A Target-Centric Formal Model For Insider Threat and More

Ramkumar Chinchani, Anusha Iyer, Hung Ngo, Shambhu Upadhyaya
Department of Computer Science and Engineering
State University of New York at Buffalo
Buffalo, NY 14260
Email: {rc27, aa44, hungngo, shambhu}@cse.buffalo.edu

Abstract

The diversity of cyber threat has grown over time from network-level attacks and password-cracking to include newer classes such as insider attacks, email worms and social engineering, which are currently recognized as serious security problems. However, attack modeling and threat analysis tools have not evolved at the same rate. Known formal models such as attack graphs perform action-centric vulnerability modeling and analysis. All possible atomic user actions are represented as states, and sequences which lead to the violation of a specified safety property are extracted to indicate possible exploits. While attack graphs are relevant in the context of network level attacks, they are ill-equipped to address complex threats such as insider attacks. The difficulty mainly lies in the fact that adversaries belonging to this threat class use familiarity of and accessibility to their computational environment to discover new ways of launching stealthy, damaging attacks. In this paper, we propose a new target-centric model to address this class of security problems and explain the modeling methodology with specific examples. Finally, we perform quantified vulnerability analyses and prove worst case complexity results on our model.

1 Introduction

The purpose of threat assessment from a security standpoint is to expose weaknesses in the infrastructure before an actual attack occurs. It allows a security analyst to take appropriate countermeasures to prevent/deter any actual attacks. Currently, this is accomplished either through lessons learned from security audits or formal models such as attack graphs [1–4], which allow representation of a computational environment and subsequent analysis for security vulnerabilities. Each approach has its benefits and disadvantages. Penetration testing replicates real world scenarios, and it gives near-realistic indication of the resistance offered by the currently deployed security systems against an attacker. Security audits can cover not only network level vulnerabilities but also weaknesses arising due to human factors. A downside is that the overall process is slow and expensive, and limited in scope to known exploits. On the other hand, formal threat models such as attack graphs offer an elegant alternative in terms of a symbolic machinery to appropriately represent a computational environment and analyze it for security weaknesses. Recently, researchers have used model checking techniques [3] to automate the process of generating traces which represent successful attacks. In the model, each state transition denotes an atomic action or event and each state corresponds to the completion of a sequence of actions. Safety properties are specified as invariants which should not be violated. Basic modeling in [3] was followed by quantified analyses [4], where minimizations were proposed to find smallest sequences to effectively launch an attack. Assuming that such sequences are most likely to occur, an analyst can identify and eliminate a critical subset of states, thereby, preventing these attacks. All the metrics used for quantification are state-based and each atomic action is assumed to incur a unit effort. It is well known that the

mileage obtained by model checking is dependent on the granularity of the specification. Finer granularity gives better coverage but introduces a large number of states and the state explosion problem becomes very pronounced. Another group of researchers [5] argued that approximations can be introduced to aggregate some details and they reported significant gains in performance, specifically, a polynomial time running time vulnerability analysis instead of exponential.

The primary focus of all these models is network level attacks. A recent survey [6] says that other types of attacks such as insider threat are gaining prominence. Insider threat has long been considered a security problem [7], however, it is only recently that organizations are beginning to acknowledge and attribute financial losses to this threat. Therefore, the threat is no longer a hypothetical one. These types of threat are generally considered as practical problems which require practical solutions, mainly because there is an interplay of both technological and social factors. Systems-based approaches such as honeypots [8] have been proposed to tackle this threat, but these are detection mechanisms for live attacks and cannot replace preventive threat assessment. In general, good models are required to understand the problem well from a local and global perspective rather than relying on stopgap solutions. However, to the best of our knowledge, no prior work exists which effectively models insider threat and social engineering attacks. The difficulty in modeling these kinds of attacks lies in the fact that insiders are very familiar with their computational environment. They also have a greater set of stealthy actions at their disposal and can use an organization's own resources against itself. In this context, since it is impractical for a security analyst to think of all possible scenarios and corresponding actions, action-centric models are not suitable for these types of attacks.

In this paper, we propose a target-centric modeling methodology motivated by the fact that insiders typically pursue lucrative targets to cause damage or gain leverage. It is based on a higher level description of an organization's infrastructure and less detail-intensive as compared to the attack graph model. Some of the basic ingredients of this model are as follows. We assume that an insider is any legitimate user who is very familiar with his computational environment, and hence, the reconnaisance phase before a successful attack is very short or non-existent. In order to do damage, he must seek the so-called "jewels" such as proprietary source code and other confidential information. Attacks on a target are launched through a communication channel and if such a channel is not directly available, an insider may collude with other insiders or compromise sub-targets first. Deterrence to such attacks is offered only by deployed security systems or enforced policies. Our model allows representation of all this information in a very intuitive way. To summarize, our model is a target-centric approach and less detail-intensive as opposed to attack graphs which are action-centric. While our approach can be easily adapted to model common network level attacks, the main benefits are seen in modeling complex threats due to insider attacks, social engineering and colluding adversaries.

1.1 Overview

Figure 1 presents an overview of the model specification and analysis process. A security analyst who understands the layout of the organization and its infrastructure produces a model specification. We defer the details of the model to later sections.

1.2 Paper Organization

The rest of the paper is organized as follows. Section 2 discusses related work by peers and puts our work in perspective. Section 3 gives a detailed description of our formal model. Section 4 discusses various complex scenarios that may be modeled using our approach highlighting the coverage of our model over existing approaches. Section 5 presents natural questions that follow from this model and their preliminary hardness results. Section 6 discusses heuristics to arrive at approximate solutions, and

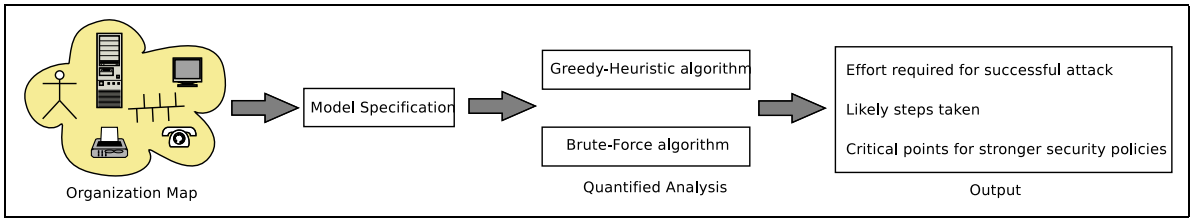

Figure 1: Overview of Model Specification and Analysis

Section 7 shows some experiments and results using these heuristics. Section 8 closes with conclusions and suggestions for future work.

2 Background and Related Work

We present an overview of the state-of-the-art practices and accomplished research results that currently exist on threat modeling and assessment. In this section, we highlight the strengths and weaknesses of these approaches as well as draw comparisons to put our work in perspective.

2.1 Security Audit Tools

A multitude of commercial tools [9–11] are available which perform both host and network level security assessment. The audit process involves performing a laundry list of security checks and running exploits against known vulnerabilities. Since real attacks are launched, they bear a strong resemblance to real-world scenarios. However, the coverage is limited to only known vulnerabilities. Automatically running these tools only reveals superficial security problems and it is difficult to understand their overall impact. On the other hand, a detailed and thorough security audit is often time-consuming and expensive.

2.2 Formal Models

Formal models offer an elegant solution to vulnerability assessment. The key to creating an effective abstract modeling of a system is finding a balance between flexibility, simplicity, and usability. The specification must be sufficiently flexible to capture different types of systems examined at different degrees of detail. It must also be sufficiently simple in order to represent the various participating entities in the model. Finally, a security analyst must be able to use it without incurring a significant learning curve.

Fault trees [12] are the first generation of formal models for system failure analysis. A high-level undesired event is taken as the root of the logic tree. Then, each possible cause is added to the tree as a series of logic expressions, creating an AND-OR tree. Failure probabilities are associated with the edges to calculate the probability of single faults or a combination of them, which could eventually cause the undesired event. While fault trees are suitable to model a disjunction and conjunction of faults, they lack the expressive power to capture attacks.

State transition based approaches model penetrations and attacks as a series of state transitions from some initially secure state to some compromised state. Porras et al. [13, 14] proposed one of the earliest rule-based state transition approaches for attack representation and detection. The attack graph [2,15,16] is a popular state-transition model to represent and analyze security problems. All possible atomic actions are represented as transitions and the goal is to find a path from an initial safe state to one or more final states denoting successful attacks. Attack graphs are essentially an extension of fault trees but

with the additional benefit that they can handle cyclic dependencies which are commonly encountered when modeling actions. Since all possible actions have to be represented, this puts a unrealistic burden on the analyst. Sheyner et al. [3] showed that model checking can be used to automatically generate attack graphs. Their technique used a modified model checker to extract *all* possible traces that lead to a successful compromise. Although the process is automated, model checking suffers from state explosion and this affects scalability. In fact, their paper reports that even for a small network and a few atomic attacks, attack graph generation took several hours. In their followup paper [4], they performed minimization analyses to assist a security analyst in deciding which minimal set of states should be eliminated to secure the network. Ammann et al. [5] argued that modeling at granularity of atomic actions represents far more details than necessary for a security analyst; instead, they use a higher level attribute based representation. Also, by relaxing the "undo" when backtracking during graph traversal, a polynomial time vulnerability analysis algorithm is presented. In our model, we also use a coarse grained representation and this translates to scalable model specification. Since our model is different from traditional attack graphs, the notion of successful attacks is not a path, and we have devised both exponential and polynomial time algorithms to perform vulnerability analysis.

Dacier et al. [17] introduced the concept of privilege graphs. Each node in the privilege graph represents a set of privileges owned by a user or a group of users, and each arc represents privilege escalation through some vulnerability. Attack state graphs are converted from privilege graphs by exploring the various ways in which an attacker may gain the required privileges. This process also suffers from a state explosion problem. Ortalo et al. [18] describe an experimental evaluation of the privilege graph framework under certain assumptions on the memory state of an attacker.

Petri nets [19, 20] are a formal model for information flow. In this token-based model, there are two types of nodes: places and transitions. Places contain tokens, and directed arcs connect places to transitions and transitions to places as pre- and post-conditions to taking transitions. The position of the tokens indicates a particular state of execution. Multiple tokens may exist in the Petri net at any point in execution signifying that multiple states hold true. This aspect of the Petri net is similar to our model, where an attacker may have compromised multiple sub-targets.

2.3 Metrics

The efficacy of a model is severely limited f it does not support quantification. Quantification allows an analyst to pose questions and obtain useful answers based on which critical infrastructural decisions can be made. The crux of quantification in most security mechanisms is not offering an absolute certainty of any given property but rather making guarantees given some assumption of the system and/or the attacker. Some research efforts have attempted to provide metrics in the context of security.

Dacier et al. [17] defined a metric called *mean effort to failure* or METF on attack state graphs. METF is based on associating transition probabilities on edges of the attack state graph representing the likelihood of a given transition. Exploring a particular path in the attack state graph gives the probability of occurrence of that path. In our model, metrics are integer values rather than probabilities. We associate an integer to estimate the resistance offered by a security system to an attacker's efforts.

Manadhata and Wing propose another metric [21] for evaluating the relative security of a system based on its *attack surface*. Again the system is modeled as a state machine and the adversary's goal as a state predicate. The point they make in the paper is that security need not be measured absolutely; relative metrics are effective too. Attack surfaces evaluate a relative comparison between different versions or flavors of the same system to determine if the attack surface of the system is more exposed. Intuitively, the more exposed the attack surface of a system, the more prone it is to attack. Similarly, in our model, we make relative guarantees rather than absolute ones.

3 Formal Model

In our model, we assume that an attacker is goal-oriented. Also, he is aware of the location of his potential targets and how to reach them. These assumptions closely model an insider and this is one of the reasons why our model is most suitable for this class of threats. We also assume that a successful compromise of a target is not possible if there is no channel of interaction. Finally, an attacker may not be able to use an existing channel of interaction with a potential target due to a strong security mechanism in place on that channel. This may force him to seek alternate routes to reach the target. Each sub-target that is compromised requires additional effort but provides the attacker with additional information/capability and another front to continue the attack. Given a model specification, the goal of vulnerability analysis is to exhaustively find the different ways in which attacker can reach the target.

3.1 Preliminaries

An instance of the model is constructed by a security analyst who is aware of the computational infrastructure. Prior to the formal definition of our model, which we call a *key challenge graph*, we describe the various components. Figure 2 shows the basic building block of the key challenge graph.

- Any physical entity on which some information or capability can be acquired is represented as a *vertex* of the graph. Let the set of vertices be denoted by V. Typically, vertices are points in the network where some information may be gained such as a database server or simply any computer system whose resources can be used or misused.

- Each piece of information or capability that is present at any vertex is represented as a *key*. Let the set of keys be denoted as K. For example, records in a database, passwords stored on a computer, or computational resources of a computer can be represented as keys. When an attacker visits a vertex, he is empowered with this additional information or capability.

- If there is a channel of access or communication between two physical entities which facilitates interaction, then a *directed edge* is created between the two corresponding vertices, pointing to the direction of the allowed interaction. Multiple channels of communication are possible, hence there can be more than one edge between two vertices. Let the set of edges be denoted by E. For example, assume a ssh server and a client computer. A channel of communication exists from the client to the server and a directed edge is drawn.

- The presence of a security measure or an enforced security policy protects the resources and allows only authorized interaction. This deterrence is represented as a *key challenge* on the corresponding channel of communication. An example of a key challenge is the password authentication required prior to accessing to a server.

- If a user does not have the right key to the key challenge, then he incurs a significant *cost* in breaking or circumventing the security policy; legitimate access incurs only a smaller cost of meeting the key challenge. For example, when password authentication is used, if a user knows the password, he incurs little or no cost, while another user who doesn't know the password will incur a higher cost in breaking the password. The cost metric is a relative quantity signifying the amount of deterrence offered by one security measure over another. It has been abstracted as a non-negative integer for the purposes of our model.

- The starting point of an attack could be one or more vertices in the graph, which are assumed to be initially in the control of an adversary. Let this set be denoted as V_0.

- The target of an attack could also be one or more vertices in the graph. In case of multiple targets, the goal is to compromise all of them. Let the set of target vertices be denoted by V_s. An example of a target is a source code repository for a commercial product.

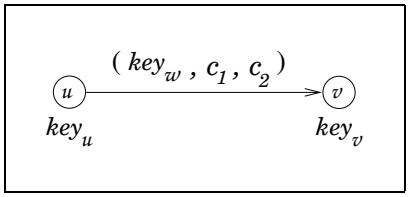

Figure 2: Basic building block of a key challenge graph

Definition 3.1 (Key Challenge Graph). A *Key Challenge Graph* or KG is a tuple:

$$KG = (V, E; K, V_0, V_s, \pi, \delta),$$

where V is the set of vertices, E is the set of edges, V_0 is the initial set of compromised vertices, V_s is the set of target vertices, $\pi : V \to K$ is a function that assigns keys to vertices, $\delta : E \to K \times \mathbb{N} \times \mathbb{N}$ is a function that assigns key challenges and costs to edges, and $\mathbb{N}$ is the set of natural numbers.

For example, $\pi(v_1) = k_0$ means that the key k_0 can be obtained at vertex v_1, $\delta(e_1) = (a, c_1, c_2)$ implies an assignment of a key challenge to edge e_1, which requires an attacker to produce the key a. If he cannot do so, then he incurs a cost c_1; otherwise, he incurs a smaller cost c_2.

An adversary begins his attack at some point in the set of compromised nodes in the graph and proceeds by visiting more and more vertices until the target(s) is reached. At each visited vertex, the attacker adds the corresponding key to his collection of keys picked up at previous vertices. Once an attacker compromises a vertex, he continues to have control over it until an attack is completed. Therefore, any vertex appears exactly once in the attack description. While a trivial attack can be performed by visiting all vertices until the target is reached, cost constraints occlude such free traversals. We shall now develop useful metrics for analysis of our model.

Lets assume that initially V_0 and V_s are disjoint sets, that is, an attacker has not successfully compromised any targets yet. We can define a successful attack as follows.

Definition 3.2 (Successful Attack). A *successful attack* is defined as a finite ordered sequence of a subset of vertices $(v_1, v_2, \ldots, v_m)$, where $V_s \subseteq \{v_1, \ldots, v_m\}$, and $V_0 \cap \{v_1, \ldots, v_m\} = \emptyset$.

In other words, a successful attack is a sequence of zero or more vertices not in the initial set V_0 but eventually containing all the target nodes in V_s. (Note that this sequence in general does not represent a path or a walk. We elucidate this point in illustrations in the following sections.)

The next important aspect of the model is the cost metric. Although an attack is defined exclusively in terms of vertices, the cost incurred by the attacker at a vertex is mainly dependent on the edge that he chooses to visit the vertex. We first define the cost of traversing an edge and then the cost of visiting a new vertex. The latter is the basic unit of cost metric in our model.

Definition 3.3 (Cost of Traversing an Edge). Let V^* be the set of visited vertices so far, including the initially compromised vertices, i.e. $V_0 \subseteq V^*$. For $u \in V^*$ and $v \notin V^*$, the *cost of traversing the edge* $e = (u, v) \in E$, given that $\delta(e) = (k, c_1, c_2)$, is c_1 if $k \notin \{\pi(w) \mid w \in V^*\}$. Otherwise, it is c_2. (In general, $c_1 > c_2$.) If $(u, v) \notin E$, for technical convenience we assume that $c_1 = c_2 = \infty$, and that k is some unique key no node has.

Definition 3.4 (Cost of Visiting a New Vertex). Define V^* as above. The *cost of visiting a new vertex* $v \notin V^*$ is defined to be

$$c(v, V^*) = \min\{\text{cost of traversing } (u, v) \mid u \in V^*\}. \tag{1}$$

Let NEW-VERTEX-COST be a procedure that computes this value. (Note that the cost of traversing an edge is implicit in this computation.)

The cost of an entire attack is measured as a sum of the effort required to compromise individual vertices by attempting to counter the key challenges on the edges with or without the keys that an attacker has already picked up.

Definition 3.5 (Cost of an Attack). The *cost of an attack* $v_1 v_2 \ldots v_m$ is defined as:

$$\sum_{i=1}^{m} c\left(v_i, V_0 \cup \{v_1, \ldots, v_{i-1}\}\right). \tag{2}$$

We will call this computation ATTACK-COST.

4 Modeling Complex Scenarios

The key challenge graph model is based on a high level representation of a organization's network. There is a one-to-one correspondence between the entities in the model and the infrastructure, making the representation very intuitive. The modeling process begins by enumerating all the computer systems and resources in the decreasing order of their importance/value, and these are represented as vertices in our KG model. For each vertex, find all incoming access channels and these become the directed edges. On each edge, verify if a security policy or authentication mechanism is in place and evaluate the relative deterrence. For example, if we were to compare password authentication against an strong cryptosystem, then we evaluate all feasible attacks on both these schemes and their difficulty. Based on attack hardness, an integer value is associated with the corresponding edge. An example calibration is the number of computer hours required to break the security counter-measures. The security challenge posed to any user combined with the costs of either legitimately answering the challenge or breaking it, forms the description of a key challenge. Note that quantification is an integral part of our model. Finally, vulnerability analysis begins by running through various scenarios involving different attacker entry points and target nodes. We defer the actual vulnerability analysis algorithms to later sections.

We now turn to application of our model to specific examples. We have performed small scale modeling of academic and banking environments, and these examples are drawn from our experience. The goal of these particular illustrations is to show the effectiveness of the KG model in capturing the various threats and suggesting counter-measures after threat assessment.

4.1 Insider Threat

Insider threat is a potential problem in any organization that conceals or protects valuable information. Examples of such information are secrets concerning national security guarded by the military, a customer's personal finances kept private by a bank, and a student's grades protected from tampering. We choose an example, which is set in a banking environment.

Consider the following description. From his workstation, every teller can perform sundry personal accounting tasks. Each personal account transaction cannot exceed US$ 5,000. A manager has to endorse any larger transactions and personally handles accounts involving business organizations. The business

accounts database is kept separate from the one that houses personal accounts. Any bank transaction, either by the teller or the manager, which operates on a database is encrypted. The communication between the teller's computer and the personal accounts database is frequent but not of high value, so it uses a lower strength symmetric encryption. On the other hand, the manager authenticates himself to a PKI server with his credentials, and obtains a session key to talk to the database and complete the transaction. Another key piece of information known to the teller is that the manager doesn't apply security patches to his computer frequently.

Now, consider an insider threat that manifests in the form of a rogue teller and the target being the business account database. In order to understand how an insider would proceed in his attack, we model this scenario using information availability, enforced security policies and corresponding effort required. Figure 3 shows the representation. Every physical entity, where some information resides, becomes a

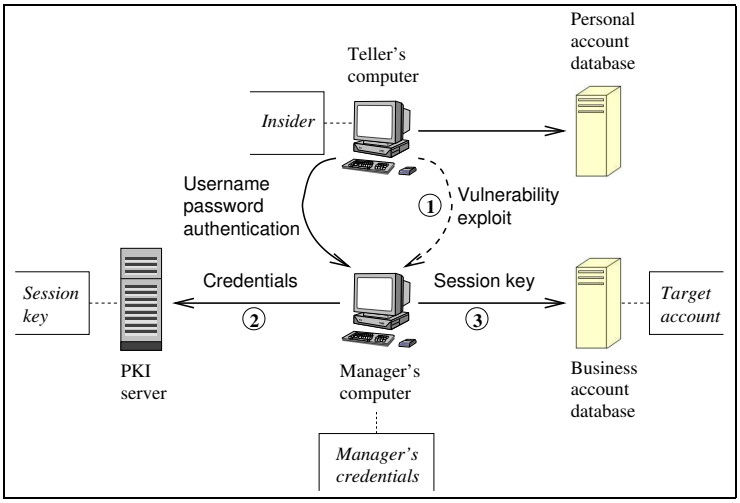

Figure 3: Modeling an example insider threat scenario

node. The physical entities are: 1) the teller's computer, where the insider attack commences, 2) the manager's computer, which holds his credentials, 3) the two databases, and 4) the PKI server. There is an edge from one entity to another if there is a channel of communication between them. Security policies may be enforced over this channel of communication. If a legitimate user has the appropriate response to these challenges, then very little effort is required to initiate communication, else significant effort has to be invested. For example, consider that the communication channel between the teller's and manager's computers requires authentication. Without knowledge of the correct username and password, a brute force attack could take substantial time and become conspicuous. However another way into the manager's computer is by using an exploit.

Now, consider that the attacker is the teller and the target is the business accounts database. Using the KG model representation, the steps taken by an insider (shown numerically in Figure 3) can be easily seen. The most likely sequence of steps is: 1) use a vulnerability exploit to compromise the manager's computer, 2) use the manager's credentials to obtain a session key from the PKI server, and 3) use the session key to attack the business account database. Simply with a key piece of information, an insider is able to launch a very damaging attack, and our KG model is able to provide a very intuitive view of the attack. The solution in this scenario is trivial, i.e., the manager's computer is to be patched.

We point out a few properties based on the example. The sequence of steps taken by the attacker is generally *not* a path. This is a striking departure from the attack graph model, where attacks appear as paths in the graph. While it is still possible to represent the example using an attack graph, the

permutations of actions implicit in the graph would result in an exponential explosion of states. Note that in our representation, only information available at various points is represented and therefore, the resulting graph is very small. Another observation is that once a particular entity is compromised (for example, the manager's computer), an attacker can always return to this entity in the future to continue down another branch in the graph.

4.2 DoS Attacks By Email Worms

We present a illustration of a denial of service (DoS) flood attack on a mail server through email worms. This attack example is based on our modeling experience in an academic environment. Transmitted via email, the most commonly used medium, these worms are one of the most effective types of social engineering attacks. They exploit a recipient's curiosity to open a virus attachment. At this point, the virus replicates and sends a copy to every entry in the recipient's address book. The flood of emails eventually overloads the mail server and results in a successful DoS attack.

Consider the following scenario (see Figure 4). A single mail server provides email service to users User1, User2 and User3 in the organization. Each user has the other user's email id in his address book. All mails to and from these users must travel via the mail server. The corresponding hosts, i.e., Host1, Host2 and Host3 from which the users work, are running resource usage based anomaly detection systems, and spikes in resource usage or outgoing connections are quickly flagged. Currently known email worms are stealthy, and they spread by using very little resources and limiting the number of connections made to the mail server. Due to these reasons, a mail flood on the mail server cannot be caused by infecting just one host. However, the maximum number of connections that a mail server can tolerate before being overwhelmed is less than or equal to the sum total of the maximum allowed outgoing rates of all three infected hosts. Therefore, a successful DoS attack entails the infection of all three host machines.

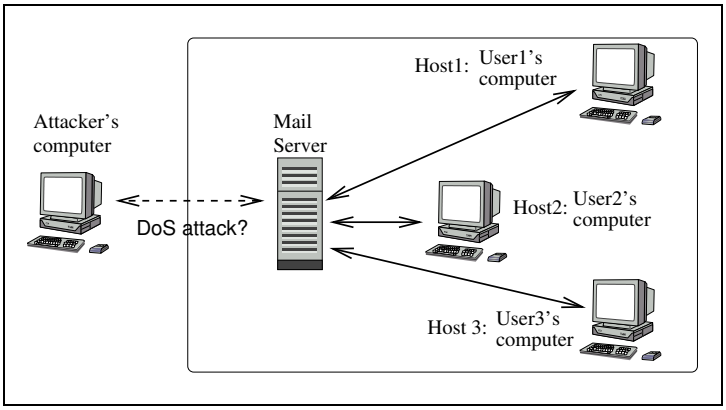

Figure 4: A Mail server configuration

The KG model specification is shown in Figure 5. Each physical entity reachable via email becomes a node. Here the nodes are: 1) the three client machines, 2) the attacker's machine, and 3) the mail server. Reaching any entity signifies that it is infected. There is an edge between two hosts (X,Y) if Y is in X's address book. Security policies enforced on individual nodes translate into the cost of traversing an edge into that node. Now, we model the threat of a DoS attack through a malicious email initiated from an attacker. There is an edge from the attacker to Host 1, signifying that the attacker will initially release the worm to Host 1. Compromising a host gives the attacker a certain capability in terms of resources,

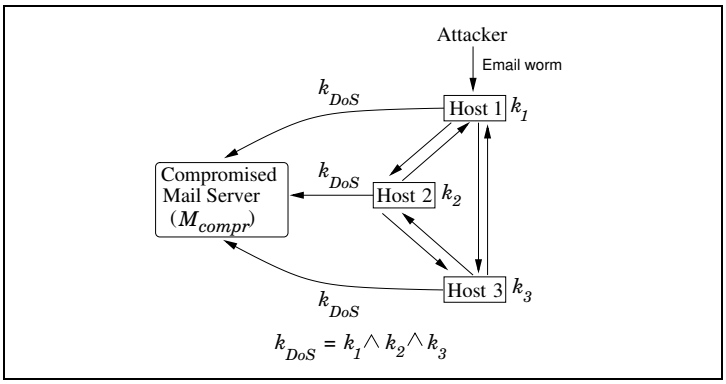

Figure 5: Modeling the email scenario

represented by the keys K_i, $i = \{1,2,3\}$. Finally, bringing down the mail server requires the compromise of all three clients. Therefore, the minimal key challenge for any edge to achieve a flooded mail server M_{compr} is $k_{DoS} = (k_1 \wedge k_2 \wedge k_3)$. We have omitted the costs because infecting a host is trivial and there is very little cost involved.

From the KG model specification, it can be seen that a distributed denial of service attack can be prevented if the mail server is instructed to randomly drop mail connections from at least one target, i.e., at least one k_i is not reached. A more permanent fix would require updating the anti-virus signatures when they are released. In our approach, since we model DoS attacks through potential targets and accrued capability, we are able to easily ask whether an attacker has gained critical mass to launch the DoS attack. Referring to attack graphs and model checking, DoS attacks violate the liveness property that a user will eventually have access to a service. Model checking [22] in its current state has certain limitations. While modeling safety properties is easily done, modeling "liveness" properties is hard.

4.3 Social Engineering Attacks

Social engineering attacks is a class of attacks which exploits factors outside the domain of computational infrastructure. Often, humans in the loop are the weakest link and social engineering attacks commonly target their gullibility. A typical example of this type of attacks involves an attacker interacting with some personnel inside the organization though a phone call and pretending to be someone important and tries to extract valuable information[1]. The key challenge graph model allows for a very general kind of communication channel. This means that it is possible to represent not only wired and wireless network media, but also channels such as telephone lines, and this still falls within the framework of our model. For example, when a customer calls a credit card company, a representative poses a key challenge in the form of date of birth or social security number, and divulges information only when the right information is presented.

4.4 Colluding Attackers

In order to improve the chances of a successful attack, two or more attackers controlling different vertices may collude and share the keys that they possess. In this case, the set V_0 contains all these vertices and $|V_0| > 1$. This situation is no different from the one where a single attacker may initially have control

[1]Such attacks have been perpetrated by notorious hackers like Kevin Mitnick in the 90's, which reportedly caused significant damage to various companies' assets.

over multiple vertices. This would not complicate the analysis of the model as an attack is represented not as a path but rather as a key sequence.

5 Theoretical Analysis

We have presently completed the basic formulation of the key challenge graph model as well as presented realistic scenarios using our model. Since we abstract out several action-centric details, model representation is much simpler and scales linearly with the input in terms of nodes and edges. Generating this abstract representation of a system and demonstrating its usefulness is only the first step. The next important aspect is setting up various attack scenarios and analyzing the model to reveal vulnerabilities.

One can look at this task from a algorithmic complexity viewpoint: a good attacking strategy is, in some sense, equivalent to a good algorithm or approximation algorithm [23] to find a minimum-cost attack on the key challenge graph.

5.1 On the Complexity of Analyzing Key Challenge Graphs

Given a key challenge graph, we may consider the problem of finding an attack with the minimum cost. Recall that an attack is a sequence of vertices $(v_1, \ldots, v_m)$ such that $V_s \subseteq \{v_1, \ldots, v_m\}$. The objective function is the cost of orchestrating the attack. We will call this problem KEYSEQ.

We shall first show that KEYSEQ is NP-hard by showing that its decision version is NP-complete [24]. The decision version of KEYSEQ asks if there is an attack whose cost is at most some given positive integer C. (In fact, the optimization problem KEYSEQ is difficult even to approximate up to a certain ratio.)

Lemma 5.1. *The decision version of* KEYSEQ *is* NP*-complete.*

Proof. First, we prove that the decision version of KEYSEQ is in NP. Given an instance $G = (V, E, K, V_0, V_S, \pi, \delta)$ of the problem, and a cost upper bound C, one can guess an attack $\mathcal{S} = (v_1, \ldots, v_m)$, where $v_i \in V$. The verification that this is a good attack can be done by checking that $V_s \subseteq \{v_1, \ldots, v_m\}$, and that the total cost of the attack is at most C using formulas (1) and (2). This verification certainly can be accomplished in polynomial time (at most $O(|V|^2)$).

Next we prove that KEYSEQ is NP-hard via a reduction from 3-SAT [24]. In the 3-SAT problem, one is given a set of n boolean variables $X = \{x_1, ..., x_n\}$ and a 3-CNF formula ϕ over X. The formula ϕ consists of clauses C_1, ..., C_m of size 3. The problem is to determine if there is a truth-assignment to x_1, ..., x_n such that ϕ is evaluated to be true. We construct an instance of KEYSEQ corresponding to ϕ as follows. Let the set of vertices V consist of the following nodes:

1. the literals $x_1, ..., x_n$, and $\bar{x}_1, \ldots, \bar{x}_n$
2. the clauses C_1, ..., C_m
3. a starting state v_0, i.e., $V_0 = \{v_0\}$
4. a success state v_s, i.e $V_s = \{v_s\}$, and
5. $n - 1$ intermediate nodes v_1, .. v_n.

Let the set of keys be $K = V$, and the vertex to key mapping π be the identity function. The set of edges E and the corresponding key challenges are constructed as follows:

1. for $i = 1, \ldots, n$, construct the following edges: (v_{i-1}, x_i) with $\delta(v_{i-1}, x_i) = (x_i, 1, 0)$, $(v_{i-1}, \bar{x}_i)$ with $\delta(v_{i-1}, \bar{x}_i) = (\bar{x}_i, 1, 0)$, (x_i, v_i) with $\delta(x_i, v_i) = (v_i, 1, 0)$, and $(\bar{x}_i, v_i)$ with $\delta(\bar{x}_i, v_i) =$

$(v_i, 1, 0)$. The main idea is that, to get from v_{i-1} to v_i, one has to pay a cost of at least 2, while obtaining at least one of the keys x_i or $\bar{x}_i$. If both of these keys are obtained, then it must have been the case that a cost of at least 3 was paid.

2. three edges (v_n, C_1) each representing a variable in the clause C_1. The key challenge on each of these edges is of the form $(l_i, \infty, 0)$, where l_i a literal in C_1. The infinity cost could be any large enough integer ($\geq 3n$, e.g.).
3. similarly for $j = 2, \ldots, m$, three edges (C_{j-1}, C_j), and literals in the clause C_j appear in the key challenge.
4. A final "free" edge (C_m, v_s) signaling that all constraints have been met. ("Free" here means both costs are zero, having the key challenge or not.)

It is now straightforward to see that there is an attack of cost $\leq 2n$ in this instance *iff* ϕ is satisfiable. □

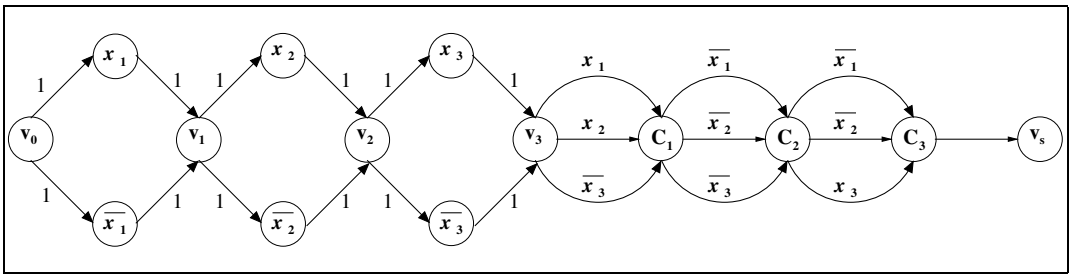

Figure 6: A sample reduction from 3-SAT

We give a short example to illustrate the reduction. Consider $\phi = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$. The key challenge graph corresponding to this problem is given in Figure 6. Notice that there is a satisfying assignment $\{x_1 = 1, x_2 = 0, x_3 = 1\}$. A successful key sequence of cost 6 is v_0, x_1, v_1, $\bar{x}_2$, v_2, x_3, C_1, C_2, C_3, v_s. Given this complexity result, our aim is now to derive efficient heuristics and approximation algorithms to analyze key challenge graphs.

Corollary 5.2. KEYSEQ *is* NP*-hard.*

We have seen that an optimal attack is in general difficult to obtain in a reasonable amount of time, unless P=NP.

6 Algorithms

We believe that solving KEYSEQ to optimality is very hard. However, it is possible to get an estimate of the optimal solution using heuristics. We present a brute force algorithm along with a heuristic algorithm for the purposes of comparison.

The brute force algorithm BRUTE-FORCE (see Table 1) generates all possible permutations of attack sequences and finds the minimum cost among them. Without loss of generality, let $V_0 = \{v_0\}$ and $V_s = \{v_s\}$. Given a set S, let PERMUTE(S) signify the set of all possible permutations of elements of S without repetitions. The running time of this algorithm is super-exponential but it solves the KEYSEQ problem to optimality.

We now describe our polynomial-time heuristic called GREEDY-HEURISTIC (see Table 2) which is based on the observation that a key sequence is structurally a path from some initial vertex to a final target

```
BRUTE-FORCE(KG)
1  min_cost ← 0
2  for each S ⊆ V − (V_0 ⋃ V_s)
3  do for each s ∈ PERMUTE(S)
4     do cost ← ATTACK-COST(v_0 s v_s)
5        if cost < min_cost
6           then min_cost = cost
7  return min_cost
```

Table 1: A brute force algorithm to find cost of optimal key sequence

vertex with zero or more branches from this backbone path, taken to collect additional keys. We use a greedy approach with the all-pairs shortest path (APSP) as the core decision-making procedure. Given a $n \times n$ adjacency matrix of a graph $G = (V, E)$, the APSP algorithm computes the all-pairs shortest path matrix, which gives the shortest path between any pair of vertices in the graph G. However, we cannot use this algorithm directly since the input that is available to us is a key challenge graph and not a weighted graph. We now briefly describe the algorithm UPDATED-ADJ-MATRIX, which converts a key challenge graph to a weighted graph. The main idea is that when an attacker acquires a new key, then weights on all edges having this key in the corresponding key challenge will reduce to the lower cost, otherwise they reduce to a higher cost. GREEDY-HEURISTIC proceeds by evaluating which neighboring key if acquired would give a shorter backbone path from the source vertex to the target vertex than the one currently seen. After at most $|V|$ rounds of decision-making, the algorithm returns a cost which cannot be reduced further. Invocation of the APSP algorithm inside the loop results in a worst case running time of $O(n^5)$. An insight into the algorithm is that we use both local (neighboring keys) and global (shortest path) factors to find the approximate solution. At this point, we do not know what the approximation ratio this algorithm yields and can only evaluate it empirically.

7 Experiments

We have performed empirical evaluations to compare BRUTE-FORCE with GREEDY-HEURISTIC. Our experiments were conducted on a Pentium 4/3GHz/1GB RAM running RedHat Linux 9.1. Since the BRUTE-FORCE algorithm has a prohibitively expensive running time, we have limited the size of the input to only 15 nodes. Our test data set consists of 1000 simulation runs and each run generates a random instance of a key challenge graph. We have compared the quality of the solutions (see Figure 7) computed by the two algorithms as well as their running times (see Figure 8).

When comparing the attack costs returned by the two algorithms, we have used randomly generated key challenge graphs of exactly 15 nodes for all the 1000 runs. GREEDY-HEURISTIC worked very well when the final attack sequence is very small (3-4 nodes) and this seems to be the best case scenario for the heuristic algorithm. However, when the attack sequence is long (10-15 nodes), the heuristic produces a larger gap from the optimal solution. The explanation we give for this observation is that longer the sequence, greater the chances that the decision-making procedure will make errors which are compounded.

When comparing the running time, the 1000 runs had a cross-section of varying graph sizes (3-20 nodes). The running time of the BRUTE-FORCE algorithm becomes very large even for small values of 13 to 15. Clearly, this is the expected behavior as the running time of this algorithm is $O(n!)$, which

```
GREEDY-HEURISTIC(KG)
 1  S ← V_0
 2  M ← UPDATED-ADJ-MATRIX(KG, π(S))
 3  A ← APSP(M)
 4  min_cost ← A[v_0, v_s]
 5  for round ← 1 to |V|
 6  do
 7      flag ← 0
 8      for each v_i ∈ NEIGHBOR(S)
 9      do vertex_cost ← NEW-VERTEX-COST(v_i)
10          M' ← UPDATED-ADJ-MATRIX(KG, π(S ⋃ {v_i}))
11          ∀v_j ∈ S,  M'[v_j, v_i] ← 0,  M'[v_i, v_j] ← 0
12          A' ← APSP(M')
13          if (vertex_cost + A'[v_0, v_s]) < min_cost
14            then min_cost ← vertex_cost + A'[v_0, v_s]
15                 new_vertex ← v_i
16                 flag ← 1
17      if flag = 1
18        then
19             S ← S ⋃ {new_vertex}
20        else return min_cost
21  return min_cost
```

Table 2: A greedy heuristic to find cost of near-optimal key sequence

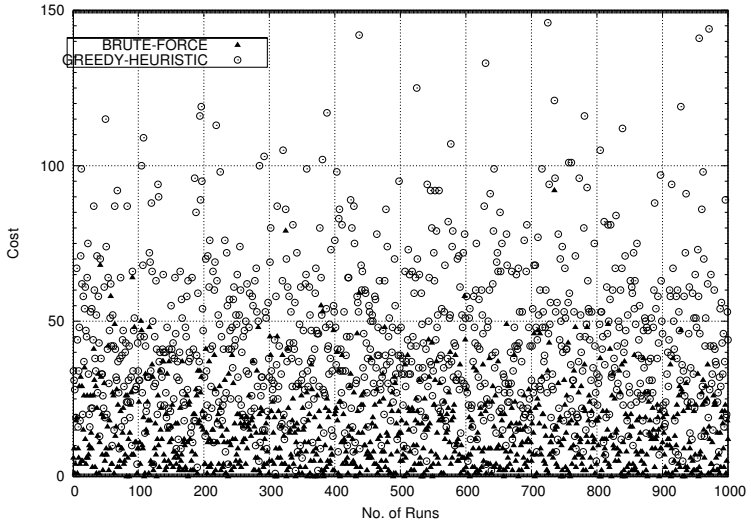

Figure 7: GREEDY-HEURISTIC vs BRUTE-FORCE: Minimum cost of an attack

is worse than exponential. On the other hand, the GREEDY-HEURISTIC has polynomial running times for the same input. Even for graphs with a large number of nodes (200-300 nodes), we have observed a

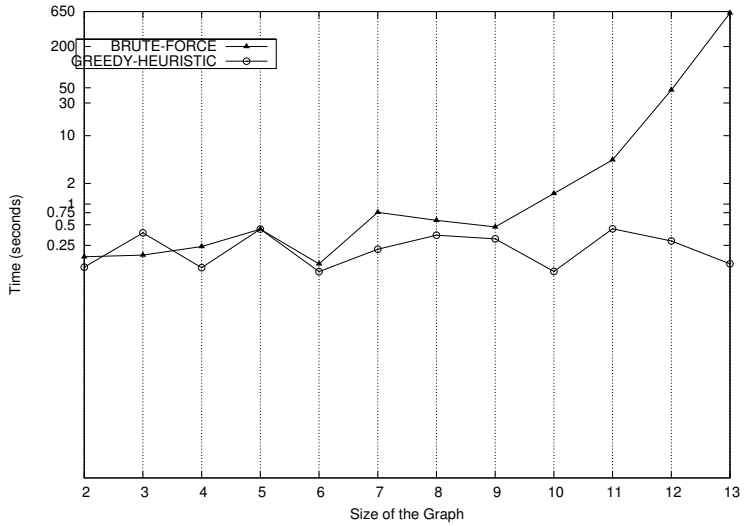

Figure 8: GREEDY-HEURISTIC vs BRUTE-FORCE: Running time behavior (on log-scale)

running time of only a few minutes (15-20 minutes).

8 Conclusions and Future Work

In this paper, we have presented a target-centric formal threat assessment model. The nature of the model makes it suitable to address complex threats such as insider threat, DoS attacks, and social engineering. Although these security problems are becoming very common, to the best of our knowledge no known threat models can effectively capture and evaluate these classes of threats. The work we have presented is aimed at advancing the state of threat research towards addressing these problems.

While our model appears different from action-centric models, the difference is only superficial. It is possible to expand each key challenge graph instance to an attack graph but at the price of a blowup in the number of states and the information that is represented. Therefore, ours is in fact a very general yet compact modeling methodology. Another way to look at the two models is that they lie at opposite ends of the spectrum, one which is very action-intensive, while the other is more target-oriented. The advantage that our model has gained is a scalable representation which is very close to the actual network and infrastructure, but on the downside, we lose the precise details of an attacker's actions.

The main goal of threat assessment is identification and then quantification of the threat. The cost metric built into our model yields a qualified measure of the vulnerability of a system, allowing a security analyst to ask natural questions such as, "What is the corresponding minimum effort required of an attacker to perform this attack?". Also, given this metric, our model can simulate the attack graph notion of stealthy versus detectable attack sequences.

With respect to complexity, we have shown that although the model representation is very succinct, analysis based on the semantics of an attacker's actions is very hard. We have devised a polynomial-time heuristic to overcome this hurdle and are working towards improved approximation results. An interesting corollary of the hardness of quantitative analysis is that given an entry point and a set of targets, not only the security analyst but also an attacker will find it extremely difficult to plot an optimal attack in the worst case.

The directions of future work are primarily focused on improvements in the key challenge graph

model. In the context of quantified analysis, we would like to investigate the effects of embellishing the basic model with additional information on its expressive power and hardness of analysis. We are also looking at devising better heuristics to close the gap to the optimal solution. From a theoretical computer science point of view, We have posed some conjectures on the hardness of the KEYSEQ problem and speculate that if the problem is harder than we believe it currently is, then it may find applications in other domains such as cryptography or serve as another fundamental problem in theoretical computer science.

References

[1] C. Phillips and 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 Press, 1998.

[2] C. Phillips and L. P. Swiler, "A Graph-Based System For Network-Vulnerability Analysis," in *Proceedings of 1998 New Security Paradigms Workshop*, (Charlottesville, Virginia), pp. 71 – 79, 1998.

[3] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated Generation and Analysis of Attack Graphs," in *Proceedings of the IEEE Symposium on Security and Privacy*, (Oakland, CA.), May 2002.

[4] S. Jha, O. Sheyner, and J. Wing, "Two Formal Analyses of Attack Graphs," in *15th IEEE Computer Security Foundations Workshop (CSFW'02)*, (Cape Breton, Nova Scotia, Canada), pp. 49–63, 2002.

[5] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, graph-based network vulnerability analysis," in *Proceedings of the 9th ACM conference on Computer and communications security*, pp. 217–224, ACM Press, 2002.

[6] "2003 CSI/FBI Security Survey." Computer Security Institute, 2003. http://www.gocsi.com/forms/fbi/pdf.html.

[7] A. R. Clyde, "Insider threat identification systems," in *Proceedings of the 10th National Computer Security Conference*, Sept. 1987.

[8] L. Spitzner, "Honeypots: Catching the insider threat," in *Proceedings of the 19th Annual Computer Security Applications Conference*, (Las Vegas, NV, USA), 2003.

[9] SAINT, "Vulnerability Scanning Engine." http://www.saintcorporation.com/.

[10] Nessus, "Security Scanner for Various Flavors of Unix and Windows." http://www.nessus.org/intro.html.

[11] INSECURE.ORG, "Top 75 Security Tools," 2003. http://www.insecure.org/tools.html.

[12] J. Gorski and A. Wardzinski, "Formalizing Fault Trees," *Achievement and Assurance of Safety*, pp. 311–327, 1995.

[13] P. A. Porras and R. A. Kemmerer, "Penetration State Transition Analysis A Rule-Based Intrusion Detection Approach," in *Proceedings of the Eighth Annual Computer Security Applications Conference*, (San Antonio, Texas), pp. 220–229, December 1992.

[14] K. Ilgun, R. A. Kemmerer, and P. A. Porras, "State Transition Analysis: A Rule-Based Intrusion Detection Approach," *IEEE Transactions on Software Engineering*, 1995.

[15] C. Meadows, "A Representation of Protocol Attacks for Risk Assessment," in *DIMACS Series in Discrete Mathematics and Theoretical Computer Science: Network Threats* (R. N. Wright and P. G. Neumann, eds.), vol. 38, December 1998.

[16] L. P. Swiler, C. Phillips, D. Ellis, and S. Chakerian, "Computer-Attack Graph Generation Tool," in *DARPA Information Survivability Conference and Exposition (DISCEX 11'01)*, vol. 2, June 2001.

[17] M. Dacier, *Towards Quantitative Evaluation of Computer Security*. PhD thesis, Institut National Polytechnique de Toulouse, December 1994.

[18] R. Ortalo, Y. Dewarte, and M. Kaaniche, "Experimenting With Quantitative Evaluation Tools For Monitoring Operation Security," *IEEE Transactions on Software Engineering*, vol. 25, pp. 633–650, September/October 1999.

[19] E. W. Mayr, "An Algorithm For The General Petri Net Reachability Problem," *SIAM J. Computing*, vol. 13, pp. 441–460, August 1984.

[20] K. Jensen, *Colored Petri Nets: Basic Concepts, Analysis Methods, And Practical Use*, vol. 1.2. Berline, Heidelberg: Springer-Verlag, 1992.

[21] M. Howard, J. Pincus, and J. Wing, "Measuring Relative Attack Surfaces," in *Proceedings of Workshop on Advanced Developments in Software and Systems Security*, 2003.

[22] E. Kindler, "Safety and liveness properties: A survey," *Bulletin of the European Association for Theoretical Computer Science*, vol. 53, pp. 268–272, 1994.

[23] D. S. Hochbaum, ed., *Approximation Algorithms for NP Hard Problems*. Boston, MA: PWS Publishing Company, 1997.

[24] M. R. Garey and D. S. Johnson, *Computers and Intractability*. San Francisco, Calif.: W. H. Freeman and Co., 1979. A Guide To The Theory Of NP-Completeness, A Series of Books in the Mathematical Sciences.