Proceedings of the 2014 Winter Simulation Conference
A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, eds.

MODELING AND ANALYSIS OF STEPPING STONE ATTACKS

David M. Nicol

Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
619 South Wright Street
Urbana, IL 61801, USA

Vikas Mallapura

Department of Computer Science
University of Illinois at Urbana-Champaign
201 North Goodwin Ave
Urbana, IL 61801, USA

ABSTRACT

Computer exploits often involve an attacker being able to compromise a sequence of hosts, creating a chain of "stepping stones" from his source to ultimate target. Stepping stones are usually necessary to access well-protected resources, and also serve to mask the attacker's location. This paper describes means of constructing models of networks and the access control mechanisms they employ to approach the problem of finding which stepping stone paths are easiest for an attacker to find. While the simplest formulation of the problem can be addressed with deterministic shortest-path algorithms, we argue that consideration of what and how an attacker may (or may not) launch from a compromised host pushes one towards solutions based on Monte Carlo sampling. We describe the sampling algorithm and some preliminary results obtained using it.

1 INTRODUCTION

A common technique by which intruders compromise computers within defended networks is through so-called *stepping stone* attacks. In these an attacker gains access to a computer, from that position attacks and gains access to some computer he could not originally access, from there attacks and gains access to yet another computer not previously accessible, and so on, until some valued target is gained. There is benefit in system administrators knowing which stepping-stone pathways are most easily taken, so as to best allocate resources in protecting critical assets.

We are interested in using modeling and analysis to assess how difficult it is for an attacker to reach a critical asset through stepping stone pathways, to find for a given critical asset stepping stone pathways that offer the least resistance, and finally to find for a given quantified level of critical asset value, stepping stone pathways that reach critical assets most easily. To accomplish these goals we need to identify which stepping stone paths are possible, and to quantify their difficulty. Whether or not a given compromised computer can reach and compromise another depends on the connectivity of the two, and also depends on whether, given access, there are any vulnerabilities on the target which can be compromised with the privileges the attacker has on the attacking host.

One can imagine describing a stepping stone attack as a path through a multi-graph where nodes represent hosts, and each edge represents one (of potentially many) ways in which an attacker who is resident on the source host can gain a foothold through an exploit on the destination host. We furthermore can imagine putting a weight on that edge which scores the difficulty of accomplishing the attack, potentially making that weight depend on some attribute of the attacking host or capability of the attacker. The score of a stepping stone path might then be defined to be the sum of its edge weights. A obvious and natural question asks where these weights come from—a question we answer in more detail later, but for now will say that there are industry standard databases of vulnerability metrics that form the basis for such weights.

If this were all there is to the story, the kinds of questions we posed before could be solved using ordinary shortest-path algorithms. In reality though, the difficulty of achieving a particular exploit can be sensitive to history. Consider: the first time the attacker encounters a vulnerable service he may spend a lot of time analyzing it, perhaps crafting an exploit for it. If after having compromised that service he encounters it again, deeper in the network, the difficulty of exploiting it a second time may be much less. The attacker has the knowledge, or perhaps a toolset he did not have before. The attacker may have gathered information along the way (e.g., password files, later cracked) from hosts compromised on the path that make the difficulty of exploiting a particular vulnerability less than it otherwise would have been. Conversely, by exploiting some vulnerability an attacker may become detected, triggering a number of responses that may make further exploits harder than they would have been otherwise. The main point is that costs associated with compromising a vulnerability may depend on history and/or previous vulnerabilities exploited on the stepping stone path.

The next natural question asks whether, given a more realistic and complex model of stepping stone path costs, finding the path with least cost is computationally tractable. As we will see, when the cost of an edge weight on a directed path depends on the path prefix which leads to the edge, the answer is "no". This realization forces us to consider heuristic methods to address our problem. The objective of this paper is to outline the problem we face, describe a software tool that begins to address it, and describe some preliminary algorithms for finding the most vulnerable stepping stone attacks possible in a given network.

The remainder of this paper is organized as follows. We lead in Section 2 with a description of CVSS, an industry standard way of describing known exploits. Next in Section 3 we develop a notational model of hosts, connectivity, and exploitable vulnerabilities. We state the theorems we've proven elsewhere which show that finding least-cost stepping stone attacks is computationally intractible, thereby motivating heuristic approaches such as the simulation based ones proposed in this paper. Section 4 describes a software system we have developed which enables our results to be employed in realistic contexts. Section 5 describes the Monte-Carlo-based heuristics we've developed to look for low-cost stepping-stone paths, and Section 6 describes our preliminary results with those heuristics. Section 7 outlines a body of related work, while Section 8 summarizes this paper and sketches future research directions.

2 CVSS

The Common Vulnerability Scoring System (CVSS) (Mell, Scarfone, and Romanosky 2006; Mell, Kent, and Romanosky 2007) is an industry standard vulnerability scoring system designed to provide an open and standardized method for rating IT vulnerabilities. A number of organizations use CVSS to provide their assessment of vulnerabilities, including, for instance, the National Institute of Standards and Technology, who publish a publicly accessible National Vulnerability Database (NVD).

CVSS describes a vulnerability with over a dozen metrics, organized into three groups. The *Base* metric group quantifies aspects of the vulnerability that are intrinsic, fundamental, and are unaffected by time or location. Within this group are descriptions of how one accesses the vulnerability, how complex the vulnerability is to exploit, and the number of times one has to authenticate in order to exploit the vulnerability. All of these sub-metrics have bearing on our goal of scoring stepping-stone paths. The *Temporal* metric group represents attributes of a vulnerability that may change in time. A relevant metric for us in this group describes the availability of code to exploit the vulnerability. The *Environmental* metric group contains no metrics related to difficulty of accessing the vulnerability, but (like the Base group) has metrics that speak to the impact of an exploit, which would be of interest in an analysis, for example, of finding stepping-stone paths that maximize impact subject to a bounded level of difficulty in access.

The NVD database we use[1] provides only the Base metrics, which are described in Table 1.

Each has a qualitative part, and a corresponding quantitative part; both parts come from the CVSS standard. *AccessVector* (A_v) describes how close the attack must be to the victim host in order to gain

[1] https://nvd.nist.gov

Table 1: Three base group CVSS metrics.

AccessVector (A_v)	local access required	0.395
	accessible from adjacent network	0.64
	accessible from remote network	1.0
AccessComplexity (A_c)	high	0.35
	medium	0.61
	low	0.71
Authentication (A_a)	requires multiple authentications	0.45
	requires single authentication	0.56
	no authentication required	0.704

access to the exploit. *AccessComplexity* (A_c) qualifies the difficulty of executing an exploit. A "high" is given if specialized conditions are required, e.g. the attacker has administrative privileges, or the exploit requires some sort of spoofing. It may be given if the attack needs to get inside of a race condition with a narrow window of opportunity. A "medium" score is given if the conditions are somewhat specialized, e.g., requires a user with some particular level of authorization, some information has to be gathered before an attack can be launched, or a small amount of social engineering is needed. A "low" score is given if specialized conditions don't exist. *AccessAuthentication* (A_a) differentiates based on the number of authentications required to exploit the vulnerability.

According to the CVSS standard these three metrics are combined to create a composite *Exploitability Score*,

$$\varepsilon = 20 \times A_v \times A_c \times A_a.$$

By construction, $0 < \varepsilon \leq 10.0$; the larger ε, the easier it is to exploit the vulnerability.

CVSS scores also address the impact of the vulnerability on confidentiality, integrity, and availability. We use these to limit our attention to vulnerabilities with impact.

3 MODEL

A host h_i has an associated set of known applications or services that are remotely accessible, called S_i. For each $a \in S_i$ there is a set (possibly empty) of known vulnerabilities in a, denoted $v(a)$, and a set of access points (e.g. open IP ports), denoted $pts(a)$, through which users (legitimate or otherwise) may access a. $chan_{i,j}$ denotes the *channel* from host h_i to h_j, and is the set of application vulnerabilities in h_j which the system's routers and firewall rules permit h_i to access. More formally,

$$v \in chan_{i,j} \iff \exists\ a \in S_j \text{ with } v \in v(a) \text{ and some } p \in pts(a)$$
such that the system supports h_i connecting to h_j through p.

Given a set of hosts and the channels between them we construct a *vulnerability multi-graph* where an edge exists from h_i to h_j for each unique (v,a,p) triple where $v \in chan_{i,j}$, $v \in v(a)$, $a \in S_j$, and $p \in pts(a)$ the source host satisfies the AccessVector qualification of e (e.g., access from adjacent network). In short, an edge corresponds to a vulnerability that allows an attacker on h_i to compromise h_j. It is important to note that the existence of this edge depends on the full connection details; the vulnerable service needs to have an open port, using a protocol, by which the underlaying access control rules permit h_i to reach h_j.

A *stepping stone path* is a path through a vulnerability multi-graph, where the edges denote the vulnerabiliteis exploited by the attacker to reach the last host in the path from the first. We describe each step in a stepping stone path by the triple $(h_i, e_{i,j}, h_j)$ which identifies the source and destination hosts and the edge used to connect them. We aim to construct a scoring function for stepping stone paths that quantifies the difficulty of the sequence of attacks it describes. A natural first approach is to weight each edge with

a value derived from the associated vulnerability's CVSS score. That derivation removes the dependency of the score on the vulnerability's AccessVector, since access is coded into the existence of the edge; it also transforms the score so that easier exploitability is coded by smaller values. If $\varepsilon = 20 \times A_v \times A_c \times A_a$ is the edge vulnerability's exploitability score, the transformed score is $\varepsilon' = 10 - \varepsilon/A_v$, which again has a range between 0 and 10.0. We call this the *exploit complexity score* for the vulnerability.

One might define the cost of a stepping-stone path to be the sum of the costs of the edges of the path. The easiest successful stepping-stone attacks can then be efficiently found using well-understood shortest path algorithms, e.g., finding the k shortest paths (Yen 1971). Our software provides this analysis, but the point of this paper is observe that this formulation is not true to experience, and to consider other formulations and simulation-based methods of solution.

As an attacker penetrates more deeply into a system, two different kinds of things may occur.

- The attack gains more knowledge of the system, in particular, how a particular vulnerability may be exploited. The attack may acquire software tools to accelerate his penetration. Thus, as the attack progresses some as-yet-to-be-exploited vulnerabilities may actually become easier to exploit. In graphical terms, the weight of an edge in a potential next exploit may depend on the path used to reach it. In particular, the weight on that edge may decrease from the original ε' CVSS value we might ascribe to it.
- The execution of some particular exploit may be detected, with the result that defender activity is triggered that make vulnerabilities *harder* to access. For example, network configuration may be dynamically changed to contain the attacker (thereby changing the graph's channels), different versions of services may be swapped in (changing which vulnerabilities might be accessed) or even hot patches might be applied, eliminating vulnerabilities altogether. Thus, at the attack progresses some as-yet-to-be-exploited vulnerabilities may become inaccessible, or harder to exploit. In particular, as a function of the stepping-stone path taken the topology of the graph may change, and weights on some edges may increase.

If we include in the cost function the possibility that exploiting a vulnerability in may make exploiting another vulnerability in a different host either more expensive or less expensive, we can prove that the problem of finding the least-cost path becomes computationally intractable. This result is stated precisely by the two theorems below.

Theorem 1 Let (G,E) be a directed multi-graph, with every edge $e \in E$ labeled with non-negative weight $w(e)$. Suppose $E_1, E_2, \ldots, E_{k-1}$ is a non-cyclic stepping stone path and consider a host h_k not yet visited but is accessible from the last host h_{k-1} in the path. For every v in $chan_{k-1,k}$ suppose that if v appears in any prior step, then the cost of exploiting v is no larger than its edge weight in (G,E), and may be smaller, as a function of $E_1, E_2, \ldots, E_{k-1}$. Then the problem of finding a min-cost stepping stone path between any two hosts is NP-hard.

Theorem 2 Let (G,E) be a directed multi-graph, with every edge $e \in E$ labeled with non-negative weight $w(e)$. Suppose $E_1, E_2, \ldots, E_{k-1}$ is a non-cyclic stepping stone path and consider a host h_k not yet visited but is accessible from the last host h_{k-1} in the path. For every v in $chan_{k-1,k}$ suppose that the cost of exploiting v is at least as large as its edge weight in (G,E), as a function of $E_1, E_2, \ldots, E_{k-1}$. Then the problem of finding a min-cost stepping stone path between any two hosts is NP-hard.

Formal proof of these claims is beyond the scope of this paper, however we can sketch the strategy. The "monotone exact satisfiability" problem (monotone XSAT) accepts a conjunction of three clauses, each the disjunction of three uncomplemented literals, and seeks a Boolean assignment to the literals with the constraint that in every clause at exactly one of the literals is "True". This problem is known to be NP-Complete (Schaefer 1978). For both theorems we show how to transform an arbitrary monotone XSAT problem into a min-cost path vulnerability path problem under the restrictions named in this theorem. Since XSAT is formally "hard" by virtue of being in NP-Complete, the min-cost path problem with path

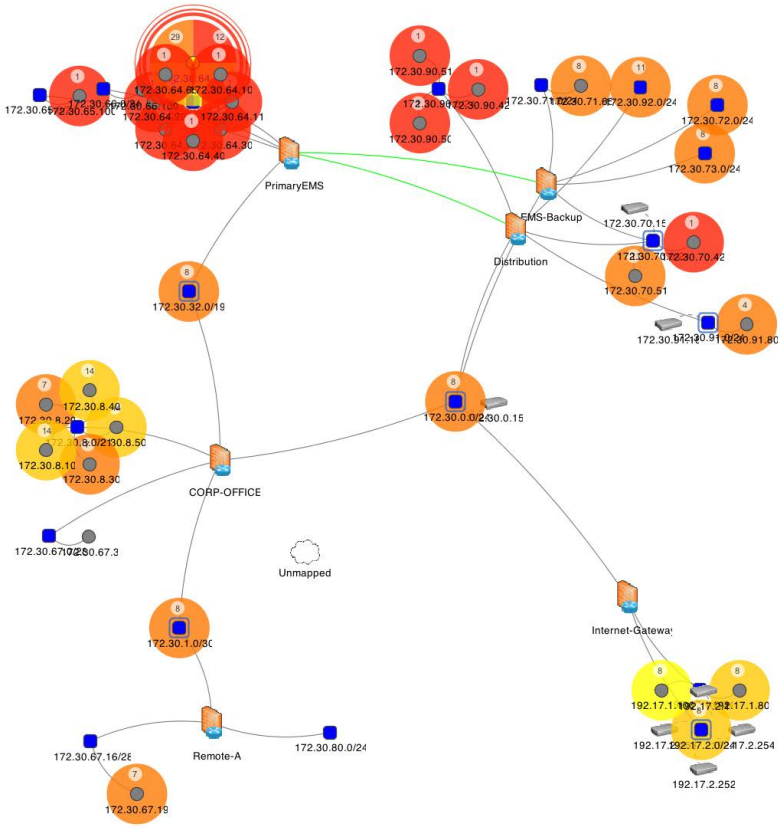

Figure 1: Screen-shot of NP-View analysis software showing stepping stone vulnerabilities.

dependent edges problem is at least as hard. This fact motivates our work in seeking simulation-based solutions to the problem.

4 NP-VIEW

We have embodied the analysis described in this paper within a software tool, **NP-View**, orginally developed under research contracts at the University of Illinois, now licensed by Network Perception. [2] **NP-View** is marketed for use in security audits performed in the power industry, but has more general application. From the configuration files of routers, switches, and firewalls it infers the network topology, and augments this with evidence of additional hosts obtainable from the scanning tool **nmap** (Lyon 2009). It then computes all of the connectivity the configurations permit; for our purposes, in particular, it computes which pairs of hosts (h_s, h_d) exist such that h_s can send a message that reaches h_d, and all of the ports (and protocols) involved in those transfers. Figure 1 presents a screen shot of NP-View on a sample network, and then the results of a stepping-stone analysis based solely on hop-count distance between attacked host and networks from which attacks might be launched.

nmap is an active scanning tool capable of discovering and reporting a significant amount of information about the state of a host. **NP-View** can import an **nmap** report from which it extracts information about open services on the host, and the Common Vulnerabilities and Exposures (CVE) identity of their vulnerabilities. It then looks up the CVSS score for each CVE entry from the NIST NVD database, and uses it in the kind

[2] http://network-perception.com

of analysis developed in this paper. The most easily accessible pathways to vulnerable hosts are reported and (ultimately displayed in ways similar to that in Figure 1).

5 HEURISTIC

We are interested in finding low-cost paths from a given attacking host h_a to a given victim host h_v (other variants are easily accommodated, such as finding the set of attacking hosts that can reach h_v with low effort). Given the computational difficulty of finding the least-cost stepping stone path, we turn to heuristics. The one we consider is based on Monte Carlo simulation. One of the attractive features is ease of trading off the amount of computation. for the quality of the solution; more explorations yields more opportunities to find lower cost paths. We first describe the data structure used by the computation, and then the algorithm applied to that data structure.

5.1 Graph Construction

The NP-View tool computes a precise connectivity matrix for hosts that are separated by one or more firewalls. For each host h_s it finds every host h_d such that there there is an internet protocol p, a set of ports P_s on h_a and a set of ports P_d on h_d such that a message sent from some port in P_s on h_s is delivered to some port in P_d on h_d, using protocol p. NP-View records the full particulars of the ports and protocols that allow such flows.

Using this graph, a breadth-first search identifies the set $T_f(h_a, j)$ of the hosts that are reachable from h_a in j hops or fewer moving forward, and a set $T_b(h_v, j)$ of hosts that can reach h_v in j hops or fewer. Any path from h_a to h_v using j or fewer hops can visit hosts only in the intersection set $T_f(h_a, j) \cup T_b(h_v, j)$; after choosing the j of interest our attention is restricted to nodes in this set.

Next we transform the connectivity matrix into a multi-graph. For every pair of hosts h_s and h_d in the set of interest and every vulnerability v associated with h_d, we create a directed edge from a node representing h_s to a node representing h_d, labeled by the CVE vulnerability identity, and the associated CVSS score.

Next we describe the algorithm used to find stepping stone attack paths.

5.2 Finding Low-Cost Paths

The basic idea is to stochastically sample paths and remember the the low cost ones. The randomness of path selection causes the algorithm to sometimes take what appear to be costly steps, against the possibility that by doing so an attacker gains access to a region where it becomes very easy to reach the target victime.

Given a stepping stone path $E_1, E_2, \ldots, E_{k-1}$ that is rooted in h_a the algorithm chooses randomly which of the edges lead away from the last host visited and do not lead back to a host already in the path. The probability distribution used to govern that sampling tries to skew sampling towards edges that appear from its limited vantage point to lead efficiently towards h_v. The technical details lay in building the graph (given already), and sampling paths from it.

Figure 2 illustrates the choice to be made. From the current position in the stepping stone pathway, one of several next vulnerabilities to exploit is to be selected. In Figure 2 the linear sequence of nodes beginning with h_a illustrate the path already traversed, with the "current position" being the last node in that sequence, with the several branching options shown, each labeled with a cost c_i. The algorithm's logic is inspired by the principle of dynamic programming. If, for each of the reachable nodes h_i (in the figure the nodes directly reachable from the current position) we knew the least cost of a path from h_i to the victim h_v, then the minimum cost of reaching h_v from here will be the minimum among summing the edge cost c_i with the minimum remaining cost L_i. The challenge for us is that the edge costs beyond the layer of next reachable nodes depend on the paths used to reach them, and so at this point are unknown. However, if we *assume* that as yet unseen edges costs are invariant, then standard shortest path algorithms can be used to *estimate* the L_i values. Those invariant costs may be a function of the path seen to date, but are assumed

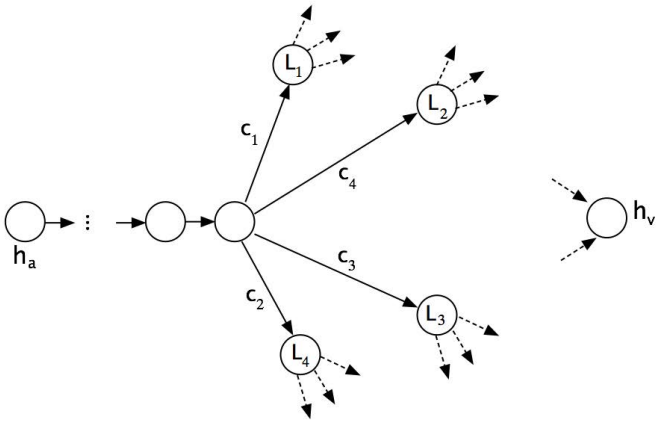

Figure 2: Edge sampling selection.

to not change as the path lengthens. For example, if we assume that once a vulnerability is exploited then any future exploit of that same vulnerability has cost 0, we'd compute the L_i values by zeroing out costs of every edges labeled by a vulnerability seen in the path leading to h_c and the vulnerability on the edge chosen from h_c.

In a shortest path algorithm on graphs with static edge costs, we'd select that i which minimizes $c_i + L_i$. In the sampling version we build a probability distribution that favors the selections where $c_i + L_i$ is lower than other. Construction of this distribution has the following steps.

1. Compute $S = \sum_{i=1}^{n}(c_i + L_i)$, where n is the number of edges.
2. Compute probabilities for each edge that are proportional to its cost: For each i compute $q_i = (c_i + L_i)/S$.
3. Invert the probabilities to give greater weight to the edges with lower costs: for each i compute $a_i = (1 - q_i)/(n-1)$.
4. Rescale probabilities to either emphasize choices leading to shortest paths, or to make the distribution more uniform.

The rescaling step is done parametrically, based on parameter α, with $0 \leq \alpha \leq 2.0$. For the purposes of exposition we assume that the i are presented so that the a_i values are in increasing sorted order. For $i = 1, 2, \ldots n$ define $A_i = \sum_{j=1}^{i} a_j$ to denote the cumulative distribution function. For any discrete distribution we must have $0 \leq A_i \leq \frac{i}{n}$, with the upper bound being the CDF of a uniform distribution. Depending whether $\alpha \leq 1.0$ or not, the rescaling moves A_i (for all but $i = n$) closer to either its lower bound of 0, or its upper bound of $\frac{i}{n}$. When $\alpha \leq 1.0$, we rescale $A'_i = \alpha A_i$, and when $1 < \alpha$ we rescale $A'_i = A_i + (2 - \alpha) * (\frac{i}{n} - A_i)$. In both cases we can interpret the rescaling as taking the distance between A_i and its bound, and retaining α (or in the case of $1 < \alpha$, $2 - \alpha$) of that difference.

With α as a control parameter, we can skew the sampling to be as close to the deterministic min-cost selection as we like, or as close to uniform as we like.

6 EXPERIMENTAL RESULTS

We now present some preliminary results. The experimental setting is the network illustrated in Figure 1, where we've seeded hosts with a set of vulnerabilities with a variety of CVSS scores. We look at how well the algorithm identifies low-cost paths between a particular attacker and victim, where a minimum of two

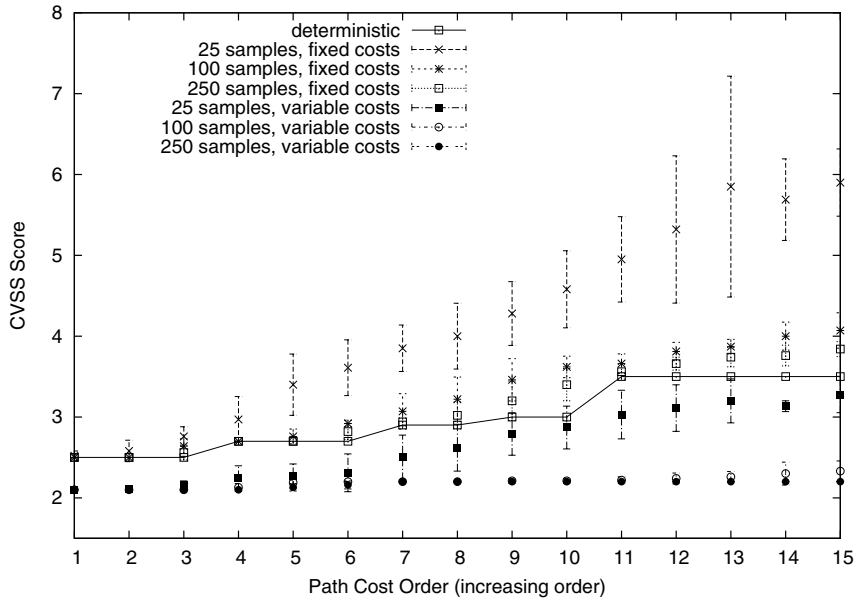

Figure 3: Tracking of minimal costs as a function of Monte Carlo replications.

penetrations are needed for the attacker to gain access to the victim. We're interested in the algorithm's behavior as a function of two parameters: the number of replications, and the skewedness of the sampling distribution.

As finding the optimal shortest paths is combinatorially complex when edge costs change dynamically, we ask instead how effective the sampling approach is when the edges don't change. We can then compare the path lengths found with optimal path lengths obtained from a deterministic algorithm.

The x-axis of Figure 3 orders the path lengths, with 1 being least, 2 being next least, and so on. The y-axis gives the sum of CVSS scores on the path. A given experiment computes the 15 least cost unique paths seen based on some number of Monte Carlo trials: 25, 100, and 250. Of course the results of an experiment will depend on the initial random number seeding. So, for each number of Monte Carlo trials we perform 10 experiments, and compute for each ordinal position the mean and standard deviation of the cost for that position. The figure also plots the optimal minimum cost for each ordinal position, using Yen's minimum k shortest path algorithms (Yen 1971).

We plot costs obtained both under the assumption of fixed edge weights, and under a model with variable edge weights. In the latter model the cost of the *first* exploit of a given vulnerability is its original edge weight, but thereafter every exploit of that same vulnerability is assumed to have cost 0.

The main take-away points for the fixed edge cost sets of experiments are that for the first handful of ordinal positions, all numbers of trials work equally well, but that more significant differences are seen—particularly for the 25 sample case—with the most costly paths. The second point is the comparative difference between 100 trials and 250 trials is not large. This really has to do with the size of the network; the main insight here is there appears to be a sweet spot in the space trading off closeness to the optimal and the number of replications; one expects each problem to have its own sweet spot, and one of the challenges will be to automate ways of finding it. Looking at behavior with variable edge weights we again see a large difference between using 25 and 100 samples, but much less difference for 250 samples. The most interesting point though is that for these larger sample runs, the minimal cost for larger ordinal

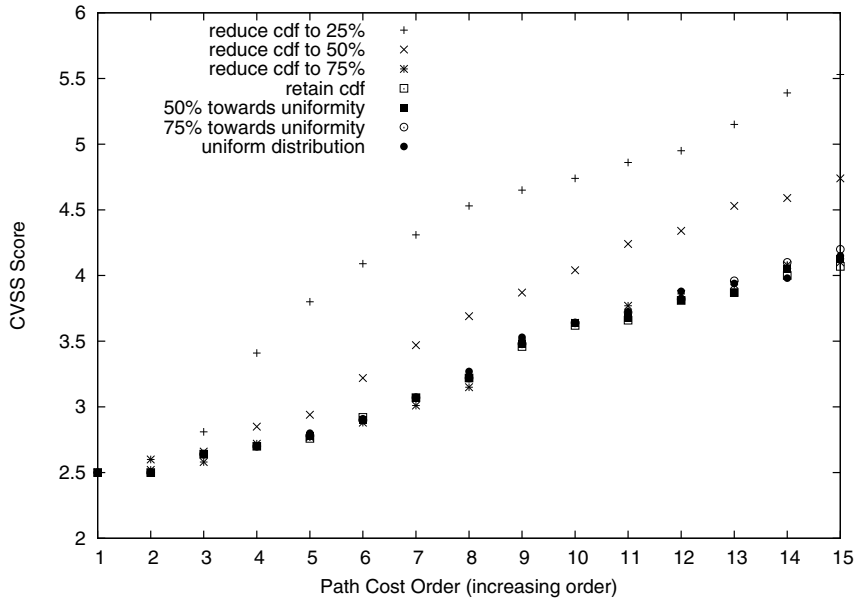

Figure 4: Effect of re-scaling sampling distribution.

positions is essentially constant, with very little variation between experiments. What is happening here is that the lowering of costs of already exploited vulnerabilities makes a difference class of paths attactive, ones that repeatedly use the same exploit—the one with least cost in the model. This gives evidence to the intuition that a sampling approach is useful to find these paths that are not identified by existing deterministic algorithms.

Figure 4 looks at the impact of smoothness parameter α on the algorithm's performance. For these plots we used 100 Monte Carlo replications, and we omit the standard deviation for visual clarity. Different values of α push the distribution closer to selecting the estimated min-cost next edge, or becoming closer to uniform. What stands out from this data is the seeming insensitivity to α, except for values that skew the sampling strongly towards the min-cost edge. The explanation for significantly higher costs at larger ordinal positions is simply that few unique paths are explored. Deeper inspection shows that the distributions before rescaling are already close to uniform to begin with, which is an artifact of the relatively small network we're analyzing and the costs of the vulnerabilities in this model. Furthermore will include larger models and greater variation in the vulnerabilites embedded in it.

7 RELATED WORK

For over fifteen years researchers have considered ways to represent the vulnerabilities in a system in such a way that an attacker's pathways to compromise critical assets might be analyzed (Phillips and Swiler 1998). These data structures have historically become known as *attack graphs*. The nodes in an attack graph typically represent the state of a system during an attack, which will incorporate how far the intruder has penetrated, and the capabilities the attack has at this point in the attack. Edges exist from node s_1 to another state s_2 if there is a vulnerability that can be exploited which transforms the system from s_1 to s_2. Problems studied include automated construction of attack graphs (Ou, Boyer, and McQueen 2006; Swiler et al. 2001; Ingols, Lippmann, and Piwowarski 2006) correlating observations on a network with

an attack model for that network (Noel, Robertson, and Jajodia 2004), discovery of attack sequences that can compromise a targeted critical asset (Sheyner et al. 2002; Ammann, Wijesekera, and Kaushik 2002), and development of metrics that score the insecurity of a network as a function of its known vulnerabilities and their relationship to attack graph (Wang et al. 2008).

While stepping stone attack paths and paths through an attack graph are related, they are not identical because the concepts represented by a node are different. In an attack graph, following an edge from one state to another may correspond in a stepping stone path to launching an attack from some compromised node earlier in the path, not necessarily the most recently compromised node. In this attack graphs are more general, with the main difference being that they embody the possibility of an attacker gaining capabilities by proceeding non-linearly that enable him to execute some exploits he could not otherwise accomplish. In the end the difference comes down to how one models the attacker, and for the model we adopt there is no advantage to using the classical attack graph formulation.

It turns out that under the most general assumptions about an attack graph, the problem of determining whether a given critical asset can be compromised is NP-complete (Sheyner et al. 2002). However, Ammann, Wijesekera, and Kaushik (2002) had the important insight that under an intuitive constraint that problem can be solved in polynomial time. The constraint—called monotonicity—is easily understood. It says that it is never the case that a successful attack somehow limits the capability of the attacker to later exploit a vulnerability he otherwise could have exploited. This makes sense in a context or model where the attacker and his attacks are invisible to the defenders, so that there is no cost to the attacker of exercising any particular exploit. We bring this point up because of result of our own that relates to it. We have shown that under the stepping stone model, while the existence of a stepping stone path that compromises a given critical asset can be determined with computational efficiency, under intuitive and realistic assumptions about costs ascribed to a path, the problem of finding the stepping stone path with least cost is computationally intractable. The models we analyze may enjoy the monotonicity property, but finding min-cost stepping stone attacks remains computationally hard.

The application of **NP-View** resembles the research project NetSPA (Lippmann et al. 2006; Ingols, Lippmann, and Piwowarski 2006; Lippmann et al. 2006) developed at MIT Lincoln Labs. The stated motivation for NetSPA was to work backwards from hosts with known vulnerabilities to determine whether those vulnerabilities are accessible from outside the firewalls protecting those hosts. In this **NP-View** accomplishes the same capabilities. However, from a network analysis perspective **NP-View** provides more detailed information (e.g., comparison of potential flows with global policy, explitict identification and analysis of rules that admit flows, etc.), and like almost all other attack graph projects, NetSPA is focused on the *existence* of attack vectors, not the identification of those that pose the least difficulty to the attacker. NetSPA appears to have been inactive for several years.

8 SUMMARY

This paper describes an approach for finding the most vulnerable pathways by which an attacker might reach a target victim in a protected network. The idea is that a defender can use this information to decide how best to increase defense of critical assets. The approach is built into a software tool, **NP-View**, which determines the connections that are supported by firewall and routing configurations, and reports from a scanning tool **nmap** which discovers live hosts and the services running on them. The National Vulnerability Database (NVD) identfies the known vulnerabilities associated with the identified services, and Common Vulnerability Scoring System (CVSS) scores are obtained from the NVD. These scores serve as the basis of edge weights on a graph where paths correspond to exploits that form a stepping-stone attack.

We have argued that the costs of exploiting vulnerabilities in reality can depend on what the attacker has done before, and learned, as well as the increased risk of being detected by exploiting vulnerabilities that might be detected. These facets make computationally intractable the problem of finding a path that minimizes the sum of weights of vulnerabilities exploited. In this paper we propose and begin a preliminary exploration of an approach that uses Monte Carlo sampling to look for lowest cost stepping stone pathways.

The work we propose might serve as the basis for an analysis that identifies those vulnerabilities that contribute most significantly to low-cost stepping stone attacks, and/or provide an evaluation engine for performing simulation based optimization that seeks to maximally improve performance subject to some sort of time or budget constraint. Other future work includes looking at variance reduction techniques such as splitting to squeeze more information out of a set among of computational effort. This makes particular sense given the relatively high cost we have of computing a selection distribution at each path extension.

ACKNOWLEDGMENTS

This material is based upon work supported by the Army Research Office under Award No. W911NF-13-1-0086.

REFERENCES

Ammann, P., D. Wijesekera, and S. Kaushik. 2002. "Scalable, graph-based network vulnerability analysis". In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 217–224. ACM.

Ingols, K., R. Lippmann, and K. Piwowarski. 2006. "Practical attack graph generation for network defense". In *Proceedings of the 22nd Annual Computer Security Applications Conference*, 121–130. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Lippmann, R., K. Ingols, C. Scott, K. Piwowarski, K. Kratkiewicz, M. Artz, and R. Cunningham. 2006. "Validating and restoring defense in depth using attack graphs". In *Proceedings of the 2006 Military Communications Conference*, 981–990. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Lyon, G. F. 2009. *nmap Network Scanning: The Official nmap Project Guide to Network Discovery and Security Scanning*. Insecure, USA.

Mell, P., K. A. Kent, and S. Romanosky. 2007. "The common vulnerability scoring system (CVSS) and its applicability to federal agency systems". Technical Report 7435, National Institute of Standards and Technology Interagency Report.

Mell, P., K. Scarfone, and S. Romanosky. 2006. "Common vulnerability scoring system". *IEEE Security & Privacy* 4 (6): 85–89.

Noel, S., E. Robertson, and S. Jajodia. 2004. "Correlating intrusion events and building attack scenarios through attack graph distances". In *Proceedings of the 20th Annual Computer Security Applications Conference*, 350–359. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Ou, X., W. F. Boyer, and M. A. McQueen. 2006. "A scalable approach to attack graph generation". In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, 336–345. ACM.

Phillips, C., and L. P. Swiler. 1998. "A graph-based system for network-vulnerability analysis". In *Proceedings of the 1998 Workshop on New Security Paradigms*, 71–79. ACM.

Schaefer, T. J. 1978. "The complexity of satisfiability problems". In *Proceedings of the 10th annual ACM Symposium on Theory of Computing*, 216–226. ACM.

Sheyner, O., J. Haines, S. Jha, R. Lippmann, and J. M. Wing. 2002. "Automated generation and analysis of attack graphs". In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, 273–284. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Swiler, L. P., C. Phillips, D. Ellis, and S. Chakerian. 2001. "Computer-attack graph generation tool". In *Proceedings of the 2001 DARPA Information Survivability Conference*, Volume 2, 307–321. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Wang, L., T. Islam, T. Long, A. Singhal, and S. Jajodia. 2008. "An attack graph-based probabilistic security metric". In *Data and Applications Security XXII*, edited by V. Atluri, Volume 5094 of *Lecture Notes in Computer Science*, 283–296. Springer.

Yen, J. Y. 1971, July. "Finding the k Shortest Loopless Paths in a Network". *Management Science* 17 (11): 712–716.

AUTHOR BIOGRAPHIES

DAVID M. NICOL is the Franklin W. Woeltge Professor of Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign, and is Director of the Information Trust Institute. He holds a B.A. in mathematics from Carleton College (1979), and M.S. and Ph.D. degrees in computer science from the University of Virginia (1983,1985). Prior to joining UIUC, he taught at The College of William & Mary, and Dartmouth College. He has served in many roles in the simulation community (e.g., Editor-in-Chief of ACM TOMACS, General Chair of the Winter Simulation Conference Executive Board of the WSC), was elected Fellow of the IEEE and Fellow of the ACM for his work in discrete-event simulation, and was the inaugural recipient of the ACM SIGSIM Distinguished Contributions award. His current research interests include application of simulation methodologies to the study of security in computer and communication systems. His email address is dmnicol@illinois.edu.

VIKAS B. MALLAPURA is a Master's student in the Department of Computer Science at University of Illinois, Urbana-Champaign. His research interests include network security. He received his B.E. in Computer Science and Engineering from Visvesvaraya Technological University, Karnataka, India in 2010. He worked at Oracle India for 3 years prior to his Master's studies. His email address is mallapu2@illinois.edu.