Aggregating Vulnerability Metrics in Enterprise Networks using Attack Graphs

John Homer[1], Su Zhang[2], Xinming Ou[2], David Schmidt[2], Yanhui Du[3], S. Raj Rajagopalan[4], and Anoop Singhal[5]

[1]Abilene Christian University, U.S.A.
[2]Kansas State University, U.S.A.
[3]Chinese People's Public Security University, P.R. China
[4]HP Labs, U.S.A.
[5]National Institute of Standards and Technology, U.S.A.

Abstract

Quantifying security risk is an important and yet difficult task in enterprise network security management. While metrics exist for individual software vulnerabilities, there is currently no standard way of aggregating such metrics. We present a model that can be used to aggregate vulnerability metrics in an enterprise network, producing quantitative metrics that measure the likelihood breaches can occur within a given network configuration. A clear semantic model for this aggregation is an important first step toward a comprehensive network security metric model. We utilize existing work in attack graphs and apply probabilistic reasoning to produce an aggregation that has clear semantics and sound computation. We ensure that shared dependencies between attack paths have a proportional effect on the final calculation. We correctly reason over cycles, ensuring that privileges are evaluated without any self-referencing effect. We introduce additional modeling artifacts in our probabilistic graphical model to capture and account for hidden correlations among exploit steps. The paper shows that a clear semantic model for aggregation is critical in interpreting the results, calibrating the metric model, and explaining insights gained from empirical evaluation. Our approach has been rigorously evaluated using a number of network models, as well as data from production systems.

Keywords

risk assessment, vulnerability metrics, attack graphs, enterprise network security management

1 Introduction

Currently, the evaluation and mitigation of security risks in an enterprise network appears to be more an art than a science. System administrators operate by instinct and experience, often without any verifiable way to gauge the full ramifications of any changes in the network. Without an objective measurement of risk, there is no straightforward and reliable method to answer fundamental questions, such as: "How likely is it that an attacker could gain privilege X?", "Where is our network most vulnerable?" and "If we change A, will our network be more or less secure as a result?" These questions must be asked when judging how scarce resources can best be utilized to improve the security of an enterprise network. Oftentimes, improvement of security also comes with cost to functionality or ease of use; thus, it is important to understand how much reduction in overall security risk a proposed change can achieve. Answering these questions requires

a quantitative model of security with clear measurements of risk and easy comparison of different network states.

Much work has already been done in analyzing network configuration data and identifying network vulnerabilities to construct attack graphs [2, 6, 7, 8, 14, 15, 16, 19, 20, 21, 29, 30, 31, 32, 35, 37, 40, 41, 42, 44]. Attack graphs illustrate the cumulative effect of attack steps, showing how series of individual steps can potentially enable an attacker to gain privileges deep into the network. One limitation of attack graphs, however, is the assumption that any existing vulnerability can be exploited. In reality, there is a wide range of probabilities that different vulnerabilities could be profitably exploited by an attacker, depending on the skill of the attacker and the difficulty of the exploit. Attack graphs show what is possible without any indication of what is likely.

Recently, there has been significant progress in standardizing and developing metrics for individual vulnerabilities, such as the Common Vulnerability Scoring System (CVSS) [27]. These risk measurements consider both specific qualities of vulnerabilities, such as the skill necessary to exploit the weakness, and known information about the availability of an exploit. The key limitation of such individual vulnerability metrics is that it is not possible to capture the security interactions the vulnerabilities have within the context of the enterprise network. For example, a vulnerability may have a high CVSS score (indicating it represents high risk to a system when the vulnerability is exposed to an attacker). But the vulnerability may reside at a location that is highly difficult for an attacker to access. Likewise, a vulnerability may have a lower CVSS score but reside at a location that is relatively easy for an attacker to approach. To accurately reflect the security risks that vulnerabilities introduce to an enterprise network, both measurement of individual vulnerabilities' properties and the context within which they appear must be taken into account.

Since attack graphs represent the logical inter-relationship among possible attack steps, it is natural to combine attack graphs and individual vulnerability metrics to calculate security risk metrics for an enterprise network. The ground-breaking work by Dacier *et al.* [6] and Ortalo *et al.* [29] applies a Markov-chain model on an attack graph (called "privilege graph" in their work) to calculate metrics such as mean time to security failure (MTTF) and mean effort to security failure (METF). This approach needs to first transform the attack graph to an "attack state graph" suitable for the Markovian model. Depending on the assumptions made about attacker behavior, the attack state graph could quickly become too large to be built [20, 29]. More recent works adopt probabilistic reasoning on dependency attack graphs [3, 9, 45, 47] which can be generated more efficiently. Frigault and Wang adopt Bayesian networks as the underlying model for calculating attack success likelihood on such graphs [9, 10]. However, directed cycles that are commonplace in attack graphs [31, 45] cannot be handled through a black-box application of Bayesian reasoning. Wang *et al.* made a number of crucial observations about cyclic attack graphs and proposed a customized probabilistic reasoning method that can handle cycles in the calculation [45]. However, when combining probabilities from multiple attack paths, the method uses a formula that assumes the multiple probabilities are independent. Our observations of attack graphs produced from production systems indicate that multiple attack paths are rarely independent and often share some common exploits. Such dependency needs to be accounted for to prevent distortion of results.

1.1 Contributions

Our contribution is a security metric model that can be used for aggregating vulnerability metrics such as CVSS. This model has clearly defined semantics and a sound computation algorithm with respect to the semantics. The soundness ensures that the output of the metric model has a clear meaning with respect to the input. The input to the metric model, *component metrics*, represent conditional probabilities of success for exploiting an individual vulnerability. The output of the metric model, *cumulative metrics*, indicate

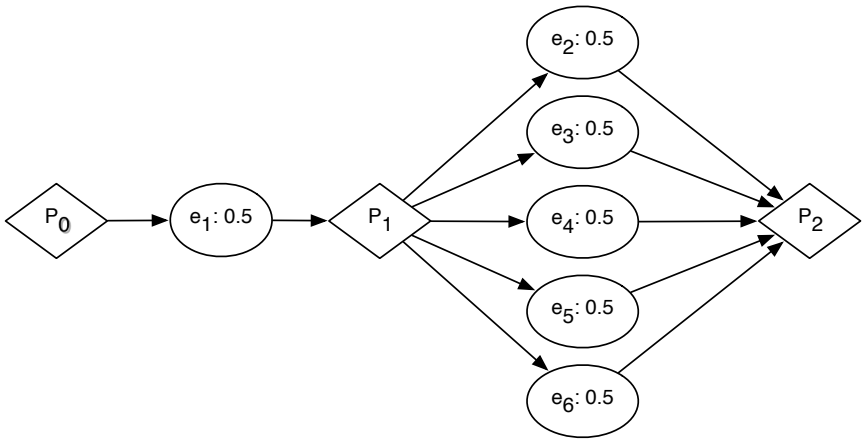


Figure 1: Sample attack graph showing the importance of accounting for shared dependency

the absolute probability that a specific network privilege could be obtained by an attacker, taking into consideration all possible multi-step attacks that can lead to that privilege.

The aggregation of individual vulnerabilities' metrics provides a measurement of the "exposure" of the system to attacks from a probability perspective (and can be viewed as a measurement of *attack surface* as defined in the literature [22, 23]). It does not account for other important risk factors such as asset value and threat model. Even if a system has a large attack surface, the risk is not necessarily high; this also depends on the value of the asset in the network and the likelihood that an adversary will seek after it. However, we believe our metric model is an important first step towards a comprehensive security metric model for enterprise networks. The above mentioned additional risk factors can be plugged in as extensions to the basic model, and provide metrics useful for decision making. For example, the asset value can be multiplied with the probability that the asset may be compromised by the attacker, providing an expected loss in monetary term. The threat model can be modeled as an adjuster on certain prior probabilities in the metric model. A clear metric semantics of our aggregation method provides a solid foundation for such extensions and comprehensive security risk metrics for enterprise networks.

1.1.1 Why a clear metric semantics is important

One challenge for security metrics is that many times the input parameters to the metric model are inevitably imprecise. For example, knowing the likelihood that a vulnerability can be successfully exploited is important when deciding upon mitigation priorities. This likelihood can be characterized as the probability of success assuming certain attacker skills and resources. However, such probabilities are virtually impossible to obtain and so we must rely upon imprecise estimates. A clear semantic model for metric calculation makes it possible to interpret the result even with the influence of imprecise input. Without clear semantics, one would not be able to tell whether a seemingly unintuitive metric result is due to the fact that the input is imprecise or that the assumptions made in the metric model are not realistic. Work continues to progress in refining metrics for better capturing the properties of network vulnerabilities. A clear metric semantic model for combining these component metrics can actually help make them more useful and more precise: a cumulative assessment that is thought to be faulty can be traced back to either the imprecision in the input component metrics, or incorrect assumptions made in the aggregation model. This provides an important feedback loop for refinement and calibration.

Once a semantic model is chosen, it is important that the metric computation algorithm adhere to the semantics. One may think that a slightly "unsound" calculation may not affect the result greatly and so

question the importance of ensuring that the model's calculation is accurate, especially when the input is imprecise. In Figure 1, we present an attack graph segment with a situation commonly found in attack graphs generated from production systems we studied. The attacker's initial privilege is p_0. He can launch exploit e_1 to achieve privilege p_1. Given p_1, he can launch one of the exploits $e_2, ..., e_6$ to gain privilege p_2. In this example, we assume that each exploit has a success likelihood of 0.5 — the *conditional probability* for him to succeed in that attack step given that the pre-condition privilege has been achieved. The probability that the attacker is able to achieve p_2 can be calculated as:

$$Pr[p_2] = Pr[p_2|p_1] \cdot Pr[p_1] = (1 - 0.5^5) \cdot 0.5 = 0.48$$

The probability for achieving p_2 is the product of the probability of achieving p_1 and the *conditional probability* that the attacker can achieve p_2 given that p_1 is achieved. However, if we assume all the attack paths from p_0 to p_2 are independent and combine the probabilities from each path (0.25) in a simplistic way, we would get the following result:

$$(1 - 0.75^5) = 0.76$$

Obviously this would create a deviation significant enough to lead to different decisions.

1.2 Challenges in metric calculation under probabilistic semantics

Enterprise networks typically enable a great deal of interconnectedness between network hosts. Multiple attack paths leading to a given network privilege are rarely independent but more likely share some dependencies, as in Figure 1. This interconnectedness often leads to the appearance of cycles in an attack graph. An accurate assessment of the probability that some privilege could be gained by an attacker should handle shared dependencies and cycles based on realistic assumptions to prevent skewing the final result. Doing so while controlling the computation complexity has been a major challenge in this work.

2 Related Work

The issue of security metrics has long attracted much attention [18, 24, 25], and recent years have seen significant effort on the development of quantitative security metrics [1, 4, 33]. There is also skepticism on the feasibility of security metrics given the current software and system architectures [5]. While we may still be far from achieving objective quantitative metrics for a system's overall security, a practical method for aggregating vulnerability metrics in an enterprise network is highly valuable in practice. Although forming only one aspect of a system's overall security risk, such cumulative metrics can provide much needed automated guidance on how to allocate limited IT management resources and how to balance security and usability in a meaningful manner. Our work provides an important first step in automated decision making through sound models and algorithms for aggregating vulnerability metrics using attack graphs.

Attack graphs have emerged as a mainstream technique for enterprise network vulnerability analysis [2, 13, 14, 16, 20, 21, 31, 35, 41]. Recent years have also seen effort toward computing various metrics from attack graphs [10, 28, 39, 45, 47, 48]. Our work builds upon results from some of these previous works but is unique in that it provides a model with clearly defined probability semantics and a sound algorithm for calculating the semantics, which we believe is critical in refining and calibrating the metric models.

Idika and Bhargava [12] make a number of crucial observations on the limitations of existing attack graph-based security metrics. The authors propose combining the existing metrics when comparing two enterprise networks, and use additional metrics (called "assistive metrics") when the main metrics (called "decision metrics") cannot correctly differentiate the security levels of two systems. We believe the limitations observed

by the authors are due to the lack of clear and meaningful semantics in the existing security metric models. We believe our approach addresses these issues and we demonstrate the effectiveness of our metric model by applying it on production systems.

The Common Vulnerability Scoring System (CVSS) [27] provides an open framework for communicating the characteristics and impacts of IT vulnerabilities. CVSS consists of three metric groups: Base, Temporal, and Environmental. Each of these groups produces a vector of compressed textual representation that reflects various properties of the vulnerability (*metric vector*). A formula takes the vector and produces a numeric score in the range of 0 to 10 indicating the severity of the vulnerability. The metric vector is a very important contribution of CVSS; it provides a wide range of vulnerability properties that are the basis for deriving the numerical scores. The main limitation of CVSS is that it provides scoring of individual vulnerabilities but does not provide a methodology for aggregating the metrics for a set of vulnerabilities in a network to provide an overall network security score. The overall security of a network configuration running multiple services cannot be determined by simply counting the number of vulnerabilities or summing the CVSS scores. Our work provides a clearly defined mathematical model based on attack graphs that can be used to aggregate CVSS metrics to reflect the cumulative effect of vulnerabilities in an enterprise environment.

Frigault *et al.* [9, 10] utilize the combination of attack graphs and Bayesian Networks (BN) in measuring network security. The major limitation of using BN is that it does not allow directed cycles, which are common in attack graphs. Our approach does not use BN reasoning as a black box. Instead, we utilize the key concept of d-separation in BN inference and design customized algorithms for probabilistic reasoning on attack graphs. Our approach not only handles cycles within the context of risk assessment but also takes into consideration special properties of our metric's semantics (*e.g.*, that no real-time evidence needs to be considered, the monotonicity property, and so on) which can eliminate some unnecessary overhead and complexity in a general BN inference engine.

Wang *et al.* [45] recognize the presence of cycles in an attack graph and present useful ideas about propagating probability over cycles. However, their probability calculation seems to assume that probabilities along multiple paths leading to a node are independent, which is not generally true for attack graphs. Our approach correctly handles both cycles and shared dependencies in attack graphs.

Anwar *et al.* [3] introduce an approach to quantitatively assessing security risks for critical infrastructures. The approach captures both static and dynamic properties in the network, contained in network and workflow models. However, the work did not provide a semantic model to explain what the calculated metrics mean. Our risk metric has a clear semantics quantifying the likelihood an attacker can succeed in achieving a privilege or carrying out an attack. Our metric algorithm provides a sound linkage between the input component metrics and the output cumulative metrics based on this semantics.

Sawilla and Ou design an extended Google PageRank algorithm and apply it to attack graphs to rank graph nodes by usefulness to an attacker for achieving his perceived goals [39]. Numeric values computed from this algorithm only indicate *relative* ranks and cannot be used to indicate absolute security risks. Mehta *et al.* also apply Google PageRank on a different type of attack graph [26]. While this metric could be used in calculating absolute risk based on probability semantics, the exponential nature [31] of the underlying attack-graph model makes it difficult to use in practice. Our work provides a more practical method for aggregating the absolute security exposure from vulnerabilities in an enterprise network, based on a more efficient attack-graph model.

Wang *et al.* [46] introduce an approach that assumes cost metrics are present for all nodes in an attack graph and use this information to identify a minimum-cost network hardening solution. Dewri *et al.* [8] formulate security hardening as a multi-objective optimization problem, using a genetic algorithm to search for an optimal solution based on costs of security hardening and potential damage. Homer and Ou [11]

demonstrate the effectiveness of using MinCostSAT as a basis for automated network reconfiguration, with numeric cost values being assigned to each configuration setting and reachable privilege in the attack graph. Noel *et al.* [28] propose using attack graphs to calculate security risk metrics and using the metrics to do cost-benefit analysis to support decision making. Our metric model and calculation algorithm can benefit such effort to apply security metrics in practical network security management. Accurate probability relationship between input and output of the metric model can improve the reliability and trustworthiness of the cost-benefit analysis and the suggested hardening options.

3 Problem Overview

We utilize the MulVAL attack graph [31] as a structural basis for aggregating vulnerability metrics, although our approach should be easily transferable to other tools that produce attack graphs with similar semantics [14, 16, 37].

3.1 An example scenario

Figure 2 shows an example enterprise network, which we will use to introduce a number of terminologies used in our security metrics algorithm.

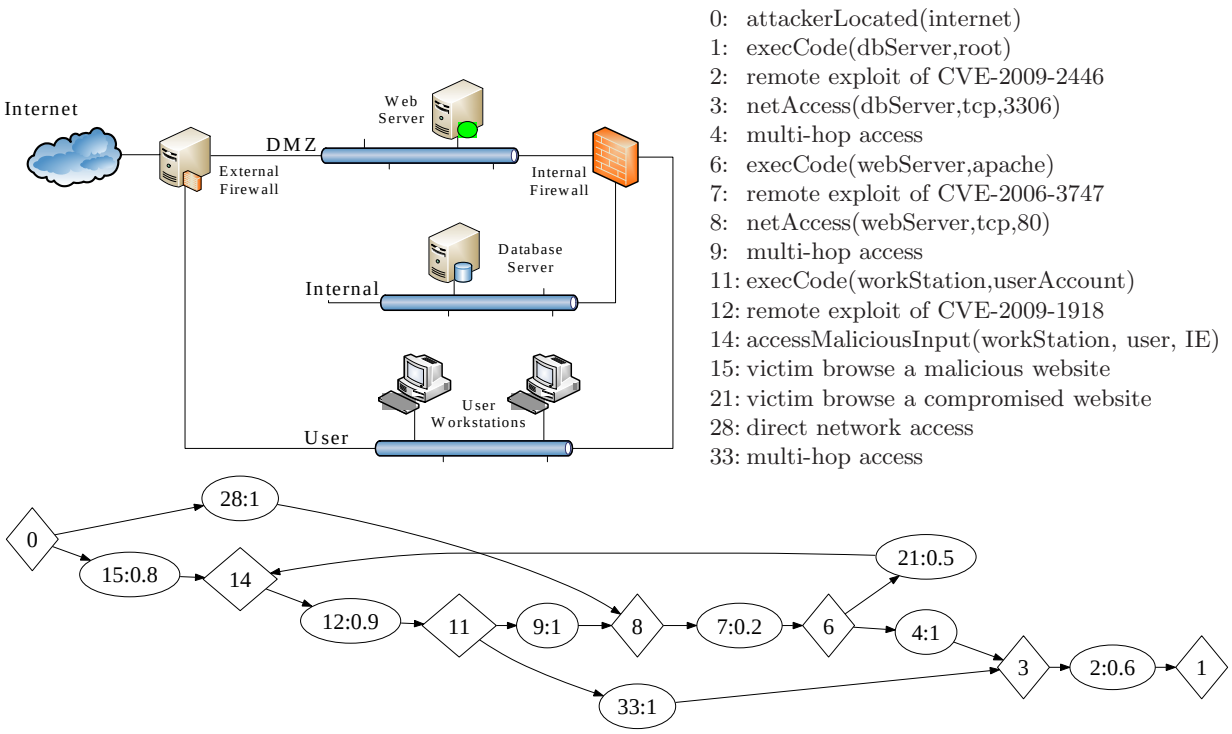


Figure 2: Example scenario and a simplified attack graph

Host and network reachability: There are three subnets mediated by an external and an internal firewall. The web server is in the DMZ subnet and is directly accessible from the Internet through the external firewall. The database server is located in the Internal subnet and holds sensitive data. It is only accessible from the web server and the User subnet. The User subnet contains the user workstations used by the company's employees. The external firewall allows all outbound traffic from the User subnet.

Vulnerabilities: The web server contains the vulnerability CVE-2006-3747[1] in the Apache HTTP service,

[1]Common Vulnerabilities and Exposures (CVE) is a dictionary of common names (i.e., CVE Identifiers) for publicly known

by which a remote attacker could gain privileges to execute arbitrary code on the machine. The database server contains the vulnerability CVE-2009-2446 in the MySQL database service, which could allow an attacker to gain administrative access. The user workstations contain the vulnerability CVE-2009-1918 in Internet Explorer. If a user accesses malicious online content using the vulnerable IE browser, the user workstation can be compromised. When a system admin is faced with these vulnerabilities, the first questions he will ask is: are any of these critical problems, and which one shall I deal with first? *Without a quantitative model that captures the vulnerabilities' cumulative effects on the network, it is hard to answer such questions in an objective way.*

Attack graph semantics: An attack graph G consists of a set of nodes G_N of three types: (1) attack-step nodes (collectively, set G_C), represented within the graph as circular-shaped AND-nodes. Each node in this set represents a single attack step which can be carried out when all the predecessors (preconditions to the attack which are either configuration settings or network privileges) are satisfied; (2) privilege nodes (collectively, set G_D), represented within the graph as diamond-shaped OR-nodes. Each node in this set represents a single network privilege. The privilege can be achieved through any one of its predecessors (representing attack steps leading to the privilege); (3) configuration nodes, which are not shown in this graph. Each node in this set represents a fact about the current network configuration that contributes to one or more attack possibilities. For probability reasoning, however, the configuration nodes can be removed since they are known to be true and have no variance in probability. Excluding the configuration nodes leaves us with an AND/OR graph, where an AND-node (attack-step) is only preceded by OR-nodes (privilege) and always has exactly one successor OR-node. An OR-node, however, may have multiple successor AND-nodes, representing different attack steps requiring this privilege as a preceding condition.

The bottom of Figure 2 displays the MulVAL attack graph produced from the model of this network configuration, with the configuration nodes removed. The attack graph shows several alternative sequences for attack. For example, an attacker can first compromise the web server and use it as a stepping stone to attack the database server, or he can first compromise the workstation and attack the database server from there. Node labels can be found at the right-hand side of the network diagram. The label for the OR nodes is in the general form of "predicate(parameters)." For example, node 1's label is "execCode(dbServer,root)", where "execCode" is the predicate and "dbServer" and "root" are two parameters, meaning "the attacker can execute arbitrary code with root privilege on host dbServer". Another example: "netAccess(dbServer,tcp,3306)" means "the attacker has network access to host dbServer" on port 3306 through protocol tcp". The AND-nodes' labels are descriptors of the attack, *e.g.* remote exploit of a vulnerability, or user actions like browsing a malicious website. These attack-step nodes are also associated with a probability (following the colon), representing the likelihood of success given all preconditions; these are explained further in the next subsection.

3.2 Component metrics

The input to the metric model are the component metrics, associated with each attack-step node, which indicate the severity of a single vulnerability. The metric represents the conditional probability that a single attack step will succeed when all the prerequisite conditions are met. For example, the component metric for node 7 represents the probability that the attacker can successfully exploit the vulnerability CVE-2006-3747 when he already has obtained the precondition for the exploit (represented by node 8): network access to the web server which runs the Apache service. We use the CVSS metrics to derive the component metrics for our metric model. Specifically, we take the Access Complexity (AC) sub-metric in CVSS and map it to a conditional probability of exploit success. The AC metric takes values in {low, medium,

information security vulnerabilities (http://cve.mitre.org/)

high} indicating the complexity of exploiting the vulnerability. In our experiment, we use the mapping $\{\text{low} \rightarrow 0.9, \text{medium} \rightarrow 0.6, \text{high} \rightarrow 0.2\}$. Intuitively, the more complex it is to exploit a vulnerability, the less likely an attacker will succeed. The AC metrics for the Apache, MySQL, and IE vulnerabilities in this example are: high, medium, and low respectively. There are also a few other component metrics that do not come from CVSS. For example, node 15 is the attack step that involves tricking a user on a workstation into accessing malicious content. The likelihood of succeeding in this is strongly affected by the security awareness of users and thus is context-specific. To provide these metric values, a risk analysis tool can conduct an initial survey asking multiple-choice questions like, "How likely is it that a system user will visit a malicious website?" Based on the answers provided by the user (system administrator or security analyst), a set of component metrics representing the above likelihood can be derived and used in subsequent analyses.

This simple component-metric scheme is used for experimentation, when our focus is on the cumulative metric computation model; further research is needed on how best to assign the component metrics. For example, time is a critical factor in component metrics: over a longer time period, it is more likely that a user will at some point click a malicious link. Other factors include the presumed attacker's skill level, resources, and so on.

3.3 Cumulative metrics

In our metric model, the cumulative metrics represent the aggregate effect of all the vulnerabilities in the network. Specifically, we would like to capture the likelihood a privilege can be obtained by a *dedicated attacker*, who will try all possible paths identified by the attack graph to achieve the privilege. The cumulative metric indicates the probability that he will succeed in achieving the privilege in at least one of the paths.

Here we give some basic notations used in formulating the calculation of the cumulative metrics. We use G_M to denote the set of relevant component metric values; each attack-step node $c \in G_C$ will have an assigned metric value denoted as $G_M[c]$. Thus, for some attack-step node e with predecessor set P, the probability of e given set P is represented by the component metric value: $Pr[e|P] = G_M[c]$. Additionally, the attack graph will have an assumed prior risk value, G_V, representing the probability that attacks will be attempted against the network. G_V is associated with the root node of the graph, G_R (here, node 0). We will assume $G_V = 1$ in the subsequent discussions.

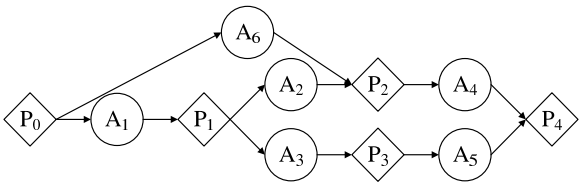


Figure 3: An attack graph without a cycle

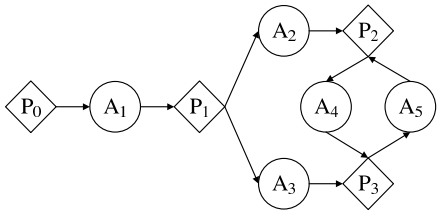


Figure 4: An attack graph with a cycle

3.3.1 Handling shared dependency

In an attack graph, it is common to see multiple attack paths leading to a single network privilege. In the example attack graph of Figure 2, node 3 has two paths leading into it, both of which depend upon node 11. Such shared dependency must be correctly accounted when calculating the joint probability that the attacker succeed in both paths, to prevent distortion of results (see Figure 1).

To simplify discussion, we use a hypothetical attack graph shown in Figure 3, in which privilege P_4 can be obtained by an attacker using either of two attack steps — A_4 or A_5. Privilege P_4 will be unobtainable if an attacker cannot successfully carry out either A_4 or A_5. $Pr[A_4]$ and $Pr[A_5]$ are the probabilities that

A_4 and A_5, respectively, can be successfully carried out. If the paths to A_4 and A_5 are independent, we would calculate the probability that an attacker might gain privilege P_4 to be: $Pr[P_4] = Pr[A_4] + Pr[A_5] - Pr[A_4] \cdot Pr[A_5]$. However, it is *incorrect* to assume that A_4 and A_5 are independent. Looking at Figure 3, it is easily seen that attack step A_4 is potentially affected by privilege P_1, and attack step A_5 fully depends upon it. Because of this shared dependence on P_1, A_4 and A_5 are not independent and the above formula would skew the effect that privilege P_1 has on the final result. In other words, *assuming that all attack paths in an attack graph are independent will lead to biased results in aggregating vulnerability metrics.*

To correctly account for shared dependencies among attack paths, we will employ the notion of *d-separation* within a causal network (such as a Bayesian Network) [17]. The concept of d-separation can be utilized to establish conditional independence between node sets. The metric aggregation problem on attack graphs is a different problem than generic Bayesian-Network reasoning (see section 3.3.3); therefore, the concept of d-separation is customized here for our specific application.

Definition 1 *In an attack graph, two distinct node sets S_1 and S_2 are d-separated by an intermediate node set $V \subseteq G_N$ (distinct from S_1 and S_2) if along every diverging path between S_1 and S_2, there is some $v \in V$ such that v is the point of divergence.*

A d-separating set for two nodes always exist; naively, the set of graph nodes in the two nodes' Markov blankets [34] together d-separate the two nodes. Practicality, however, requires that a minimal set be found to reduce calculation time. Because of attack graphs' semantics, only *shared dependencies* (points of divergence in shared paths) need to be considered in the construction of a d-separating set. For example, in Figure 3, there is a diverging path $A_2 \leftarrow P_1 \rightarrow A_3$ between nodes A_2 and A_3; the point of divergence, P_1, d-separates these two nodes. Even though A_2 and A_3 are not independent (they are both influenced by P_1), when P_1 is fixed they become *conditionally independent*. The determination of a minimal d-separating set is addressed in Section 4.1.

Nodes with multiple successor nodes are called "branch nodes"; the set of all branch nodes is denoted as G_B. Since the configuration nodes have been removed and attack-step nodes have exactly one successor each, the set of branch nodes must necessarily be a subset of the privilege nodes — $G_B \subseteq G_D$. The d-separating set D for two node sets will then be a subset of G_B, so that the elements in D "block" all diverging paths between the two node sets. To identify a minimal set of nodes that d-separates two node sets, we only need to consider branch nodes along the paths from the root node G_R to the nodes. Consider A_4 and A_5 in Figure 3 as an example; here, we only need to consider the branch nodes along the paths to A_4 and A_5, which is $D = \{P_0, P_1\}$. We can see from the figure that D d-separates A_4 and A_5. Once the nodes in D are fixed, A_4 and A_5 become conditionally independent, and we can then calculate the joint distribution.

Theorem 3.1 *Let D, N be node sets such that D d-separates any pair of nodes in N. Then:*

$$Pr[N] = \sum_D (\prod_{n \in N} Pr[n|D]) \cdot Pr[D]$$

Proof:

$$\begin{aligned} Pr[N] &= \sum_D Pr[N, D] \\ &= \sum_D Pr[N|D] \cdot Pr[D] \\ &= \sum_D (\prod_{n \in N} Pr[n|D]) \cdot Pr[D] \end{aligned}$$

By Bayes theorem, $Pr[N|D] \cdot Pr[D]$ produces the joint probability $Pr[N, D]$. Summing over all possible values of D will *marginalize* D from the joint distribution $Pr[N, D]$. Since D d-separates any pair of nodes

in N, all nodes $n \in N$ are conditionally independent given D: $Pr[N|D] = \prod_{n \in N} Pr[n|D]$.

Using the above theorem, we have a way to calculate the joint probability $Pr[\overline{A_4}, \overline{A_5}]$, which is needed to calculate $Pr[P_4]$. We can sum over all possible values of d-separating set $\{P_0, P_1\}$ to solve $Pr[\overline{A}_4, \overline{A}_5]$, and we can decompose a joint conditional distribution to singleton conditional probabilities:

$$\begin{aligned} Pr[P_4] &= 1 - Pr[\overline{A_4}, \overline{A_5}] \\ &= 1 - \sum_{P_0, P_1} Pr[\overline{A}_4|P_0, P_1] \cdot Pr[\overline{A}_5|P_0, P_1] \cdot Pr[P_0, P_1] \end{aligned}$$

The formation and use of calculations will be discussed in greater detail in later sections.

3.3.2 Handling cycles

It is also common to find cycles in attack graphs. The example attack graph in Figure 2 contains a cycle (6-21-14-12-11-9-8-7-6). To simplify discussion, we use another small hypothetical attack graph shown in Figure 4, where some graph nodes comprise a cycle — $\{P_2, A_4, P_3, A_5\}$. When evaluating the probability of a node within a cycle, such as $Pr[P_2]$, we must be careful that node P_2 does not affect its own probability of occurrence via cyclic attack paths. According to the graph, it is possible that an attacker can use attack step A_2 to obtain privilege P_2, then use attack step A_4 to obtain privilege P_3, and then use attack step A_5 to obtain privilege P_2 again. Although this path does technically exist within the graph, one must take care so that the existence of such cycles do not distort the probability calculation result. For example, even though an attacker could traverse a cycle to return to a prior privilege, this shall not increase his success likelihood in obtaining the privilege. We must be able to evaluate nodes within the cycle while eliminating any cyclic influence in the probability calculation. One straightfoward approach is to unfold any cyclic graph into an equivalent acyclic graph such that each node appears exactly once in any path, but this procedure typically results in an exponential blowup in the size of the unfolded graph. Unfolding the graph is actually not necessary if we apply a data flow analysis to the cyclic nodes so that we can evaluate the same probabilities as on the unfolded graph, but without actually unfolding it. Through dynamic programming and other optimizations in the data flow analysis process, we can avoid some increase in the complexity of the calculation.

3.3.3 Relationship to Bayesian Networks

Our method conducts Bayesian reasoning on attack graphs for the purpose of caculating the cumulative metrics under the dedicated-attacker semantics. Existence of directed cycles in attack graphs preclude direct application of Bayesian Network (BN) reasoning systems, which require the graphical model to be acyclic. Our algorithm does adopt the core concept of Bayesian Network reasoning, d-separation, in calculating the probability metrics. It handles directed cycles (which cannot be handled by a general BN system) based on the application semantics, and utilizes specific features of our metric semantics to simplify computation. Thus our algorithm is neither a specialization nor a generalization of the standard Bayesian Network reasoning system. It overlaps with BN in that both utilize the notion of d-separation in calculating probabilities.

4 Cumulative Metric Computation

4.1 Definitions

For describing our approach of computing cumulative vulnerability metrics, it is convenient to define and employ the following notations.

Function 1 *For a node $n \in G_N$, $\phi(n)$ represents the absolute probability that node n is true (i.e., the probability that the privilege/attack step represented by node n can be successfully gained/launched). Similarly, $\phi(\overline{n})$ represents the probability that node n is false. $\phi(n) + \phi(\overline{n}) = 1$.*

For example, in Figure 3, $\phi(P_4)$ is the probability that node P_4 is true, while $\phi(\overline{P}_4)$ is the probability that P_4 is false. We extend the notation to represent joint probability for multiple nodes. For example, $\phi(\overline{A}_4, \overline{A}_5)$ is the joint probability that nodes A_4 and A_5 are both false.

Function 2 *For a node $n \in G_N$, $\psi(A, n)$ represents the conditional probability that node n is true given the condition A. We also extend the notation to a set of nodes similarly as before.*

For example, in Figure 3, $\psi(\{P_0, P_1\}, P_2)$ is the probability that P_2 is true given that P_0 and P_1 are true. This eliminates any influence that A_1 has on node P_2. We use e_i to denote the component metric value for each attack-step node A_i, *i.e.* $\forall A_i \in G_C, G_M[A_i] = e_i$, so $\psi(\{P_0, P_1\}, P_2) = e_2 + e_6 - e_2 e_6$. Another example: $\psi(\{P_0, \overline{P}_1\}, P_2)$ is the probability that P_2 is true given that P_0 is true and P_1 is false. $\psi(\{P_0, \overline{P}_1\}, P_2) = e_6$.

Function 3 *For $n \in G_N$, $\chi(n) = \{b \mid b \in G_B$ and b appears in at least one attack path leading to node $n\}$.*

$\chi(n)$ is the set of all branch nodes that appear in at least one attack path to n and so affect the probability of n. For example, in Figure 3, $\chi(A_2) = \{P_0, P_1\}$. The χ set is used to identify a minimal d-separating set for a node set. For any two nodes m and n, the set $\{\chi(m) \cap \chi(n)\}$ includes all branch nodes that lie on paths leading to both nodes; fixing the values of these nodes will d-separate m and n.

Definition 2 *Within the attack graph, for any node n, a logical dominator is any node d such that n is true only if d is true. This relationship is denoted $d \Leftarrow n$.*

Function 4 *For $n \in G_N$, $\delta(n) = \{d | d \in G_B$ and $d \Leftarrow n\}$.*

$\delta(n)$ is the set of all branch nodes that logically dominate (appear in all attack paths to) n, so that $\forall d \in \delta(n),\ \psi(\overline{d}, n) = 0$. In other words, n is false if any $d \in \delta(n)$ is false. In Figure 3, node $P_1 \Leftarrow A_5$ since A_5 is true only when P_1 is true: all attack paths to A_5 must first accomplish P_1. But P_1 does not dominate A_4, since there is a path to A_4 through A_6 which does not require P_1. The δ set is used to optimize calculations over a d-separating set.

Employing these notations, we will now consider how to calculate the probability values for every node within an attack graph.

4.2 Specification of the computation

The following propositions set forth recursively-defined equations for the calculation of the functions defined in the previous section. These propositions provide a sound basis for the evaluation for any node n, accounting for the effect of all preceding nodes in all possible paths leading to n. The root node of the attack graph, G_R, will be initialized at the beginning of the algorithm and will serve as an anchor for the ϕ recursion. When calculating $\psi(A, n)$, the recursion must reach a point where $n \in A$ (so that n must be true), $\bar{n} \in A$ (so that n must be false), or $A \cap \chi(n) = \emptyset$ (so that n is independent of A, assuming A contains branch nodes only); these base cases will serve to halt the recursion.

Proposition 4.1 *For any privilege node $n \in G_D$ with immediate predecessor set W,*

$$\begin{aligned}
\phi(n) &= 1 - \phi(\overline{W}) \\
\psi(A, n) &= 1 - \psi(A, \overline{W}) \\
\chi(n) &= \bigcup_{w \in W} \chi(w) \\
\delta(n) &= \bigcap_{w \in W} \delta(w)
\end{aligned}$$

A privilege node n will be true when at least one of its predecessors is true; conversely, it will be false only when all of its predecessors are false. The χ set for n is the set of branch nodes that affect at least one path to some $w \in W$ and so at least one path to n; the δ set for n is the set of branch nodes that affect *all* paths to n (and so logically dominate n). Thus χ can be found by a union over the χ sets for all predecessors, and δ can be found by intersecting the δ sets for all predecessors. Since the predecessors of a privilege node are all attack nodes, they cannot be branch nodes themselves.

Proposition 4.2 *For any attack-step node $n \in G_C$ with immediate predecessor set W,*

$$\begin{aligned}
\phi(n) &= G_M[n] \cdot \phi(W) \\
\psi(A, n) &= G_M[n] \cdot \psi(A, W) \\
\chi(n) &= (\bigcup_{w \in W} \chi(w)) \cup (G_B \cap W) \\
\delta(n) &= (\bigcap_{w \in W} \delta(w)) \cup (G_B \cap W)
\end{aligned}$$

When all predecessors are true, an attack step node n is true with conditional probability $G_M[n]$. Similarly, given set A, n is true with conditional probability $G_M[n]$ when all predecessors are conditionally true. The χ set for n is the set of branch nodes that affect at least one of its predecessors together with any of its predecessors that are branch nodes themselves. Since n requires that all predecessors be true, the δ set for n is the set of branch nodes that logically dominate *any* predecessor (and so logically dominate n) as well as *any* one of its predecessors that are branch nodes themselves.

Pseudocode specifications of the computation algorithms are provided below; sample computations, presented in the Appendix, demonstrate the application of this algorithm over both acyclic and cyclic graphs.

4.3 Algorithm for Cumulative Metric Computation

Algorithm 1 presents the core algorithm for vulnerability metric aggregation over an attack graph. This algorithm consists primarily of a controlling loop that will iteratively consider each individual non-cyclic node or set of cyclic nodes in the graph. This loop will terminate only when a risk assessment evaluation has been performed for every node.

The algorithm assumes a MulVAL attack graph node set, with configuration nodes removed and a new privilege node (representing the attacker's starting point) added to serve as a root node for the graph. Cycles in the graph are identified using Tarjan's algorithm for strongly connected components [43]. We also use dynamic programming to store partial results of the computation to improve efficiency.

Before entering the control loop, the data set is initialized for the graph root node. The probability that this node is true is exactly the same probability that the network will be attacked: G_V. Because this is the root node, the χ and δ sets are empty; no other nodes will affect the root node.

Within the loop, we attempt to find an individual, non-cyclic node n that is itself unevaluated but for which all predecessors have already been evaluated. Then, depending on whether the node is a privilege or attack step node, we calculate the probability of n and the χ and δ sets based on Proposition 4.1 or Proposition 4.2 respectively. If n is a branch node, we also set up the base case for $\psi(n, n)$.

Definition 3 *For some set $C \subset G_N$ comprising a strongly connected component (cycle) within the graph, the* entry nodes *are the set of nodes Q such that $Q \cap C = \{\}$ and $\forall q \in Q, \exists c \in C, (q, c) \in G_E$. That is, the entry nodes are not in the cycle, but each entry node does have an arc leading into the cycle.*

Algorithm 1: Pseudocode for cumulative metric computation

Identify cyclic subsets
$n \leftarrow G_R$ {Begin with graph root node}
$\phi(n) \leftarrow G_V$
$\chi(n) \leftarrow \emptyset$
$\delta(n) \leftarrow \emptyset$
$U \leftarrow G_N - n$ { Initialize set of unevaluated nodes, excluding root node}
while $U \neq \{\ \}$ **do** { some nodes remained unevaluated }
    if $\exists\ n \mid n \in U, \forall p \mid [p,n] \in G_E, p \notin U$ **then** { node n is ready to be evaluated }
        $P \leftarrow \{p \mid [p,n] \in G_E\}$ { Immediate predecessors of n }
        if $n \in G_D$ **then**
            $\phi(n) = 1 - evalProb(\overline{P})$ { Call Algorithm 2 }
            $\chi(n) = \{\bigcup_{p \in P} \chi(p)\}$
            $\delta(n) = \{\bigcap_{p \in P} \delta(p)\}$
            if $n \in G_B$ **then**
                $\psi(n,n) \leftarrow 1$
            end if
        else { $n \in G_C$ }
            $\phi(n) \leftarrow G_M[n] \cdot evalProb(P)$ { Call Algorithm 2 }
            $\chi(n) = (G_B \cap P) \cup \bigcup_{p \in P} \chi(p)$
            $\delta(n) = (G_B \cap P) \cup \bigcup_{p \in P} \delta(p)$
        end if
        $U \leftarrow U - n$ { Mark node n as evaluated }
    else { a cycle is ready for evaluation }
        $C \leftarrow$ cyclic set ready for evaluation (all non-cyclic predecessors evaluated)
        $M \leftarrow evalCycle(C)$ { Call Algorithm 4 }
        $U \leftarrow U \setminus M$ { Mark node set M as evaluated }
    end if
end while

If no individual node n can be found ready for evaluation, it must be the case that at least one cycle in the graph is ready for evaluation. A cycle C is "ready" for evaluation when all entry nodes to the cycle (Definition 3) have been evaluated. In this case, Algorithm 4 will be called to handle the cyclic node set C; the return value will be set $M \subset C$, the subset of nodes in set C that have multiple immediate predecessors. Once these are solved, all remaining (single-predecessor) cyclic nodes can be solved without consideration of the cycle [45]. After a cycle has been evaluated, it will be abstracted to a single node for representation in the graph. In this way, successor nodes following the cycle can be treated acyclically.

After each loop iteration, a single node n or a node set M will be removed from the set of remaining, unevaluated nodes. The loop will re-iterate at this point, continuing until calculations have been made for all nodes in the graph.

Algorithm 2: Pseudocode for computing $evalProb(N)$

Require: Parameter N, such that
$N = \{n_0, n_1, ..., n_j\} \subseteq G_N$, such that $\forall n_i \in N, \phi(n)$ has already been evaluated

{ Find d-separating set D for node set N }
$D \leftarrow \bigcup_{m,n \in N} \chi(m) \cap \chi(n)$
if $D = \emptyset$ **then**
    return $\prod_{n \in N} \phi(n)$
else
    return $\sum_{D} evalCondProb(D, N) \cdot evalProb(D)$
end if

Algorithm 3: Pseudocode for computing $evalCondProb(D, N)$

Require: Parameters D, N, such that
$N \subseteq G_N, |N| \geq 1$, such that $\forall n_i \in N, \phi(n)$ has been evaluated
$D \subseteq G_N$, such that D d-separates all $n \in N$

if $|N| > 1$ **then**
    return $\prod_{n \in N} evalCondProb(D, \{n\})$
else if $N = \{\overline{n}\}$ **then** { N contains exactly one negative element }
    return $1 - evalCondProb(D, \{n\})$
else { $N = \{n\}$, so that N contains exactly one positive element }
    $J \leftarrow \{j \mid j \in D\}$ { All positive elements in D }
    $K \leftarrow \{k \mid \overline{k} \in D\}$ { All negative elements in D }
    if $n \in J$ **then**
        return 1
    end if
    { If n or a dominator of n is negated in D }
    if $n \in K$ or $K \cap \delta(n) \neq \emptyset$ **then**
        return 0
    end if
    { If set D does not affect the value of n }
    if $D \cap \chi(n) = \emptyset$ **then**
        return $\phi(n)$
    end if
    $P \leftarrow \{p \mid [p, n] \in G_E\}$ { Immediate predecessors of n }
    if $n \in G_D$ **then**
        return $1 - evalCondProb(D, \overline{P})$
    else { $n \in G_C$ }
        return $G_M[n] \cdot evalCondProb(D, P)$
    end if
end if

Algorithm 2 calculates $\phi(N)$ — the joint probability of an acyclic node set N, when each element of N's probability has already been calculated. Dynamic programming is used in the implementation; thus if the size of N is 1 the value of $\phi(N)$ is already computed and the cached result will be returned immediately. Otherwise the algorithm finds a d-separating set D such that all $n \in N$ are conditionally independent given D. Based on Definition 1, it suffices to construct D by finding all branching nodes that may affect two or more elements in N. If D is an empty set, then all $n \in N$ are fully independent. In this case, the probability of N is equal to the product of the probabilities of all $n \in N$. Otherwise, the probability is calculated according to Theorem 3.1.

Algorithm 3 – $evalCondProb(D, N)$ – calculates the conditional probability of node set N given d-separating set D. If the node set N contains multiple nodes, then the conditional probability of N given D is equal to the product of the conditional probability of n given D, over all $n \in N$. This is due to the nature of d-separating set D, so that all $n \in N$ are conditionally independent given D. When N contains exactly one element, the algorithm first checks whether a base case has been reached (line 8, 12, 16). If none of the base cases is reached, the algorithm recursively calls itself on the predecessors of n (line 21 and 23).

Algorithm 4: Pseudocode for $evalCycle(C)$ - assessment over nodes in cycle

Require: Parameters C, such that
$C = \{c_0, c_1, ..., c_k\} \subset G_N$ comprises a strongly connected component in G

1: { Trace all acyclic reaching paths }
2: $AllPaths \leftarrow \emptyset$
3: **for all** $\{(p, a) \mid p \notin C,\ a \in C, [p, a] \in G_E\}$ **do**
4: $\quad AllPaths \leftarrow AllPaths \cup tracePaths(a, \{p\}, C)$ { Algorithm 5 }
5: **end for**
6:
7: {Solve all nodes in cycle with multiple predecessors}
8: $Q \leftarrow$ set of entry nodes (non-cyclic nodes, leading into cycle)
9: $M \leftarrow \{m \mid m \in C,\ \exists p, q.(p \neq q, [p, m] \in G_E, [q, m] \in G_E)\}$
10: **for all** $m \in M$ **do**
11: $\quad P \leftarrow \{p \mid p \in AllPaths\ and\ p\ ends\ with\ m\}$
12: $\quad V \leftarrow$ set of possible instances of m (such that each $v \in V$ is attainable by exactly one path $p \in P$)
13: $\quad$ { Find set of nodes appearing in multiple acyclic paths in set P }
14: $\quad T \leftarrow \bigcup_{p,q \in P} (p \cap q)$
15: $\quad$ { Identify d-separating set within cycle for all paths to m }
16: $\quad D \leftarrow (G_C \cap T) \setminus Q$
17: $\quad$ { Find the probability of m, transitively d-separating by Q and D }
18: $\quad$ **if** $m \in G_D$ **then** { m is an OR-node }
19: $\quad\quad \phi(m) \leftarrow 1 - \sum_Q \left(evalProb(Q) \cdot \sum_D \left[\left(\prod_{d \in D} (G_M[d]) \right) \cdot evalCycleNode(Q \cup D, \overline{V}) \right] \right)$
20: $\quad$ **else** { m is an AND-node }
21: $\quad\quad \phi(m) \leftarrow G_M[m] \cdot \left[1 - \sum_Q \left(evalProb(Q) \cdot \sum_D \left[\left(\prod_{d \in D} (G_M[d]) \right) \cdot evalCycleNode(Q \cup D, \overline{V}) \right] \right) \right]$
22: $\quad$ **end if**
23: **end for**
24: **return** M

Handling cycles

Figure 4 contains an attack graph that we will now use as an aid to explain and demonstrate cumulative metric computation over an attack graph containing cycles. This attack graph contains one cycle, node set $\{P_2, P_3, A_4, A_5\}$. Figure 5 shows an equivalent representation of the same attack graph. This unfolded attack graph is acyclic, containing all the unique, acyclic paths that traverse the cycle. Node P_2, for example, can be reached as P_{2A} or P_{2B}; the dotted-line arcs indicate that reaching either of these instances means that P_2 has been reached. In other words, P_{2A} and P_{2B} can be viewed as "partial values" for P_2 and P_2 is true when either of them is true. Intuitively, we will identify all acyclic paths traversing the cycle and use this knowledge to logically identify unique possible instances of cyclic nodes. We can then d-separate over the set of reaching paths to calculate the probability that the cyclic node will be reached.

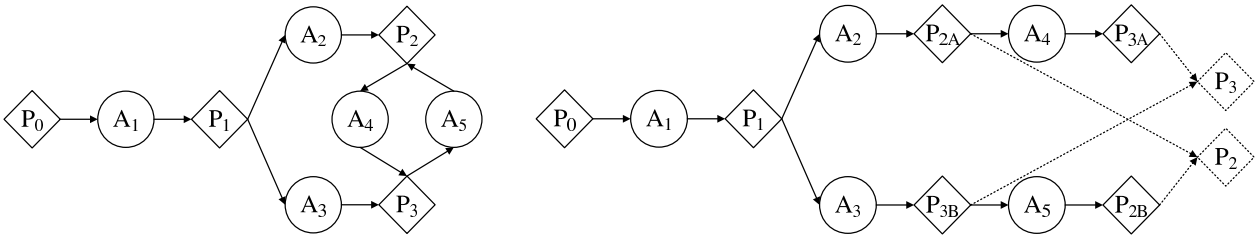


Figure 5: Cyclic attack graph (left, from Figure 4) shown with cycle unfolded (right)

We can simplify the handling of cycles by calculating values only for cyclic nodes with multiple immediate predecessors [45]. For a node N with only one immediate predecessor P, P necessarily dominates N and thus N cannot have any influence on P's probability. The probability of N is then completely dependent on the probability of P according to Propositions 4.1 and 4.2, without double-counting any node's influence. For this reason, once we have computed the probabilities for the nodes with multiple predecessors, the rest of the nodes can be handled in the same manner as in the acyclic case.

Algorithm 4 controls probability calculation for certain nodes within a cyclic node set C. This algorithm is called from the main algorithm [Algorithm 1] to consider cyclic components within the attack graph. Given a cyclic node set C, this algorithm first identifies all acyclic reaching paths that lead from outside of the cycle to a node within the cycle [Algorithm 5]. It then calculates a risk assessment value for each $m \in C$ such that m has multiple predecessors (multiple arcs leading to m).

For each multi-predecessor node $m \in C$, there is a set V of "partial values," or different instantiations of m reached by different paths. The node m will be true except when all of these "partial values" are unreachable. To find a d-separating set D within the cycle for all $v \in V$, the algorithm identifies all attack-step nodes appearing on multiple paths in P, excluding the entry nodes Q (which are not in the cycle). Recall that the d-separating set for any node is earlier stated to be a subset of G_B (Section 3.3.1). However, each node $B \in G_B \subset G_D$ along a specified partial path will have probability equal to its preceding attack-step node (Proposition 4.1); the attack-step nodes appearing on multiple paths will precede any branching nodes and thus, for purposes of calculation, represent a sufficient set to determine a d-separating set D. Because all possible acyclic paths are considered, the set of attack-step nodes lying along each path is unique to that path. We will not use the full probability of these nodes, but only the individual component metrics associated with them, calculating the likelihood of success for each path within the cycle. Together with a set of cycle entry nodes Q, we can now solve for the probability of node m.

If m is a privilege node, then m is true except when all $v \in V$ are false. To find this value, we must marginalize over sets Q and D, eliminating their conditional influence on m. If m is an attack-step node, it is similarly computed, considering also the individual component metric value $G_M[m]$. Because we are using

only the component metric values for D, the order and derivation of each attack path is unimportant; only the product of the individual probabilities is needed.

Once probabilities have been calculated for all multi-predecessor nodes $M \subset C$, the set M will be returned to the main algorithm to be marked as evaluated. Again, all single-predecessor cycle nodes will be treated as acyclic, to reduce overall run-time of the algorithm. In this way, the cycle is properly evaluated, so that no node influences its own probability.

Algorithm 5: Pseudocode for $tracePaths(n, P, C)$ - tracing acyclic paths through cycle

Require: Parameters n, P, C, such that
$n \in C$, a strongly connected component
$P = \{p_0, p_1, ..., p_m\} \subset Q \cup C$ is a sequence of nodes comprising a simple path to node n

1: $P \leftarrow P :: n$ { Append n to P }
2: $Paths \leftarrow \{P\}$
3: { Get set of successors to n not appearing in path set P }
4: $S \leftarrow \{s \mid s \in C, s \notin P, [n, s] \in G_E\}$
5: **for all** $s \in S$ **do**
6: $\quad Paths \leftarrow Paths \cup tracePaths(s, P \cup n, C)$
7: **end for**
8: **return** $Paths$

Algorithm 5 – $tracePaths(n, P, C)$ – performs a logical "unfolding" of the strongly connected graph component C, comparable to the graphical unfolding discussed in Section 3.3.2 in that all acyclic reaching paths are identified for each node $c \in C$. The paths are primed by cycle entry nodes. As a node is visited, a unique, non-cyclic path set (sufficient for obtaining one instance of that node) will be saved for future reference, representing one non-cyclic path to that node. This algorithm will never produce duplicate paths to any one node. Whenever the algorithm discovers a connecting node that is already in the path set P, this avenue of exploration is ceased, to prevent cyclic paths from occurring. In this way, all of the exploratory paths will eventually end, so the algorithm will terminate successfully. These partial paths and values are stored for use in calculating probability for multi-predecessor cycle nodes in Algorithm 4.

Algorithm 6 – $evalCycleNode(D, N)$ – calculates conditional probability for a set N of node instances given a d-separating set D. D is the union of the set of cycle entry nodes and the attack-step nodes appearing on multiple partial paths within the cycle. This algorithm is called from $evalCycle(N)$ [Algorithm 4]. It is similar in many respects to $evalCondProb(D, N)$ [Algorithm 3], but utilizes the acyclic reaching paths identified within the cycle to perform its probability calculations.

If node set N contains multiple nodes, then the product of the conditional probability for all $n \in N$ is calculated and returned. If node set N contains exactly one node n, and that node is disabled, then $\overline{n}$ holds except when n is conditionally true. This value, the inverse, will be calculated and returned.

If node set N contains exactly one node n, and that node is enabled, we must calculate a conditional probability for n given D. Set P contains all attack-step nodes in entry nodes or cycle nodes that lie along the acyclic reaching path to node instance n. If some node in path P is negated in D, then n cannot be reached by path P and so has probability zero. Otherwise, we must remove from P all nodes that are forced true in D, since these can have no effect on the probability of n by path P.

Algorithm 6: Pseudocode for $evalCycleNode(D, N)$

Require: Parameters D, N, such that
D d-separates all $n \in N$, and
$N = \{n_0, n_1, ..., n_j\}$ are partial values for some node $n \in C$

if $|N| > 1$ **then**
  return $\prod_{n \in N} evalCycleNode(D, \{n\})$
else if $N = \{\overline{n}\}$ **then**
  return $1 - evalCycleNode(D, \{n\})$
else { $N = \{n\}$, so that N contains exactly one element, enabled }
  $P \leftarrow$ partial path (set of attack-step nodes) leading to node instance n
  $J \leftarrow \{j \mid j \in D\}$ { All enabled nodes in D }
  $K \leftarrow \{k \mid \overline{k} \in D\}$ { All disabled nodes in D }

  { If node in path is negated, n cannot be reached by path P }
  if $K \cap P \neq \emptyset$ **then**
    return 0
  end if

  { Discard any path nodes forced true }
  $P \leftarrow P \setminus J$

  return $\prod_{p \in P} G_M[p]$
end if

Once all such fixed values have been accounted for, we know that the node is reachable along path P. Because the attack-step nodes in P are treated independently, knowing the probability that all will jointly succeed gives us the likelihood that an attacker will succeed along path P. The algorithm therefore calculates the product of the component metric values for all remaining nodes in path P; this value is the probability that n is true by path P, given set D.

5 Evaluation Results

We have implemented our cumulative metric algorithm in the Python language. To evaluate the effectiveness of using the metric model for risk assessment, we carried out three lines of study:

- Testing how the metrics can help making security hardening decisions.
- Evaluation on a production system to gain empirical experience of the metric model.
- Testing the scalability of metric computation.

5.1 Evaluating the use of metrics to guide hardening decisions

For this evaluation, we used the small example network in Figure 2. For this configuration, the cumulative metrics result is shown in the "Initial scenario" column of Table 1. In the table, the numbers indicate the likelihood various machines can be successfully compromised by an attacker.

Host	Initial scenario	Patch web server	Patch db server	Patch workstations	Change network access
Web server	0.2	0	0.2	0.2	0.2
Database server	0.47	0.43	0	0.12	0.12
Workstations	0.74	0.72	0.74	0	0.74

Table 1: Probabilities of compromise for hosts in Figure 2 (columns reflect different scenarios)

Consider again the sample network configuration (and associated attack graph) shown in Figure 2. When considering improvements in network security, a network administrator is constrained in terms of money and time. For example, some changes, though preferable, may not be feasible because of the time necessary to make the change and the system downtime that would occur while the change was made. Considering the network topology in this example, it is not immediately clear which of the vulnerabilities should be patched first, assuming that a fix is available for each of the three, or what other changes could be made to reduce security risk. The columns in Table 1 show new metric values based on various mitigation options: patching different vulnerabilities or changing the network access rules so that the user workstations cannot access the database server.

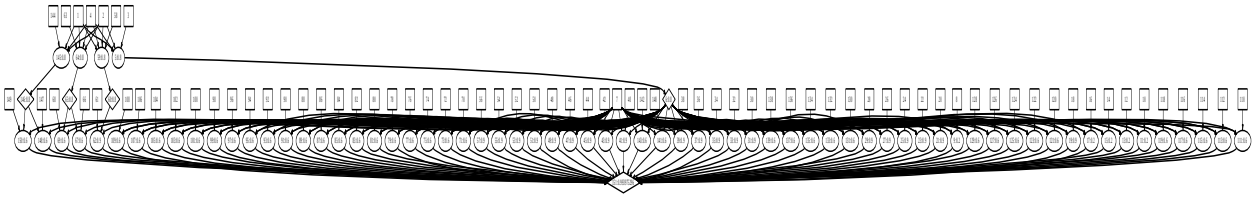
Patching the vulnerability on the web server would eliminate the known risk of compromise for the web server, but have little effect on the other two hosts. The web server does not contain sensitive information, so protecting this host first may not be the best choice.

Patching the vulnerability on the database server would eliminate the known risk of compromise for the database server, but have no effect on the risk in the other two hosts, since privileges on the database server do not enable new attacks on the other hosts. This option would secure the sensitive data on the database server, which may be most desirable, but at the cost of having a period of downtime on the database server which may affect business revenues.

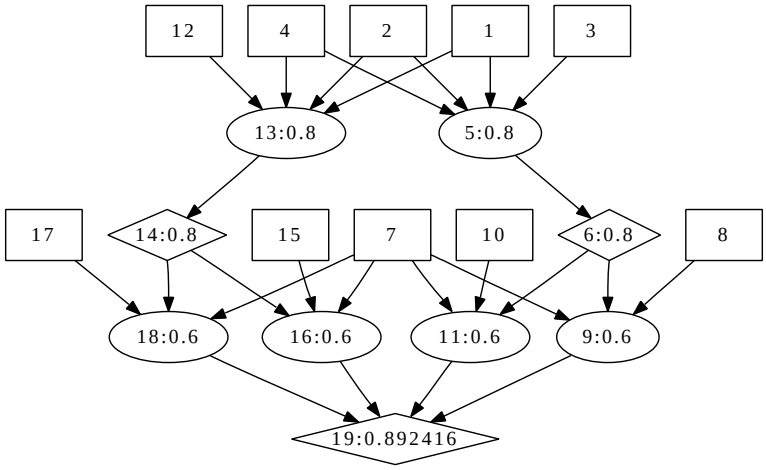
Patching the vulnerability on the user workstations would eliminate the risk to the workstations, as well as significantly reduce the risk to the database server, but the risk to the web server would remain unchanged. This may be a more feasible solution since downtime on the workstations is less costly than on the server, especially if the patching can be done outside of normal working hours.

Network configuration changes can also have drastic effects on the security risk. The final column in the table shows the effect of blocking network access from the workstations to the database server. This option eliminates an attack path to the database server that depends on privileges on the workstations, lowering the risk of compromise for the database server, but it leaves the web server and workstations vulnerable. Depending on other resource constraints and asset valuations, this may also be a viable solution.

There may not be a single "best" option for all organizations. Indeed, different administrators could easily make different choices, based on the perceived importance of the hosts and the expected time necessary to enact proposed changes, as well as human resources available. The quantitative security metrics make clear the effects emerging from each of these possible changes, thereby providing a network administrator with objective data beneficial for judging the relative value of each option. Our cumulative metrics could also be combined with quantitative asset values and costs of various mitigation options and used as input in an optimization engine such as proposed by earlier works [11, 28, 38, 46] to automatically compute optimal hardening options.



(a)



(b)

Figure 6: Attack graphs from two production servers

5.2 Insights gained from evaluation on a production system

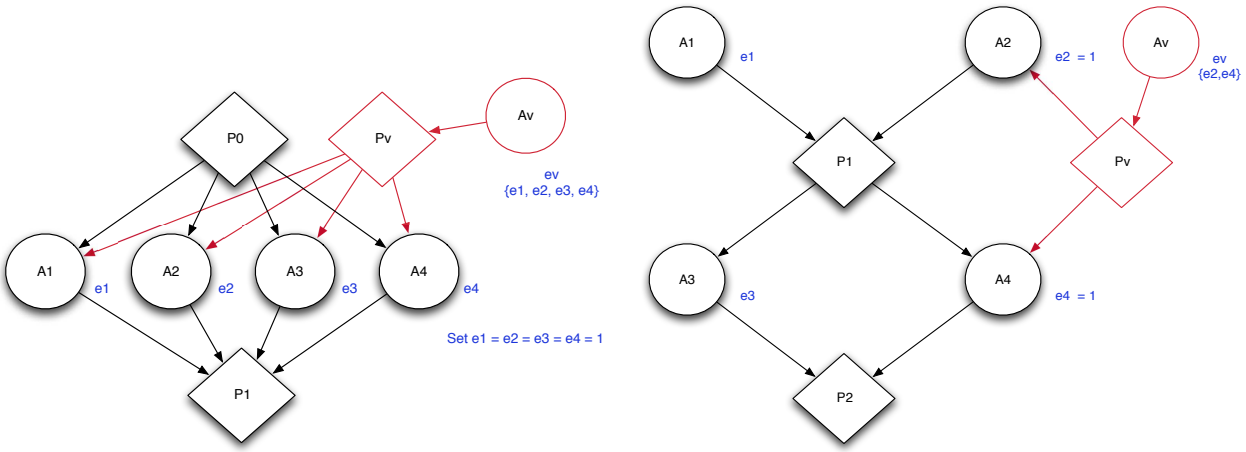
To study how the metric model works on production systems, we have conducted an OVAL vulnerability scan on all the Windows servers and workstations in the CIS departmental network of Kansas State University. OVAL[2] is part of the SCAP standard [36] for communicating security information. It is a language for reporting discovered known vulnerabilities on a host. The OVAL scan is performed periodically, and the vulnerability assessment reports are automatically sent to a central data repository, providing continuous fresh data for evaluating our metric model.

The departmental network has a fairly simple network topology. There is no firewall control in the internal network, so all the servers can talk to each other. The servers are well-managed so that most of the service program vulnerabilities have been patched. However, there are still a large number of client-side and local vulnerabilities on each machine. These vulnerabilities pose relatively low perceived risk, since it is very unlikely that a user will access the server to launch those client programs, and as long as no user is compromised, the local vulnerabilities cannot pose any danger to the systems. For this reason, these systems are good candidates for evaluating the security metric methods — there is a significant amount of residual risk that needs to be quantified. The calculated security metrics can be used in comparison to the system administrator's rationale for delayed patching of these non-critical vulnerabilities.

As we ran our metric algorithm on the model, a problem quickly became obvious. It is best illustrated by the results in Figure 6, which shows the attack graphs for two servers.[3] Server (a) has many more vulnerabilities than (b), as can be seen immediately from the density of the attack graphs. The attack graph for (a) is so wide that it is shown almost like a line in the limited space on paper. However, when it comes

[2](http://oval.mitre.org/)

[3]The square nodes are configuration nodes which have been omitted in the previous attack-graph examples.



(a) (b)

Figure 7: Modeling artifacts for capturing hidden correlations

to cumulative security metrics results, the two do not show a significant difference: 0.99 vs. 0.89. Indeed, most of the machines we have evaluated had at least a dozen attack paths, which raises the likelihood of attack success to almost 1. This necessarily prompts the question: is this a realistic measurement of risk? We presented the result to the system administrator. His opinion was that it depends on the underlying differences in the existing vulnerabilities. A machine with 10 vulnerabilities in a single application has a lower risk than a machine with 10 vulnerabilities in 10 different applications, because vulnerabilities in the same application may not give an attacker significant advantage in exploiting them. The exploits for these vulnerabilities may not be truly independent: if the attacker lacks skill or experience in exploiting a specific application, the presence of more vulnerabilities in that application will not help a lot. On the other hand, if the 10 vulnerabilities are dispersed across 10 different applications, the chance that the attacker possesses the skill to exploit at least one of them will be significantly higher. This dependency among exploits based on the similarity of applications is not captured by the attack-graph structure. For the attack graph in (a), there are only four vulnerable applications but 67 distinct vulnerabilities, of which 62 are in the same application (Firefox). For (b), there are four vulnerabilities in two applications. Intuitively, (a) has a much higher risk than (b), more than the difference between the two calculated metric numbers indicate.

This observation has also led to another manifestation of the hidden-correlation problem. Suppose the same vulnerability appears on two different hosts and the attacker needs to exploit both of them in a multi-stage attack. If he has succeeded in exploiting the first one, he will very likely succeed with the second. This dependency also is not captured by the attack graph, which could lead to a lower evaluation of risk than really exists. If the chance of success for the attacker to exploit the vulnerability is 0.6, the likelihood for him to succeed in the two-stage attack chain should be very close to 0.6, since a successful attack in the first step will lead, with a likelihood of almost 1, to success in the second. Based on the attack-graph model, however, our metric algorithm will produce a result of 0.36, by multiplying the two probabilities.

Modeling artifacts for capturing hidden correlations To correctly account for such hidden correlations among attack steps, we introduced additional modeling artifacts in attack graphs so that the hidden correlations become explicit. Essentially, we grouped vulnerabilities for the same application and introduced a virtual modeling node to capture the case in which an attacker has succeeded in exploiting the application. This applies to both vulnerabilities on the same host (Figure 7.a) and on different hosts (Figure 7.b). The success likelihood of exploiting this vulnerability is associated with the added virtual exploit node A_v, and

the original exploit nodes are associated with a likelihood of 1. This makes the hidden correlation explicit in the graphical model. In (a), if the attacker fails in exploiting the vulnerability, he will fail on all four instances $A_1, ..., A_4$, thus avoiding an incorrect increase in the success likelihood of P_1 to almost 1. In (b), if the attacker succeeds in exploiting the vulnerability, he will succeed in both attacks A_2 and A_4, thus correctly evaluating the success likelihood of the two-step chain.

After this revision of the graphical model, our risk assessment algorithm produces significant different metrics for the two systems: 0.98 vs. 0.73; the difference between the two values is more consistent with the intuitive assessment provided by experienced system administrators. Moreover, the number of applications that enable an attacker to compromise the system also became obvious in the new attack-graph model. The results of this grouping are shown in Figure 8.

However, the new metric numbers would still imply that both hosts were at high risk and the vulnerabilities should be addressed. This was not adopted by the system administrator, after looking at the applications that had the vulnerabilities — most of them existed in client applications rarely used on the servers. Thus the likelihood a user will launch one of those vulnerable programs was very low. In our evaluation we used a fixed value (0.8) to represent this likelihood, resulting in the high metric values. This indicates that to obtain more realistic security metrics, we need to assign this input parameter based on the knowledge of whether and how often a client application is used, which we leave as future work.

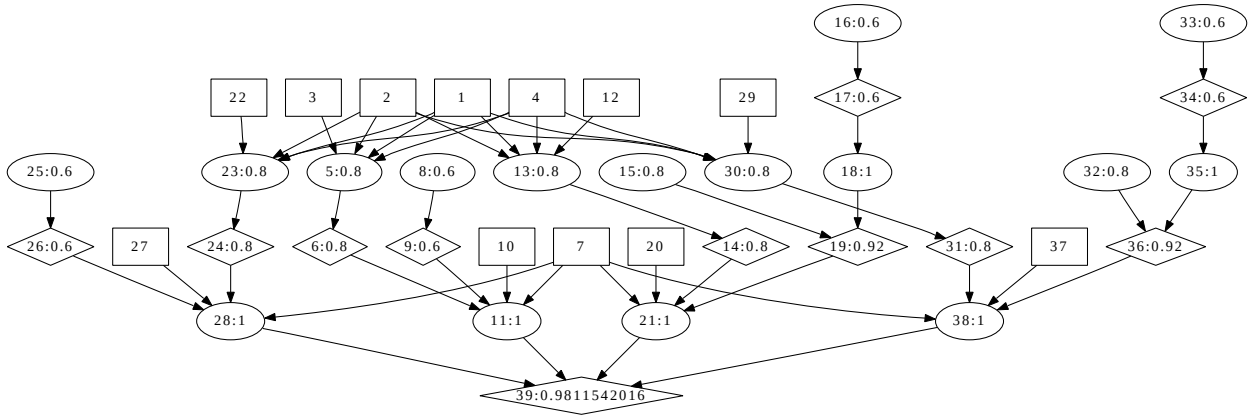
We stress that *the reason we can refine our metric model and interpret the results based on such empirical observations is largely due to the fact that the metric calculation is sound.* This ensures that when we get a non-intuitive result, we can easily trace it back to the root cause, without having to wonder whether it was due to errors introduced in the calculation.

5.3 Testing scalability

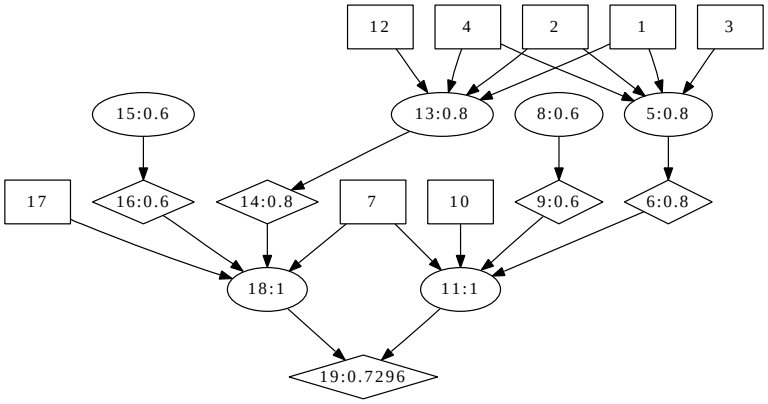
Table 2: **Network Reachability**

source	*destination*	*protocol*	*port*
Webservers	Database Servers	tcp	3306
Internet	Webservers	tcp	80
Workstations1	Internet	*	*
Workstations1	Fileservers	nfs	
Workstations2	Internet	*	*
Workstations2	Historians	*	*
Workstations2	Fileservers	nfs	
Workstations2	Mailservers	tcp	25
Mailservers	Internet	tcp	25

In order to test the scalability of our approach, we constructed several testing models based on networks of varying sizes and complexity, created MulVAL input files representing each network, and evaluated them with the current implementation of our algorithm. Figure 9 indicates the general network topology of the hypothetical network configurations used in our testing. The reachability information between various groups of machines is given in Table 2 (* is wild card). The vulnerability information can be found in Table 3. We performed our tests based on abstracted network models. In an abstracted model, each host represents a group of hosts having the same network reachability and similar configuration features (*e.g.*, they may be under the same software package management server). Therefore, every machine in Table 2 and Figure 9 represents a group of hosts. The run-time for metric computation of this network model was extremely short (less than a second).



(a)



(b)

Figure 8: Attack graphs with modeling artifacts for capturing hidden correlations

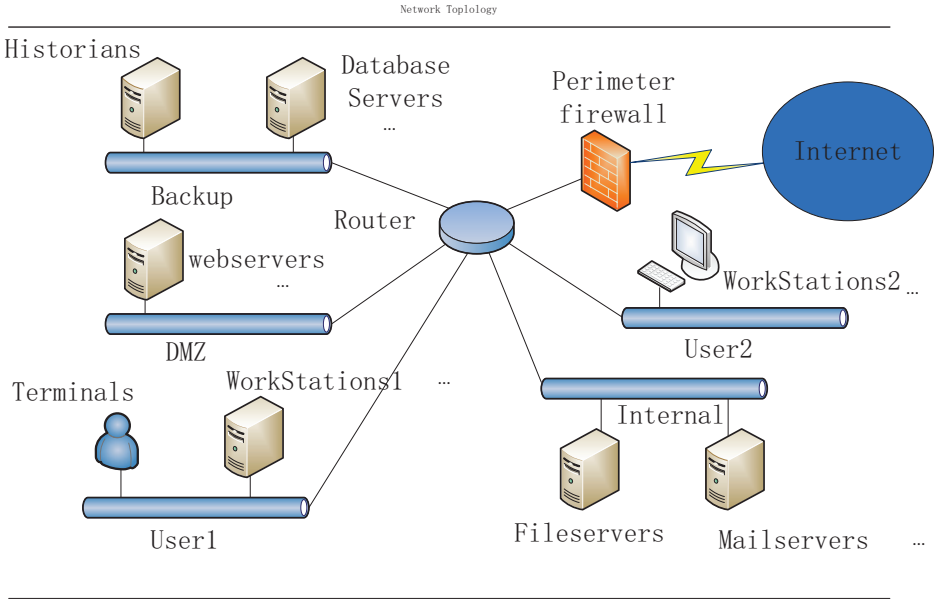


Figure 9: General network topology

Table 3: **Host vulnerabilities**

Host	# of vulnerabilities		
	Local	Remote Client	Remote Server
Webservers	0	1	1
Database servers	0	0	1
workstations1	0	1	1
Fileservers	1	0	1
Workstations2	0	1	1
Historians	0	0	1
Mailservers	0	0	1

To further test the limit of the algorithm's scalability, we picked 10 and 100 as the number of host groups in each subnet, where every machine can reach another subnet. Each time, we ensure that the host group with the deepest attack path (the one having the largest number of inter-subnet hops from the initial location of the attacker) is in the set of attack goals. The deepest goal will take the longest running time on the metric calculation. We also added 10 vulnerabilities per host with all three types (local, remote client, and remote server). The running time of the algorithm for these scenarios is shown in Table 4.

Table 4: **Scalability of Risk Assessment**

Num of host groups per subnet \ depth of the deepest inter-subnet attack hops	1	2	3
10	2s	2s	3s
100	1m38s	16m14s	46m36s

The limiting factors in the current algorithm and implementation are the size of the d-separating set (the number of nodes which must be marginalized in calculating conditional probability values) and the number of paths that must be considered in the calculation of each multi-predecessor node within a cycle. As either of these increases, the number of recursive calls made by the algorithm increases, and the evaluation time grows correspondingly. In the worst case, the computational increase could be exponential. However, as Table 4 shows, for realistic network settings, our algorithm can finish metric calculation for sufficiently large network configurations. The biggest case in the configuration consists of 100 host groups per subnet, 3 inter-subnet attack steps in the longest attack path, and 10 vulnerabilities of all three types per host. We believe this is a reasonable estimation on the large cases the tool will need to handle in reality. It is believed that most enterprise intrusions will take no more than three inter-subnet steps. After grouping and abstraction, 100 host groups per subnet and 10 vulnerabilities with all types per host represents a significantly large scenario for risk assessment analysis. For the worst-case scenario, our implementation of the algorithm can finish computation in less than an hour.

6 Conclusion

We have presented an approach to aggregating vulnerability metrics in an enterprise network through attack graphs. Our approach is sound in that, given component metrics which characterize the likelihood that individual vulnerabilities can be successfully exploited, the model computes a numeric value representing the cumulative likelihood that an attacker could succeed in gaining a specific privilege or carrying out an attack in the network. Our method handles both cyclic and shared dependencies in attack graphs correctly, surpassing

previous efforts on this problem. Preliminary testing results show the effectiveness and practicality of the approach and how it can be used to help system administrators decide between risk mitigation options.

7 Acknowledgment

This material is based upon work supported by U.S. National Science Foundation under award no. 0954138 and 1018703, and HP Labs Innovation Research Program. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation, or Hewlett-Packard Development Company, L.P.

References

[1] Ehab Al-Shaer, Latif Khan, and M. Salim Ahmed. A comprehensive objective network security metric framework for proactive security configuration. In *ACM Cyber Security and Information Intelligence Research Workshop*, 2008.

[2] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of 9th ACM Conference on Computer and Communications Security*, Washington, DC, November 2002.

[3] Zahid Anwar, Ravinder Shankesi, and Roy H. Campbell. Automatic security assessment of critical cyber-infrastructures. In *Proceedings of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, July 2008.

[4] Davide Balzarotti, Mattia Monga, and Sabrina Sicari. Assessing the risk of using vulnerable components. In *Proceedings of the 2nd ACM workshop on Quality of protection*, 2005.

[5] Steven Bellovin. On the brittleness of software and the infeasibility of security metrics. *IEEE Security & Privacy*, 2006.

[6] Marc Dacier, Yves Deswarte, and Mohamed Kaâniche. Models and tools for quantitative assessment of operational security. In *IFIP SEC*, 1996.

[7] J. Dawkins and J. Hale. A systematic approach to multi-stage network attack analysis. In *Proceedings of Second IEEE International Information Assurance Workshop*, pages 48 – 56, April 2004.

[8] Rinku Dewri, Nayot Poolsappasit, Indrajit Ray, and Darrell Whitley. Optimal security hardening using multi-objective optimization on attack tree models of networks. In *14th ACM Conference on Computer and Communications Security (CCS)*, 2007.

[9] Marcel Frigault and Lingyu Wang. Measuring network security using Bayesian network-based attack graphs. In *Proceedings of the 3rd IEEE International Workshop on Security, Trust, and Privacy for Software Applications (STPSA'08)*, 2008.

[10] Marcel Frigault, Lingyu Wang, Anoop Singhal, and Sushil Jajodia. Measuring network security using dynamic Bayesian network. In *Proceedings of the 4th ACM workshop on Quality of protection*, 2008.

[11] John Homer and Xinming Ou. SAT-solving approaches to context-aware enterprise network security management. *IEEE JSAC Special Issue on Network Infrastructure Configuration*, 2009.

[12] Nwokedi Idika and Bharat Bhargava. Extending attack graph-based security metrics and aggregating their application. *IEEE Transactions on Dependable and Secure Computing*, 9(1), 2012.

[13] Kyle Ingols, Matthew Chu, Richard Lippmann, Seth Webster, and Stephen Boyer. Modeling modern network attacks and countermeasures using attack graphs. In *25th Annual Computer Security Applications Conference (ACSAC)*, 2009.

[14] Kyle Ingols, Richard Lippmann, and Keith Piwowarski. Practical attack graph generation for network defense. In *22nd Annual Computer Security Applications Conference (ACSAC)*, Miami Beach, Florida, December 2006.

[15] Sushil Jajodia and Steven Noel. Advanced cyber attack modeling analysis and visualization. Technical Report AFRL-RI-RS-TR-2010-078, Air Force Research Laboratory, March 2010.

[16] Sushil Jajodia, Steven Noel, and Brian O'Berry. Topological analysis of network attack vulnerability. In V. Kumar, J. Srivastava, and A. Lazarevic, editors, *Managing Cyber Threats: Issues, Approaches and Challanges*, chapter 5. Kluwer Academic Publisher, 2003.

[17] Finn V. Jensen and Thomas D. Nielsen. *Bayesian Networks and Decision Graphs*. Springer Verlag, 2 edition, 2007.

[18] Daniel Geer Jr., Kevin Soo Hoo, and Andrew Jaquith. Information security: Why the future belongs to the quants. *IEEE SECURITY & PRIVACY*, 2003.

[19] Wei Li, Rayford B. Vaughn, and Yoginder S. Dandass. An approach to model network exploitations using exploitation graphs. *SIMULATION*, 82(8):523–541, 2006.

[20] Richard Lippmann and Kyle W. Ingols. An annotated review of past papers on attack graphs. Technical report, MIT Lincoln Laboratory, March 2005.

[21] Richard P. Lippmann, Kyle W. Ingols, Chris Scott, Keith Piwowarski, Kendra Kratkiewicz, Michael Artz, and Robert Cunningham. Evaluating and strengthening enterprise network security using attack graphs. Technical Report ESC-TR-2005-064, MIT Lincoln Laboratory, October 2005.

[22] P.K. Manadhata and J.M. Wing. An attack surface metric. *IEEE Transactions on Software Engineering*, June 2010.

[23] Pratyusa Manadhata, Jeannette Wing, Mark Flynn, and Miles McQueen. Measuring the attack surfaces of two FTP daemons. In *Proceedings of the 2nd ACM workshop on Quality of protection*, 2006.

[24] John McHugh. Quality of protection: measuring the unmeasurable? In *Proceedings of the 2nd ACM workshop on Quality of protection (QoP)*, Alexandria, Virginia, USA, 2006.

[25] John McHugh and James Tippett, editors. *Workshop on Information-Security-System Rating and Ranking (WISSRR)*. Applied Computer Security Associates, May 2001.

[26] Vaibhav Mehta, Constantinos Bartzis, Haifeng Zhu, Edmund Clarke, and Jeannette Wing. Ranking attack graphs. In *Proceedings of Recent Advances in Intrusion Detection (RAID)*, September 2006.

[27] Peter Mell, Karen Scarfone, and Sasha Romanosky. *A Complete Guide to the Common Vulnerability Scoring System Version 2.0*. Forum of Incident Response and Security Teams (FIRST), June 2007.

[28] Steven Noel, Sushil Jajodia, Lingyu Wang, and Anoop Singhal. Measuring security risk of networks using attack graphs. *International Journal of Next-Generation Computing*, 1(1), July 2010.

[29] Rodolphe Ortalo, Yves Deswarte, and Mohamed Kaâniche. Experimenting with quantitative evaluation tools for monitoring operational security. *IEEE Transactions on Software Engineering*, 25(5), 1999.

[30] Xinming Ou. *A logic-programming approach to network security analysis*. PhD thesis, Princeton University, 2005.

[31] Xinming Ou, Wayne F. Boyer, and Miles A. McQueen. A scalable approach to attack graph generation. In *13th ACM Conference on Computer and Communications Security (CCS)*, pages 336–345, 2006.

[32] Xinming Ou, Sudhakar Govindavajhala, and Andrew W. Appel. MulVAL: A logic-based network security analyzer. In *14th USENIX Security Symposium*, 2005.

[33] Joseph Pamula, Sushil Jajodia, Paul Ammann, and Vipin Swarup. A weakest-adversary security metric for network configuration security analysis. In *Proceedings of the 2nd ACM workshop on Quality of protection*, 2006.

[34] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman, 1988.

[35] Cynthia Phillips and Laura Painton Swiler. A graph-based system for network-vulnerability analysis. In *NSPW '98: Proceedings of the 1998 workshop on New security paradigms*, pages 71–79. ACM Press, 1998.

[36] Stephen Quinn, David Waltermire, Christopher Johnson, Karen Scarfone, and John Banghart. *The Technical Specification for the Security Content Automation Protocol (SCAP): SCAP Version 1.0*. The National Institute of Standards and Technology Special Publication 800-126, 2009.

[37] Diptikalyan Saha. Extending logical attack graphs for efficient vulnerability analysis. In *Proceedings of the 15th ACM conference on Computer and Communications Security (CCS)*, 2008.

[38] Reginald Sawilla and Craig Burrell. Course of action recommendations for practical network defence. Technical Report TM-2009-130, Defence Research and Development Canada, 2009.

[39] Reginald Sawilla and Xinming Ou. Identifying critical attack assets in dependency attack graphs. In *13th European Symposium on Research in Computer Security (ESORICS)*, Malaga, Spain, October 2008.

[40] Oleg Sheyner. *Scenario Graphs and Attack Graphs*. PhD thesis, Carnegie Mellon, April 2004.

[41] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 254–265, 2002.

[42] Laura P. Swiler, Cynthia Phillips, David Ellis, and Stefan Chakerian. Computer-attack graph generation tool. In *DARPA Information Survivability Conference and Exposition (DISCEX II'01)*, volume 2, June 2001.

[43] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[44] T. Tidwell, R. Larson, K. Fitch, and J. Hale. Modeling Internet attacks. In *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security*, West Point, NY, June 2001.

[45] Lingyu Wang, Tania Islam, Tao Long, Anoop Singhal, and Sushil Jajodia. An attack graph-based probabilistic security metric. In *Proceedings of The 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSEC'08)*, 2008.

[46] Lingyu Wang, Steven Noel, and Sushil Jajodia. Minimum-cost network hardening using attack graphs. *Computer Communications*, 29:3812–3824, November 2006.

[47] Lingyu Wang, Anoop Singhal, and Sushil Jajodia. Measuring network security using attack graphs. In *Third Workshop on Quality of Protection (QoP)*, 2007.

[48] Lingyu Wang, Anoop Singhal, and Sushil Jajodia. Measuring the overall security of network configurations using attack graphs. In *Proceedings of 21th IFIP WG 11.3 Working Conference on Data and Applications Security (DBSEC'07)*, 2007.

Appendices

A Sample Symbolic Computations

A.1 Computation for Acyclic Graph

We will now work through the sample graph shown in Figure 3, demonstrating our approach and showing its effectiveness at recognizing and correctly handling shared dependencies within the graph. This graph is acyclic, so we can recursively calculate the probability value for each node, utilizing previously calculated individual probability values and computing joint probabilities only as needed. A table showing all calculated values will be included at the end of this example.

We begin with the root node, P_0. The probability that node P_0 is true is G_V which is assumed to be 1. Thus $\phi(P_0) = 1$. As the root node, P_0 has no preceding nodes, so $\chi(P_0) = \{\ \}$ and $\delta(P_0) = \{\ \}$.

Now that we have calculated $\phi(P_0)$, we can calculate for either A_1 or A_6. Let us next calculate for A_6. A_6 has exactly one predecessor, P_0. So, $\phi(A_6) = G_M[A_6] \cdot \phi(P_0) = e_6 \cdot 1 = e_6$; furthermore, $\chi(A_6) = \delta(A_6) = \{P_0\}$.

We cannot yet evaluate node P_2, because not all of its predecessors have been evaluated. We will return, then, to evaluate node A_1. Similar to the calculation for A_6, $\phi(A_1) = G_M[A_1] \cdot \phi(P_0) = e_1$; $\chi(A_1) = \delta(A_1) = \{P_0\}$.

We can now evaluate node P_1. Node P_1 also has only one predecessor. Thus, $\phi(P_1) = 1 - \phi(\bar{A}_1) = \phi(A_1) = e_1$; $\chi(P_1) = \delta(P_1) = \{P_0\}$.

From this point, we could evaluate either A_2 or A_3. Let us next calculate for A_2. $\phi(A_2) = G_M[A_2] \cdot \phi(P_1) = e_2 \cdot e_1 = e_1e_2$; $\chi(A_2) = \delta(A_2) = \{P_0, P_1\}$.

Now both predecessors to P_2 have been solved and we can calculate for this node. Proposition 4.1 specifies the calculation for a privilege node with multiple predecessors. So, $\phi(P_2) = 1 - \phi(\{\overline{A}_2, \overline{A}_6\})$. In previous cases with single predecessors, we already knew the probability of the predecessor, but in this case we do not yet know the joint probability of $\phi(\{\overline{A}_2, \overline{A}_6\})$ and so must solve for it. To calculate $\phi(\{\overline{A}_2, \overline{A}_6\})$, we must find a d-separating set for these two nodes so that we can utilize Theorem 3.1. One such set can be found by taking the intersection of the χ sets for these nodes, so that $D = \chi(A_2) \cap \chi(A_6) = \{P_0\}$. D contains all branch nodes that diverge to paths leading to A_2 and A_6, which should be sufficient to d-separate the nodes (Definition 1). Using the set D, we can now solve for $\phi(\{\overline{A}_2, \overline{A}_6\})$:

$$
\begin{aligned}
&= \sum_{P_0} \psi(\{P_0\}, \overline{A}_2, \overline{A}_6)\phi(\{P_0\}) \\
&= \sum_{P_0} \psi(\{P_0\}, \overline{A}_2)\psi(\{P_0\}, \overline{A}_6)\phi(\{P_0\}) \\
&= \psi(\{P_0\}, \overline{A}_2)\psi(\{P_0\}, \overline{A}_6)\phi(\{P_0\}) + \\
&\quad \psi(\{\overline{P}_0\}, \overline{A}_2)\psi(\{\overline{P}_0\}, \overline{A}_6)\phi(\{\overline{P}_0\}) \\
&= (1 - e_1e_2)(1 - e_6)(1) + (1)(1)(0) \\
&= 1 - e_1e_2 - e_6 + e_1e_2e_6
\end{aligned}
$$

Then, $\phi(P_2) = 1 - \phi(\{\overline{A}_2, \overline{A}_6\}) = 1 - (1 - e_1e_2 - e_6 + e_1e_2e_6) = e_1e_2 + e_6 - e_1e_2e_6$. Also, $\chi(P_2) = \chi(A_2) \cup \chi(A_6) = \{P_0, P_1\}$, and $\delta(P_2) = \delta(A_2) \cap \delta(A_6) = \{P_0\}$.

Nodes A_3, A_4, A_5, P_3 are calculated very similarly to nodes we've already seen here, so we will skip over the details of these. The resulting values are in Table 5.

Finally, we evaluate for node P_4, a privilege node with multiple predecessors, so again we will apply Proposition 4.1: $\phi(P_4) = 1 - \phi(\{\overline{A}_4, \overline{A}_5\})$. The d-separating set for $\{A_4, A_5\}$ is $D = \chi(A_4) \cap \chi(A_5) = \{P_0, P_1\}$. Using the set D, we can now solve for $\phi(\{\overline{A}_4, \overline{A}_5\})$:

$$
\begin{aligned}
&= \sum_{P_0,P_1} \psi(\{P_0,P_1\},\overline{A}_4)\psi(\{P_0,P_1\},\overline{A}_5)\phi(\{P_0,P_1\}) \\
&= \psi(\{P_0,P_1\},\overline{A}_4)\psi(\{P_0,P_1\},\overline{A}_5)\phi(\{P_0,P_1\}) + \\
&\quad \psi(\{P_0,\overline{P}_1\},\overline{A}_4)\psi(\{P_0,\overline{P}_1\},\overline{A}_5)\phi(\{P_0,\overline{P}_1\}) + \\
&\quad \psi(\{\overline{P}_0,P_1\},\overline{A}_4)\psi(\{\overline{P}_0,P_1\},\overline{A}_5)\phi(\{\overline{P}_0,P_1\}) + \\
&\quad \psi(\{\overline{P}_0,\overline{P}_1\},\overline{A}_4)\psi(\{\overline{P}_0,\overline{P}_1\},\overline{A}_5)\phi(\{\overline{P}_0,\overline{P}_1\}) \\
&= (1 - e_4(e_2 + e_6 - e_2e_6))(1 - e_3e_5)(e_1) + \\
&\quad (1 - e_4e_6)(1)(1 - e_1) + (1)(1)(0) + (1)(1)(0) \\
&= 1 - e_1e_2e_4 + e_1e_2e_4e_6 - \\
&\quad e_1e_3e_5 - e_4e_6 + e_1e_3e_4e_5(e_2 + e_6 - e_2e_6)
\end{aligned}
$$

Then, $\phi(P_4) = 1 - \phi(\overline{A}_4, \overline{A}_5) = e_4e_6 + e_1(e_2e_4 + e_3e_5) - e_1e_2e_4e_6 - e_1e_3e_4e_5(e_2 + e_6 - e_2e_6)$. Also, $\chi(P_4) = \chi(A_4) \cup \chi(A_5) = \{P_0, P_1\}$, and $\delta(P_2) = \delta(A_2) \cap \delta(A_6) = \{P_0\}$.

We have now solved for the probability of each node in the graph. The computed ϕ values for individual nodes are shown in Table 5, together with the χ and δ sets for each node. In our implementation, joint and conditional probability values are calculated only as needed, to reduce the amount of computation performed. We also apply dynamic programming techniques to cache the calculated values.

Table 5: Risk assessment calculations for Figure 3

N	$\phi(N)$	$\delta(N)$	$\chi(N)$
P_0	1	{}	{}
P_1	e_1	$\{P_0\}$	$\{P_0\}$
P_2	$(e_1e_2 + e_6 - e_1e_2e_6)$	$\{P_0\}$	$\{P_0, P_1\}$
P_3	e_1e_3	$\{P_0, P_1\}$	$\{P_0, P_1\}$
P_4	$(e_4e_6 + e_1(e_2e_4 + e_3e_5) - e_1e_2e_4e_6 - e_1e_3e_4e_5(e_2 + e_6 - e_2e_6))$	$\{P_0\}$	$\{P_0, P_1\}$
A_1	e_1	$\{P_0\}$	$\{P_0\}$
A_2	e_1e_2	$\{P_0\}$	$\{P_0, P_1\}$
A_3	e_1e_3	$\{P_0, P_1\}$	$\{P_0, P_1\}$
A_4	$e_4(e_1e_2 + e_6 - e_1e_2e_6)$	$\{P_0\}$	$\{P_0, P_1\}$
A_5	$e_1e_3e_5$	$\{P_0, P_1\}$	$\{P_0, P_1\}$
A_6	e_1	$\{P_0\}$	$\{P_0\}$

A.2 Computation for Cyclic Graph

In the previous example, we showed that the probabilities for graph nodes depend on the probabilities of their predecessors, so a recursive approach can be employed for this calculation. Within a cycle, however, recursing backward through predecessor sets will create an infinite loop. It is clear that a different approach is needed for cyclic nodes.

We will now work through the sample graph shown in Figure 4, demonstrating our approach and showing its effectiveness at recognizing and correctly handling cycles within the graph. A table showing all calculated values will be included at the end of this example.

Nodes P_0, A_1, P_1, A_2, A_3 will be calculated very much as demonstrated in Section A.1 and so we will not go through the detailed calculations for those nodes. Once these have been calculated, however, the remaining graph nodes comprise a cycle and therefore must be handled differently.

First, we will trace all acyclic paths through the cycle, to determine all valid ways that these nodes can be reached. This trace essentially performs a logical unfolding of the graph, marking unique passes through each node. The acyclic paths through this cycle are:

$$P_{2A} = \{A_2\} \qquad P_{3A} = \{A_2, P_{2A}, A_4\}$$
$$P_{3B} = \{A_3\} \qquad P_{2B} = \{A_3, P_{3B}, A_5\}$$

There are two unique instances of node P_2 in this logical unfolding of the graph, P_{2A} and P_{2B}. The probability that node P_2 is true will equal the probability that at least one of these instances is true, or $\phi(P_2) = 1 - \phi(\overline{P}_{2A}, \overline{P}_{2B})$.

To calculate $\phi(\overline{P}_{2A}, \overline{P}_{2B})$, we must calculate the joint probability of the set of entry nodes $\{A_2, A_3\}$ and we will also need to identify a d-separating set D within the cycle, to ensure that the instances of P_2 are conditionally independent. A cyclic d-separating set can be found by intersecting the sets of possible paths leading to the node instances and identifying common attack-step nodes within the cycle. In this case, a cyclic d-separating set is not needed for nodes P_{2A} and P_{2B}; because the cycle is so small, these partial paths are already conditionally independent, given the entry points into the cycle. The formula of computation is shown below.

$$\begin{aligned}
& \phi(\overline{P}_{2A}, \overline{P}_{2B}) \\
= & \sum_{A_2, A_3} \phi(\{A_2, A_3\})\psi(\{A_2, A_3\}, \overline{P}_{2A})\psi(\{A_2, A_3\}, \overline{P}_{2B}) \\
= & \ \phi(\{A_2, A_3\})\psi(\{A_2, A_3\}, \overline{P}_{2A})\psi(\{A_2, A_3\}, \overline{P}_{2B}) + \\
& \ \phi(\{A_2, \overline{A}_3\})\psi(\{A_2, \overline{A}_3\}, \overline{P}_{2A})\psi(\{A_2, \overline{A}_3\}, \overline{P}_{2B}) + \\
& \ \phi(\{\overline{A}_2, A_3\})\psi(\{\overline{A}_2, A_3\}, \overline{P}_{2A})\psi(\{\overline{A}_2, A_3\}, \overline{P}_{2B}) + \\
& \ \phi(\{\overline{A}_2, \overline{A}_3\})\psi(\{\overline{A}_2, \overline{A}_3\}, \overline{P}_{2A})\psi(\{\overline{A}_2, \overline{A}_3\}, \overline{P}_{2B}) \\
= & \ (0) + (0) + (e_1 e_3 (1 - e_2))(1)(1 - e_4) + \\
& \ (e_1(1 - e_2)(1 - e_3) + 1 - e_1)(1)(1) \\
= & \ 1 - e_1 e_2 - e_1 e_3 e_4 + e_1 e_2 e_3 e_4
\end{aligned}$$

So, $\phi(P_2) = 1 - (1 - e_1e_2 - e_1e_3e_4 + e_1e_2e_3e_4) = e_1e_2 + e_1e_3e_4 - e_1e_2e_3e_4$. By a similar calculation, $\phi(P_3) = e_1e_3 + e_1e_2e_5 - e_1e_2e_3e_5$. Once these have been solved, it is easy to see that $\phi(A_4) = G_M[A_4] \cdot \phi(P_3) = e_4(e_1e_3 + e_1e_2e_5 - e_1e_2e_3e_5)$ and $\phi(A_5) = G_M[A_5] \cdot \phi(P_2) = e_5(e_1e_2 + e_1e_3e_4 - e_1e_2e_3e_4)$.

The full results are shown in Table 6. Nodes P_2, P_3, A_4, A_5 do not have χ or δ sets, because these values are not used for evaluation within a cycle. Once a cyclic node set is calculated, however, successor nodes can include a virtual node representing the cycle in their χ and δ sets.

Table 6: Risk assessment calculations for Figure 4

N	$\phi(N)$	$\delta(N)$	$\chi(N)$
P_0	1	{}	{}
P_1	e_1	{}	{}
P_2	$e_1e_2 + e_1e_3e_4 - e_1e_2e_3e_4$	–	–
P_3	$e_1e_3 + e_1e_2e_5 - e_1e_2e_3e_5$	–	–
A_1	e_1	{}	{}
A_2	e_1e_2	$\{P_1\}$	$\{P_1\}$
A_3	e_1e_3	$\{P_1\}$	$\{P_1\}$
A_4	$e_4(e_1e_3 + e_1e_2e_5 - e_1e_2e_3e_5)$	–	–
A_5	$e_5(e_1e_2 + e_1e_3e_4 - e_1e_2e_3e_4)$	–	–