Analyzing uncertainty in TG protection graphs with TG/MC

Article *in* Journal of Computer Security · August 2010

DOI: 10.3233/JCS-2009-0378 · Source: DBLP

Jim Alves-Foss
University of Idaho
150 PUBLICATIONS **1,485** CITATIONS

Analyzing uncertainty in TG protection graphs with TG/MC

James R. Conrad[1] and Jim Alves-Foss

Department of Computer Science, University of Idaho

Moscow, Idaho 83844–1010 USA

E-mail: {conr2286, jimaf}@uidaho.edu

Sauchi Stephen Lee

Department of Statistics, University of Idaho

Moscow, Idaho 83844-1104 USA

We introduce TG/MC, a Monte-Carlo approach for evaluating the impact of uncertainty about vulnerabilities upon forecasts of security for a real-world system modeled by a protection graph. A TG/MC model defines a vulnerability as a potential change to an otherwise safe initial protection graph that, if exploited, leads to an unauthorized state, a violation of the system's security policy through the application of TG rules. TG/MC captures uncertainties about vulnerabilities as probability distributions and forecasts the probability of a specific security violation. TG/MC extends beyond the rigid yes/no analysis of safety in a TG protection graph to consider uncertainty in questions of security for real-world systems.

Keywords: Take-Grant, TG, protection graph, Monte-Carlo, security

1. Introduction

The Take-Grant (TG) Protection Model addresses questions about the transfer of rights and information in a computer security model [1, 3, 19]. TG is notable because specific questions about the safety of a protection graph, G, having an initial state G_0, are decidable in time linear with the size of G [2]. However, when applying TG to real-world systems, much about the security

[1]Corresponding author: 5350 W. Banker Dr., Boise, ID 83714 USA, Tel.: +1.208.396.4842.

analysis remains uncertain. The real-world system may not implement all of its claimed features, it may implement undocumented features, or it may not implement the features exactly as claimed. Likewise, an attacker's capabilities, resources, objectives or timing may be uncertain. While analysts can model many expected behaviors of a real-world system with a protection graph, the graph likely includes significant uncertainty. Uncertainty can lead a model to misrepresent its associated real-world system, leaving the analyst unable to infer the system's security from the model's safety. The Take-Grant Monte-Carlo (TG/MC) approach seeks to address questions of security in an uncertain system over the foundation of TG's formal reasoning.

TG/MC evaluates a security risk, the probability of a security violation, by capturing uncertainty in modeling parameters and simulating the security risk's impact upon the model's resulting forecast. Knowledge of security risks is particularly important to business decision makers when the investment opportunities exceed the available resources. In order to make an informed decision regarding investment opportunities, business leaders need to know more than just an attacker *can* do something "bad;" they also need to understand the probability of that "bad" outcome [9].

TG/MC employs probability distributions to define uncertainty in the modeling parameters representing the real-world real system. The Monte-Carlo simulation repeatedly samples the TG modeling parameters, builds a trial protection graph, evaluates a TG model to address questions of safety about the graph, and collects the result. Each trial protection graph represents an iteration of the Monte-Carlo simulation, an experiment testing the safety of a possible configuration of the protection graph. The TG/MC approach evaluates the safety of a particular trial protection graph in the TG model, while, at a higher level, the Monte-Carlo simulation expresses uncertainty about the real-world system. The TG framework remains unmodified, allowing all of the model's existing rules and theorems to remain available for reasoning about questions of safety in the protection graph. TG/MC augments the TG safety analysis to consider the probability of a security violation in the real-world system.

The remainder of this paper provides an overview of the TG/MC approach. Section 2 reviews the Take-Grant model. Section 3 notes the sources of uncertainty in security models. Section 4 provides an overview of the Monte-Carlo technique. Section 5 discusses the simulation of uncertainty in security models with the TG/MC approach. Section 6 illustrates an example application of the TG/MC approach for the computing equipment located in an electrical substation. Section 7 summarizes the benefits and future directions for TG/MC.

2. Overview of the Take-Grant Protection Model

A *protection model* abstracts essential behaviors of a *protection system*, a mechanism implementing a security policy. A model is said to be *safe* if it cannot enter a state in which an unauthorized flow of rights or information occurs [1]. While the safety of a protection system is undecidable in general cases [15], those expressed within a TG model are decidable in time linearly proportional to the size of their protection graphs [19]. TG provides a framework for investigating the safety properties of protection systems. To promote readability, this paper adopts terminology, graphics and labeling consistent with those used previously by Bishop in the TG literature [2].

The TG Protection Model consists of a set of rules defining permissible operations on a *protection graph*, a directed graphical representation of the associated real protection system. *De jure* rules govern the transfer of authority while the *de facto* rules address the transfer of information [1]. This paper focuses on transfers of authority (rights).

Named vertices in the protection graph represent either *subjects* or *objects* in real-world systems. The illustrations in this paper follow the TG literature convention of plotting subjects as solid-filled circles, objects as unfilled circles, and using cross-filled circles when a distinction is unnecessary. Subjects can *act*, that is, they can invoke a *de jure* rule to change the state of the protection graph. Subjects typically represent actors or processes in a real-world system. Objects cannot act and thus cannot change the state of the protection graph. Objects typically represent data in a real-world system because data can be trusted not to act. Note that this trust can be misplaced as exemplified in a model of the buffer-overflow attack in Section 2.2.

Different protection systems are represented by variations in their associated protection graphs. The TG rules never change; only the protection graph changes from one model to the next.

This paper refers to the initial state of a protection graph, G, as G_0. A *witness* describes a sequence of *de jure* rule applications updating the graph to produce G_1, $G_2 \dots G_n$, in succession, from the initial graph, G_0. A single change of state of the protection graph can be denoted by $G_i \vdash G_{i+1}$, and a series of rule applications can be written, $G_i \vdash^* G_n$.

A labeled edge in a protection graph represents the rights of that edge's origin vertex over the destination. The TG Protection Model defines two distinguished rights, *take* and *grant*, usually abbreviated in the literature as t and g respectively. The TG literature describes, for example, a subject **x** having the *take* right to vertex **v** as, "**x** has (t to **v**)." Models of information flow consider two additional rights, *read* and *write*, often abbreviated as r and w respectively.

Discussions of TG protection graphs often describe paths through a graph. A *tg-path* is defined as a path composed entirely of edges (in either direction) labeled as either t or g between vertices of a protection graph. Two vertices are said to be *tg-connected* if there exists a *tg-path* between them. Finally, an *island* is defined as a graph *component*, a maximally-connected subgraph constructed soley with *tg-connected* subjects.

2.1 The Graph Rewriting Rules

The *de jure* rules include the four important *graph rewriting* operations [1] summarized below. The application of these rules change the state of a protection graph. These four rules establish a framework for subsequent theorems within the TG model.

2.1.1 The Take Rule

Figure 1 illustrates an application of the *take* rule in G_0 in which subject $\mathbf{x}$ takes access rights (α to $\mathbf{z}$) from vertex $\mathbf{y}$, where $\alpha \subseteq \beta$, producing an updated protection graph G_1. Subject $\mathbf{x}$ is capable of action and is furthermore authorized to act because it possesses the t right to $\mathbf{y}$. However subject $\mathbf{x}$ can only *take* (copy) rights possessed by $\mathbf{y}$; afterward, $\mathbf{y}$ retains (β to $\mathbf{z}$). Note that although $\mathbf{x}$ must be a subject, the other vertices may be subjects or objects as their role is passive.

2.1.2 The Grant Rule

Figure 2 illustrates an application of the *grant* rule in G_0 in which subject $\mathbf{x}$ grants access rights (α to $\mathbf{z}$) to vertex $\mathbf{y}$, where $\alpha \subseteq \beta$, producing an updated protection graph G_1. Note that subject $\mathbf{x}$ is again the only actor and is authorized to act because it possesses the g right to $\mathbf{y}$; $\mathbf{x}$, in this example, can only *grant* rights already in its possession. Afterward, $\mathbf{x}$ retains (β to $\mathbf{z}$). Again, $\mathbf{x}$ must be a subject but the other vertices can be either subjects or objects.

2.1.3 The Create Rule

Figure 3 illustrates an application of the *create* rule in G_0 in which subject $\mathbf{x}$ creates a new vertex $\mathbf{y}$ and edge (α to $\mathbf{y}$) producing an updated protection graph G_1. Vertex $\mathbf{x}$ must be a subject.

2.1.4 The Remove Rule

Figure 4 illustrates an application of the *remove* rule to G_0 in which subject $\mathbf{x}$ removes the rights (α to $\mathbf{y}$), where $\alpha \subseteq \beta$, producing an updated protection graph G_1. If an edge's set of rights becomes empty then the edge is removed from the graph. The TG Protection Model does not define a rule for removing a vertex.

2.2 Modeling the Buffer Overflow Attack with TG

The ubiquitous buffer overflow attack [17,23] provides a convenient example illustrating how to apply the TG *graph rewriting* rules to model a current computer security issue. The buffer overflow attack can occur when a defender's trusted software accepts a meticulously crafted and oversized data object from an untrusted user, the attacker. The attacker constructs the object with knowledge of the application's storage map, allowing the oversized data to overwrite an address variable in the application's data segment, often the *return address* to a calling procedure which will be later dereferenced by the application. When the trusted application eventually de-references the attacker-supplied content of the address variable, it typically branches to attacker-supplied code (often within the oversized data object) rather than to trusted code. After the system has begun executing the attacker-supplied code, the attacker *owns* (controls) the defender's system.

The buffer overflow attack is an effective weapon in the attacker's arsenal.

The defender takes no risk other than executing a vulnerable software program to process the attacker-supplied data content. The data can be as innocent as a form posted to a web server program or even a digital image viewed by a browser program. The buffer overflow attack is particularly insidious because defenders and their tools (including many protection graphs) often model data as inanimate objects incapable of execution. But the buffer overflow attack transforms the attacker's inanimate data into an executable program. Defense against the buffer-overflow attack is particularly challenging because defenders may be limited to a small choice of historically vulnerable software and may have little or no choice of the external data they accept.

Church defines a model, the λ-function in the Lambda Calculus, of the transformation of inanimate data into an executable function [14]. While much has been written about buffer overflow attacks, the λ-like feature transforming the attacker-supplied passive data object into an active executing function is central to our analysis of the buffer overflow attack in which the attacker and defender have different views of the protection system. Figure 5 illustrates a defender's view of the protection graph in which **s** represents the defender's software, object **b** represents a passive data buffer, and **a** represents the attacker. The defender's software **s** has α access to another resource, **v**. The attacker has (w to **b**) access enabling **a** to modify the buffer object. The defender's software has (r to **b**) access enabling **s** to read the buffer object. This model might represent a real web server (**s**), its content (**v**), one of its buffers (**b**) and its attacker (**a**).

Because the TG model does not have a λ-rule promoting an object to become a subject, TG cannot model the promotion itself, only the two separate views of the protection graph. TG/MC works around the lack of a λ-rule in TG by portraying the possibility of a buffer overflow using uncertainty about the type (object or subject) of the buffer vertex.

The defender's view of the protection graph is safe because no transfer of rights is possible (there exists no edge labeled with either t or g so rights cannot flow even though information can flow along the authorized (w to **b**) and (r to **b**) path) to the attacker, **a**. The defender's view provides solely for the transfer of information.

The attacker views the protection system as depicted in Figure 6. In the attacker's view, **b** is an active subject, not a mere passive object. The attacker takes advantage of the (w to **b**) right to supply executable code to **b** that will then, per the attacker's craft, *take* (α to **v**) from **s** and in turn *grant* the (α to **v**) right to **a**. Figure 7 models the aftermath, G_v, of this *witness* enabling the attacker to acquire (α to **v**). Assuming this state is unauthorized, the attacker's view of the protection graph is not safe.

The buffer overflow example provides an opportunity to reflect upon what went wrong in the defender's security assessment. Figure 5 represents a defender's safe view of the protection system as specified, but not as-built. Note that the defender's view anticipates only the transfer of *information* through the buffer as there exists no *tg-path* through **b**. The defender expects the attacker will write data into the buffer to be later read by the defender's software, **s**.

The attacker views the protection system differently. The attacker's view emphasizes the transfer of access rights with information playing only a brief role in the buffer overflow attack enabling the attacker to transfer an executable program to the buffer. The possibility for the attacker to promote the buffer object **b** to become a subject is the crucial departure from the defender's view.

Even the attacker's view of Figure 6 does not completely represent the protection system software as-built because **s** may have the (r to **b**) right. The protection system as-built is a composition of both the defender's and the attacker's view (Figure 11), plus whatever undiscovered anomalies remain. As with Bishop's distinction between the model's *safety* and the real system's *security* [1], even though the defender's model of Figure 5 is *safe*, the analyst cannot infer the *security* of the real-world system because (amongst other possibilities) the real-world system may not exactly implement the model.

Between the reality of the as-built real-world protection system and the decidable TG protection model lies a field of uncertainty in which many things *could* happen, the field of risk analysis. We examine the uncertainties of the TG model using risk analysis with the Monte-Carlo technique.

2.3 Additional De Jure Results

TG describes additional lemmas and results governing the transfer of rights. The reader is referred to Bishop's work for a thorough overview of the TG Protection model [1].

2.3.1 Duality

Rights can flow in either direction along a *tg-path* between cooperating subject endpoints.

2.3.2 canShare

The predicate $canShare(\alpha, x, y, G_0)$ defines the complete set of conditions under which $\mathbf{x}$ can acquire (α to $\mathbf{y}$) through a witness $G_0 \vdash^* G_n$ composed entirely of *de jure* rules. The *canShare* predicate addresses transfers of rights including those in which the subjects of G cooperate (e.g., *grant* the right α to another vertex) with $\mathbf{x}$ in the transfer.

2.3.3 canSteal

The predicate $canSteal(\alpha, x, y, G_0)$ defines the set of conditions under which $\mathbf{x}$ can acquire (α to $\mathbf{y}$) through a witness $G_0 \vdash^* G_n$ composed entirely of *de jure* rules such that no subject initially possessing (α to $\mathbf{y}$) applies any rule. The *canSteal* predicate addresses the theft of rights, mainly a transfer of authority without the cooperation of the other (non-attacking) subjects in the graph.

3. Sources of Uncertainty in Security Models

We consider uncertainty arising from both aleatoric and epistemic sources where aleatoric uncertainty refers to an outcome of a truly-random (stochastic) process while epistemic uncertainty refers to the lack of knowledge about the exact behavior of an unknown process or parameter [18].

The familiar brute-force guessing attack on a secret credential exemplifies aleatoric uncertainty. Aleatoric events have stochastic models describing the probability of their occurrence. While a stochastic model cannot predict exactly when an aleatoric event will occur, it can predict the probability of its occurrence within a specified interval. For example, *war-dialing* refers to a guessing attack on a secret target phone number and can be modeled with the following assumptions:

- The guessing attack consists of one or more independent, discrete trials, each resulting in a success or failure outcome for a single attempt to guess the secret.
- A uniformly distributed population of candidate numbers provide each trial with the same probability of success or failure as any other trial.
- Each trial is drawn from a finite and fixed size pool of candidate phone-numbers.

- The attack need not re-consider failed trials (*non-replacement* sampling assumption) because the target phone number is unchanging.

These characteristics suggest using the hypergeometric process [28] as a stochastic model of the war-daling attack [10]. And, for this process, the inverse hypergeometric distribution describes the uncertainty about the required number of failed trials before the attacker succeeds by correctly guessing the target phone number. While this model can't predict the exact number of trials required, it can forecast the probability of a successful outcome within an interval of n trials.

Epistemic uncertainty arises from a lack of knowledge regarding an underlying process or its modeling parameters. Many forecasts of human behavior have epistemic qualities. For example, analysts may not know the exact resources available to an attacker, the level-of-effort an attacker is willing to invest, or the frequency of attempts. Thus, quantitative descriptions of epistemic uncertainty often include expert estimates in lieu of specific data. Because the underlying processes are unknown, epistemic uncertainty is sometimes modeled using rough tools such as the triangle, PERT or uniform distribution.

The total uncertainty in a real-world system may include both aleatoric and epistemic qualities. For example, the aleatoric hypergeometric model of the war-dialing attack may include epistemic uncertainty in terms of the size, M, of the phone-number pool, and the number, D, of target modems available to the attacker. The attacker might, for example, be able to reduce the size of the pool by removing phone numbers known to be assigned to other applications. Or the number of modems on the target system may be unknown. Without definitive knowledge of these parameters, an analyst will likely resort to expert estimates for the epistemic parameters of an otherwise aleatoric model.

In subsequent sections, we illustrate the use of the Monte-Carlo technique for modeling both the aleatoric and the epistemic uncertainty appearing in TG protection graphs.

4. Overview of the Monte-Carlo Technique

A Monte-Carlo simulation can be viewed as an experiment comprised of a set of independent trials in which each trial is known as an *iteration*. The experiment includes the familiar independent and dependent variables, and is controlled by a Monte-Carlo *engine*. The analyst defines each experimental parameter (an independent variable) as either a constant or a probability distribution. The collected results of the iterations form a population, sometimes known as the *forecast*, the experiment's dependent variable.

In every iteration of the simulation, the Monte-Carlo engine samples each parameter and evaluates the experimental procedure which in TG/MC is a predicate of the TG security model [9, 11]. The TG/MC predicate decides the safety of each iteration's protection graph and returns this result (safe/unsafe) to the engine. The Monte-Carlo engine collects the results of each iteration into a forecast presented to the analyst at the conclusion of the simulation experiment. Thus, the Monte-Carlo engine functions as a harness around the security model, supplying the model with sampled parameters, controlling its execution, and capturing each result into the forecast.

Note that the harnessed security model remains intact; the Monte-Carlo approach can be applied without any fundamental changes to the TG Protection Model. Because no changes are made to the model itself, all of the existing TG lemmas remain available.

The Monte-Carlo technique is not without difficulty. Monte-Carlo simulations require the model to be supplied with wholly independent parameters or, at a minimum, the analyst must address the joint probabilities of interrelated parameters. For example, an analyst wishing to study the effect of two boolean parameters p_1 and p_2 upon a forecast, R, might be tempted to treat p_1 and p_2 as two independent variables. But this only works when their independence reflects the real-world system. If the system constrains the p_1 value to be always *false* whenever p_2 is *false*, then these parameters clearly are not independent. Without care in their specification, the Monte-Carlo engine may eventually sample an impossible set of values for p_1 and p_2. A common workaround for this issue expresses one correlated parameter as a function of the other.

The independent parameter issue may be the trickiest aspect encountered in the development of a Monte-Carlo simulation. Vose offers a rule to guide the analyst through this and many related issues, "Every iteration of a [Monte-Carlo] risk analysis model must be a scenario that could physically occur" [28]. We cannot expect our security model to render meaningful forecasts if we allow the Monte-Carlo engine to supply it with non-sensical parameter values.

5. Simulating Uncertainty with TG/MC

We consider two approaches, each differing in what is uncertain, for applying the Monte-Carlo technique to the TG Protection Model.

The first approach (not adopted by TG/MC) decides whether to apply a specific rule prior to *each* state change in a witness. The uncertainty in this approach lies in whether to apply the rule at each element of a witness originating from the initial protection graph, G_0. Sometimes the rule will be available to make the state change and sometimes it will not. This approach may be most meaningful if, in the development of a long witness, a specific edge might be applied in one occasion but not in another. This approach simulates a real-world system having a dynamic protection graph (independent of any possible attack); edges become available or unavailable to the graph as time passes. Such a system might, for example, have random malfunctions causing subgraphs to appear and disappear from its protection graph as time passes. This approach models systems in which the attacker is not the only force behind state transitions in the protection graph.

In the second approach (adopted by TG/MC), the uncertainty lies entirely in the definition of the initial protection graph, G_0. Given a specific graph, G_0, the analyst is confident that its safety can be decided within the TG Protection Model. What is uncertain is the topology of the initial graph itself, not the individual state transitions in a witness originating with G_0. TG/MC employs this approach because it readily simulates the *epistemic* uncertainty in a real-world system — the analyst often can not be certain whether a protection graph truly represents the real system as-built. In this approach, the analyst is more concerned with their knowledge of the real system than with the dynamic behavior of the real system.

Figure 8 illustrates a methodology for analyzing uncertainty in protection systems using TG/MC. The methodology begins with an investigation of the real system and concludes with a forecast as described in the following sections.

5.1 Modeling a Real System

Figure 5 illustrates an initial protection graph representing a specific system as designed and a model of the way the system is intended to function. This initial protection graph is used in the second step of the TG/MC methodology, the threat analysis.

5.2 Threat Analysis

The threat analysis process investigates potential violations of security and produces a list of vulnerabilities whose exploitation would implement the threats.

Informally, a *violation* is an attacker's goal, an undesirable outcome of an exploit or a violation of security policy. More specifically,

Definition 1 *A TG/MC* violation *is an edge from vertex* $\boldsymbol{a}$ *to another vertex* $\boldsymbol{v}$ *labeled* α *in a protection graph* G *such that vertex* $\boldsymbol{a}$ *is not authorized to possess* (α *to* $\boldsymbol{v}$).

For the buffer overflow example, Figure 9 illustrates a TG/MC *violation* in which the subject $\mathbf{a}$ has acquired the unauthorized right, (α to $\mathbf{v}$).

In TG/MC, a *vulnerability* represents a change to an otherwise safe initial protection graph that, if exploited, enables an attacker to realize a threat and create a *violation*. More specifically,

Definition 2 *A graph* D *is said to be a* vulnerability *for* G_0 *if* $canShare(\alpha, \mathbf{a}, \mathbf{v}, G_0)$ *is false and* $canShare(\alpha, \mathbf{a}, \mathbf{v}, D \cup G_0)$ *is true.*

For example, the addition of graph D (Figure 10) to the initially safe protection graph G_0 (Figure 5) produces an unsafe protection graph (Figure 11) having a witness (Figure 12) from $D \cup G_0$ to G_v, and an aftermath (Figure 7) containing the security *violation* of Figure 9. The elements of *vulnerability* D, when incorporated with those of G_0, leave the resulting graph $D \cup G_0$ unsafe.

Note the uncertainty about the existence of a TG/MC vulnerability arises from uncertainty about the initial state of the protection graph, the possibility of $D \cup G_0$ rather than G_0 alone representing the system as built.

5.3 Disaggregating Vulnerabilities

Disaggregation breaks a complex risk analysis problem into a set of primitive elements enabling an improvement of the model at lower levels and the discovery of correlated parameters [28]. A vulnerability graph, D, can be disaggregated into the elements of its *minimal vulnerability*, D_{min}, isolating just those elements essential to the vulnerability.

Definition 3 *A* minimal vulnerability*,* $D_{min} \subseteq D$*, for a violation* Q_D *is a minimal subgraph of* D *such that* D_{min} *is a vulnerability in which* $(G_0 \cup D_{min}) \vdash^* G_v$ *containing the violation,* $Q_D \subseteq G_v$*.*

In the buffer overflow example, the *vulnerability*, D, of Figure 10, includes three primitive TG elements, all of which are required to enable the buffer overflow attack. Thus, $D_{min} = D$ in this example. The three primitive elements are:

1. V[**b**,s]. This vertex promotes the protection graph's labeling of **b** from an object to a subject, allowing it to act. It models the possibility for **b** to execute code supplied by **a** through the edge **a** has (w to **b**).

2. E[**b**,**s**,t], an edge from **b** to **s** labeled t modeling the ability of attacker-supplied code in vertex **b** to obtain any resource, **v**, owned by the server, **s**.

3. E[**b**,**a**,g], an edge from **b** to **a** labeled g. The attacker-supplied code controlling **b** "phones home" to pass information or authority to **a**.

The edges, E[**s**,**v**,α] and E[**a**,**b**,w], are members of the initial protection graph, G_0, but are not members of the buffer overflow vulnerability, $D_{min} = D$, of Figure 10.

The Monte-Carlo simulation will use the uncertainty about the time-to-failure (TTF) of disaggregated elements as parameters supplied to the simulation. In TG/MC, the TTF term refers to the time required for the attacker to acquire the primitive TG elements of the vulnerability, D; each element of the disaggregated vulnerability may have a different TTF:

Definition 4 *TG/MC defines $TTF(D_i)$ as the time-to-failure of element i of vulnerability D where $1 \leq i \leq n$ and n is the number of elements in the disaggregated vulnerability.*

The uncertainty about any $TTF(D_i)$ can be expressed as a probability distribution. Because of the intent to use TTF distributions in parameters to a Monte-Carlo simulation, the analyst must take care to identify and accommodate any correlated parameters. Correlated parameters exist, for example, whenever one minimal vulnerability D_{min} is composed of multiple, distinct subgraphs, $[D_x, D_y, ...]$ such that the TTF of one subgraph, D_i, is correlated with the TTF of another subgraph, D_j. If the TTF of every D_i is wholly independent of every D_j then there is no issue for the simulation. Correlation, when it does exist, can often be accommodated by expressing the TTF of one correlated parameter as a function of the TTF of another parameter.

5.4 Quantifying Uncertainty

This section introduces techniques for choosing probability distributions to quantify the uncertainty in parameters supplied to a Monte-Carlo simulation of a TG Protection Graph for several well-known attacks.

Consider again the brute force guessing attack on a static, secret credential. This type of attack was previously modeled with a hypergeometric process for which the aleatoric uncertainty about the number of required trials (before a successful guess) is given by an inverse hypergeometric distribution. However, this model becomes unwieldy when a large population of candidates is present because large factorials burden the calculations. However, in a large population, the non-replacement policy becomes less significant because the effect of replacing a trial's unsuccessful sample has little impact on the subsequent trial. Under this condition, the replacement policy of the binomial process and its associated (and readily calculated) geometric distribution approximates the inverse hypergeometric distribution.

The dictionary refinement of the guessing attack is another case to consider when the complimentary information is unavailable to the attacker [2]. The dictionary attack recognizes the candidate values may not be uniformly distributed and focuses the attack trials on the more probable candidates identified by a dictionary. This attack proceeds rapidly if the secret credential resides in the dictionary, but resorts to time-consuming brute-force guessing if it is not. Considerable epistemic uncertainty exists about whether the secret will be found in the dictionary or not, and there also exists aleatoric uncertainty about how many trials will be required (using either method). One approach for simulating the total uncertainty of the dictionary attack is to model the simulation as a mixture of two underlying processes:

1. A discrete distribution models the epistemic uncertainty about the size of a population (either the size of the dictionary or the cardinality of the set of all possible values of the secret credential). For example, if there is a 68% chance of finding the secret credential in a dictionary of 2×10^9 entries [21], then:

$$P(N = |D|) = 0.68 \qquad \text{P(success with dictionary)} \tag{1}$$

$$P(N = |S|) = 0.32 \qquad \text{P(resort to brute force)} \tag{2}$$

   where N = Size of the population to search, D = Set of entries in the dictionary, S = Set of all possible values for the secret. In a single iteration i of the simulation,

$$p_D = 1/|D| \qquad \text{P(successful dictionary trial)} \tag{3}$$

$$p_S = 1/|S| \qquad \text{P(successful brute force trial)} \tag{4}$$

$$p_i = \text{Bernoulli}(0.68, p_D, p_S) \qquad \text{P(successful trial in iteration i)} \tag{5}$$

where $f_i(x)$ is the probability density function describing the uncertainty about the number of failures before a successful trial (guess). Each iteration of the Monte-Carlo simulation first addresses the epistemic uncertainty in p_i using a Bernoulli distribution, randomly assigning either p_D or p_S to p_i with the $P(p_D) = 0.68$.

2. After assigning a value to p_i, the simulation addresses the aleatoric uncertainty in the number of failures before a successful trial using a geometric distribution for $f_i(x)$ in iteration i:

$$f_i(x) = p_i(1 - p_i)^x \qquad \text{P(x failures before success)} \tag{6}$$

This simulation of the dictionary attack incorporates a composite of both epistemic uncertainty about p_i (depending upon whether the secret is an $\in D$) and the aleatoric uncertainty of the binomial process. The simulation of the binomial process benefits from the simplifying assumption of a constant p_i within any iteration, i. But the simulation also models the epistemic uncertainty in p_i by varying its value from one iteration to the next.

Epistemic uncertainty is also important when simulating the time required for an attacker to acquire a new exploit. Analysts modeling the arrival of a new exploit as a Poisson process may encounter epistemic uncertainty about the mean arrival rate, λ. In many cases, a modeling parameter such as a λ, or even the choice of a probability distribution for a Monte-Carlo simulation parameter will be a subjective decision made by an expert. Experts estimate a parameter such as λ using historical trends and/or their judgement. Borgonovo and Smith [5], Knudsen and Smith [18], and Vose [28] describe approaches for quantifying the uncertainty in modeling parameters where the $MTTF$ is uncertain. Rather than obscuring this uncertainty, the TG/MC process captures the uncertainty in the experts' estimates, simulates its impact upon the TG model, and discloses the impact on the resulting forecast.

5.5 Monte-Carlo Simulation

The TG/MC simulation brings together an initially safe protection graph, G_0, vulnerabilities D_i quantified by their time-to-failure probability distributions, $TTF(D_i)$, potential violations Q_i, arising from the vulnerabilities, and the TG predicates. The Monte-Carlo engine samples each TTF distribution at the beginning of each iteration in the TG/MC simulation to determine which vulnerabilities to include in each particular iteration's trial protection graph. The engine then constructs the trial protection graph from the initially safe graph (G_0) with the vulnerability chosen by sampling. Note that a trial protection graph has a scope of a single iteration while an initial protection graph has a scope of a complete simulation. For example, Figure 11 illustrates a possible trial protection graph including the Buffer Overflow vulnerability while Figure 5 illustrates a possible trial protection graph without the example vulnerability. Monte-Carlo sampling of a Bernoulli distribution determines whether to include the vulnerability or not in the trial protection graph; we describe this process in more detail in Section 6.3.

Next, the Monte-Carlo engine tests the safety of the trial protection graph using a Prolog implementation of the TG *canShare* predicate:

$canShare(\alpha, \mathbf{a}, \mathbf{v}, G_t)$

An implementation of the *canShare* predicate determines if subject **a** can obtain (α to **v**) in G_t, a potentially vulnerable trial protection graph. The programmatic implementation of the TG *canShare* predicate is invoked during each iteration (trial) of the Monte-Carlo simulation.

6. The Substation Example Application of TG/MC

This section illustrates the application of TG/MC for analyzing the security of equipment commonly found in the control and management of a prototypical electrical substation (Figures 13 through 23). A Commercial Off-The-Shelf (COTS) Monte-Carlo engine was extended to harness a Prolog implementation of the TG predicates. The COTS package samples the parameter distributions, while the extension builds the trial protection graphs, executes the TG predicates, and captures the resulting forecast of safety from each iteration.

6.1 System Description

The system in the substation example (Figure 13) illustrates a subset of a typical Supervisory Control and Data Acquisition (SCADA) application similar to those investigated by Conte de Leon et al [12]. Substations are critical components of electrical transmission and distribution systems, and may include protection equipment (circuit breakers), voltage adjusting transformers, electrical phase adjusting equipment (capacitors), instruments for measuring operating parameters, relays controlling the protection equipment, and communications processors providing remote access. Some of the substation equipment contains embedded computing devices with associated communication pathways between the computing equipment within the substation, between substations, and between a substation and the utility's central office. A substation may contain computing equipment with one or more Internet Protocol (IP) Addresses and/or a modem. Substations and similar critical infrastructures can be vulnerable to an electronic "cyber" attack on their relays or other remotely controlled and configured equipment [22]. A "cyber" attack on a circuit breaker through its relay could result in the improper operation of the circuit breaker leading to equipment damage or even loss of life.

6.2 Modeling the Example Protection System

Figure 14 illustrates a TG/MC protection graph developed by the authors for the substation example of Figure 13. In order to consolidate the drawing, both the initial protection graph (G_0) and the assorted vulnerabilities appear in a single figure. The TTF of a black edge in the TG/MC protection graph is effectively 0, while the TTF of a grey (vulnerability) edge is some uncertain (and typically non-zero) value. The model in the example investigates the threat for an attacker, **a**, to obtain the right (α to **ck**) in the potentially vulnerable G_0, allowing it to operate the substation's circuit breaker.

The TG/MC graph represents the four physical locations in Conte de Leon's system of Figure 13, the substation, the central office, a remote engineer's home, and the attacker. The following explains the roles of various members of the substation example's protection graph [12].

a The attacker.

c The substation communication processor.

cc The substation communication processor's secret authentication credentials. The processor, **c**, grants all rights in its possession to anyone possessing the processor's credentials, **cc**. The attacker, **a**, and the remote engineer, **e**, have different access paths to the credentials, modeled here through different paths. The attacker must calculate, guess or otherwise obtain the secret credentials, but the remote engineer's computer has a login script programmed with the credentials.

ck The substation's circuit breaker. The relay, **r**, has the right, (α to **ck**), to operate the circuit breaker..

cm The attacker's view of the substation modem and its "secret" phone number. The attacker, **a**, and the remote engineer, **e**, have different access to the modem, represented here through different paths through the protection graph. The attacker must find the modem's "secret" phone number (e.g. through "war dialing" or social engineering) before taking access to the modem's rights.

cm' The remote engineer's view of the substation modem. The remote engineer's computer autodials the substation modem.

d A desktop workstation located in the utility's central office. The workstation grants any access right in its possession to anyone acquiring its secret credentials.

db A potentially vulnerable buffer located within an application on the workstation.

dc The secret credentials authorizing access to the workstation.

e The engineer's remote (home) computer system.

eb A buffer in the remote engineer's system potentially vulnerable to a buffer overflow attack.

fw The electrical utility's firewall. The associated VPN service grants authorized authenticated external hosts (vertices) access to the utility's LAN.

ln The electrical utility's central office LAN. External access to the LAN is guarded by the firewall's secret credentials.

r The substation's relay having the right to operate the circuit breaker, **ck**.

s The SCADA Master Server located in the utility's central office.

sc The SCADA Master Server's secret credentials. The SCADA Master grants all rights in its possession to anyone possessing its credentials.

vc The office VPN's secret credentials. The firewall grants any right in its possession to any external vertex having the VPN credentials.

Figure 14 includes a potential buffer overflow vulnerability within a desktop workstation located in the utility's central office. The utility's security policy permits unrestricted browsing by workstations to external web sites. The model graphs the mechanism implementing this policy with a dashed w edge from **a** to **db**. Recall that dashed lines represent edges that *might* exist. The attacker, **a**, must wait for the workstation to establish a connection; **a** will then have an opportunity to write information (e.g. in response to an http GET issued by the workstation) to the workstation's buffer, **db**. The TG/MC model quantifies the uncertain wait time with a TTF distribution. The buffer overflow vulnerability has three uncertain elements; the potential to promote object **db** to become a subject, the potential for **db** to take rights from **d**, and the potential for **db** to grant rights to **a**. In this example, the attacker creates an attractive web site and waits for a vulnerable workstation to infect itself.

The attacker has other opportunities to attack the utility's central office. A successful attack on the VPN credentials, **vc**, allows the firewall, **fw**, to pass along any right available on the LAN, **ln**. The attacker, **a** can also physically travel to the vicinity of the utility's wireless LAN and attack the wireless credentials to obtain the rights available on the LAN. While either of these approaches provide the attacker with "inside" (the firewall) LAN access, the attacker still must overcome the mechanisms protecting the equipment attached to the LAN (represented in Figure 14 as the desktop workstation and SCADA Server's credentials).

In addition, **a** can war dial the communication processor modem and attack the processor's authentication credentials, **cc**.

Alternatively, the attacker, **a**, can exploit a potential buffer overflow vulnerability in an engineer's home computer. This exploit requires the attacker to attract someone using the home computer to accept information supplied by **a**. Because the engineer's home computer will auto-dial and login to the communications processor, the attack proceeds relatively rapidly without need to overcome the secret modem number nor the secret communications processor's credentials — the convenience of auto-dialing and auto-login scripts may prove costly. The engineer's VPN client, however, requires interactive entry of the VPN credentials, making the **e** to **vc** better defended.

The TG/MC graph of Figure 14 includes potential TG connections (the dashed, gray lines) between vertices; the simulation of this model will express the uncertainty about each of these in terms of their TTF. Analysis of the graph illustrates that even a strong defense (such as the VPN's credentials) can be overcome or bypassed by vulnerabilities such as the buffer overflow in the desktop workstation. The graph quantifies the potential value of a secret phone number in the war-dialing (**a** to **cm**) edge as another impediment requiring a little more time to overcome. Figure 14 also illustrates how the engineer's auto-dial and login scripts provide near unfettered access to the substation for anyone who "owns" the remote engineer's system. Unlike some system-level [11, 27] and even some graphical approaches [10, 26], TG/MC models important details about the potential flow of authority in an attack.

6.3 Simulating Uncertainty in the Example System

TG/MC uses probability density functions illustrated here as frequency charts to describe uncertainties about the vulnerability parameters. Using a sample of the corresponding probability density function, each iteration of the Monte-Carlo simulation calculates the probability of each vulnerability in Table

1. This section describes how these probabilities are calculated and how they are used to determine whether to include the associated vulnerabilities in an iteration's trial protection graph. The more probable vulnerabilities appear in the trial protection graphs more frequently.

Figure 19 illustrates a frequency chart for a successful attack on the communication controller's secret phone number, the *cmwd* vulnerability, as a function of the number of trials (attempts). The example models the uncertainty about the duration of the war-dialing attack (ending with the "failure" of the secret credential when successfully guessed) with a binomial process. The uncertainty about the number of unsuccessful trials (guesses) in a binomial process is described by a geometric distribution configured for *cmwd* in this example as follows:

$$N_{cmwd} = 10000 \qquad \text{Size of population to search} \tag{7}$$

$$r_{cmwd} = 2880 \qquad \text{Number of trials in one day} \tag{8}$$

$$p_{cmwd} = 1/N_{cmwd} \qquad \text{P(successful guess from 1 trial)} \tag{9}$$

$$F_{cmwd} = 1 - (1 - p_{cmwd})^{r_{cmwd}-1} \qquad \text{P(successful guess during one day)} \tag{10}$$

where N_{cmwd} is the estimated size of the phone number population (the cardinality of the set of all phone numbers to search) in the attack, r_{cmwd} is the estimated mean number of trials during an interval of one day, p_{cmwd} is the probability of successfully guessing the phone number with any single trial, and F_{cmwd} is the cumulative probability of a successful guess during an interval of one day consisting of at most $r_{cmwd} - 1$ failures before a success.

Finally, TG/MC simulates a boolean value, Z^i_{cmwd}, in each iteration, i, indicating whether to include the *cmwd* vulnerability in this iteration's protection graph, G^i_t. The Monte-Carlo engine chooses a $true$ or $false$ value for Z^i_{cmwd} by sampling a Bernoulli distribution configured with F_{cmwd}:

$$Z^i_{cmwd} = \text{Bernoulli}(F_{cmwd}) \qquad \textit{true} \text{ or } \textit{false} \tag{11}$$

Z^i_{cmwd} determines whether or not the simulation will include the *cmwd* vulnerability in G^i_t, the trial protection graph for iteration i.

In order to simplify the example, both N_{cmwd} and r_{cmwd} appear as constants. In many real-world applications, at least some epistemic uncertainty will exist in both the phone number pool and the mean guessing rate. The uncertainty in these too can be modeled with probability distributions.

Figures 15 through 18 illustrate the frequency charts for successful attacks on the the *ccat, dcat, scat* and *vcat* credential vulnerabilities. Each chart plots the probability of the number of failed trials before a successful guess of the credential. The example estimates the *strength* of each credential arising from its associated *entropy* quantified as bits. The estimates use the NIST guidelines [6] for human-generated passwords, an alphabet of 94 symbols, and the example's password policy:

ccat Policy of 6 symbols without any checks provides 14 bits of randomness to yield a population, $N_{ccat} = 16384$.

dcat Policy of 8 symbols checked by a dictionary for common words provides 24 bits of randomness to yield a population, $N_{dcat} = 16777216$.

scat Policy of 8 symbols without any checks provides 18 bits of randomness to yield a population, $N_{scat} = 262144$.

vcat Policy of 8 symbols with both dictionary and composition checks provides 30 bits of randomness to yield a population, $N_{vcat} = 1.074 * 10^9$.

The example models the attacker's desire to elude detection by limiting the maximum possible guessing rates:

$$r_{ccat} = 3 \text{ trials/minute} \tag{12}$$

$$r_{dcat} = 60 \text{ trials/minute} \tag{13}$$

$$r_{scat} = 60 \text{ trials/minute} \tag{14}$$

$$r_{vcat} = 20 \text{ trials/minute} \tag{15}$$

As with *cmwd* above, the epistemic uncertainty about the estimates for these populations and rates could be modeled with additional distributions although the example employs scalar values for simplicity. The description of the *dbow* vulnerability illustrates an approach for handling epistemic uncertainty.

Figure 23 illustrates the uncertainty in the example's *wcat* parameter. Unlike the previous models of uncertainty for the *cmwd, ccat, dcat, scat* and *vcat* parameters, the model for *wcat* sums the uncertain parameters, attack time and travel time, because the attack must be launched near the utility's campus. The example begins by simulating the probability density, $f_{tta}(t)$, describing uncertainty in the attack time, as follows:

$$MTTA^i = \text{uniform}(0.00208, 0.25) \tag{16}$$

$$\lambda^i_{wcat} = 1/MTTA^i \tag{17}$$

$$f_{tta}(t) = \lambda^i_{wcat} * \mathrm{e}^{-\lambda^i_{wcat} * t} \tag{18}$$

where $MTTA^i$ is the expected attack time for iteration i of the simulation, and λ^i_{wcat} is the expected attack rate for iteration i. WEP may be attacked using active or passive methods. In March of 2005, a team of United States Federal Bureau of Investigation Agents reportedly demonstrated a successful three minute attack on WEP [8,29]. Ossmann reports nearly comparable results [24,25] using active methods. Passive methods require more time, but offer greater protection to the attacker against detection. We used the uniform distribution to express the epistemic uncertainty in the example's WEP attack time, estimating a minimum of only 3 minutes (0.00208 days) using active tools and 6 hours (0.25 days) for a passive eavesdropping attack on a working network [4, 7, 16, 24, 25]. The uniform distribution predicts little about the attacker's expected behavior, but this estimate can be refined if empirical data are available about the trade-offs of speediness vs. covertness in real-world WEP attacks. To minimize the chance of detection, an attacker may not choose the fastest (active) method. The epistemic uncertainty in the MTTA models the attacker's choice while the $f_{tta}(t)$ density function models the aleatoric uncertainty of the cryptanalysis.

The Monte-Carlo simulation samples $f_{tta}(t)$ in each iteration, i, obtaining a TTF^i_{tta} expressed as the time required to attack WEP in this iteration. The example also estimates the travel time for the attacker to reach the utility's campus and samples the uniform distribution in each iteration, i, obtaining a $travelTime^i$, the travel time for this iteration:

$$travelTime^i = \text{uniform}(0.042, 0.333) \tag{19}$$

between 0.042 and 0.333 days. Later, the attack and travel times are summed to obtain the total time to failure in this iteration, i, for the *wcat* vulnerability:

$$TTF^i_{wcat} = TTF^i_{tta} + travelTime^i \tag{20}$$

Unlike in the previous examples whose exponential shapes were introduced above, the TTF_{wcat} distribution takes on a more *normal* shape as a result of an application of the Central Limit Theorem with the summation in Equation 20 [28].

Figures 20 and 21 illustrate the uncertainty in the example's *dbow* and *ebow* parameters modeling the times required for the attacker to acquire and deliver a buffer overflow exploit to the utility's desktop (*dbow*) and the engineer's home (*ebow*) workstations. The example simulates *dbow* and *ebow* with identical but independent variables. The example disaggregates the problem into three sources of uncertainty: the mean time required for the attacker to acquire an exploit, AT_{dbow}, a Poisson Process modeling the aleatoric time to the next acquisition event, $TTAE_{dbow}$, and the time required to deploy the exploit, DT_{dbow}, to its intended victim.

The example models the epistemic uncertainty in the AT_{dbow} and DT_{dbow} estimates with triangle distribution. During the simulation, the Monte-Carlo engine samples the triangle distributions to obtain values of AT^i_{dbow} and DT^i_{dbow} for each iteration, i. The simulation then calculates the mean acquisition rate, AR^i_{dbow}.

$$AR^i_{dbow} = 1/AT^i_{dbow} \tag{21}$$

$$f_{TTAE}(t) = AR^i_{dbow} * \mathrm{e}^{-AR^i_{dbow}*t} \tag{22}$$

The simulation continues by sampling the exponential probability density function for the Poisson Process, $f_{TTAE}(t)$, to obtain this iteration's time to acquire an exploit for the *dbow* vulnerability, $TTAE^i_{dbow}$. The example simulation obtains this iteration's time to deploy the exploit to its intended victim, DT^i_{dbow}, by sampling a triangle distribution which estimates the deployment time for the *dbow* vulnerability, DT_{dbow}. The simulation sums $TTAE^i_{dbow}$ and DT^i_{dbow} to obtain the total time to failure, TTF^i_{dbow}, for exploiting the *dbow* vulnerability in this iteration, i. Finally, the simulation employs a Bernoulli distribution, similar to the model of the *cmwd* vulnerability, to choose whether to include the *dbow* vulnerability in iteration, i:

$$\lambda^i_{dbow} = 1/TTF^i_{dbow} \tag{23}$$

$$P^i_{dbow} = 1 - \mathrm{e}^{-\lambda^i_{dbow}} \tag{24}$$

$$Z^i_{dbow} = \text{Bernoulli}(P^i_{dbow}) \tag{25}$$

where λ^i_{dbow} is the failure rate of the *dbow* vulnerability associated with the iteration's sampled TTF^i_{dbow}, the P^i_{dbow} is the cumulative probability of vulnerability *dbow* appearing in the one day interval of this iteration, i, and Z^i_{dbow} is a boolean determination of whether to include the *dbow* vulnerability in the protection graph of iteration, i.

For the final vulnerability, the example simply estimates the time the attacker must wait for the remote engineer to establish an authenticated VPN connection from the home system, *vcct*, using a triangle distribution (Figure 22). Each iteration of the simulation samples the triangle distribution to obtain a wait time, WT^i, and then samples a Bernoulli process to obtain, Z^i_{vcct} which determines whether to include the *vcct* vulnerability in this iteration, i.

6.4 Substation Example Result

The example simulation forecasts the probability of subject **a** acquiring (α to **ck**) during an interval of any one day, a successful attack on the circuit breaker:

$$P(\mathbf{a} \text{ acquires } (\alpha \text{ to } \mathbf{ck})) = 0.33 \tag{26}$$

This result reflects the uncertainty about the security of the real-world system given the simulation parameters for the vulnerabilities (Figures 15 through 23).

The result suggests that an external attack on the circuit breaker has a high probability of success despite several meaningful precautions (the use of a VPN protected by a very strong password policy, and a meaningful password policy for desktop workstations) illustrated in the example. However, the strength of these precautions is circumvented by other vulnerabilities illustrated in the example:

- The example wireless system, protected by the WEP secret credential, can fail in a matter of hours, even when taking into account the attacker's travel time and motivation to avoid detection with a passive exploit. The failed wireless system then exposes the internal hosts to electronic attacks. This route could be denied to the attacker by moving the wireless access point outside of the firewall and requiring mobile employees to VPN into the internal hosts.
- The example's password policy focuses on IT equipment, including the VPN and the employee's desktop workstations, where it provides relatively strong protection. However, the weak password policy for engineering equipment (including the substation's communication controller and the SCADA server) leaves these vital systems vulnerable to dictionary attacks.
- The secret phone number of the example's substation modem provides only a weak defense that is quite vulnerable to a war dialing attack.

The example assumes the attacker, **a**, exists. Real-world questions about a *potential* attacker's success would also include uncertainty about the very existence of the attacker, **a**, in the protection graph.

7. Benefits, Discussion and Future Directions

Unlike previous approaches, TG/MC benefits from the formalized TG Protection System, in which questions of safety are decidable using a set of rules and predicates. But TG/MC goes beyond TG's rigid yes/no analysis of safety as represented in a protection graph to analyze the uncertainty of a real-world system's security by introducing the concept of potential *vulnerabilities* into each iteration of a Monte-Carlo simulation. TG/MC analyzes the effect of uncertainty in the initial protection graph, enabling it to address many real-world issues including the existence of specific vulnerabilities and an individual attacker's behavior. TG/MC quantifies the uncertainty of *vulnerabilities* with time-to-failure distributions, and forecasts the probability of a security violation in the simulated real system.

Conrad, Oman and Conte de Leon analyzed uncertainty in a weighted graphical security model [10] and compared the time-to-failure of various paths through the graph. Frank and Bishop investigated a mechanism for comparing the costs of paths through a TG protection graph [13]. Their computational approach associates a *cost* for each rule application and computes the total cost of a witness as a function (summation) of the individual rule applications. Frank and Bishop's approach redefines a protection graph as a weighted cost graph with little consideration of uncertainty. Marc noted the need for a probabilistic approach in the assessment of operational security [20]. A combination of the Monte-Carlo approach described by Conrad, Oman and Conte de Leon with the weighted graph of Frank and Bishop might offer an alternative approach for investigating uncertainty in protection graphs.

This paper applied the TG Protection Model to only two patterns of attack, the buffer overflow attack and an attack on authentication credentials. It would be useful to identify more attack patterns and how to best model them with TG.

While TG/MC inherits the limitations of the expressive power of the TG model, the overall approach might be applicable for dealing with uncertainty in other formal security models. We hope this paper contributes to the broader understanding of uncertainty in security models while offering insight into the application of TG/MC models to important real-world problems.

References

[1] M. Bishop, Theft of information in the take-grant protection model, *Journal of Computer Security* **3**(4), (1994/1995), 283–309.

[2] M. Bishop, *Computer Security: Art and Science*, Addison-Wesley, Boston, MA, 2003.

[3] M. Bishop and L. Snyder, The transfer of information and authority in a protection system, in: *SOSP '79: Proceedings of the seventh ACM symposium on Operating systems principles*, ACM Press, New York, NY, USA, 1979, ISBN 0-89791-009-5, 45–54.

[4] A. Bittau, M. Handley and J. Lackey, The final nail in wep's coffin, in: *SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P'06)*, IEEE Computer Society, Washington, DC, USA, 2006, ISBN 0-7695-2574-1, 386–400.

[5] E. Borgonovo and C. Smith, A case study: Two components in parallel with epistemic uncertainty, *Saphire Facets* **4**(2), (1999), 2–6, URL http://saphire.inl.gov/guest_area/FACETS_Vol-4_No-2.pdf.

[6] W. E. Burr, D. F. Dodson and W. T. Polk, Electronic authentication guideline, Technical Report 800-63, National Institute of Standards and Technology (NIST), Gaithersburg, MD 20899-8930, April 2006.

[7] N. Cam-Winget, R. Housley, D. Wagner and J. Walker, Security flaws in 802.11 data link protocols, *Commun. ACM* **46**(5), (2003), 35–39.

[8] M. A. Computing and N. Services, Msu security, Technical report, Michigan State University, East Lansing, MI USA, 2007, URL http://security.msu.edu/cgi-bin/index.pl?best.html#wlan.

[9] J. R. Conrad, Analyzing the risks of security investments with monte-carlo simulations, in: *Fourth Workshop on the Economics of Information Security (WEIS05)*, Harvard University (USA), June 2005, URL http://infosecon.net/workshop/pdf/13.pdf.

[10] J. R. Conrad, P. Oman and D. Conte de Leon, A monte-carlo approach for analyzing uncertainties in weighted graphical security models, accepted for publication in the International Journal of Critical Infrastructures.

[11] J. R. Conrad, P. Oman and C. Taylor, Managing uncertainty in security risk model forecasts with RAPSA/MC, in: *IFIP 11.1 and 11.5 Joint Conference*, December 2005.

[12] D. Conte de Leon, J. Alves-Foss, A. Krings and P. Oman, Modeling complex control systems to identify remotely accessible devices vulnerable to cyber attack, in: *ACM Workshop on Scientific Aspects of Cyber Terrorism (SACT)*, Washington D.C., November 2002, URL http://www.cs.uidaho.edu/~krings/SACT-2002-D.pdf.

[13] J. Frank and M. Bishop, Extending the take-grant protection system, Technical report, University of CA — Davis, December 1996, URL http://citeseer.ist.psu.edu/frank96extending.html.

[14] C. Hankin, *An Introduction To Lambda Calculi For Computer Scientists*, King's College Publications, February 2004.

[15] M. A. Harrison, W. L. Ruzzo and J. D. Ullman, Protection in operating systems, *Communications of the ACM* **19**(8), (1976), 461–471.

[16] K. J. Hole, E. Dyrnes and P. Thorsheim, Securing wi-fi networks, *Computer* **7**(38), (2005), 28–34.

[17] M. Howard and D. LeBlanc, *Writing Secure Code*, Microsoft Press, Redmond, WA, 2nd edition, 2003.

[18] J. K. Knudsen and C. L. Smith, Estimation of system failure probability uncertainty including model success criteria, in: *6th International Conference on Probabilistic Safety Assessment and Management*, International Association for PSAM, June 2002, URL http://nuclear.inl.gov/docs/papers-presentations/psam_6_modeluncertainty.pdf.

[19] R. J. Lipton and L. Snyder, A linear time algorithm for deciding subject security, *J. ACM* **24**(3), (1977), 455–464.

[20] D. Marc, A petri net representation of the take-grant model, in: *IEEE Computer Security Foundations Workshop VI*, IEEE Computer Society Press, June 1993, 99–108.

[21] A. Narayanan and V. Shmatikov, Fast dictionary attacks on passwords using time-space tradeoff, in: *Proceedings of the 12th ACM Conference on Computer and Communications Security CCS '05*, ACM Press, New York, NY, USA, 2005, ISBN 1-59593-226-7, 364–372.

[22] P. Oman, E. O. Schweitzer III and D. Frincke, Concerns about intrusions into remotely accessible substation controllers and scada systems, in: *Proc. 27th Annual Western Protective Relay Conferences*, October 2002, URL http://www.csds.uidaho.edu/deb/SCADA.pdf.

[23] A. One, Smashing the stack for fun and profit, Technical report, Phrack, 1996, URL http://www.phrack.org/archives/49/P49-14.

[24] M. Ossmann, WEP: Dead again, part 1, 2004, URL http://www.securityfocus.com/infocus/1814.

[25] M. Ossmann, WEP: Dead again, part 2, March 2005, URL http://www.securityfocus.com/infocus/1824.

[26] C. Phillips and L. P. Swiler, A graph-based system for network-vulnerability analysis, in: *NSPW '98: Proceedings of the 1998 workshop on New security paradigms*, SIGSAC, ACM Press, New York, NY, USA, 1998, ISBN 1-58113-168-2, 71–79.

[27] C. Taylor, A. Krings and J. Alves-Foss, Risk analysis and probabilistic survivability assessment (RAPSA): An assessment approach for power substation hardening, in: *ACM Workshop on the Scientific Aspects of Cyber Terrorism*, ACM, Washington, D.C., November 2002, URL http://www.cs.uidaho.edu/~krings/publications.html.

[28] D. Vose, *Risk Analysis: A Quantitative Guide*, John Wiley and Sons, West Sussex, England, 2nd edition, 2000.

[29] J. S. White, Cracking WEP "wired equivalent privacy" in 3 easy steps, Technical report, State University of New York Institute of Technology, March 2006, URL http://clubs.sunyit.edu/telecom/buzz/TBV2I22006.pdf.

Table 1: The Substation Example Vulnerabilities

Name	PDF	Description
ccat	Figure 15	Time to Failure for the communication controller's credentials.
dcat	Figure 16	Time to Failure for the desktop computer's credentials.
scat	Figure 17	Time to Failure for the SCADA server's credentials.
vcat	Figure 18	Time to Failure for the VPN's credentials.
cmwd	Figure 19	Time to Failure for the substation's phone number secret.
dbow	Figure 20	Time to acquire and exploit a buffer overflow in the desktop computer.
ebow	Figure 21	Time to acquire and exploit a buffer overflow in the engineer's home computer.
vcct	Figure 22	Time to wait for the engineer to establish a VPN connection from home.
wcat	Figure 23	Time to Failure for the WEP (wireless) credentials.

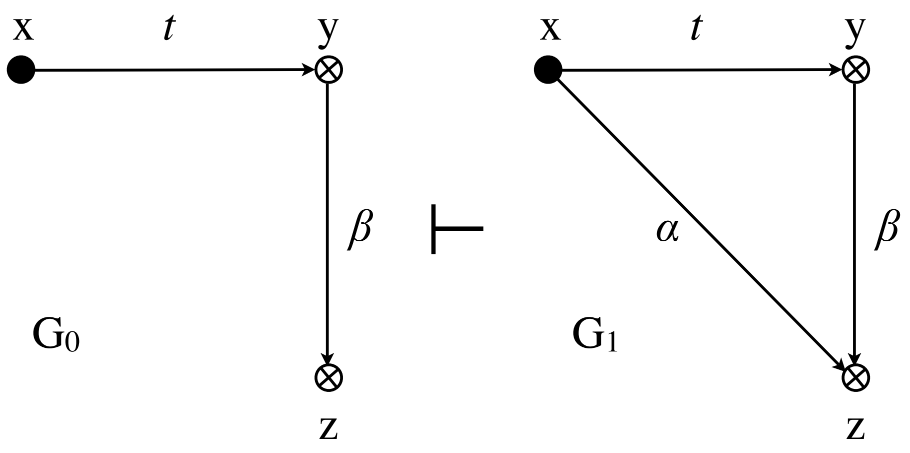


Figure 1: Subject **x** Takes (α to **z**) from Vertex **y**

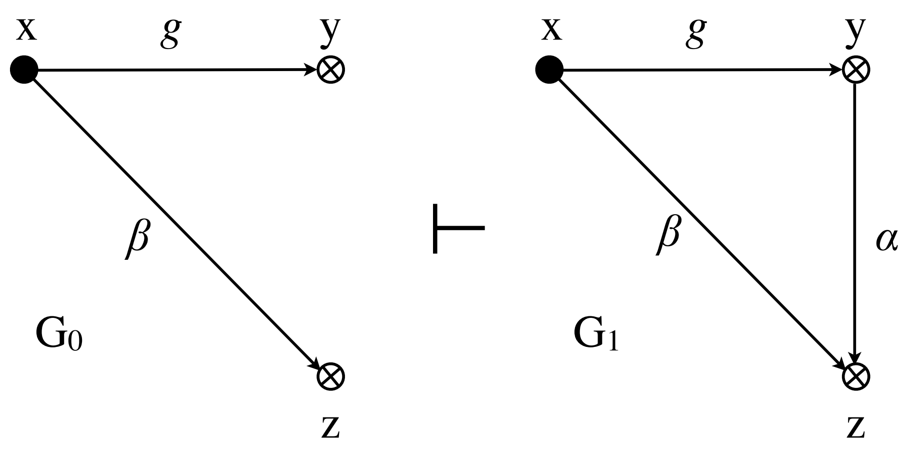


Figure 2: Subject **x** Grants (α to **z**) to Vertex **y**

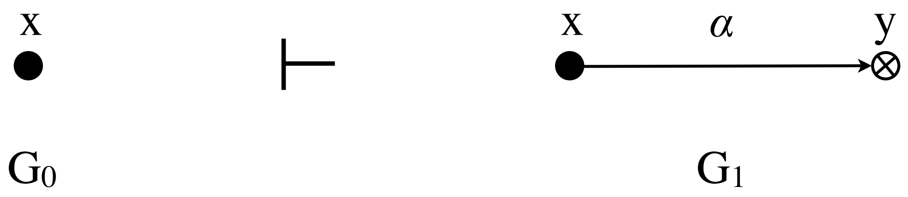


Figure 3: Subject **x** Creates (α to New Vertex **y**)

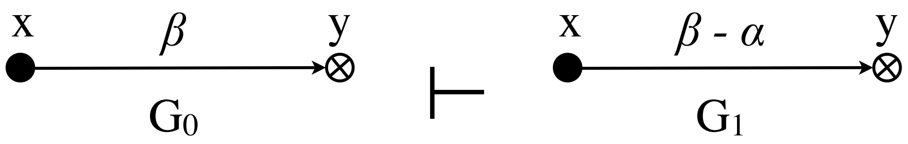


Figure 4: Subject **x** Removes (α to **y**)

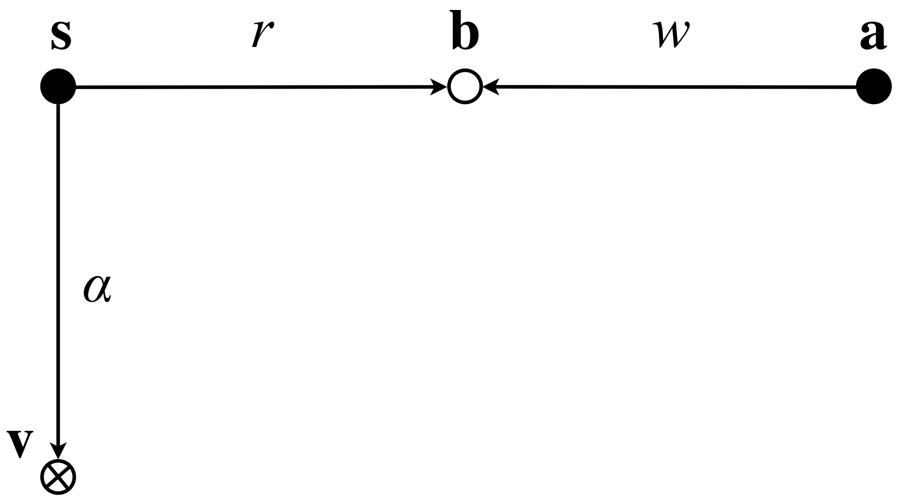


Figure 5: Defender's Initial View of the Buffer Overflow Attack, G_0

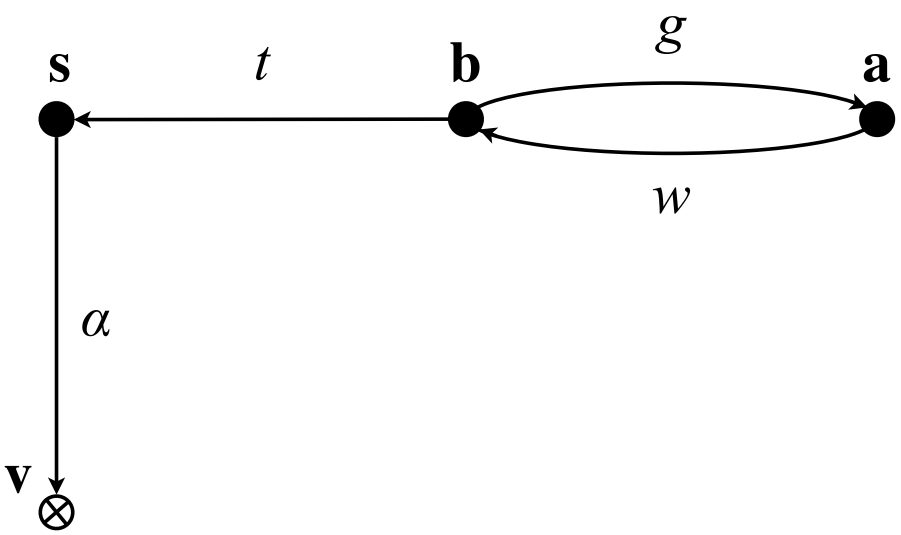


Figure 6: Attacker's Initial View of the Buffer Overflow Attack

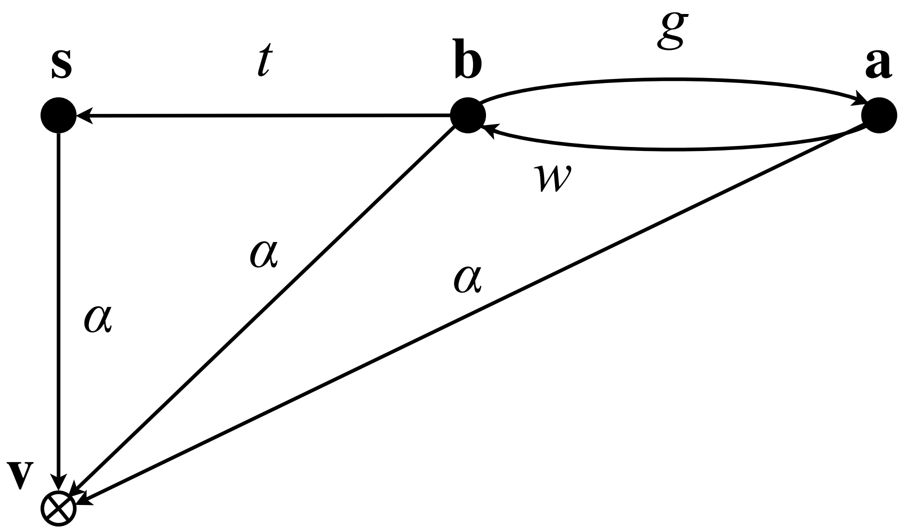


Figure 7: The Aftermath, G_v, of the Buffer Overflow Attack

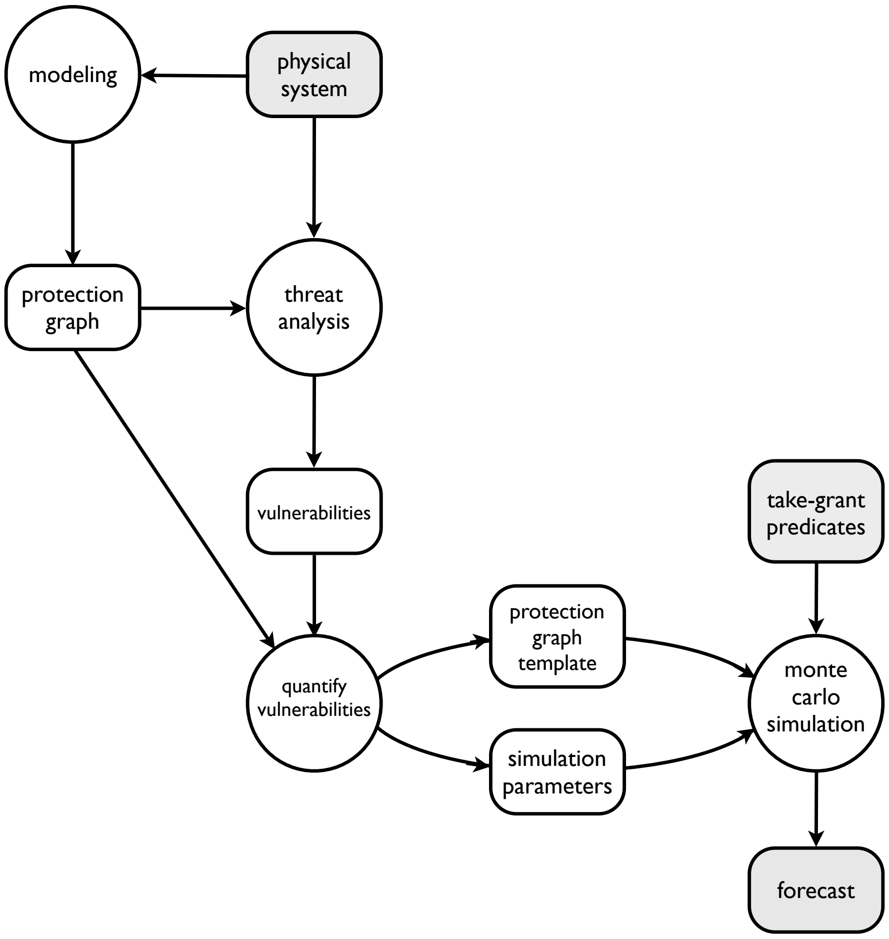


Figure 8: Applying the TG/MC Technique

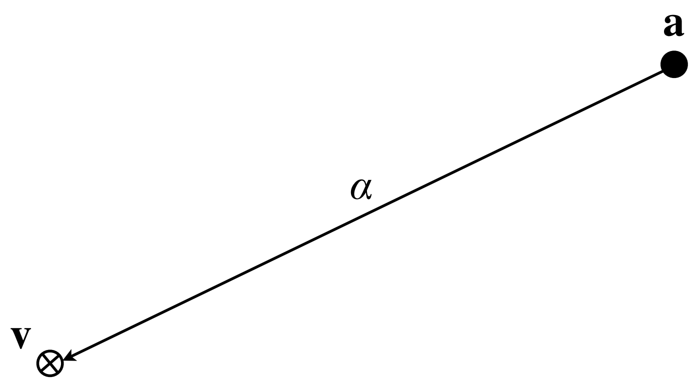


Figure 9: A Security Violation, Q, Providing an Attacker, **a** with (α to **v**)

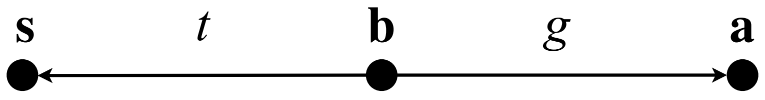


Figure 10: The Buffer Overflow Vulnerability, D

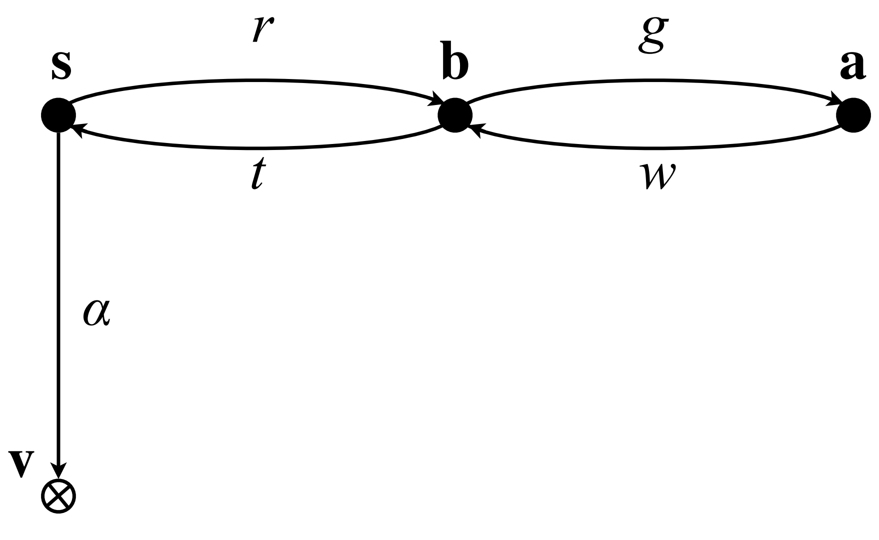


Figure 11: An Unsafe Protection Graph, $D \cup G_0$

1. **b** takes (α to **v**) from **s**
2. **b** grants (α to **v**) to **a**

Figure 12: The Buffer Overflow Witness

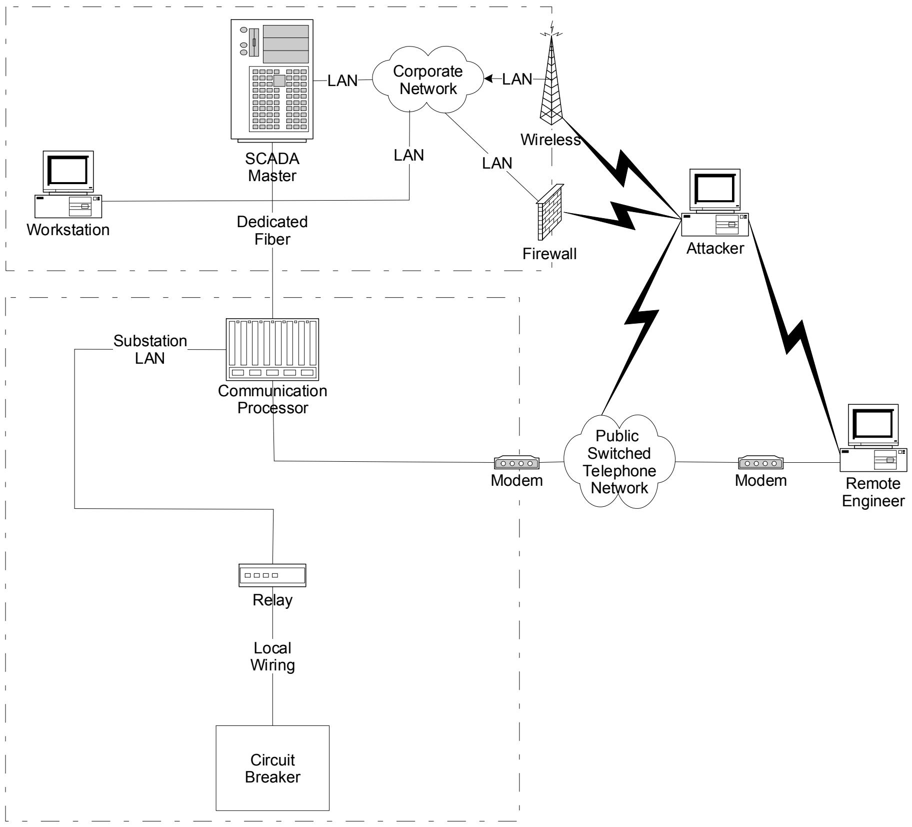
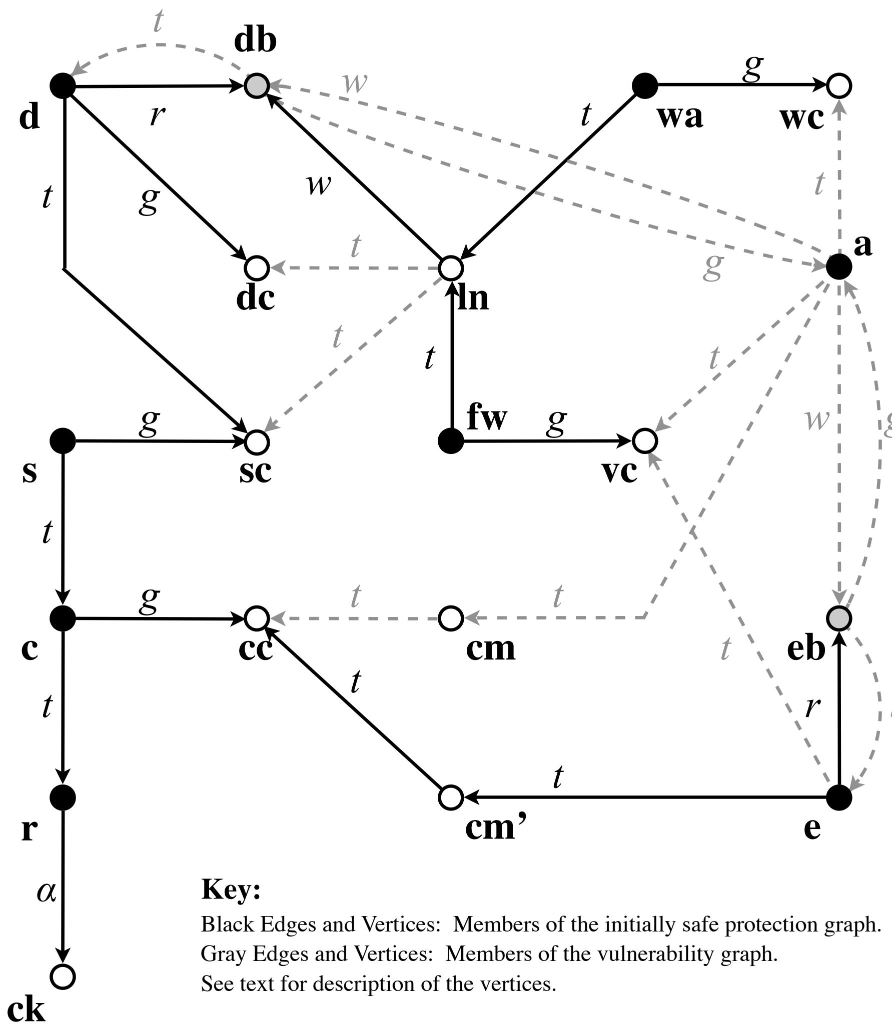


Figure 13: Diagram of the Example Substation System



Key:

Black Edges and Vertices: Members of the initially safe protection graph.
Gray Edges and Vertices: Members of the vulnerability graph.
See text for description of the vertices.

Figure 14: TG/MC Model of the Example Substation System

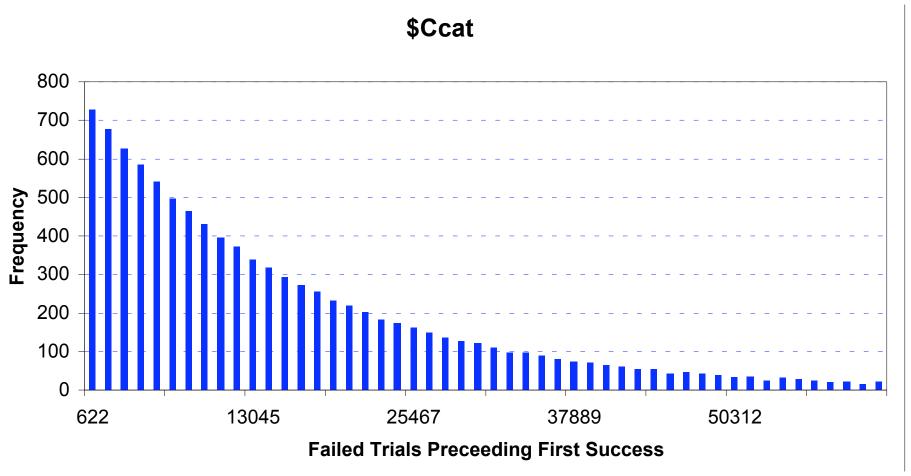


Figure 15: Frequency Chart for 10,000 Trial Attacks on the Communication Processor's Credentials (*ccat*)

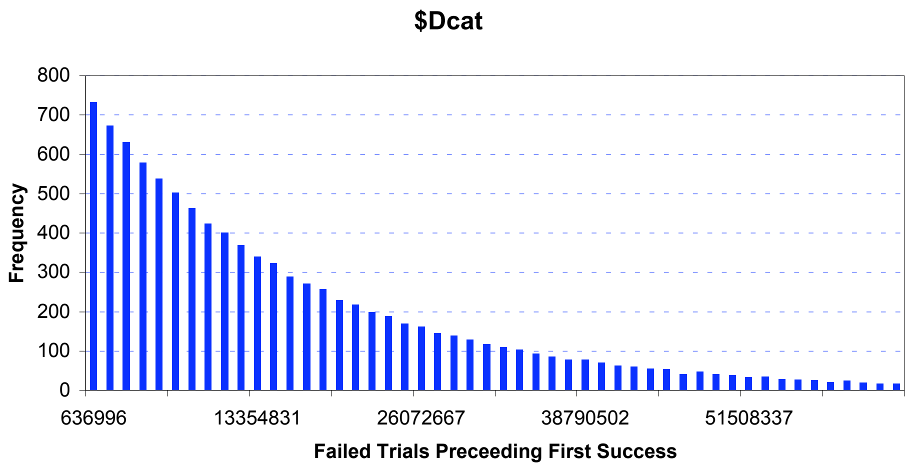


Figure 16: Frequency Chart for 10,000 Trial Attacks on the Desktop Workstation's Credentials (*dcat*)

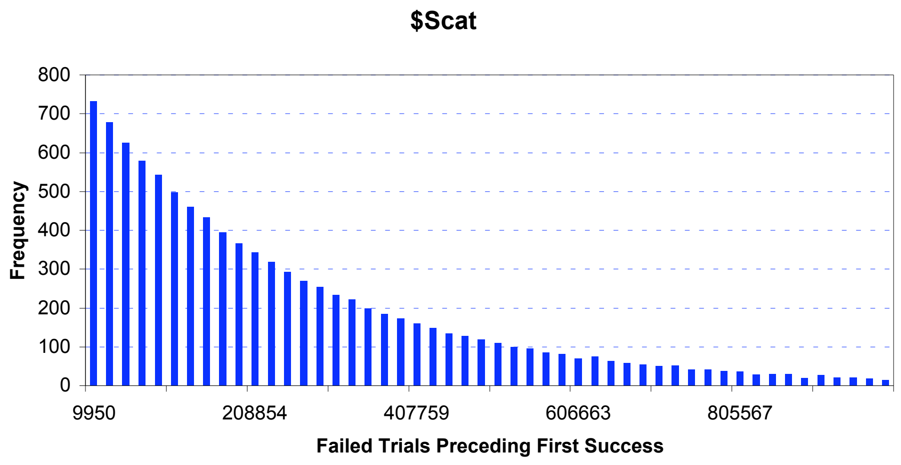


Figure 17: Frequency Chart for 10,000 Trial Attacks on the SCADA Server's Credentials (*scat*)

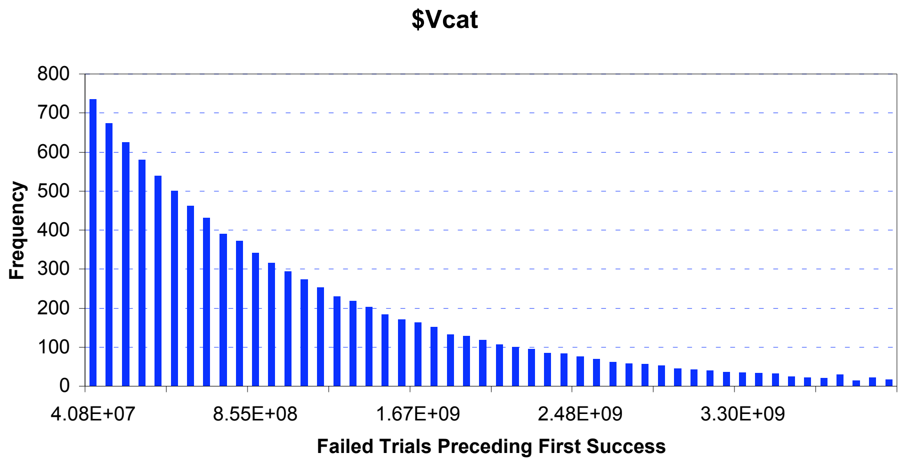


Figure 18: Frequency Chart for 10,000 Trial Attacks on the VPN Credentials ($vcat$)

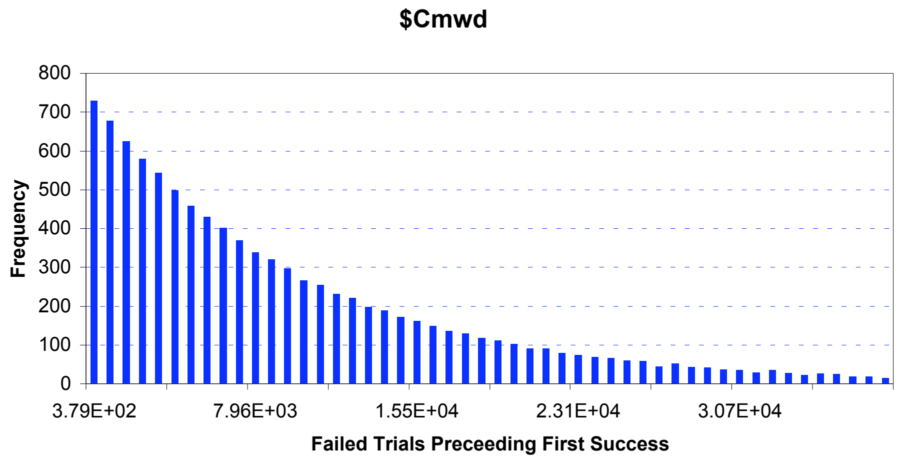


Figure 19: Frequency Chart for 10,000 Trial Attacks on the Substation's Modem Secret Phone Number ($cmwd$)

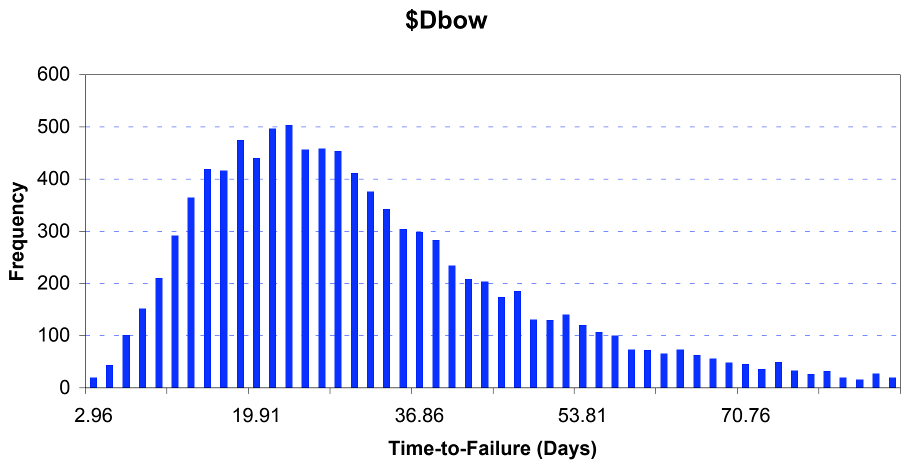


Figure 20: Time to Implement a Buffer Overflow Attack on Desktop Workstation (*dbow*)

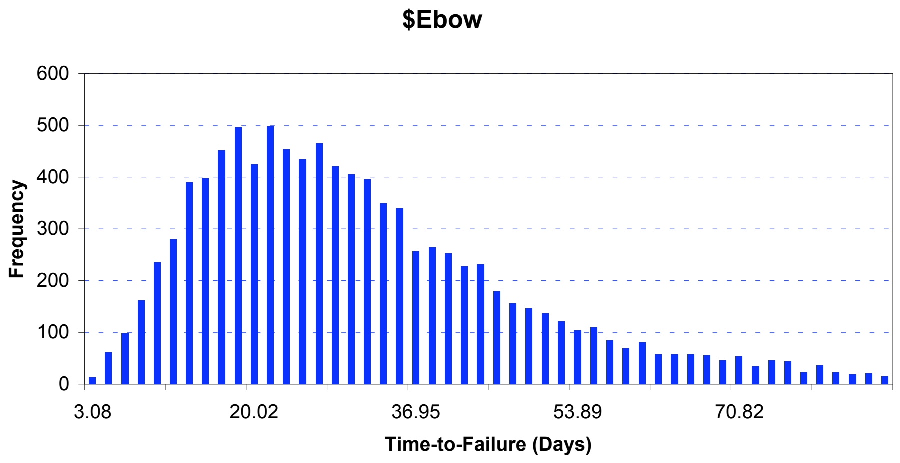


Figure 21: Time to Implement a Buffer Overflow Attack on the Engineer's Home Workstation (*ebow*)

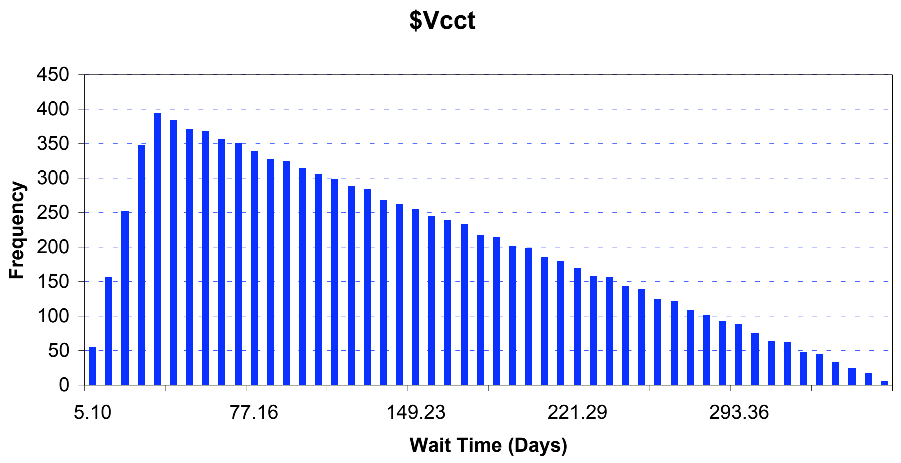


Figure 22: Elapsed Time Between Remote Engineer's VPN Connections (*vcct*)

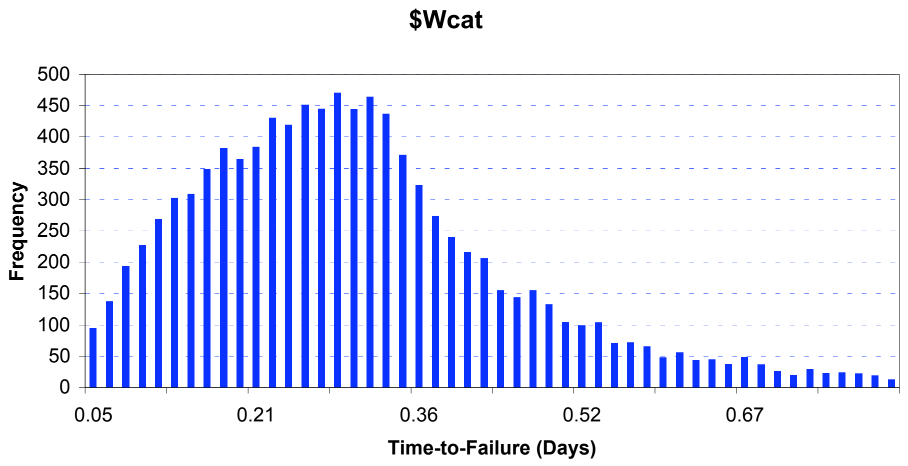


Figure 23: Failure of WEP (*wcat*)