A Kolmogorov Complexity Approach for Measuring Attack Path Complexity

Nwokedi Idika and Bharat Bhargava

Purdue University, Department of Computer Science
305 North University Street,
West Lafayette, IN 47907 USA
{nidika,bb}@cs.purdue.edu

Abstract. The difficulty associated with breaching an enterprise network is commensurate with the security of that network. A security breach, or a security policy violation, occurs as a result of an attacker successfully executing some attack path. The difficulty associated with this attack path, then, is critical to understanding how secure a given network is. Currently, however, there are no consistent methods for measuring attack path complexity that make the assumptions of a modeler explicit while providing flexibility in how the modeler models the attack path. To provide these desirable attributes, we propose a regular-expressions-inspired language whose rationale for attack path complexity measurement is based on Kolmogorov Complexity. After detailing our Kolmogorov Complexity-based method, we demonstrate how it can be applied to a novel security metric: the K-step Capability Accumulation metric–a metric that defines the security of a network in terms of the network assets attainable for attack effort exerted.

Keywords: security, security metrics, attack graphs, exploitability

1 Introduction

A completely secure network is one where no attacker can violate a security policy of that network. Since such a system is currently impractical, an approximation to it would be one where the attacker has extreme difficulty violating the network's security policies. Thus, one way of asking "how secure is this network?" would be to ask "how *difficult* is it to violate a security policy in this network?" When violations occur, they happen as result of a weakness, or a series of weaknesses, that the attacker leverages. These weaknesses in the network are referred to as vulnerabilities. The complexity associated with exploiting a vulnerability is then critical to the security of a network.

The ease with which an attacker can successfully take advantage of a vulnerability is referred to as exploitability. The importance of exploitability is reflected by its inclusion in all the well-known vulnerability scoring systems. For instance, the Common Vulnerabilities Scoring System (CVSS) [1] computes exploitability as a linear combination of qualitative values. The Computer Emergency Response Team/Coordination Center (CERT/CC) [2] produces a numeric

vulnerability score based on a series of questions–one of which is, "How easy is it [the vulnerability] to exploit?" The SANS Critical Vulnerability Analysis Scale Rating [8] includes the ease of exploitation in defining two of its four ratings.

Security metrics that rely on exploitability measure security in terms of the effort or skill the attacker needs to cause a security policy violation. When security violations occur as a series of vulnerability exploits, the dependencies among these exploits can be incorporated into security metrics that utilize the attack graph. An attack graph is an abstraction that reveals the ways an attacker can use identified vulnerabilities interdependently in a network to violate a security policy. In this paper, we use condition-oriented attack graphs [9] where nodes correspond to hosts, and edges correspond to vulnerabilities. Examples of attack graphs are displayed in Figure 3. Security metrics based on attack graphs are referred to as attack graph-based security metrics. An instance of such a metric would the Shortest Path metric. This metric [4] denotes the security of a network as the path from the attack graph that is of least of resistance. Another example attack graph-based security metrics is the Average Path Length metric. This metric [10] denotes the security of a network as the mean difficulty associated with all paths from the attack graph.

In the attack graph, attack path difficulty (or complexity) is represented as path length. Path length may be represented as the number of edges between the attacker's initial state and goal state or as a more complicated algebraic permutation of vulnerability scores along attack paths. While such flexibility in path length representation may be appropriate and necessary for a network administrator to model the threat of a given network, such flexibility hinders consistent communication across organizations and among researchers. If a method more complex than simply counting the edges along a path is used, some qualitative information is being employed. Because quality can be subjective, different organizations and researchers may score vulnerabilities differently and potentially compute path length differently. These differences are fine as long as the assumptions being made are explicit and grounded in theory.

We propose an approach to maintain flexibility in how attack path complexity is modeled, make associated modeling assumptions explicit, and provide a theoretical basis for attack path measurement. Our approach is called the Kolmogorov Complexity Method (KCM). KCM allows attack paths to be modeled quantitatively or qualitatively using a regular-expressions-inspired language. After describing this methodology, we show how these methods can be applied to a novel security metric called the K-step Capability Accumulation (KCA) metric.

The remainder of the paper will have the following organization. Section 2 describes KCM. Section 3 specifies the KCA metric and how KCM can be used with the metric. Section 4 gives related work associated with attack path complexity and with our novel attack graph-based security metric. Section 5 details our conclusion.

2 Kolmogorov Complexity Method (KCM)

Given an alphabet and the infinite number of strings that can be produced from that alphabet, some strings are more complex than others. Kolmogorov Complexity is an approach for determining string complexity. Thus, given two strings, Kolmogorov Complexity determines which string is more complex.

Kolmogorov Complexity determines a string's complexity using the size of the smallest program that can produce that string [5]. Let K represent the Kolmogorov Complexity function. In comparing two strings, x_1 and x_2, if $K(x_1) < K(x_2)$, then x_2 is more complex than x_1, because a larger program is needed to describe x_2. The use of Kolmogorov Complexity to monitor security was first promulgated by Evans et al. in [6]. Kolmogorov Complexity has also been applied to spam filtering [14] by Spracklin and Saxton. We are applying Kolmogorov Complexity to security assessment–and more specifically, the complexity of attack paths.

Kolmogorov Complexity provides a systematic approach for determining complexity of attack paths. There is, however, a caveat to its use. Ming et al., in [5], have shown that determining a lower bound K is impossible. With respect to attack path complexity, this finding means that we can never find the "true" effort needed for any attacker to exploit a vulnerability. This constraint need not be a hindrance to its use if reasonable effort estimates can be obtained. Nonetheless, KCM provides a standard way for communicating attack path complexity that is based upon a sound theory of complexity. Such formalism provides consistency among researchers when comparing measurements. Below, we provide an attack path complexity description language inspired by the language of regular expressions.

Alphabet

1. A corresponds to the exploits (i.e., instances of vulnerabilities) found in all attack graphs being considered.

Constants

1. ϵ corresponds to the empty string.
2. $e_i \in A$ denotes an exploit from one of the attack graphs being considered.
3. $\emptyset$ corresponds to the empty set.

Operations

1. Let S and T be two strings comprised of characters from A.
2. Let E_1 and E_2 be expressions of the language.
3. ST evaluates to the concatenation of strings S and T.
4. () provides priority ordering of evaluation.
5. $(S)^+$ states that the expression S *may* repeat one or more times but must appear once.

6. S^k evaluates to k instances of S concatenated together.
7. $E_1^{[k]}E_2$ evaluates to inserting E_1 at index k in E_2, where the first character in E_2 corresponds to the zeroth index. This can be generalized to $E_1^{[k_1],[k_2],[k_3],\ldots,[k_{n-1}],[k_n]}E_2$. Note: In general, this rule would not be used independently. It would only be used as part of the following two rules.
8. $E_1^{l,[k]}E_2$ evaluates to concatenating E_1^l to E_2, and inserting E_1 into index k of E_2.
9. $E_1^{l[k]}E_2$ evaluates to inserting E_1^l into index k of E_2.

The above KCM language is flexible in its ability to model the complexity of attack paths. For instance, the attack path in Figure 1 can be represented multiple ways. H1 through H6, Attacker, and Target in Figure 1 correspond to hosts, and v_j corresponds to vulnerability j. Based on operations 8 and 9 from the KCM language, a qualitative representation, KCM-qual, of this path can be represented as $v_1^{3,2[2]}v_2v_3$ yielding an attack path length of 3 vulnerabilities. An equivalent representation would be $v_1^{3,[2],[2]}v_2v_3$ yielding an attack path length of 3 vulnerabilities. In both representations, E_1 and E_2 from operations 8 and 9 corresponds to v_1 and v_2v_3 respectively. This representation suggests that once the attacker exploits H1 via v_1, attacking hosts H2, H3, H6, and Target is trivial because they, for example, require the same credentials to have their vulnerabilities exploited. Such a representation may also be appropriate if known scripts can be used to exploit vulnerabilities in the system. Ultimately the qualitative representation used depends on the subjective decision of the modeler. For instance, the attack path can alternatively be represented as $v_1^{3,[2]}v_2v_3v_1$. The path length of this representation is 4 vulnerabilities. This representation suggests that the attack path in Figure 1 is more secure than what is suggested by the two representations initially described for this attack path. The semantics of this latter representation suggests that while the same information can be used in exploiting hosts H2, H3, H4, and H6, different information is required to exploit Target. If the attack path is represented as $v_1v_1v_1v_2v_3v_1v_1$, then this would, for instance, suggest that different credentials are required for compromising each host. This representation corresponds to the quantitative representation, KCM-quant, which is equivalent to counting the edges along attack paths.

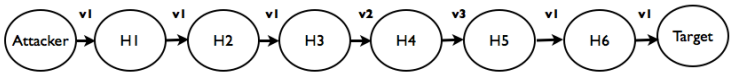


Fig. 1. A single attack path with Attacker, H1–H6, and Target corresponding to hosts and v_j corresponding to vulnerabilities

2.1 Representing Cycles in Attack Graphs

It is possible for an attack graph to contain cycles. A cycle in an attack graph corresponds to an infinite number of paths. Thus, it can be difficult to reason about such attack paths (e.g., determining attack path length). Typically, cycles are ignored when performing attack graph analysis. Using KCM-qual, a modeler can account for the path length of attack paths containing cycles.

Figure 2 shows a infinite number of attack paths because there is a cycle. Using KCM-qual, we can represent such attack paths. The infinite number of attack paths appearing in Figure 2 can be, for example, captured with $v_1^2(v_1v_2v_3)^+v_1^2$. The attack path length of this cyclic attack path is 5 vulnerabilities.

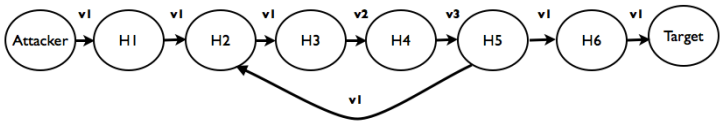


Fig. 2. An attack path containing a cycle

2.2 Qualitative Versus Quantitative Representations

The decision of whether to use the qualitative representation of KCM, KCM-qual, or the quantitative representation, KCM-quant, is predicated on the amount of information available for the security assessment. If the security engineer has no information regarding exploitation difficulty, KCM-quant is appropriate. KCM-quant assumes all vulnerabilities require distinct, yet, equal amounts of effort. This assumption is overly restrictive in practice. Therefore, practitioners may find KCM-qual most amenable for measuring attack path complexity.

If the security engineer has exploitability information about the vulnerabilities in the network under inspection, then KCM-qual would be appropriate. If the security engineer knows that two vulnerabilities require the same exact actions and information to be exploited, these exploits can be coalesced into a single vulnerability in KCM-qual. An example of this scenario would be when the attacker uses the same credentials to access different hosts within a network. Another example of when two exploits may be coalesced in KCM-qual is when some information (e.g., host name) that the attacker has already acquired or that is publicly known is all that is required to be changed when needing to exploit the second vulnerability. If the security engineer assumes that the attacker is rational, then KCM-qual corresponds more closely to what the attacker will actually experience.

The assumptions the modeler uses can be reflected in the names used for different qualitative representations. KCM-qual-C corresponds to the KCM-qual

representation where nodes in an attack path are coalesced *only* when the same credentials can be used at each node to realize an exploitation. KCM-qual-S corresponds to the KCM-qual representation where nodes in an attack path are coalesced *only* when there exists some publicly accessible software that reduces the effort of vulnerability exploitation at each node. Under KCM-qual-S, even if two nodes require different software, each node would be coalesced. However, under KCM-qual-S$^-$, a less conservative formulation of KCM-qual-S, nodes would only be coalesced if the same software could be used to exploit vulnerabilities at different nodes. KCM-qual-CS corresponds the combination of KCM-qual-C and KCM-qual-S. Under KCM-qual-CS, nodes on an attack path will be coalesced if with KCM-qual-C holds or KCM-qual-S holds. When KCM-qual is used in this paper, it is referring to the KCM-qual-CS formulation. Naturally, there is also a KCM-qual-CS$^-$. KCM-qual-CS$^-$ will coalesce two nodes on the same attack path if either KCM-qual-C holds or KCM-qual-S$^-$ holds.

If truly quantitative complexity or probability values were known for each distinct vulnerability in the attack path, then these values could be incorporated with KCM. By assigning these values to the nodes in the KCM-quant representation, algebraic operators would be available to manipulate the values along these paths. If the KCM-qual representation is used, algebraic operators would be unavailable for manipulating the values along attack paths as the result would be mathematically unsound. This soundness issue could be practically bypassed if empirical results showed KCM-qual to be in closer alignment with reality than KCM-quant. The problems of determining the complexity of exploiting a vulnerability or determining the probability of a vulnerability being exploited are critical issues, open problems, and outside the scope of this paper.

3 K-step Capability Accumulation (KCA) Metric

The K-step Capability Accumulation (KCA) metric specifies the "power" the attacker can obtain on a network in K steps. Practically, "power" refers to access level. Users with the highest access level have the most capability in a network. Such users also have the most opportunity for violating the network's security policy due to their level of access. The granularity of access levels ranges from system-level access to application-level access. This granulation is necessary in some instances as the attacker may successfully obtain system level access to machines on a network and yet still require access to specific authenticated applications. We show how to integrate such application-level accesses in the following subsection. These semantic variations in access level is best captured by the term "capability." This term differs from the notion of capability defined by Phillips and Swiler in [4]. In [4], capability is used broadly to include not only privileged actions within a network, but it also includes the skill level of the attacker and the tools or resources available to the attacker. The capabilities KCA is concerned with are those derived from the network under inspection, that is, privileges and privileged actions. The examples in this paper usage of

access levels follows the concept of privileges in [12, 13] by using system-level access on machines to represent power.

3.1 Evaluating with KCA

In determining which of two networks Sys_1 and Sys_2 are most secure using KCA, one procedure would resemble the following. First, generate attack graphs G_1 and G_2 for Sys_1 and Sys_2 respectively. Choose an integer for K that corresponds to the maximum number of steps the analysis should be carried out for if needed. K should be no greater than the minimum of the maximum path length for both attack graphs. One evaluation method is that if an attacker can obtain more capabilities on G_1 in K steps, than the attacker can on G_2 in K steps, then Sys_2 is more secure than Sys_1. KCA is given below.

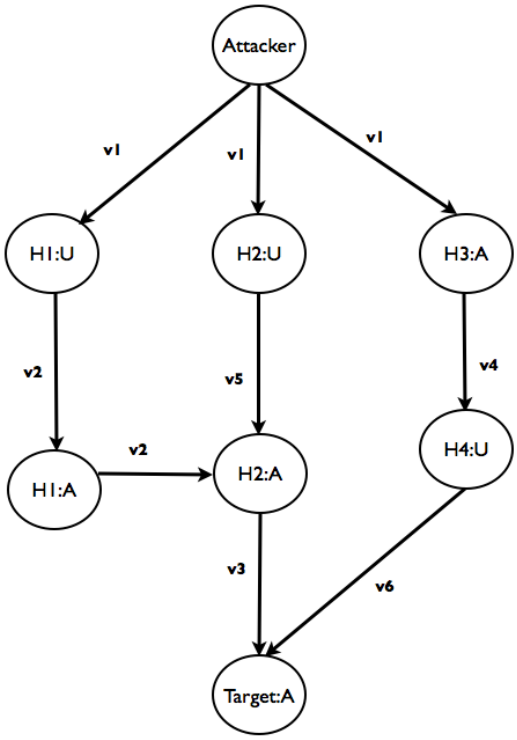
$$Cap_h(G) = \cup_h capabilities(n) \tag{1}$$

$$KCA_k(G) = \cup_{i=1}^{k} Cap_i(G) \tag{2}$$

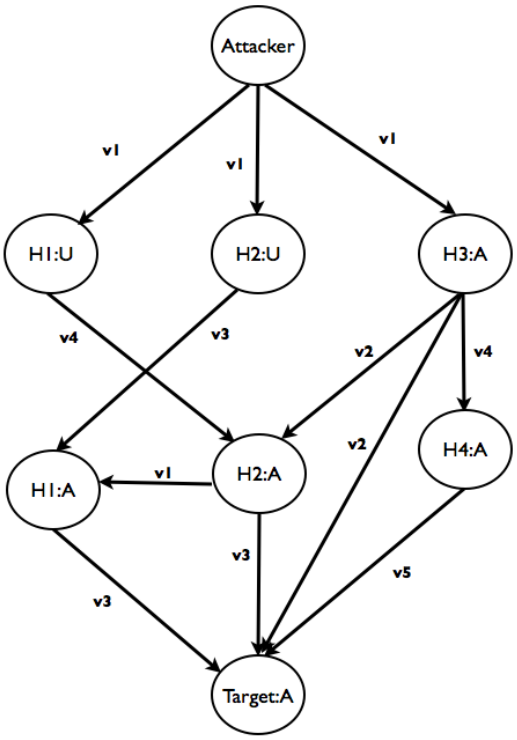
capabilities(n) returns the set of capabilities available at node n. If more granularity is desired, the *capabilities* function may be extended with a set of vulnerabilities as an input parameter. Different vulnerabilities may provide different capabilities for an attacker. This extension may be captured by extending *capabilities*(n) as *capabilities*$(\{v_1, v_2, v_3..., v_{m-1}, v_m\}, n)$. Vulnerabilities v_1 through v_m correspond to the vulnerabilities that an attacker may exploit to compromise node n. This extended representation allows for application-level access control to be modeled. Such a model also allows for different attacker profiles. That is, not all attackers have the same skill level or resources [11]. Therefore, depending on the assumptions made about the attacker, potentially only a subset of the vulnerabilities may be exploitable. Using our extended representation this scenario can be captured. $Cap_h(G)$ represents the capabilities obtained at level h in attack graph G. h represents the distance from the attacker's initial state. $KCA_k(G)$ is then simply the union of capabilities obtained at each level up to k.

Applying KCA to the attack graph in Figure 3(a), we obtain different values at each level of the attack graph. The nodes of the attack graph correspond to hosts and access levels. The labels have the format of *host:access level*. *U* corresponds to user-level access. *A* refers to administrator-level access. If access level is not specified, the attacker is assumed to have administrator-level access on the machine. In computing KCA for the attack graph in Figure 3(a), we obtain the following. Cap_1 = user-level access on H1 and H2, administrator-level access on H3. $Cap_2 = Cap_1$ and administrator-level access on H1 and H2, and user-level access on H4. $Cap_3 = Cap_2$ and administrator-level access on the Target host. The attacker can accumulate all privileges in the network in 3 levels.

When comparing two network configurations to decide which is most secure with respect to this metric, we have two options. One option is to compare



(a) A 3 path attack graph



(b) A 7 path attack graph

Fig. 3. Two attack graphs

the two networks' corresponding attack graphs only after k steps. A second option is to compare the two network's corresponding attack graphs at each level up to k. The second option is useful when a security engineer is interested in how quickly the attacker attains capabilities. For instance, after k steps in the two attack graphs being compared, the attacker may have attained the same level of capabilities. This information does not help the security engineer decide which of the two attack graphs is more secure. However, if the security engineer were to examine the two attack graphs using KCA at each level up to k the security engineer may find that in one attack graph, the attacker accumulates all of the network's capabilities after 1 step, whereas in the other attack graph the attacker accumulates all of the network's capabilities after 5 steps. This additional information would suggest to the security engineer that the network corresponding to the latter attack graph is more secure because it requires the attacker to take more steps to acquire all of the network's capabilities.

An alternative method for comparing two network configurations using KCA would be to compare KCA at each step. That is, in comparing two networks, the security engineer deems the network whose attack graph allows for the same or more valuable assets attainable in fewer steps to be less secure. This method of comparison is best exemplified through an example. Observe the attack graph in Figure 3(b). The KCA at level 1 for the attack graph in Figures 3(a) and 3(b) is the same. However, we can see that in Figure 3(b) that Target can be reached in two steps. Assuming that this is the most important host to protect, then the attack graph in Figure 3(b) is less secure because the attacker is able to obtain more power in fewer steps.

3.2 Applying Kolmogorov Complexity Method to KCA

The KCA metric uses steps or levels within the attack graph to obtain its values. The KCA metric as previously defined utilizes KCA-quant. To accommodate KCA-qual, we need to redefine *Cap*.

$$Cap_{p_i} = \cup capabilities(p_i) \tag{3}$$

This path-oriented version of Cap extracts all capabilities along an entire path p_i.

A step or level in KCA under KCM-qual uses the characters in the string representation to determine levels. If an attacker exploits a vulnerability v_j that is followed immediately by $0 \leq m < k$ v_j's, then the attacker has the capabilities associated with each host that v_j violates. If any new vulnerability v_l is encountered on the path, then for any v_j or v_l immediately following this vulnerability the attacker has the capabilities associated with each host that is violated by v_l or v_j. This pattern continues when evaluating KCA with KCM-qual. If using a KCM-qual representation that involves KCM-qual-S, then any vulnerabilities the software enables exploitation for can belong to a vulnerability superclass that would allow these vulnerabilities to be coalesced.

Let s_1 to s_n correspond to the KCM-qual string representations of attack paths p_1 to p_n. Let q_1 to q_n correspond to the KCM-quant string representations of attack paths p_1 to p_n. Let $q_j^{0..i}$ correspond to the substring of q_j that includes indices 0 through i. A similar definition exists for s_j. Let s_j^i correspond to the ith character position in s_j. A similar definition exists for q_j. Let $e(s_j^{0..i}) = q_j^{0..m}$, for some integer m such that the vulnerability at index s_j^i and q_j^m are the same, and the vulnerability does not appear in q_j^{m+1}, and the vulnerabilities present before q_j^m all appear in $s_j^{0..i}$. Then the equation for KCA using KCM-qual is given by the following:

$$KCA_k(G) = \cup_{i=1}^{k} Cap_{e(s_j^{0..i})}(G), \tag{4}$$

for all attack paths j.

This manifestation of KCA captures the fact that an attacker may obtain knowledge from one attack path and apply this knowledge to a completely separate attack path. If this version of KCA is applied to Figure 3(a), we must first determine how to represent each path. We will then define each path from left to right as the following: $v_1 v_2^2 v_3$, $v_1 v_5 v_3$, $v_1 v_4 v_6$. Thus, after one iteration, the attacker will have H1:U, H2:U, and H3:A. After two iterations, H1:A, H2:A, and H4:U will be unioned with the attacker's previous capabilities. Note that the attacker is able to obtain H2:A through both v_2 and v_5. Both are accounted for in this iteration. In the final iteration Target:A is added to the capabilities of the attacker.

4 Related Work

Because this work proposes an attack path complexity measurement method *and* a security metric, this section has two parts. The first subsection is dedicated to attack path complexity. The second subsection is focused on relevant attack graph-based security metrics.

4.1 Attack Path Complexity

In the authors' literature search, attack path complexity is a topic that has not been directly addressed. An attack path's length can simply be the number of edges (i.e., vulnerabilities) between the attacker's initial state and goal state [4]. If complexity values are assigned to each vulnerability in the form of probabilities, then the product of a given attack path would correspond to the complexity of that attack path [4]. Complexity may also be summed [3, 4].

Wang et al., in [3], noted that a relation could exist between two vulnerabilities such that the exploitation of one vulnerability decreases the complexity in exploiting the other vulnerability. KCM-qual captures this notion as the modeler is given the ability to show that exploiting one vulnerability decreases the difficulty of exploiting other vulnerabilities to zero with use of our language (e.g.,

operation 6 from the KCM language). The relation in [3], allows for more flexibility within the proposed framework in that the complexity of other affected vulnerabilities can change to arbitrary values. We believe that this flexibility, unfortunately, lends itself to the type of subjectivity that hinders the sharing of security metric data. With no rules for how vulnerability complexity values should change due to a vulnerability exploitation, there can be no expectation that researchers will assign changes in complexity values in any uniform way.

4.2 Attack Graph-based Security Metrics

There are two security metrics that have inspired KCA: the Shortest Path metric and the Network Compromise Percentage (NCP) metric. If the Shortest Path metric [4], from Phillips and Swiler, is being used under KCM-quant, then the shortest attack path in the attack graph corresponds to the path with the fewest number of edges. If KCM-qual is used, then the shortest path in the attack graph corresponds to the path that produced, through arithmetic/algebraic manipulations, the value considered to have the least resistance in comparison to other paths.

KCA and the Shortest Path metric can be similar when using a goal-oriented attack graph. However, KCA can be applied to attack graphs with no goal states. The Shortest Path metric, on the other hand, cannot be applied to attack graphs with no goal states. Thus, KCA is more versatile in its applicability to different types of attack graphs.

When there is a goal state and the semantics of the attack graph are such that this goal state has all of the asset value in the network, the KCA metric may degenerate to the Shortest Path metric. For instance, if the attacker can reach the goal state in single step and the non-attacker nodes are of little value with respect to the target node, then using the Shortest Path metric without KCA would be sufficient for determining which of the two networks is most secure. However, if other nodes are perceived as being relevant to the network's security, then the KCA metric can be used to obtain more information about the security of the network than what the Shortest Path metric can provide.

The NCP metric [7] proposed by Lippmann et al. denotes the security of the network as the percentage of compromised hosts within the network. A NCP of 100% means that the entire network can be compromised by the attacker. A NCP of 0% means that none of the network assets can be compromised by the attacker. NCP can be extended/modified to include assets that are more fine-granular than hosts, and therefore has the same representational ability as KCA. Thus, KCA's differentiating feature is its inclusion of attack effort exerted.

5 Conclusion

Measuring network security in terms of the difficulty experienced by an attacker in attempting to violate a security policy is an intuitive perspective. However, without systematic and flexible methods for modeling the complexity associated

with any given attack path, such a perspective will have limited value. In this paper, we have proposed the well-founded theory of Kolmogorov Complexity to serve as a foundation for measuring attack path complexity. We have also proposed a novel security metric that specifies network security in terms of the assets gained for the attack effort expended.

References

1. Mell, P., Scarfone, K., Romanosky, S.: Common Vulnerability Scoring System. IEEE Security and Privacy, vol. 4, 85–89 (2006)
2. Computer Emergency Response Team (CERT), `http://www.cert.org`
3. Wang, L., Singhal, A., Jajodia, S.: Measuring Overall Security of Network Configurations Using Attack Graphs. Data and Applications Security XXI, vol. 4602, 98–112 (2007)
4. Phillips, C. A., L. P., Swiler: A Graph-based System for Network-vulnerability Analysis. In: Proceedings of the 1998 Workshop on New Security Paradigms, pp. 71–79. ACM (1998)
5. Ming, L., Vitanyi, P.: An Introduction to Kolmogorov Complexity and Its Applications. Springer, (1997)
6. Evans, S., Bush, S., Hershy, J.: Information Assurance Through Kolmogorov Complexity. In: DARPA Information Survivability Conference and Exposition, (2001)
7. Lippmann, R., Ingols, K., Scott, C., Piwowarski, K., Kratkiewicz, K., Artz, M., Cunningham, R.: Validating and Restoring Defense in Depth Using Attack Graphs. In: Military Communications Conference, (2006)
8. SANS `http://www.sans.org/newsletters/risk/`
9. Noel, S., Jajodia, S.:Managing Attack Graph Complexity Through Visual Hierarchical Aggregation. In: Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security, pp. 109–118, ACM, (2004)
10. Li, W., Vaughn, R.:Cluster Security Research Involving the Modeling of Network Exploitations Using Exploitation Graphs. In: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and Grid Workshops, (2006)
11. Dantu, R., Kolan, P.:Risk Management Using Behavior Based Bayesian Networks. Intelligence and Security Informatics (LNCS), vol. 3495, 115–126 (2005)
12. Dacier, M., Deswarte, Y., Kaâniche, M.:Models and Toos for quantitative assessment of operational security. In: Proceedings of the 12th International Information Security Conference pp.177–186 (1996)
13. Ortalo, R., Deswarte, M., Kaâniche, M.:Experimenting with Quantitative Evaluation Tools for Monitoring Operational Security. IEEE Transactions on Software Engieering, vol. 25, 633–650 (1999)
14. Spracklin, L. M., Saxton, L. V.:Filtering spam using kolmogorov complexity estimates. Advanced Information Networking and Applications Workshops, pp. 321–328 (2007)