Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

Course of action recommendations for practical network defence

Reginald Sawilla and Craig Burrell

Defence R&D Canada – Ottawa

Technical Memorandum
DRDC Ottawa TM 2009-130
August 2009

Canada

Course of action recommendations for practical network defence

Reginald Sawilla
Defence R&D Canada – Ottawa

Craig Burrell
Defence R&D Canada – Toronto

Defence R&D Canada – Ottawa
Technical Memorandum
DRDC Ottawa TM 2009-130
August 2009

Principal Author

Original signed by Reginald Sawilla

Reginald Sawilla

Approved by

Original signed by Julie Lefebvre

Julie Lefebvre
Head/NIO Section

Approved for release by

Original signed by Brian Eatock

Brian Eatock
Head/Document Review Panel

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 as effective as the optimal choices computed by an exponential algorithm, making our solution especially useful in practice. We demonstrate the value of our solution on a complex cyclic attack graph generated from a representative business network.

Résumé

Grâce aux avancées récentes dans la construction et l'analyse des graphiques d'attaque, on peut maintenant compter sur de nouveaux outils pour protéger les réseaux. Cependant, l'amélioration de la sécurité des réseaux demeure une tâche incroyablement complexe. Avec la multiplication des vulnérabilités, le perfectionnement des outils qu'utilisent les attaquants et la dépendance grandissante des organisations sur l'infrastructure réseau, les outils d'automatisation et de recommandation sont essentiels. Jusqu'ici, les recherches portant sur les plans d'action à recommander présumaient qu'il était possible de sécuriser un réseau parfaitement. En fait, les administrateurs de réseau cherchent à établir un équilibre entre la sécurité du réseau et sa convivialité. Ils acceptent donc que certaines failles soient présentes et que la sécurité du réseau ne soit pas parfaite. Dans ce document, nous présentons des algorithmes de recommandation de plans d'action qui font des calculs afin de proposer des solutions efficientes et efficaces qui augmentent la sécurité des réseaux malgré certaines contraintes concrètes, dont la disponibilité des correctifs, le coût des ressources et les coûts d'utilisation. Notre solution s'inspire de la recherche de mesures visant à proposer des plans d'action qui perturbent le plus possible la capacité d'un attaquant à atteindre les cibles névralgiques choisies par l'administrateur. L'algorithme de temps polynomial fait des choix voraces afin de produire des plans d'action qui sont presque aussi efficaces que les choix optimaux calculés par un algorithme exponentiel, ce qui fait en sorte que notre solution est particulièrement utile dans la pratique. Nous illustrons la valeur de notre solution

dans un graphique complexe décrivant des attaques cycliques dans un réseau représentatif des activités de l’organisation.

Executive summary

Course of action recommendations for practical network defence

Reginald Sawilla, Craig Burrell; DRDC Ottawa TM 2009-130; Defence R&D Canada – Ottawa; August 2009.

Background: 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 varying 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.

Principal results: In this paper we present a method for generating course-of-action (COA) recommendations based on analysis of attack graphs. 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 devices. We present both a polynomial complexity greedy algorithm that estimates the best solution and a (worst-case) exponential complexity algorithm that produces the optimal solution. Both algorithms are parallelizable.

Significance of results: Our contributions in this paper are polynomial time and exponential time algorithms that leverage 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 greedy algorithm in less than a second for a representative single site corporate network; we expect to be able to compute COAs for large enterprise networks in a reasonable time period (for example, once per day). Our algorithms make effective recommendations for improving security even when practical considerations prevent the network from being completely secured.

Future work: Both the greedy and optimal algorithms we presented are parallelizable. For enterprise computer networks and large critical infrastructure networks, parallelization will enable fast computation using a hybrid of our polynomial and exponential time algorithms to compute optimal courses of action. Our future work in progress includes integrating the algorithms into an automated computer network defence system.

Sommaire

Course of action recommendations for practical network defence

Reginald Sawilla, Craig Burrell ; DRDC Ottawa TM 2009-130 ; R & D pour la défense Canada – Ottawa ; août 2009.

Contexte : Les réseaux modernes sont extrêmement complexes. Le travail visant à assurer leur sécurité l'est tout autant. Les administrateurs de réseau doivent protéger des biens à priorité variable contre des attaques dont la complexité et le degré de perfectionnement sont variables. De plus, ils disposent de ressources restreintes et sont confrontés à la difficile tâche de trouver l'équilibre entre la sécurité, la convivialité et les ressources. Dans ce document, nous présentons une solution qui aide les administrateurs de réseau à prendre les meilleures décisions en matière de sécurité, c'est-à-dire des solutions qui optimisent l'utilisation des ressources et qui protègent leur réseau en respectant le budget établi.

Principaux résultats : Dans ce document, nous présentons une méthode qui génère des recommandations de plans d'action en fonction de l'analyse des graphiques d'attaque. Nous utilisons en combinaison nos connaissances de la structure logique des attaques de réseau, des données de configuration, des attributs que présentent les failles, du degré de perfectionnement du programme d'exploitation des failles, de la priorité de l'appareil, des coûts du redressement et des contraintes budgétaires. À partir d'un graphique d'attaque qui décrit les possibilités d'attaque contre un groupe précis d'appareils dans le réseau, notre méthode empêche l'attaquant d'atteindre les dispositifs névralgiques du réseau. Nous présentons un algorithme vorace à complexité polynomiale qui trouve la meilleure solution, ainsi qu'un algorithme (du scénario du pire) à complexité exponentielle qui trouve la solution optimale. Les deux algorithmes peuvent être utilisés en parallèle.

Importance des résultats : Nos contributions à ce document sont les algorithmes de temps polynomial et de temps exponentiel qui se basent sur le graphique des attaques antérieures et la recherche de méthodes pour faire des calculs et proposer des recommandations de plans d'action multiples qui perturbent au maximum les attaquants tout en respectant le budget établi. Nos expériences démontrent que l'algorithme vorace à temps polynomial arrive à trouver des solutions pratiques en moins d'une seconde pour un réseau organisationnel représentatif à site unique. Nous nous attendons à pouvoir calculer des plans d'action pour les grands réseaux d'entreprise dans un délai raisonnable (une fois par jour, par exemple). Nos algorithmes proposent des recommandations efficaces pour améliorer la sécurité, et ce, même lorsque des considérations pratiques empêchent le réseau d'être entièrement sécurisé.

Travaux futurs : Les algorithmes vorace et optimal que nous avons présentés peuvent être utilisés en parallèle. Dans le cas des réseaux d'entreprise et des grands réseaux de l'infrastructure essentielle, les calculs se feront rapidement grâce à la parallélisation, c'est-à-dire l'utilisation combinée des algorithmes de temps polynomial et exponentiel pour trouver les plans d'action optimaux. Dans le cadre de nos travaux futurs, nous souhaitons intégrer les algorithmes à un système automatisé de protection des réseaux informatiques.

This page intentionally left blank.

Table of contents

List of figures

List of tables

List of algorithms

1 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 varying 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, 2, 3, 4, 5, 6, 7, 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. The most promising solutions describe complex network attacks by the use of *dependency attack graphs*. Dependency attack graphs illustrate how the low-level details of network configuration can be chained together by an attacker to compromise network assets. Additionally, attack graphs allow problems to be solved through the use of graph theory.

Since all of the information is contained in the graph, a network administrator could, in theory, use an attack graph to decide on an ideal course of action to improve the security of the network. However, in practice, attack graphs are locally easy to interpret but are globally overwhelmingly complex. Hence, the network defender can be inundated with information and at a loss as to how best to proceed. Furthermore, zero-day vulnerabilities, resource limitations, and operational constraints make it impossible to completely secure a network. Network defenders require a solution that maximally (but not necessarily completely) improves security within real-world constraints.

In this paper we present a method for generating course-of-action (COA) recommendations based on analysis of attack graphs. 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 devices. We present both a polynomial complexity greedy algorithm that estimates the best solution and a (worst-case) exponential complexity algorithm that produces the optimal solution. Both algorithms are parallelizable.

2 Background

A network attack graph is a mathematical abstraction, consisting of vertices and arcs, 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, 9] 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, 3, 4, 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 [11] 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 type 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. A network defender may examine the graph by beginning at an attacker goal (*e.g.*: the ability to execute arbitrary code on a particular server) and follow its out-neighbours to see which combinations of SINK vertices enable the goal. 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 an F-hypergraph [12].

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. An additional layer of analysis is required to examine the attack graph and make intelligent COA recommendations to support the network administrator.

To be useful in practice, COA recommendations should take into account various factors. First, some network configuration changes are more costly than others. The concept of *cost* is abstract in our analysis: it could refer to financial cost, time investment cost, or usability cost (insofar as certain changes will disrupt network function and impair the organization's business activity). In our model, the user can dynamically 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

1. `http://nvd.nist.gov/cvss.cfm`

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. Fourth, some network assets are more valuable to the defender than other assets, and their protection should be prioritized accordingly.

We model costs by assigning a weight, in generic cost units, to every SINK vertex in the attack graph. This weight represents the cost, in terms of time, money, and usability, incurred by changing the network configuration fact represented by the SINK vertex. We model the second factor by introducing a defensive budget which represents the network defender's resources, in the same generic cost units, for securing the network. We model the third factor by assigning a rank weight using AssetRank [13] to every vertex in the attack graph. All rank weights sum to 1 and the ranks reflect the value of the graph vertex to the attacker. The fourth factor, the selection of critical attack assets from the defender's point of view, is provided as input to the AssetRank directed graph analysis algorithm.

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 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.

Given a dually weighted dependency attack graph with an AssetRank weight assigned to each vertex and a removal cost weight assigned to each SINK vertex, we are in a position to consider how to generate optimal COA recommendations under budgetary constraints.

3 Related Work

In a number of previous papers, the authors have studied the problem of security metrics for IT networks. Some have advocated an economics-driven approach to the problem. Gordon and Loeb [14] 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 [15].

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 intent, objectives, and strategies [16]. Cavusoglu *et al.* model network traffic patterns, attacker intent, monitoring effectiveness, expected damage, and other factors [17]. 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) [18] and Return-On-Attack (ROA) [19]. 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.* [20] 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.* [21] 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.* [22] 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 [23]. 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 [24] combine MulVAL-generated attack graphs with usability requirements, and use Boolean Satisfiability Solving to find network configurations that provide secu-

rity 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.* [25], 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 produces recommendations similar to those from previous exponential approaches. However, we also present a polynomial complexity solution which gives acceptable recommendations and executes quickly.

4 Computing courses of action

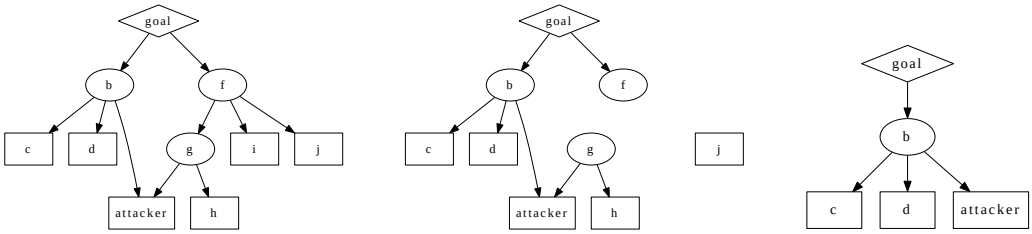
In general, we want to maximally prevent the attacker from reaching the goal vertices. We define two sets of vertices in the attack graph: the source set contains the goal vertices and the target set contains the attacker starting location(s). Our objective is then to maximally decrease the connectivity between the source set and the target set. AssetRank values reflect the global cohesion between these two sets. In order to set AssetRank values, the source set is assigned defender priorities which are positive values summing to 1. The remaining vertices are assigned the value 0 which signifies that the defender is interested in them only inasmuch as they allow the attacker to reach the defender priority vertices. Hence, AssetRank models random walkers which start at the defender priority vertices and walk the graph, choosing out-going arcs with probabilities proportional to the arc weights. The arc weights correspond to attacker preferences which are computed from the likelihood that the attacker can successfully exploit the vertex, among other factors. For full details, see Sawilla and Ou [13].

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.* [25] 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 [24, 25] 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 improvement in the security of the network). However, the problem of determining which privileges are not available to the attacker is more complicated than simply summing the invalid vertices.

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. A disconnected subgraph corresponds to vertices which become useless to the attacker in reaching the goal vertices and is an example of both unreachability cases.



(a) Example attack graph

(b) Figure 1(a) attack graph with vertex f 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.*

As an example, consider the attack graph shown in Figure 1(a). If vertex i is deleted, then the compound arc[2] incident to it is also removed, resulting in the graph shown in Figure 1(b). Vertex j is still present but it 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

2. 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.

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 COA set vertices from the graph can, because it signifies changes to the network configuration, affect legitimate network users, whereas removing the closure of the COA set only affects attackers.

The full details of the closure algorithm are presented in Algorithm 1 (VertexSetClosure). Briefly, the algorithm computes the closure of a set of vertices by:

Line 2 Removing the vertices from the graph

Line 3 Computing the AND and OR vertices unable to reach the attacker (the target set of the directed graph). This is efficiently computed by finding the vertices reachable from the target set in the graph transpose. The graph transpose is the graph with all arcs reversed, and is denoted G^T for a graph G.

Line 5 Computing the vertices unreachable from the goals (the source set of the directed graph).

Computing the vertex set closure is extremely fast and uncomplicated (see Section 5). 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 and an efficient way to measure partial security improvements.

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

Input: AND/OR directed graph $G = (V, A)$, set of vertices C whose closure is to be computed, source vertex set S that vertices must be reachable from, target 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** $G, \overline{C}$

The next task is computing the least cost set of vertices to include in the COA set that removes the maximum rank from the graph. Each SINK vertex has an associated remediation cost, so the remediation 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 [3] as the sum of the ranks of its members. We aim to optimally choose a COA set, where the optimization criteria are: first, maximization of the COA set closure rank; and second, minimization of the COA set remediation cost.

We select the final COA set using two different methods. The greedy algorithm computes the COA set closure rank for each SINK vertex. The rank sum is divided by the cost of the vertex to obtain a return on investment (ROI). 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 remaining SINK vertices whose cost is within the available budget. The full details are presented in Algorithm 2 (GreedyCOA).

The optimal solution, Algorithm 3, is computed by brute force by considering all combinations of SINK vertices that are within budget. Since Algorithm 3 has exponential complexity (see Section 5), it is not practical for large graphs. However, we take advantage of it for small problems, in particular to improve the results obtained by Algorithm 2 (GreedyCOA). Algorithm 2 sometimes computes a non-optimal result where a subset of the returned result would obtain the same rank removal, but at a lower cost. For example, if the COA is found in the order a, b, c, and if $a \notin \overline{b}, \overline{c}$ (where overlines denote closure), then $\{a, b, c\}$ could be a sub-optimal set. This will be the case if $a \in \overline{\{b, c\}}$. Even if non-optimal, we can use the COA set found by the greedy algorithm to limit the number of possible COA sets considered by the brute force algorithm. Although the details are omitted due to a lack of space, we created an additional hybrid algorithm that calls Algorithm 3 each time a new COA is added to the COA set in Algorithm 2. The OptimumCOA algorithm only considers the vertices in the COA set (rather than all SINK vertices) and removes the redundant vertices (such as a in the previous example).

5 Complexity

As mentioned in Section 4, the complexity of computing the logic clauses of each vertex is exponential. 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 [26]. The implementation was modified slightly to allow a set of vertices to be passed to the function and so compute the reachability of all vertices in the set 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

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

Algorithm 2 GreedyCOA(G,S,T,Budget) — Compute greedy course of action

Input: AND/OR directed graph $G = (V, A)$, source vertex set S, target vertex set T, Budget.

Output: Graph $G' = (V', A')$ with $\overline{COA}$ deleted, COA, $\overline{COA}$ (closure), TotalCost, TotalRank.

1: TotalCost $\leftarrow 0$, TotalRank $\leftarrow 0$, COA $\leftarrow \emptyset$, $\overline{COA} \leftarrow \emptyset$, SinkClosureComputed $\leftarrow$ **false**, $G' \leftarrow G$
2: Sinks $\leftarrow \{v | v \in G'_V, Type(v) = \text{SINK}, Cost(v) \leq \text{Budget}\}$
3: **while** Sinks $<>$ $\emptyset$ **do**
4: $MaxROI \leftarrow -\infty, m \leftarrow 0$
5: **for** $v_i \in$ Sinks **do**
6: **if** SinkClosureComputed **then** {Ensure vertex is within budget, adjust cost}
7: COAInClosure$_i \leftarrow$ COA $\cap \overline{v_i}$
8: VertexCost $\leftarrow$ VertexCost $- \sum_{u \in \overline{v_i}}$ Cost(u)
9: **if** VertexCost + TotalCost $>$ Budget **then**
10: Sinks $\leftarrow$ Sinks $-\{v_i\}$
11: **next**
12: $(G_i, \overline{COA_i}) \leftarrow VertexSetClosure(G', \{v_i\}, S, T)$
13: $COARank_i \leftarrow \sum_{u \in \overline{COA_i}} Rank(u)$
14: $ROI_i \leftarrow COARank_i$/Cost$(v_i)$
15: **if** $ROI_i > MaxROI$ **then**
16: $m \leftarrow i$
17: **if** $m = 0$ **then** {No COA found within budget}
18: Sinks $\leftarrow \emptyset$
19: **else**
20: TotalRank $\leftarrow$ TotalRank $+COARank_m$
21: TotalCost $\leftarrow$ TotalCost + Cost(v_m)
22: **if** SinkClosureComputed **then**
23: $COA \leftarrow COA-$ COAInClosure$_m$
24: $COA \leftarrow COA \cup \{v_m\}$
25: $\overline{COA} \leftarrow \overline{COA} \cup \overline{COA_m}$
26: $G' \leftarrow G_m$
27: Sinks $\leftarrow$ Sinks $- \overline{COA_m}$
28: SinkClosureComputed $\leftarrow$ **true**
29: **return** G', COA, $\overline{COA}$, TotalCost, TotalRank

Algorithm 3 OptimumCOA(G,S,T,Budget) — Compute optimum course of action

Input: AND/OR directed graph $G = (V, A)$, source vertex set S, target 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

no loops in the algorithm, the complexity is $\mathcal{O}(v + a)$.

Algorithm 1 (VertexSetClosure) is called by both Algorithm 2 (GreedyCOA) and Algorithm 3 (OptimumCOA). In Algorithm 2 (GreedyCOA), it is the highest time complexity 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.

Algorithm 3 (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. What remains are all the candidate courses of action whose total cost is within budget. The size of this

set can be much smaller than 2^s. If every SINK has a cost of 1 and the budget is b, then the number of candidate courses of action is given by $\sum_{k=0}^{b} C_k^s$ where $C_k^s = s!/(k!(s-k)!)$. As the attack graph grows in size, the discriminator of a limited budget significantly reduces the candidate space. We have implemented an efficient dynamic programming method of generating the candidate courses of action that generates only the candidate sets within budget, but we omit the details.

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 $\sum_{k=0}^{b} C_k^s$ iterations when a limited budget is specified. Specifying a budget is usually practical because Algorithm 2 (Greedy-COA) will compute a solution in polynomial time, thus giving a budget upperbound.

6 Experiments

We illustrate the value of our method using two examples.

6.1 Example 1:

The network in Figure 2 has three hosts plus the attacker's PC. Each computer has a remotely exploitable vulnerability. The likelihood of successfully exploiting each vulnerability differs due to the type of vulnerability and the attack tools available. The likelihoods are: 80% – web server; 40% – mail server; and 100% – 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.

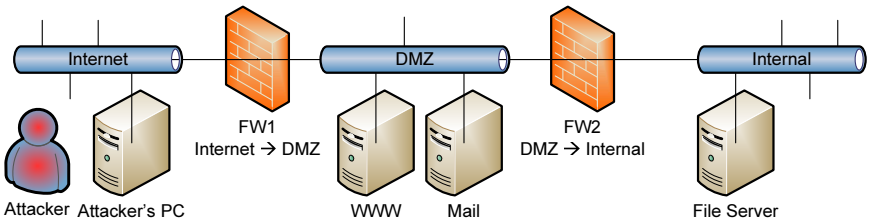


Figure 2: *Example 1 network: The attacker does not have direct access to the file server but can reach it by a multistep attack through the web server or mail server.*

Figure 3 is the attack graph containing the possible attacks for this scenario. As described earlier, 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,serviceaccount)`[4] is derived

4. `execCode(fs,serviceaccount)` states that the attacker can execute arbitrary code on the file server (fs) at the privilege of serviceaccount.

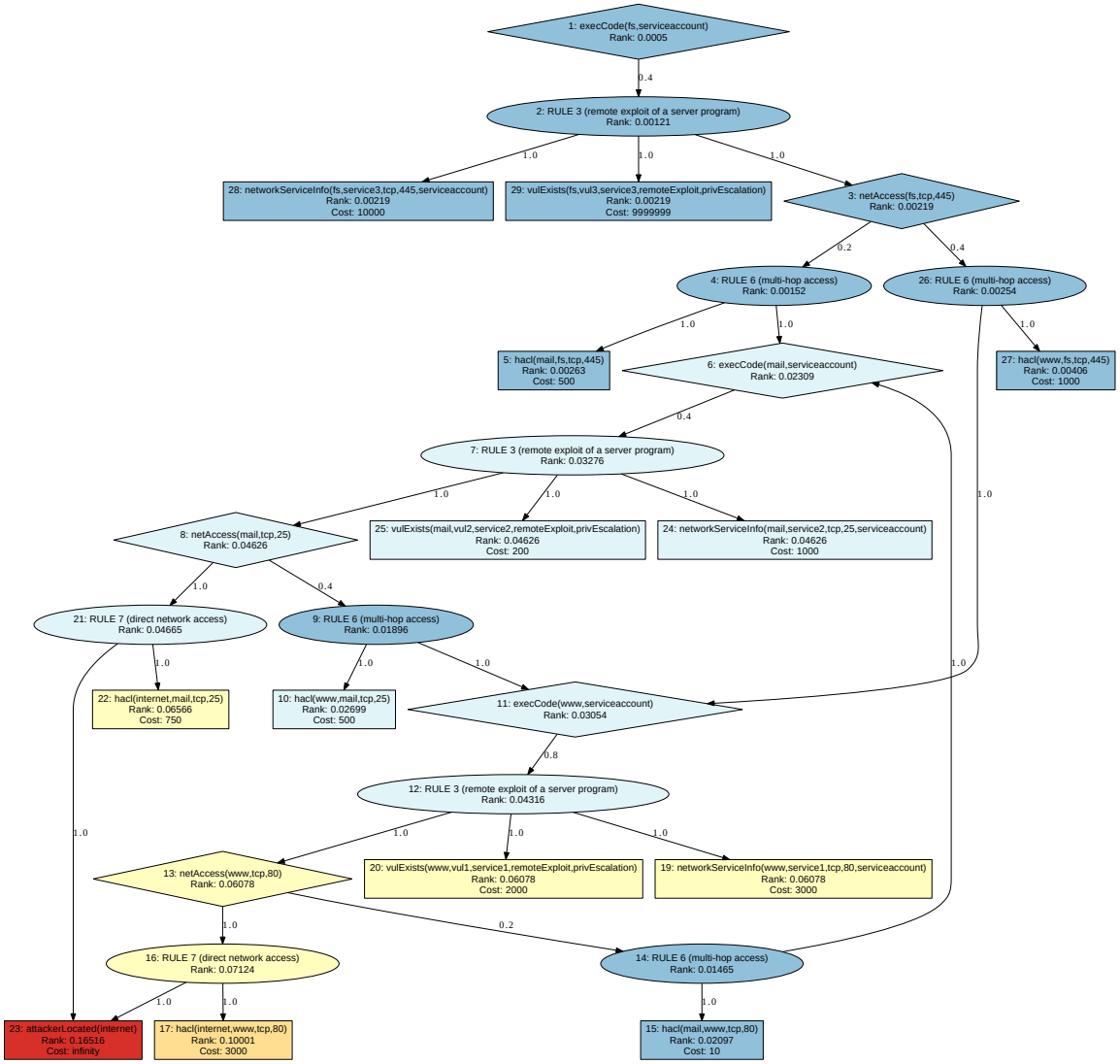


Figure 3: *Attack graph for Example 1 network: The graph gives a proof tree for how the attacker's ability to obtain code execution privileges on the file server is derived from other facts.*

from other facts. The vertices are colored to reflect their AssetRank values. The vertices with the highest rank are red and the vertices with the lowest rank are blue. In this graph, vertices close to the attacker tend to have larger rank values because they enable multiple attack paths, whereas vertices closer to the top goal vertex tend to be associated with a single attack vector.

Remediation costs are assigned to the SINK vertices. We assume that no patch exists for the vulnerability on the file server (vertex 29) and that the file sharing service (vertex 28) is critical to operations so its availability is the highest priority. We also assume that the

vulnerability on the mail server (vertex 25) has a well-tested patch available but the web server vulnerability (vertex 20) only has a workaround available. Table 1 gives the assigned vertex costs which reflect these assumptions.

Table 1: *Vertex costs for Figure 3 graph*

Vertex #	Description	Remediation Cost (operational & physical)
28	file sharing service	10000
19	www service	3000
24	mail service	1000
17	route from internet to www	3000
22	route from internet to mail	1000
10	route from www to mail	500
27	route from www to file server	500 (1000)
5	route from mail to file server	500
15	route from mail to www	10 (link is not necessary)
29	patch file server vulnerability	∞ (no patch available)
20	patch www server vulnerability	2000
25	patch mail server vulnerability	200

We generate COA recommendations for this attack graph using both Algorithms 2 (Greedy-COA) and 3 (OptimumCOA). With a budget of 500, both algorithms recommend patching the vulnerability on the mail server (vertex 25), which has a cost of 200 and eliminates 34% of the rank in the graph (see Appendix A). This course of action removes the portion of the attack graph that goes through the mail server, but leaves open an attack through the web server. No further advantageous actions can be taken without exceeding the budget.

When the budget is raised to 1000 the remediation costs of several other vertices come within range. Again the greedy algorithm finds the optimal solution: patch the mail server (vertex 25) and block the network route between the web server and the file server (vertex 27), for a total cost of 700. This course of action eliminates the entire attack graph, completely securing the file server against attack.

If the administrator decides that blocking the web server's access to the file server is too disruptive to business activities, he can raise the cost associated with that network route (to, for example, 1000). In that case the best course of action within budget (≤ 1000) is simply to patch the mail server (vertex 25). The file server is not completely secured, but given the practical constraints reflected in the budget and remediation costs, this is the maximum partial solution within budget.

6.2 Example 2:

The utility of our approach can be better appreciated by considering a more complex example. Consider the small business network shown in Figure 4. The network 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 workstations 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, and Elmer hosts a file server, source repository, and SQL database containing sensitive data.

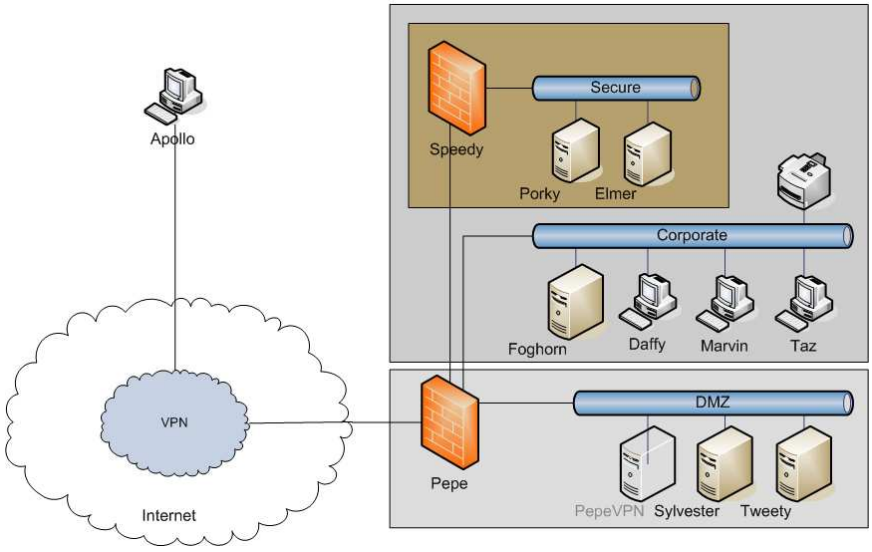


Figure 4: *Example 2 network. The attacker is located on the internet.*

All of the servers in the network have been hardened according to NIST security configuration guidelines [27]. Nonetheless, the network is vulnerable to attack because of the presence of several software vulnerabilities. The vulnerabilities present on the network are given in Table 2.

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 any of Sylvester, Tweety, Foghorn, or Elmer. The Web Application administrator account on Foghorn is also vulnerable, as is the administrator account on the VPN server Pepe. The SINK vertices in this graph are described in Appendix C.

Table 2: *Vulnerable services on the Example 2 network. Each vulnerability is remotely exploitable and leads to privilege escalation. The remediation cost for the Sendmail vulnerability is higher because we assume the patch is incompatible with customized scripts used on the network.*

Host	Vulnerable Service	Remediation Cost
Pepe	VPN	1600
Sylvester	Sendmail	5000
	DNS	800
Tweety	DNS (secondary)	800
Foghorn	Web Application	400
	Internal Mail	1600
Elmer	Database	800
	File Sharing	800

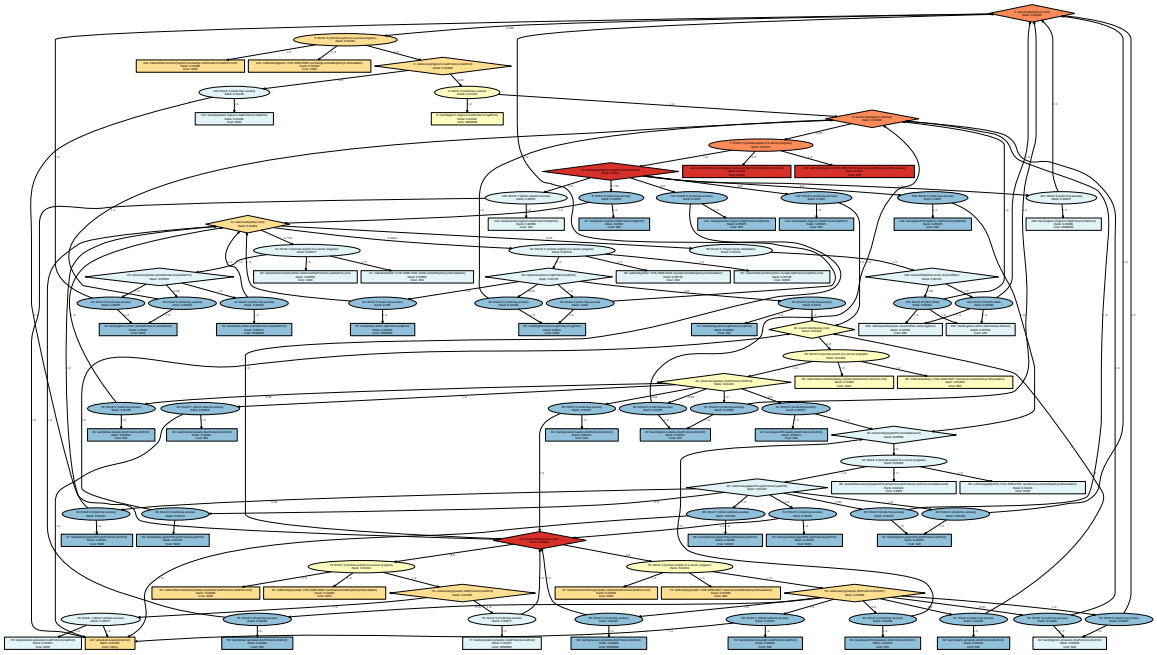


Figure 5: *Attack graph*[5] *for Example 2 network: As configured, an attacker beginning on the internet can obtain root privileges on Sylvester, Tweety, Foghorn, or Elmer.*

Remediation costs are assigned to the SINK vertices in a manner consistent with each fact's physical and operational costs. In practice, these costs would be assigned iteratively with the operator, as proposed in [24]. If an inappropriate COA is presented to the administrator, the administrator could instruct the system that the action has low, medium or high reme-

5. The figures in the electronic version of this document can be magnified in order to read the vertex contents.

diation cost. Costs could be assigned individually, per machine, or per type of fact (*i.e.*: vulnerability patch). In this example, we have assigned patch costs in a manner consistent with the maturity of the patch so that the cost reflects deployment, testing, and compatibility concerns. Network routes are assigned costs that are consistent with the value of the service with which they are associated. Most network services are of high value and should be shut down only in extreme circumstances; the VPN service and connectivity are essential and should be blocked only as a last resort. The costs assigned to the vertices in Appendix C (see also Table 2) reflect these considerations. We assume that the 2006 sendmail vulnerability on Sylvester (vertex 82) has a high cost because of customized scripts that are incompatible with the patch.

We study the greedy and optimal solutions for this attack graph at several different budget levels. Table 3 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 on a 2.4 GHz dual core MacBook Pro running Python 2.5.1.

Table 3: *Course-of-action recommendations by Algorithm 2 GreedyCOA. The CPU times are for a 2.4 GHz dual core MacBook Pro running Python 2.5.1.*

Budget	COA Cost	Rank Eliminated	COA Vertices	CPU Time (seconds)
1000	600	30.9%	105, 119, 51	0.222
2000	1800	48.9%	105, 42, 119, 73	0.367
3000	2900	62.4%	105, 98, 96, 119, 73	0.470
4000	3700	82.4%	105, 96, 119, 123, 73	0.514
5000	3700	82.4%	105, 96, 119, 123, 73	0.566
6000	3700	82.4%	105, 96, 119, 123, 73	0.636
7000	3700	82.4%	105, 96, 119, 123, 73	0.668
8000	7700	100%	79, 96, 119, 123, 105, 73	0.673

Table 4: *Course-of-action recommendations by Algorithm 3 OptimumCOA. The CPU times are for a 2.4 GHz dual core MacBook Pro running Python 2.5.1.*

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

In this example, the greedy algorithm does not converge to the optimal solution, but it does find good 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 the optimal algorithm make it impractical for real-world application, but the fast execution times and quality recommendations of the greedy algorithm make it an attractive alternative.

It is interesting to note that a large proportion of the graph rank can be removed by intelligent deletion of a relatively small set of SINK vertices. Consider the Budget $= 2000$ case as an example. The attack graph in Figure 5 has nearly 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 in Appendix C is 101,600, yet 82% of the rank can be eliminated with a cost of just 3,600 if the SINK removals are optimal.

The effectiveness of the greedy algorithm can be illustrated using the solution returned for the Budget $= 2000$ case. The recommended COA set is $\{105, 42, 119, 73\}$, shown in Appendix D. Consulting the vertex descriptions in Appendix C we see that this corresponds to patching the vulnerabilities in the Foghorn web application service (119) and the Sylvester DNS service (73), blocking DNS traffic from the Pepe firewall to the Tweety DNS server (42), and revoking Foghorn's ability to communicate with Elmer via NFS (105). The attack graph that results from removing this COA set and its closure is shown in Appendix E. This new graph is substantially simpler than the previous one, with fewer than half the original number of SINK vertices. Examination shows that each of Elmer, Foghorn, Tweety, and Sylvester is still vulnerable to attack, but the number of different attack vectors has decreased. Though it is not possible to completely secure the network with this budget, we have achieved a good, near-optimal solution for improving the network's security subject to budgetary constraints.

7 Conclusion

Our contributions in this paper are polynomial time and exponential time algorithms that leverage 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 greedy algorithm in less than a second for a representative single site corporate network; we expect to be able to compute COAs for large enterprise networks in a reasonable time period (for example, once per day). Our algorithms make 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 3 and 4 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 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.

Both the greedy and optimal algorithms we presented are parallelizable. For enterprise computer networks and large critical infrastructure networks, parallelization will enable fast computation using a hybrid of our polynomial and exponential time algorithms to compute optimal courses of action. 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. (2001), Computer-Attack graph generation tool, *DARPA Information Survivability Conference and Exposition, Vol.2*.

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

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

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

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

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

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

[8] Templeton, S. J. and Levitt, K. (2000), A requires/provides model for computer attacks, pp. 31–38.

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

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

[11] Ceri, S., Gottlob, G., and Tanca, L. (1989), What you always wanted to know about Datalog (and never dared to ask), *IEEE Transactions on Knowledge and Data Engineering*, 1(1), 146–166.

[12] Gallo, G., Longo, G., Pallottino, S., and Nguyen, S. (1993), Directed hypergraphs and applications, *Discrete Applied Mathematics*, 42(2-3), 177–201.

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

[14] Gordon, L. and Loeb, M. (2002), The Economics of Information Security Investment, *ACM Transactions on Information and System Security*, pp. 438–457.

[15] Mercuri, R. (2003), Analyzing Security Costs, *Communications of the ACM*, 46(6).

[16] Liu, P., Zang, W., and Yu, M. (2005), Incentive-Based Modeling and Inference of Attacker Intent, Objectives, and Strategies, *ACM Transactions on Information and System Security*, 8(1).

[17] Cavusoglu, H., Mishra, B., and Raghunathan, S. (2004), A model for evaluating IT security investments, *Communications of the ACM*, 47(7).

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

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

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

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

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

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

[24] Homer, J. and Ou, X., SAT-solving approaches to context-aware enterprise network security management, *IEEE JSAC Special Issue on Network Infrastructure Configuration*.

[25] Wang, L., Noel, S., and Jajodia, S. (2006), Minimum-cost network hardening using attack graphs, *Computer Communications*, 29(18).

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

[27] Scarfone, K., Jansen, W., and Tracy, M. (2008), 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.

This page intentionally left blank.

Annex A: Removal of vertex 25 in Example 1 network

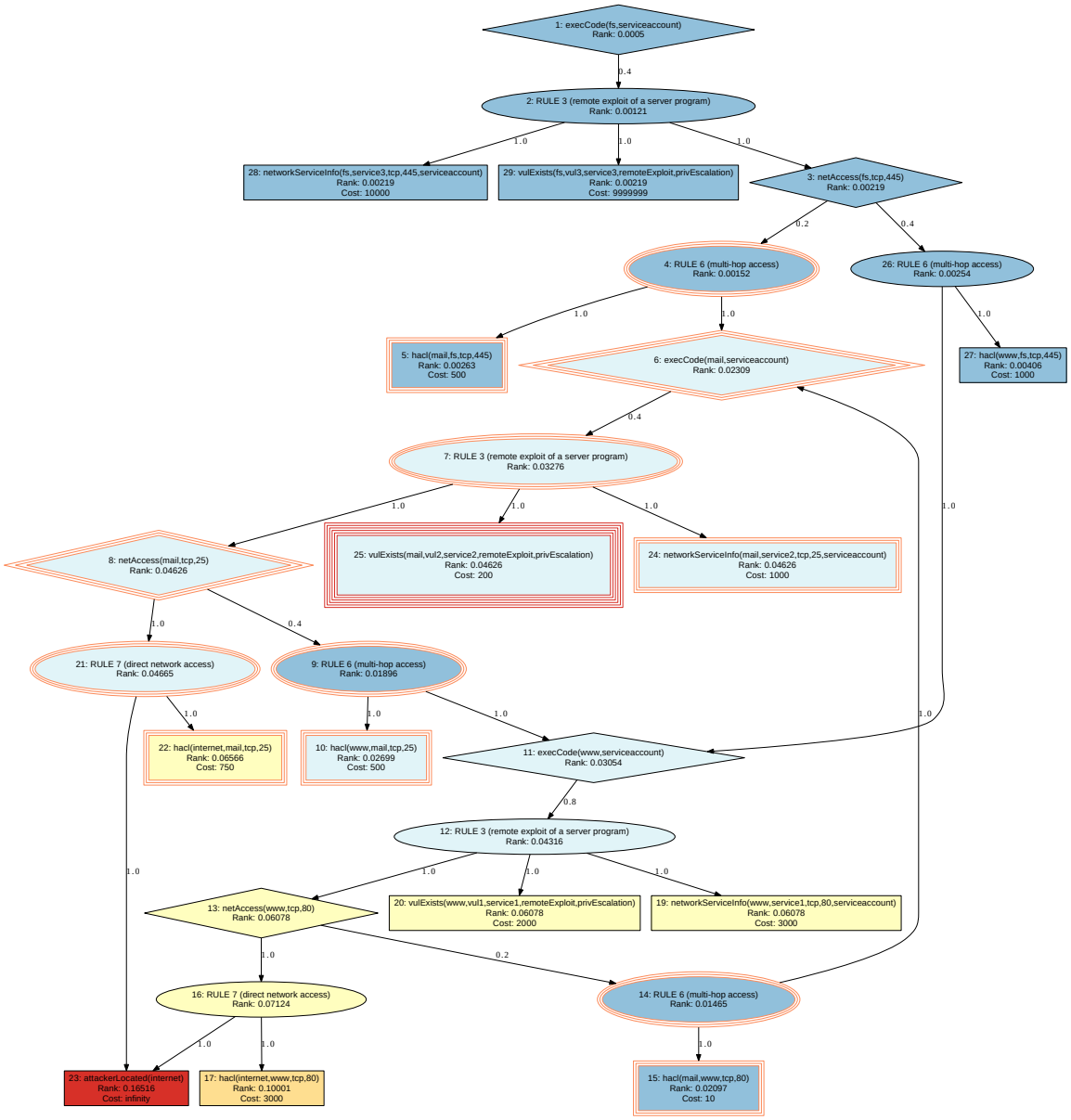


Figure A.1: *Example 1 Network: Effect of patching the mail server vulnerability (vertex 25 – red outline). When vertex 25 is removed from the graph, the closure of vertex 25 (orange outline) is also removed, indicating those vertices are no longer useful to the attacker. The closure of vertex 25 contains 34% of the graph rank.*

This page intentionally left blank.

Annex B: Removal of vertex 15 in Example 1 network

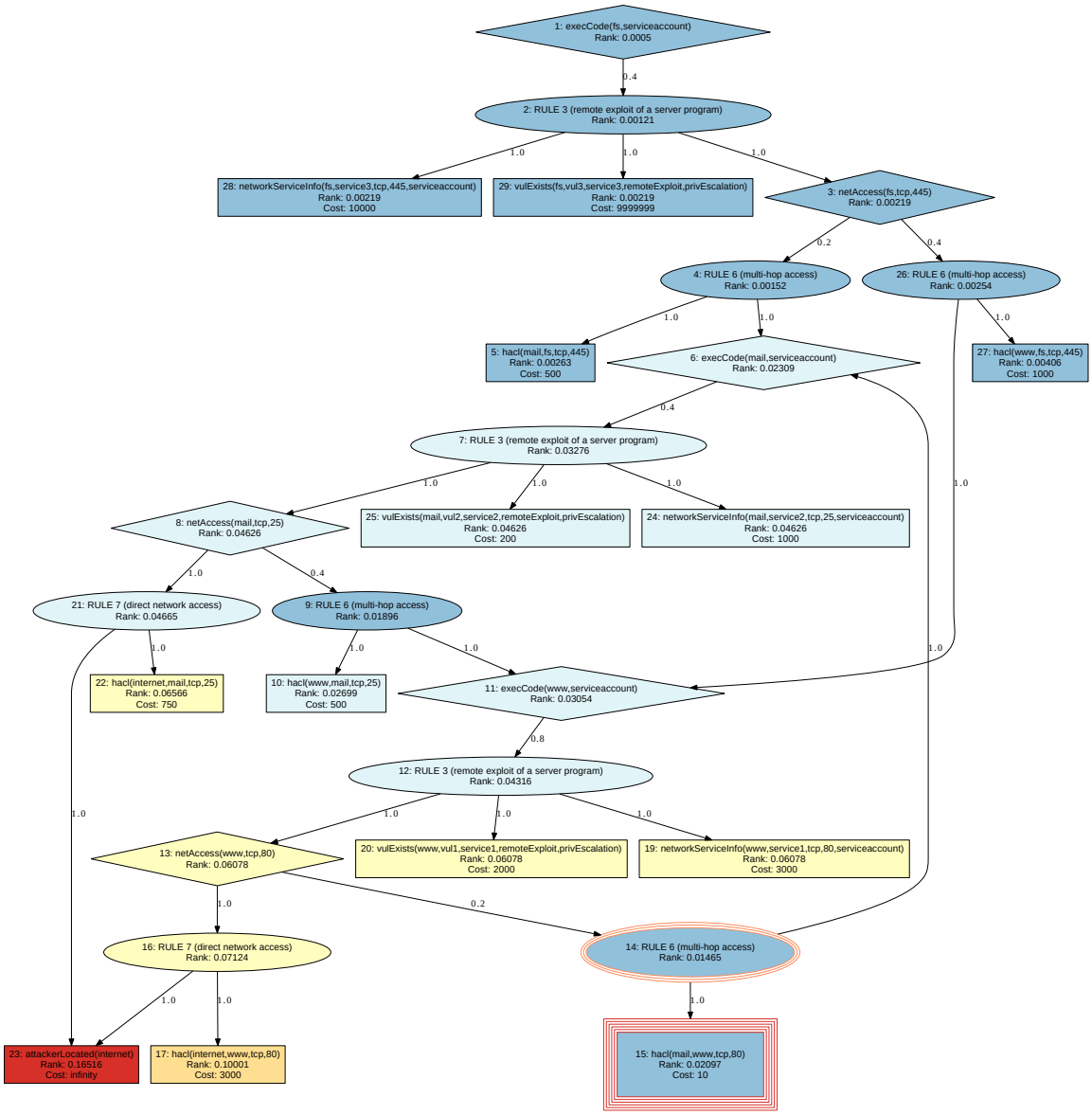


Figure B.1: *Example 1 Network: Effect of removing the connectivity between the mail server and web server (vertex 15 – red outline). When vertex 15 is removed from the graph, the closure of vertex 15 (orange outline) is also removed, indicating those vertices are no longer useful to the attacker. The closure of vertex 15 contains 3% of the graph rank.*

This page intentionally left blank.

Annex C: The Example 2 Network

Table C.1: *Vertex costs for the Figure 4 graph. The vertex descriptions are stated using MulVAL terminology:* `hacl` *(host access control list) vertices define network routes as a function of source, destination, protocol, and port;* `networkServiceInfo` *vertices define network services and the privileges under which they run;* `vulExists` *vertices describe vulnerability instances; and* `nfsExport` *vertices specify the configuration of a network file server. We assign an infinite remediation cost to self-referencing* `hacl` *vertices because self-communication is essential to functionality. Other costs are assigned according to the rationale outlined in Section 6.2.*

Vertex #	Description	Remediation Cost (operational & physical)
5	hacl(foghorn,foghorn,mailProtocol,mailPort)	∞
10	hacl(elmer,foghorn,httpProtocol,httpPort)	500
15	hacl(elmer,elmer,sambaProtocol,sambaPort)	∞
19	hacl(foghorn,elmer,sambaProtocol,sambaPort)	5000
20	networkServiceInfo(elmer,smbd,sambaProtocol,sambaPort,root)	1000
21	vulExists(elmer,'CVE-2008-1105',smbd,remoteExploit,privEscalation)	800
25	hacl(elmer,elmer,sqlProtocol,sqlPort)	∞
29	hacl(foghorn,elmer,sqlProtocol,sqlPort)	5000
31	hacl(tweety,elmer,sqlProtocol,sqlPort)	500
36	hacl(elmer,tweety,dnsProtocol,dnsPort)	500
40	hacl(foghorn,tweety,dnsProtocol,dnsPort)	500
42	hacl(pepeVPN,tweety,dnsProtocol,dnsPort)	500
47	hacl(elmer,pepeVPN,vpnProtocol,vpnPort)	2500
51	hacl(foghorn,pepeVPN,vpnProtocol,vpnPort)	100
53	hacl(sylvester,pepeVPN,vpnProtocol,vpnPort)	5000
58	hacl(elmer,sylvester,dnsProtocol,dnsPort)	500
62	hacl(foghorn,sylvester,dnsProtocol,dnsPort)	500
64	hacl(pepeVPN,sylvester,dnsProtocol,dnsPort)	500
66	hacl(sylvester,sylvester,dnsProtocol,dnsPort)	∞
68	hacl(tweety,sylvester,dnsProtocol,dnsPort)	500
70	hacl(internet,sylvester,dnsProtocol,dnsPort)	500
72	networkServiceInfo(sylvester,named,dnsProtocol,dnsPort,root)	2000
73	vulExists(sylvester,'CVE-2008-0007',named,remoteExploit,privEscalation)	800
77	hacl(sylvester,sylvester,mailProtocol,mailPort)	∞
79	hacl(internet,sylvester,mailProtocol,mailPort)	4000
81	networkServiceInfo(sylvester,sendmail,mailProtocol,mailPort,root)	4000
82	vulExists(sylvester,'CVE-2006-0058',sendmail,remoteExploit,privEscalation)	5000
84	hacl(tweety,pepeVPN,vpnProtocol,vpnPort)	5000
86	hacl(internet,pepeVPN,vpnProtocol,vpnPort)	10000
88	networkServiceInfo(pepeVPN,vpnService,vpnProtocol,vpnPort,normalAccount)	10000
89	vulExists(pepeVPN,'CVE-2009-1155',vpnService,remoteExploit,privEscalation)	1600
91	hacl(sylvester,tweety,dnsProtocol,dnsPort)	500
93	hacl(internet,tweety,dnsProtocol,dnsPort)	500
95	networkServiceInfo(tweety,named,dnsProtocol,dnsPort,root)	2000
96	vulExists(tweety,'CVE-2008-0007',named,remoteExploit,privEscalation)	800
97	networkServiceInfo(elmer,mysqld,sqlProtocol,sqlPort,root)	10000
98	vulExists(elmer,'CVE-2008-0226',mysqld,remoteExploit,privEscalation)	800
105	hacl(foghorn,elmer,nfsProtocol,nfsPort)	100
106	nfsExportInfo(elmer,'/export/files',write,foghorn)	500
108	hacl(foghorn,foghorn,httpProtocol,httpPort)	∞
110	hacl(pepeVPN,foghorn,httpProtocol,httpPort)	500
112	hacl(sylvester,foghorn,httpProtocol,httpPort)	500
114	hacl(tweety,foghorn,httpProtocol,httpPort)	500
116	hacl(internet,foghorn,httpProtocol,httpPort)	100
118	networkServiceInfo(foghorn,tomcat,httpProtocol,httpPort,tomcat)	10000
119	vulExists(foghorn,'CVE-2007-0774',tomcat,remoteExploit,privEscalation)	400
121	hacl(sylvester,foghorn,mailProtocol,mailPort)	2000
122	networkServiceInfo(foghorn,exchange,mailProtocol,mailPort,root)	4000
123	vulExists(foghorn,'CVE-2009-0098',exchange,remoteExploit,privEscalation)	1600

This page intentionally left blank.

Annex D: Closure of vertices 105, 42, 119, 73 in Example 2 network

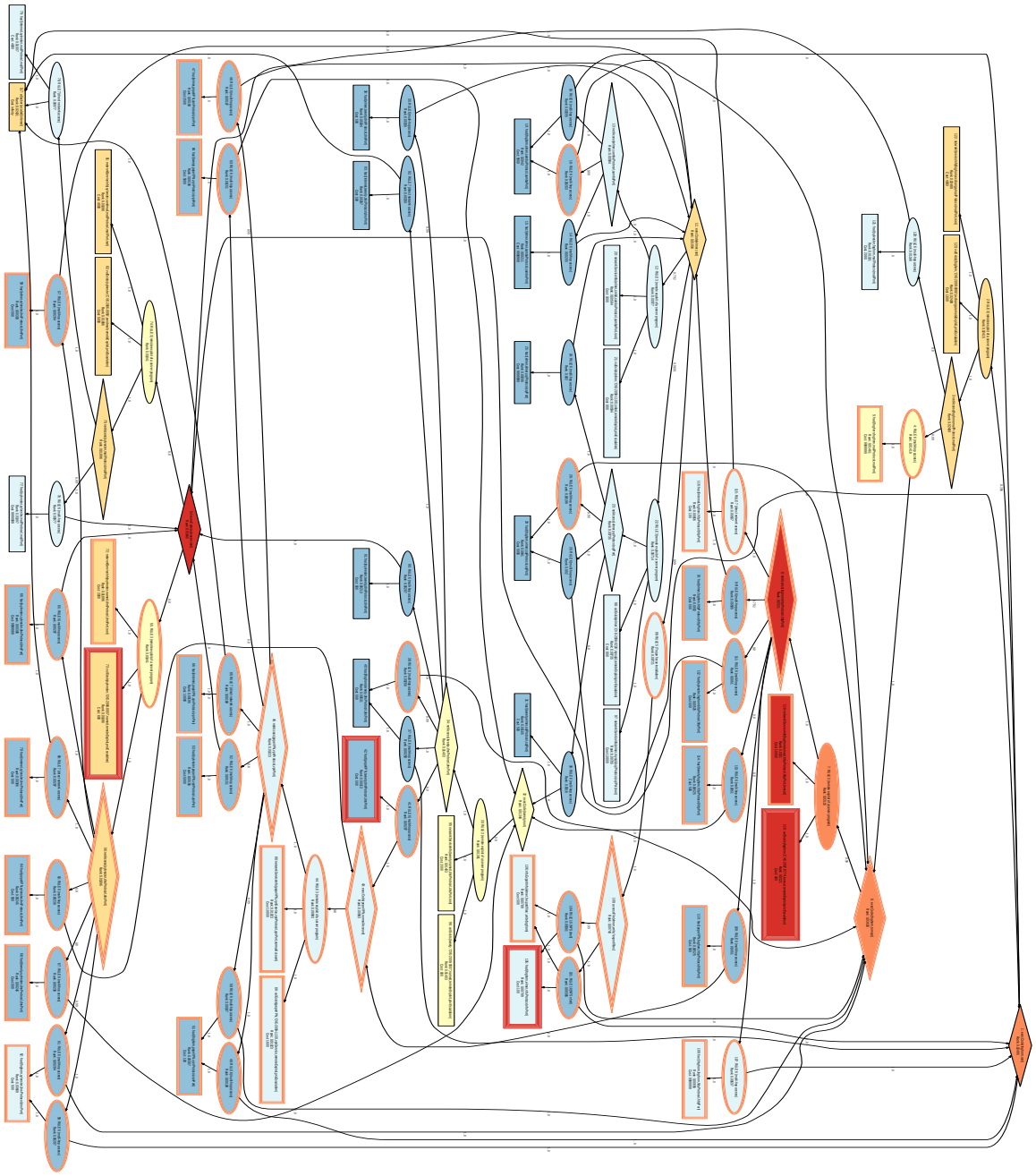


Figure D.1: *Example 2 Network: Solution of greedy algorithm for Budget* $= 2000$. *The COA set (red outline) consists of vertices 105, 42, 119, and 73. The closure of this COA set (orange outline) accounts for 45% of the rank. The graph that results when the outlined vertices are removed is shown in Appendix E.*

This page intentionally left blank.

Annex E: Removal of vertices 105, 42, 119, 73 in Example 2 network

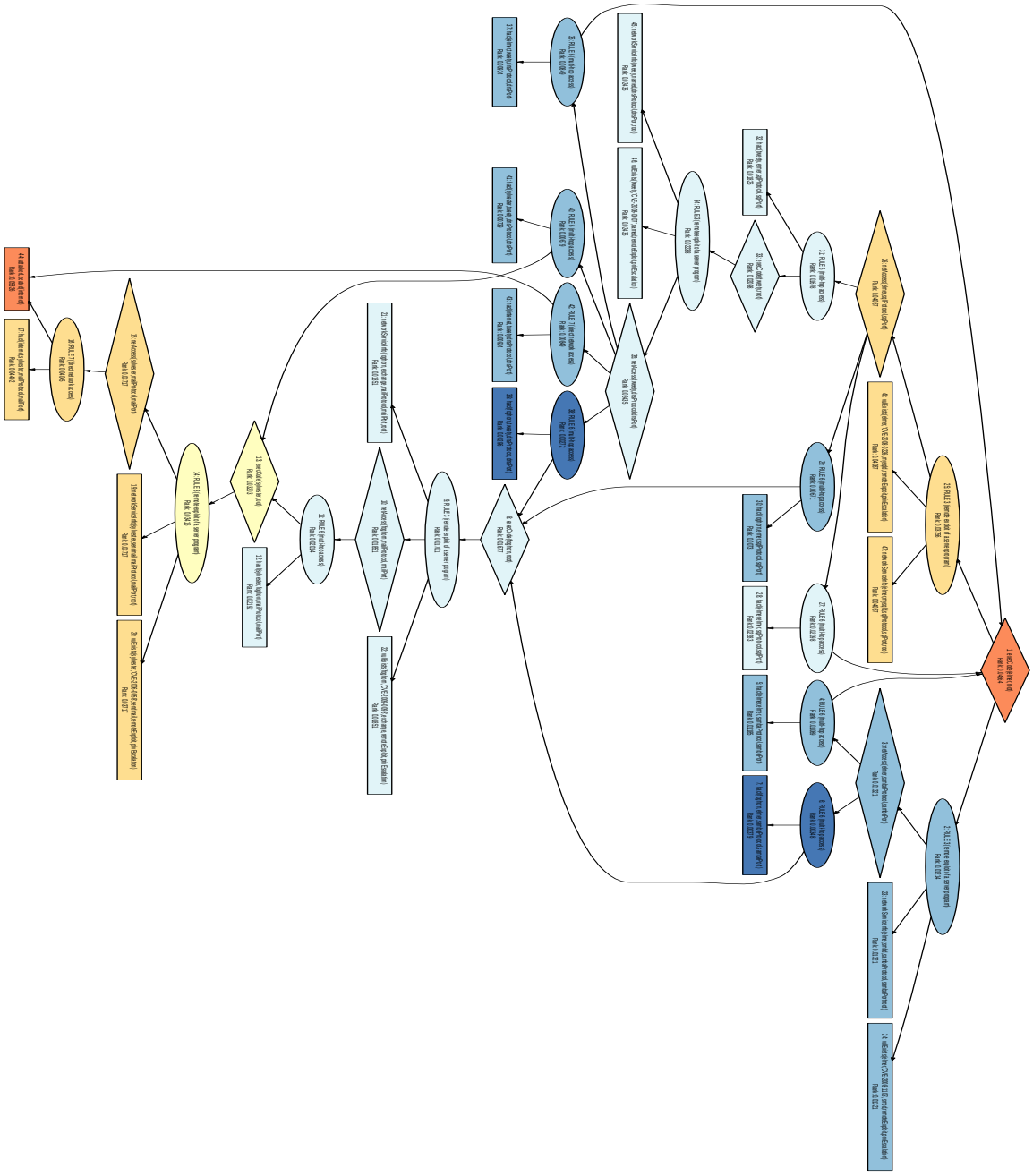


Figure E.1: *Example 2 Network: Attack graph remaining after removing COA set* $\{105, 42, 119, 73\}$ *and its closure. The graph is substantially simpler than its original form, shown in Appendix D).*

This page intentionally left blank.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when document is classified)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Defence R&D Canada – Ottawa
3701 Carling Avenue, Ottawa ON K1A 0Z4, Canada

2. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.)

UNCLASSIFIED

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)

Course of action recommendations for practical network defence

4. AUTHORS (Last name, followed by initials – ranks, titles, etc. not to be used.)

Sawilla, R.; Burrell, C.

5. DATE OF PUBLICATION (Month and year of publication of document.)	6a. NO. OF PAGES (Total containing information. Include Annexes, Appendices, etc.)	6b. NO. OF REFS (Total cited in document.)
August 2009	46	27

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Memorandum

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

Defence R&D Canada – Ottawa
3701 Carling Avenue, Ottawa ON K1A 0Z4, Canada

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)
15BB03	

10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)
DRDC Ottawa TM 2009-130	

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

(X) Unlimited distribution
( ) Defence departments and defence contractors; further distribution only as approved
( ) Defence departments and Canadian defence contractors; further distribution only as approved
( ) Government departments and agencies; further distribution only as approved
( ) Defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11)) is possible, a wider announcement audience may be selected.)

Unlimited

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

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 as effective as the optimal choices computed by an exponential algorithm, making our solution especially useful in practice. We demonstrate the value of our solution on a complex cyclic attack graph generated from a representative business network.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

attack graph, security metrics, course of action recommendation, partial or perfect security, return on investment

Defence R&D Canada

Canada's leader in Defence
and National Security
Science and Technology

R & D pour la défense Canada

Chef de file au Canada en matière
de science et de technologie pour
la défense et la sécurité nationale

www.drdc-rddc.gc.ca