulnerability on mail	0.39267	15	14	250	250
27	route: www to fs	0.0066	26	27	500	1000
28	server service on fs	1	0	0	1000	10000
29	vulnerability on fs	1	0	0	250	∞

4.2 Example 2

Only a slightly larger example is required to demonstrate the utility of the optimization algorithms and the efficiency of the greedy algorithm. Consider the small business network shown in Figure 4, based upon an actual business network. It is protected by a firewall configured to permit only traffic deemed necessary for business activities. The firewall also hosts a VPN server to permit secure connections for employees working off-site. The machines in the DMZ host services such as HTTP, DNS, and sendmail; in the Corporate subnet is an internal mail server and web application server, along with several workstation templates (together representing dozens of similarly configured computers) running various operating systems; the Secure subnet is

protected by an additional firewall and contains machines running important services: Porky is a Citrix server for remote application hosting and also hosts the company's financial information; Elmer hosts a file server, source repository, and SQL database containing sensitive data.

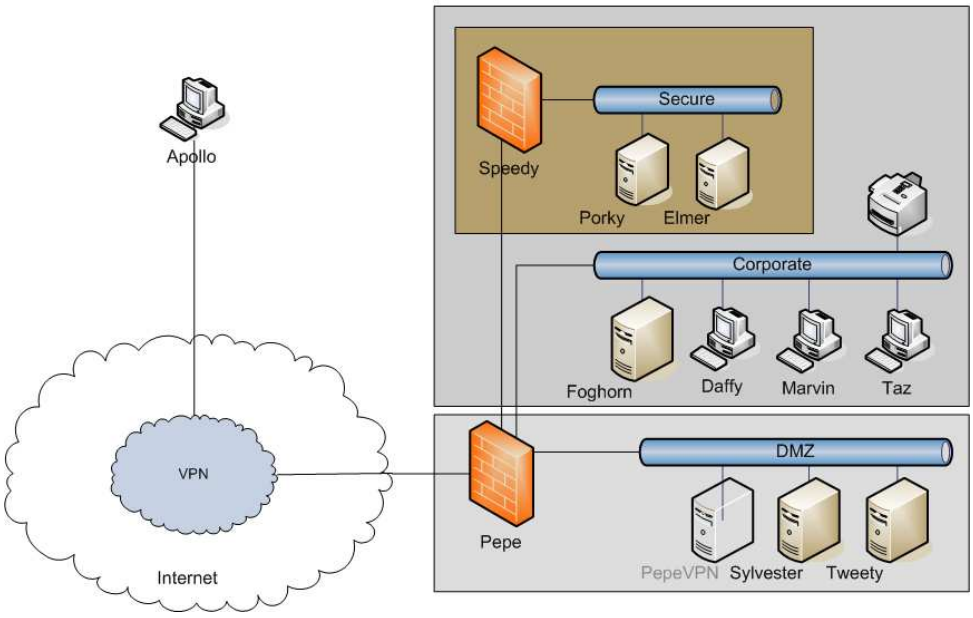


Figure 4: Example 2 network. The attacker is located on the internet. The defender's primary concern is safeguarding the six servers.

All of the servers in the network have been hardened according to NIST security configuration guidelines [21]. Nonetheless, the network is vulnerable to attack because of the presence of several software vulnerabilities.

An attack graph for this network is shown in Figure 5. The graph is complex and, unaided, it is difficult for a human to grasp the security implications of COA decisions without considerable effort. When examined closely, it shows that an attacker could gain root access on four of the servers: Sylvester, Tweety, Foghorn, or Elmer. The four `execCode` SINK vertices corresponding to root access on the servers form the goal set for the algorithms, and the attacker vertex forms the attacker set. Details about network connectivity, vulnerabilities, and costs are contained in Figure 5 (which may be magnified in the electronic version of this document) but a discussion of them is omitted since our intent in this example is to show the range of possible COA solutions and compare the algorithm performance.

We study the greedy and optimal solutions for this attack graph at several different budget levels. Table 2 presents the budget, the actual cost of the COA found within the budget, the percentage of rank eliminated from the graph, the COA vertices, and the CPU time required to compute the solution using a Python 2.5.1 implementation on a 2.4 GHz dual core processor.

Here, the greedy algorithm does not always produce the optimal solution, but it does find nearly optimal solutions efficiently. The rank removed from the graph by the greedy solution is at least 90% of the optimal value for each budget, and the greedy solution is found in a small, and decreasing, fraction of the time to find the optimal solution. As the budget increases – or, more generally, as the number of SINK vertices under consideration increases – the long execution times for optimal algorithms make them impractical for real-world application. At a budget of 4,000, there are 34 SINK vertices to consider resulting in 1,585,000 combinations

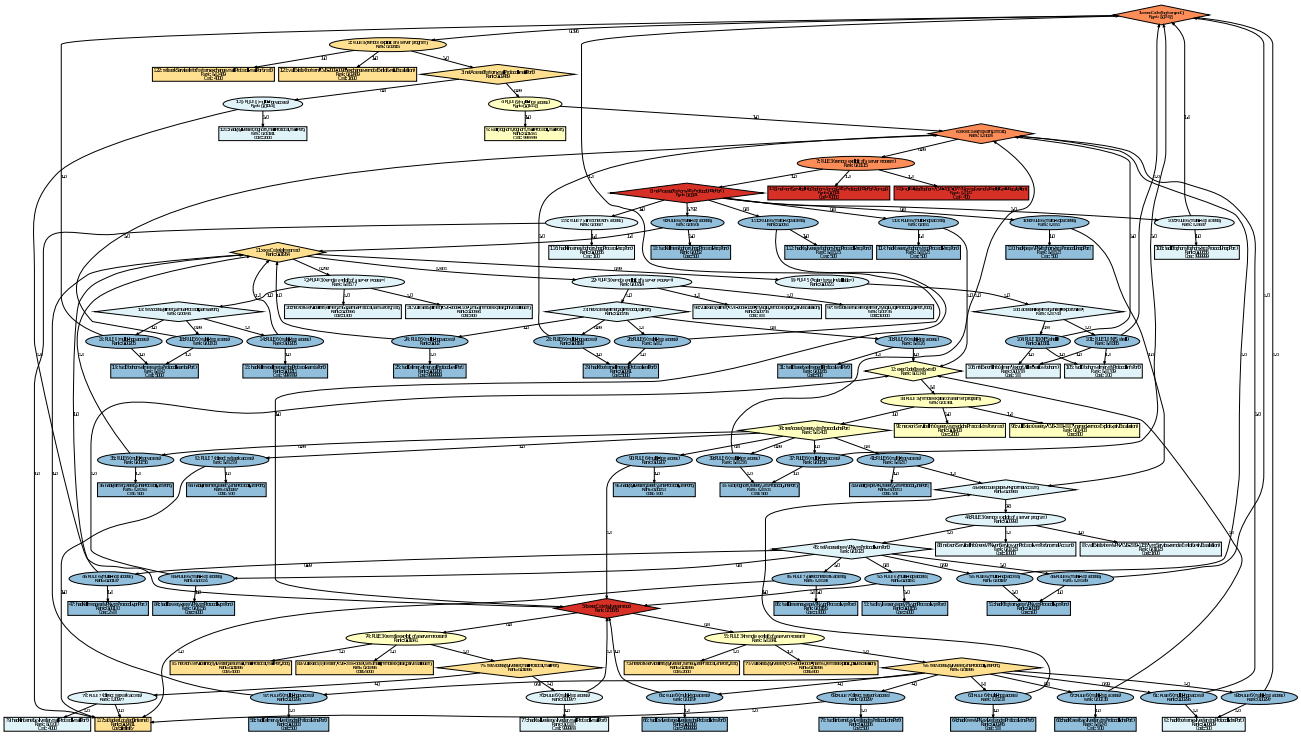


Figure 5: Attack graph for Example 2 network: As configured, an attacker beginning on the internet can obtain root privileges on Sylvester, Tweety, Foghorn, or Elmer. Note: The figures in the electronic version of this document can be magnified in order to read the vertex contents.

Table 2: Course-of-action recommendations by Algorithm 3 BFSCOA. Italicized vertices are redundant vertices that are removed by applying OptimumCOA to the BFSCOA output. CPU times are for a 2.4 GHz dual core processor and Python 2.5.1.

Budget	COA Cost	Rank Eliminated	COA Vertices	CPU Time (seconds)
1000	600	30.9%	119, 105, 51	0.222
2000	1800	48.9%	119, 105, 73, 42	0.367
3000	2900	62.4%	119, 105, 73, 96, 98	0.470
4000	3700	82.4%	119, *105*, 73, 96, 123	0.514
5000	3700	82.4%	119, *105*, 73, 96, 123	0.566
6000	3700	82.4%	119, *105*, 73, 96, 123	0.636
7000	3700	82.4%	119, *105*, 73, 96, 123	0.668
8000	7700	100%	119, *105*, 73, 96, *123*, 81	0.673

within budget for OptimumCOA to consider. Thus, the fast execution time and quality recommendations of the greedy algorithm make it an attractive and practical alternative.

Note that a large proportion of the graph rank can be removed by intelligent deletion of a relatively small set of SINK vertices. This illustrates the supermodular non-linear aspect of the optimization. Consider the Budget $= 2000$ case as an example. The attack graph in Figure 5 has fifty SINK vertices, yet removal of just three or four judiciously chosen vertices (the COA sets from the optimal and greedy algorithms, respectively) is sufficient to remove roughly half the rank in the graph, thereby significantly limiting the attacker's range of action. Similarly, the sum of the (non-infinite) costs of all SINK vertices is 101,600, yet 82% of the rank can

Table 3: Course-of-action recommendations by Algorithm 2 OptimumCOA. The CPU times are for a 2.4 GHz dual core processor and Python 2.5.1.

Budget	COA Cost	Rank Eliminated	COA Vertices	CPU Time (seconds)
1000	1000	31.2%	62, 105, 119	2.814
2000	2000	55.0%	73, 96, 119	139.147
3000	2800	67.4%	96, 119, 123	2238.819
4000	3600	82.4%	73, 96, 119, 123	17711.524

be eliminated with a cost of just 3,600, if the SINK removals are optimal.

To illustrate the suitability of Return on Investment as an evaluation function for the best first search heuristic, consider Figure 6. The solutions generated by Algorithm 2 (OptimumCOA) with up to three vertices in the COA set are plotted with the cost against the ROI. The dot colour reflects the rank sum of the COA closure with the highest rank sums in red and the lowest in blue. The four 3-vertex COAs given in Tables 2 and 3 can be picked out in the figure by identifying the highest ROI dot for each of the costs: 600, 1000, 2000, and 2800. Those COAs optimally remove the attackers' capabilities on the network to reach the defender's priorities.

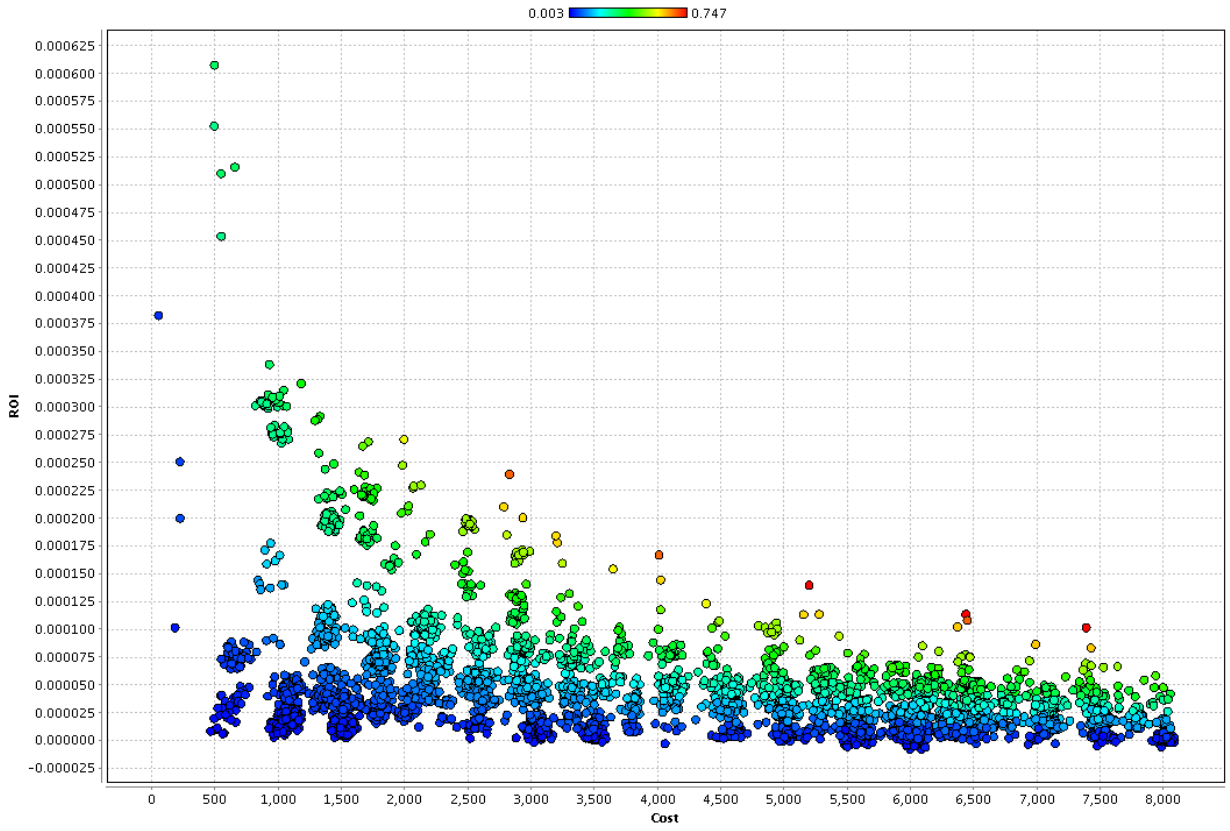


Figure 6: COAs for the Example 2 network: 1-vertex, 2-vertex, and 3-vertex COAs costing less than 8,000. The dot colour reflects the rank sum of the COA closure with the highest rank sums in red and the lowest in blue. (Jitter has been applied to differentiate stacked data points.) The figure illustrates the large number of COA combinations, the fact that cost without considering rank is not a good measure of solution effectiveness.

4.3 Scalability and accuracy

In this section we demonstrate the scalability and efficacy of the Best First Search algorithm, compared to the brute force algorithm. To do this, we randomly generate networks comprised of 10, 100, and 1000 unique configurations. One may think of each configuration as a subnet which contains multiple hosts but all hosts in the subnet share a common baseline configuration. For the purpose of generating a MulVAL attack graph, this format is sufficient since it isn't concerned with the actual number of hosts, only the number of different configurations.

A scale-free network is a network in which the vertex degree distribution follows a power law. The following equation gives the expected percentage of nodes of degree k in a graph following a power law distribution.

$$P_{out}(k) \approx k^{-\gamma_{out}} \text{ and } P_{in}(k) \approx k^{-\gamma_{in}} \tag{4}$$

The world wide web (WWW) and inter-domain networking have been shown to be well-modeled as scale-free networks [22, 23]. Researchers have modelled inter-domain networking using undirected graphs because it accurately models routing. In our case, our concern is not whether the physical infrastructure allows a connection but whether the network access control lists (ACLs) allow a connection.

If a domain allows outbound session initialization to another domain it does not imply it allows inbound session initialization. The most important factor in attack propagation is whether or not a communication session may be stood up. For example, a domain likely allows any host to initiate a communication session with Google; however, the converse would rarely be true.

Preferential attachment is used by Bollobas *et al.* [24] to generate scale-free directed graphs. Conceptually, preferential attachment is the idea that vertices with many in-neighbours are more likely to receive new connections than vertices with few in-neighbours. Likewise, vertices with many out-neighbours are more likely to make a new connection than vertices with few out-neighbours. For the WWW, we see this reflects reality. Network connectivity follows the same pattern. A server that already allows many inbound connections (for example, a DNS server) is likely to allow connections from newly added hosts. Similarly, a server that is allowed to access many other hosts (for example, a patch deployment server) will likely be allowed to access new hosts as well.

In the absence of published data giving experimentally verified power-law exponents, we use the values that Bollobas *et al.* [24] suggest for web graphs. Empirical evidence shows that enterprise network subnets are more likely to initiate outbound connections than to receive them, and this is also the behaviour of web graphs. The values given for web graphs (and used by us) are $\gamma_{out} = 2.7, \gamma_{in} = 2.1$.

4.3.1 Graph generation

First, a directed scale free graph is generated where each vertex of the graph represents a host baseline configuration. Each configuration contains a remotely exploitable vulnerability. Since the network is formed using preferential attachment, a large number of host configurations do not offer connectivity for incoming connections. Hosts sharing a baseline configuration are assumed to be in the same subnet and have unrestricted connectivity to other hosts in the subnet.

Next, an attack graph is generated for the computer network. The attacker's starting location is chosen randomly. Most often, the attacker will have no incoming connections but occasionally it will be a host configuration with dense connections.

Table 4: Distribution of inbound connectivity

Number of inbound connections	Vertex list
0	3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29
1	15, 16
2	6
4	2
8	0
25	1

The number of attack targets varies from 1 to 10% of the number of host configurations. The targets are selected by choosing the hosts with the highest in-degree (most incoming connections). Since the attacker starting location is chosen randomly, care is taken to avoid choosing the attacker's location as a target.

For display purposes, a 100 host configuration network is too large and a 10 host configuration network is too small to be very interesting so Figure 7 shows a randomly generated 30 host configuration network. The network has three attack targets. Tables 4 and 5 give the distribution of inbound and outbound connectivity for the network.

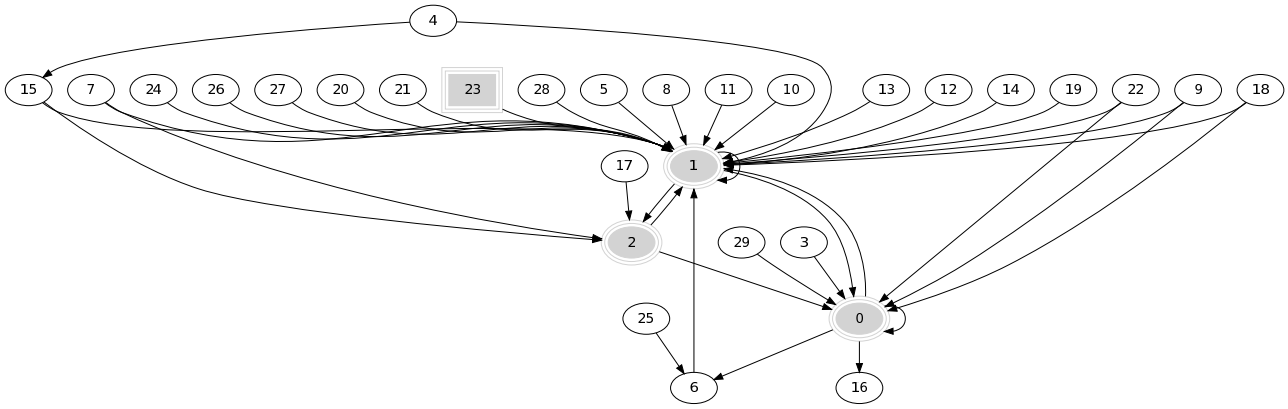


Figure 7: Randomly generated 30 host configuration network with three attack targets (vertices 0, 1, and 2). The attacker's starting location is vertex 23.

4.3.2 Course of action parameters

A loss function cost is associated with each course-of-action as follows:

- Removing a network connection from host A to B costs the number of in-neighbours of A. The rationale is that host A depends on host B for functionality. Removing the connection between A and B will adversely affect all of the hosts connecting to A.

- Shutting down a service on host A costs the number of in-neighbours of A. The same rationale is applied here as was applied for network connections.

Table 5: Distribution of outbound connectivity

Number of outbound connections	Vertex list
0	16
1	3, 5, 6, 8, 10, 11, 12, 13, 14, 17, 19, 20, 21, 23, 24, 25, 26, 27, 28, 29
2	2, 4, 7, 9, 15, 18, 22
3	1
4	0

- Patching a vulnerability costs 1/4 of the number of hosts that have the vulnerability. The cost of deploying a patch is proportional to the number of hosts that have the vulnerability.

For each attack graph, the COA budget is incremented from 1 to $0.25(numhosts)$ in increments of $\lceil 0.25(numhosts)/10 \rceil$ to give up to 10 steps.

4.3.3 Scalability and efficacy

Since the brute force algorithm (OptimumCOA) is too slow to be applied to networks with 1,000 hosts, it was only applied to networks with 10 and 100 hosts. The Best First Search (BFS) algorithm was applied to all three sizes of networks over various combinations of random networks, number of goals, and budget levels. The BFS algorithm gives the same answer as the brute force algorithm 99% of the time in networks with 10 and 100 hosts, but with a much shorter computation time. Table 6 presents the results.

Table 6: Comparison of the Best First Search algorithm versus the brute force algorithm over various combinations of random networks, number of goals, and budget levels.

Property	10 Hosts	100 Hosts	1000 Hosts
Combinations (network, goals, budget)	1504	2622	108
BFS computing time range (sec)	0.00 – 0.16	0.00 – 1.95	0.01 – 85.67
Brute force computing time range (sec)	1.08 – 87.87	1.14 – 50,652.95	N/A
Cases with different COA total rank	14 (0.9%)	28 (1.1%)	N/A

5.0 RELATED WORK

A number of previous papers have studied the problem of security metrics in IT networks. Some have advocated an economics-driven approach to the problem. Gordon and Loeb [25] study how much money should be invested in order to protect information of a certain value but they do not specify how the money can be spent effectively to secure the information. The same generality is a feature of other cost-benefit analyses from an economic perspective [26].

Previous work has used game theory to model the interactions of attackers and defenders in order to develop quantitative risk measures and security strategies. Liu *et al.* model the attacker's incentives in order to infer his intent, objectives, and strategies [27]. Cavusoglu *et al.* model network traffic patterns, attacker intent, monitoring effectiveness, expected damage, and other factors [28]. While such approaches can effectively

parametrize various aspects of network security, the large number of parameters with uncertain or unknown values makes their quantitative predictions very subjective.

Numerous groups have proposed security metrics in the context of attack graphs or trees. One strategy has been to introduce measures of Return-On-Investment (ROI) [29] and Return-On-Attack (ROA) [30]. In this scheme, ROI is a function of monetary damage resulting from attacks, estimates of risk mitigation, and the costs of security investments, while ROA measures the degree to which an attacker's gain is reduced by security investments. These measures are applied to qualitative attack trees. Wang *et al.* [31] use attacker and exploit modeling in the context of attack graphs to estimate the probability of particular network assets being compromised.

Closer to our approach is the work of Jha *et al.* [32] in which attack graphs are used to deduce a minimal set of security measures that will ensure security. Our work differs insofar as we maximize the value of privileges denied to the attacker, but within a fixed budget. We are interested in solutions that will maximally improve, but not necessarily ensure, the security of the system for a given cost. Ingols *et al.* [2] build "multiple prerequisite" attack graphs, and generate single-action recommendations based on the number of hosts secured by deleting vulnerability instances. Our work extends their work by considering costs and combinations of actions, as well as the relative value of privileges to the defender. Jajodia *et al.* [33] propose a "weakest-adversary" metric in which the security of a network is stated in terms of the weakest adversary that can successfully attack it. A benefit of this approach is that it permits comparison of the relative security of two different network configurations.

Dewri *et al.* also study the problem of how to select a set of security hardening measures that satisfy a budgetary constraint [34]. They use genetic algorithms to search for a solution to a multi-objective optimization problem that balances security improvements against cost and potential damage. Their work deals with attack trees while our work deals with the more general case of attack graphs. Recall that any two vertices in a tree are connected by exactly one path, while in a graph they may be connected by multiple and cyclic paths.

Directed graph theory defines an (x, y)-vertex cut set as a set of vertices whose removal disconnects vertices x and y (for example, representing an attacker and goal). Directed graph cut set research is not directly applicable to the problem addressed in this paper for two reasons. The first, and most important reason, is we do not assume it is possible to remove all connectivity between attackers and goals. Second, logic attack graphs are not properly represented by directed graphs but are in fact a species of directed hypergraphs. We are not aware of any published theory on vertex cuts in directed hypergraphs.

Homer and Ou [35] combine MulVAL-generated attack graphs with usability requirements, and use Boolean Satisfiability Solving to find network configurations that provide security while retaining usability. They compute the minimal cost cut-set that will completely protect a set of identified vertices while respecting specified usability requirements. If a solution exists, their method provably finds the optimal (lowest cost) configuration. The attempt to find solutions that simultaneously address security problems and preserve usability is admirable. The advantage of our work is that it can provide an effective course of action even when absolute protection of the goal vertices is not possible.

The previous work that most resembles our approach is that of Wang *et al.* [17], in which a minimum-cost algorithm is used to identify sets of network properties that enable successful attacks. They associate a propositional logic formula with each attack graph vertex that states its truth condition as a function of the truth values of network configuration facts, and they search for minimal-cost sets of facts which can invalidate the attack.

Our work extends existing approaches by employing rank weights, which permit strategic defence of network assets that are most valuable to the attacker. We also produce partial solutions when mitigation costs of a complete solution are too high. If each vertex is assumed to be of equal value to the attacker and the budget is

infinite, our exponential complexity solution is similar to previous exponential approaches.

6.0 CONCLUSION

We introduced the *closure* of a course of action (COA) in the context of attack graphs and presented a polynomial time algorithm that leverages prior attack graph and metrics research to compute multi-action COA recommendations that maximally disrupt attackers, within a specified budget. Our experiments demonstrate that practical solutions can be found by the polynomial-time best first search greedy algorithm in less than a second for a representative single site corporate network. We generated thousands of network configurations with the results showing that computing COAs for enterprise networks can be done in a reasonable time period, usually with optimal solutions. Our algorithm makes effective recommendations for improving security even when practical considerations prevent the network from being completely secured.

It is possible to use the greedy and optimal algorithms cooperatively in order to achieve better results. The greedy algorithm, because it removes the single most effective SINK vertex in each iteration, can sometimes expend part of the budget removing a vertex that will itself be indirectly removed in a later iteration. This results in a COA set that is a superset of an equivalently good solution. (For example, comparing the Budget $= 4000$ results in Tables 2 and 3 shows that the greedy algorithm includes vertex 105, which is unnecessary because it is contained in the closure of the remaining vertices.) Such supersets can be compressed by applying the optimal algorithm to them. These sets are usually small so the optimal algorithm can execute quickly, and when it succeeds in reducing the COA set (and its associated cost) the greedy algorithm can resume searching with a larger remaining budget. This optimization can be computed very quickly and we recommend the hybrid approach.

Our approach is flexible due to the use of rank and cost weights. Rank weights may reflect whatever the defender wants to deny the attacker, and cost weights may reflect whatever the defender wishes to change. For example, in place of the expressive rank and loss function cost values we presented, the defender could uniformly rank the network services and set all other vertex ranks to zero. The defender could assign a cost of 1 to each vulnerability vertex and set the remaining SINK vertices to an infinite cost. The computed COA would then maximally deny the attacker access to network services by patching as few vulnerabilities as possible. The flexibility and efficiency of our approach should be very useful in practice.

Our technique can be applied in both proactive and reactive scenarios. The difference between the two situations is the range of actions, and the cost of implementing the actions. Our future work in progress includes integrating the algorithms into an automated computer network defence system.

REFERENCES

[1] Swiler, L., Phillips, C., Ellis, D., and Chakerian, S., "Computer-Attack graph generation tool," *DARPA Information Survivability Conference and Exposition, Vol.2*, 2001.

[2] Ingols, K., Lippmann, R., and Piwowarski, K., "Practical Attack Graph Generation for Network Defense," *Proceedings of the 22nd Annual Computer Security Applications Conference (ACSAC)*, December 2006, pp. 121–130.

[3] Ammann, P., Wijesekera, D., and Kaushik, S., "Scalable, graph-based network vulnerability analysis," *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002.

[4] Ou, X., Boyer, W., and McQueen, M., "A scalable approach to attack graph generation," *13th ACM Conference on Computer and Communications Security (CCS)*, 2006, pp. 336–345.

[5] Noel, S., Jajodia, S., O'Berry, B., and Jacobs, M., "Efficient minimum-cost network hardening via exploit dependency graphs," *19th Annual Computer Security Applications Conference (ACSAC)*, 2003.

[6] Sheyner, O. and Wing, J., "Tools for generating and analyzing attack graphs," *LECTURE NOTES IN COMPUTER SCIENCE*, Jan 2004.

[7] Jajodia, S., Noel, S., and O'Berry, B., "Topological analysis of network attack vulnerability," *Managing Cyber Threats: Issues, Approaches and Challenges*, 2005, pp. 248–266.

[8] Templeton, S. and Levitt, K., "A requires/provides model for computer attacks," *Proceedings of the 2000 workshop on New security paradigms*, Jan 2001, pp. 31–38.

[9] Locasto, M. E., Burnside, M., and Bethea, D., "Pushing Boulders Uphill: The Difficulty of Network Intrusion Recovery," *The 23rd Large Installation System Administration Conference (LISA 2009)*, Feb 2009, pp. 1–13.

[10] Ou, X., Govindavajhala, S., and Appel, A. W., "MulVAL: A logic-based network security analyzer," *14th USENIX Security Symposium*, Jan 2005.

[11] Sawilla, R. E. and Ou, X., "Identifying Critical Attack Assets in Dependency Attack Graphs," *Proceedings of the 13th European Symposium on Research in Computer Security*, Vol. 13, 2008, pp. 18–34.

[12] Sheyner, O., Haines, J., Jha, S., Lippmann, R., and Wing, J., "Automated generation and analysis of attack graphs," *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, 2002, pp. 254–265.

[13] Ceri, S., Gottlob, G., and Tanca, L., "What you always wanted to know about Datalog (and never dared to ask)," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 1, No. 1, 1989, pp. 146–166.

[14] Moore, A., Ellison, R., and Linger, R., "Attack modeling for information security and survivability," *Technical Note CMU/SEI-2001-TN-001*, Jan 2001.

[15] Gallo, G., Longo, G., Pallottino, S., and Nguyen, S., "Directed hypergraphs and applications," *Discrete Applied Mathematics*, Vol. 42, No. 2-3, 1993, pp. 177–201.

[16] Hazewinkel, M., editor, *Encyclopaedia of Mathematics*, No. ISBN 1-4020-0609-8, Springer-Verlag, New York, 2002, `http://eom.springer.de/`.

[17] Wang, L., Noel, S., and Jajodia, S., "Minimum-cost network hardening using attack graphs," *Computer Communications*, Vol. 29, No. 18, November 2006.

[18] Homer, J., Manhattan, K., Ou, X., and Schmidt, D., "A Sound and Practical Approach to Quantifying Security Risk in Enterprise Networks," *people.cis.ksu.edu*, 2009.

[19] Karp, R., "Reducibility among combinatorial problems," *Complexity of computer computations*, Jan 1972.

[20] Hagberg, A. A., Schult, D. A., and Swart, P. J., "Exploring network structure, dynamics, and function using NetworkX," 2008, pp. 11–15.

[21] Scarfone, K., Jansen, W., and Tracy, M., "Guide to General Server Security: Recommendations of the National Institute of Standards and Technology," `http://csrc.nist.gov/publications/nistpubs/800-123/SP800-123.pdf`, 2008.

[22] Broder, A., Kumar, R., Maghoul, F., and Raghavan, P., "Graph structure in the web," *Computer Networks*, Jan 2000.

[23] Faloutsos, M., Faloutsos, P., and Faloutsos, C., "Power laws and the AS-level internet topology," *IEEE/ACM Transactions . . .*, Jan 2003.

[24] Bollobás, B., Borgs, C., Chayes, J., and Riordan, O., "Directed scale-free graphs," *Proceedings of the fourteenth annual ACM-SIAM . . .*, Jan 2003.

[25] Gordon, L. A. and Loeb, M. P., "The Economics of Information Security Investment," *ACM Transactions on Information and System Security*, Nov 2002, pp. 438–457.

[26] Mercuri, R., "Analyzing Security Costs," *Communications of the ACM*, Vol. 46, No. 6, June 2003.

[27] Liu, P., Zang, W., and Yu, M., "Incentive-Based Modeling and Inference of Attacker Intent, Objectives, and Strategies," *ACM Transactions on Information and System Security*, Vol. 8, No. 1, February 2005.

[28] Cavusoglu, H., Mishra, B., and Raghunathan, S., "A model for evaluating IT security investments," *Communications of the ACM*, Vol. 47, No. 7, July 2004.

[29] Bistarelli, S., Fioravanti, F., and Peretti, P., "Defense trees for economic evaluation of security investments," *Proceedings of the First International Conference on Availability, Reliability and Security*, 2006, pp. 416–423.

[30] Cremonini, M. and Martini, P., "Evaluating Information Security Investments from Attackers Perspective: the Return-on-Attack (ROA)," *Fourth Workshop on the Economics of Information Security*, 2005.

[31] Wang, L., Islam, T., Long, T., Singhal, A., and Jajodia, S., "An Attack Graph-Based Probabilistic Security Metric," *Proceedings of the 22nd Annual Conference on Data and Applications Security*, 2008, pp. 283–296.

[32] Jha, S., Sheyner, O., and Wing, J., "Two formal analyses of attack graphs," *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, June 2002, pp. 49–63.

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

[34] Dewri, R., Poolsappasit, N., Ray, I., and Whitley, D., "Optimal security hardening using multi-objective optimization on attack tree models of networks," *14th ACM Conference on Computer and Communications Security (CCM)*, 2007.

[35] Homer, J. and Ou, X., "SAT-Solving Approaches to Context-Aware Enterprise Network Security Management," *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS*, Vol. 27, No. 3, 2009, pp. 315.