Appears in Proc. Hawaii Int'l Conf. on System Sciences
Hawaii, January 2003

Balancing Safety Against Performance: Tradeoffs in Internet Security

Vu A. Ha and David J. Musliner
Honeywell Technology Center
3660 Technology Drive
Minneapolis, MN 55418
{vu.ha,david.musliner}@honeywell.com

Abstract

All Internet-accessible computing systems are currently faced with incessant threats ranging from simple script-kiddies to highly sophisticated criminal enterprises. In response to these threats, sites must perform extensive intrusion monitoring. This intrusion monitoring can have significant costs in terms of bandwidth, computing power, storage space, and licensing fees. Furthermore, when exploits are detected, the victims must take actions that can consume further resources and compromise their objectives (e.g., by reducing e-commerce server throughput). In this paper, we explore techniques for modeling the costs and benefits of various security monitoring and response actions. Given these models and stochastic expectations about the types of attacks that a site is likely to face, our CIRCADIA *automatic security control system is able to make real-time tradeoffs between the level of safety and security that is enforced, and the level of system resources/performance that are applied to the main computational objectives (e.g., e-commerce transactions). We show how* CIRCADIA *is able to dynamically adjust its security activities to account for changing threat profiles and objectives. The result: a continually-optimized balance of security-maintaining activity that reduces risk while still allowing the system to meet its goals.*

1. Introduction

In this paper, we describe CIRCADIA, the Cooperative Intelligent Real-Time Control Architecture for Dynamic Information Assurance. We are developing CIRCADIA to provide active real-time response to intrusions as they occur. CIRCADIA provides local, low-cost, autonomic defenses for computing resources by intelligently adapting threat monitoring systems and automatically responding to security threats in real time. By detecting and responding to intrusion activities within the timescale of the attacks themselves, CIRCADIA is able to defeat scripted attacks and prevent attackers from compromising protected systems.

The idea of responding automatically to attacks is not new; various researchers have developed prototype systems using rule-based, case-based, and other related inferential approaches to select attack responses [2]. CIRCADIA differs from prior efforts in several ways. First, CIRCADIA uses control-theoretic methods to *automatically synthesize* its reactive strategies, rather than relying on hand-built rules or other knowledge. This means that the system can automatically adapt its responses when faced with changing system resources, changes in security policy, or an evolving computational mission (i.e., the information processing and storage tasks the network is meant to be supporting and defending). Second, CIRCADIA reasons explicitly about the timeliness of its responses. Using models of the attacks that may occur and the available responses, CIRCADIA synthesizes reactive security control rules that are guaranteed to respond quickly enough to defeat an intruder, if possible. Third, when performance guarantees cannot be completely ensured, CIRCADIA can automatically make principled tradeoffs between the resources devoted to security and the resources devoted to handle mission processes. Finally, since CIRCADIA reasons explicitly about models of the attacks it faces and assesses the expected performance of the security controllers (reactions) it designs, CIRCADIA may also be used in an offline, system-design methodology to determine what level of security is achievable with a given set of assets and anticipated attack spectrum.

2. An Example Scenario

We begin our discussion by describing a very simplified model of a security attack on a system, modeled for CIRCADIA in Figure 1. The attack goes through several steps, modeled by "event" and "temporal" transitions that capture instantaneous or time-consuming pro-

```
;; Steps of the ping2root attack, modeled in CIRCADIA as non-volitional transitions.

(def-event ping2rootattempt-a
    :preconds ((is-ping2rootattempt-a F))
    :postconds ((is-ping2rootattempt-a T))
    :delay-distribution (uniform-distribution 10 20))
(def-temporal ping2rootsuccess-a
    :preconds ((is-ping2rootsuccess-a F) (is-ping2rootattempt-a T))
    :postconds ((is-ping2rootsuccess-a T))
    :delay-distribution (uniform-distribution 10 20))
(def-temporal new-user-added-a
    :preconds ((is-new-user-added-a F) (is-ping2rootsuccess-a T))
    :postconds ((is-new-user-added-a T))
    :delay-distribution (uniform-distribution 10 20))
(def-temporal new-user-added-failure-a
    :preconds ((is-new-user-added-a T))
    :postconds ((failure T))
    :delay-distribution (uniform-distribution 10 20))

;; Available actions to respond to the attack.
(def-action turnon-verbose-logging
    :preconds ((verbose-logging off))
    :postconds ((verbose-logging on))
    :delay-distribution (uniform-distribution 1 2))
(def-action kill-attacker-a
    :preconds ((is-ping2rootattempt-a T)(verbose-logging on))
    :postconds ((is-ping2rootattempt-a R)
                (is-ping2rootsuccess-a F)
                (is-new-user-added-a F))
    :delay-distribution (uniform-distribution 1 3))
(def-temporal recover-a
    :preconds ((is-ping2rootattempt-a R))         ;; cleaning up after
    :postconds ((is-ping2rootattempt-a F))        ;; killing the attacker
    :delay-distribution (uniform-distribution 1 3))
(def-action turnoff-verbose-logging
    :preconds ((verbose-logging on))
    :postconds ((verbose-logging off))
    :delay-distribution (uniform-distribution 1 2))

;; The initial state, before any attack.
(def-state initial-state
    :features ((failure F)
               (verbose-logging off)
               (is-ping2rootattempt-a F)
               (is-ping2rootsuccess-a F)
               (is-new-user-added-a F)))

;; System's goal: keep the logging activities minimal to maximize node A throughput.
(def-goal verbose-logging-is-off
    :goal-type :maintenance
    :features ((verbose-logging off))
    :reward 1)
```

Figure 1. CIRCADIA**'s model of a simple attack.**

cesses that are outside of CIRCADIA's control. First, the attacker attempts to execute a particular "ping2root" exploit on node A (`ping2rootattempt-a`), which may succeed after some time (`ping2rootsuccess-a`). The attack script then creates a new user on node A (`new-user-added-a`). In this simple example, we have told CIRCADIA that the system is considered to have failed if the attacker is able to add a new user that persists for some amount of time (`new-user-added-failure-a`).

To respond to this attack, the system needs to turn on a verbose logging function. Unfortunately, the verbose logging is so costly that it negatively affects node A's system performance, so there is a goal `verbose-logging-is-off`. Once verbose logging is on and the system detects the new user, it must destroy the new user account (`kill-attacker-a`) before the failure transition (described above) can occur. If the `kill-attacker-a` action is taken quickly enough, then the node will be restored to its prior operation mode (`recover-a`) and the attack may begin anew.

The model is completed with the specification of an initial state and a set of goals (which in this case contains a single goal of minimizing logging activities, thus maximizing node A throughput). Other details such as stochastic information (e.g. `delay-distribution`) and utilities (e.g. `reward`) will be discussed in the next section. Given this model, the CIRCADIA controller synthesis algorithm projects possible future states and decides, for each state, what control action is appropriate to preserve system security and retain the maximum possible quality of service to mission goals.

For example, Figure 2 depicts the state space that results from a particular automatically-generated plan. In essence, this plan is "cavalier": it reacts to the attack only when the attacker has successfully added a new account. Any further inaction will result in failure. This way, the total time the system spends in goal states (with `verbose-logging-is-off`) is maximized. At the same time, there may be a chance that starting to react in state 7 (once the new user is added) is too late; the attacker may already have enough time to cause system failure (state 10). We can imagine a "paranoid" plan that leaves `verbose-logging on` at all times, and activates `kill-attacker-a` whenever the system senses a `ping2rootattempt-a`. This plan will have higher probability of preventing `new-user-added-failure-a`, but trades off that security against a low level of node A throughput. How should we strike the right balance between these two extreme solutions? That is the main question we address in this paper, using the methodology of decision analysis.

Before continuing with discussion of the CIRCADIA

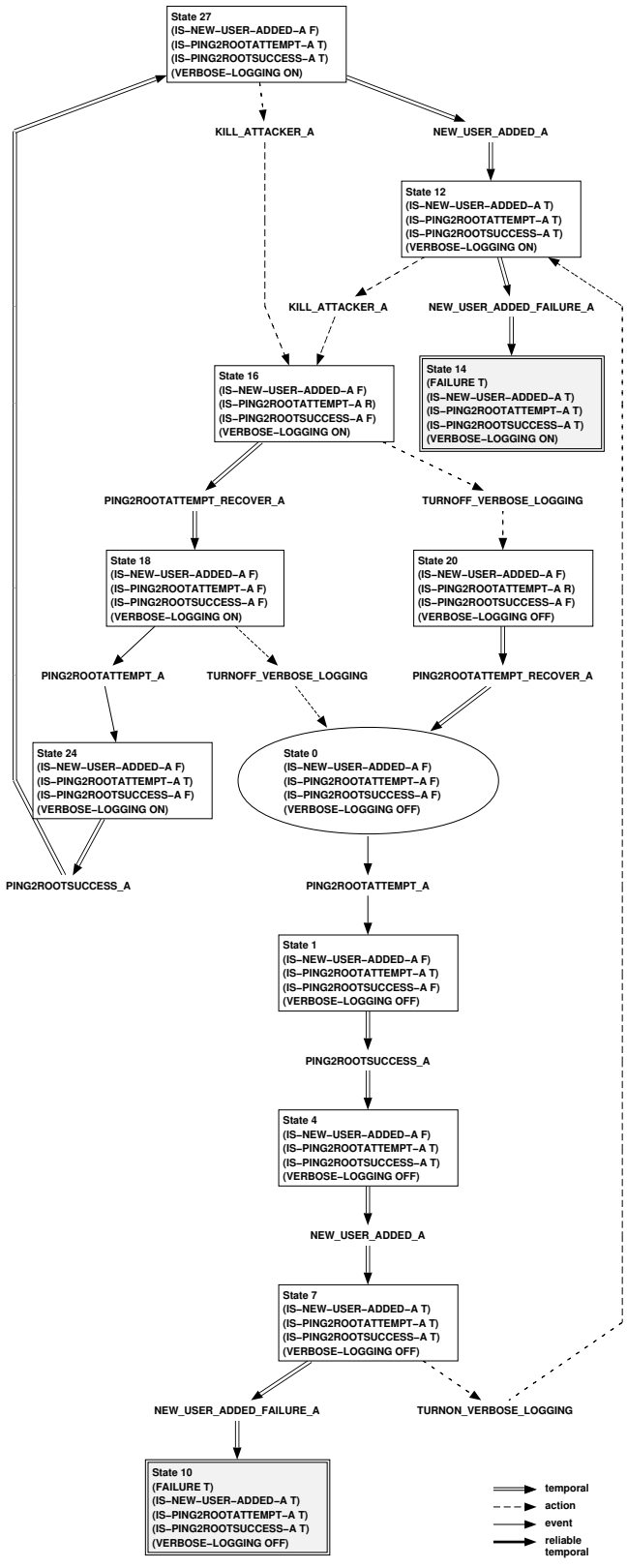

Figure 2. The "cavalier" plan.

framework, it is worth noting that we do not give CIRCADIA a sequential attack script, we give it each uncontrollable "primitive" piece (transition) of the script, and logical preconditions and postconditions that can be used to chain the primitives together. The advantage of this approach is that, as we provide CIRCADIA with more and more transitions describing exploit primitives, the system reasons about all possible combinations of individual transitions (the worst-case, including timing constraints). Thus novel attacks that combine previously-known elements in new ways are handled without further work by either CIRCADIA or the security engineer providing the input knowledge.

3. The Framework: Decision-Theoretic CIRCA

CIRCADIA is built on the foundation of the CIRCA architecture for intelligent real-time control systems [12, 13]. CIRCA automatically synthesizes and executes hard-real-time discrete event control systems for embedded applications. We provide a brief discussion of the CIRCA world model below.

3.1. The Original CIRCA World Model

The CIRCA planner searches for a plan that is guaranteed to be safe, by making sure that no failure state is reachable from initial states. Action transitions are planned to "preempt" temporal transitions to failure states. Transition τ_1 preempts τ_2 in state s if it can be proven that τ_1 will always be triggered before τ_2, independent of the history of state transitions. Safety of a plan is formally verified using model checking methods [15].

The formal non-probabilistic world model has seven elements $(S, S_F, S_0, T, E, min\Delta, max\Delta)$. S is a finite set of *states*, where each state represents a description of relevant features. $S_F \subset S$ is a set of *failure states*, which consists of all states in S that violate domain constraints or control-level goals. $S_0 \subset S$ is a set of possible *initial states*. $T = T_E \cup T_A \cup T_T$ is a finite set of *transitions*, where T_E is a set of *event transitions* representing world occurrences as instantaneous state changes, T_A is a set of *action transitions* representing actions performed by the run-time system, and T_T is a set of *temporal transitions* representing the progression of time. Each transition $\tau \in T$ is a mapping between states; $\tau : S \rightarrow S$. Function $E : S \rightarrow 2^T$ is a function mapping a state s to a set of transitions enabled in s. Functions $min\Delta, max\Delta : T \rightarrow \mathbb{R}$ map transitions to minimum and maximum trigger times.

At any particular point in time, the world is considered to occupy a single state in the model. The initial world state can be any state $s \in S_0$. The world state changes when a transition is triggered. If the current state is s and transition τ is triggered, the next state is given by $\tau(s)$. Not all transitions are necessarily enabled in all states. For each state $s \in S$, $E(s)$ is the subset of T denoting the set of transitions that can be triggered in state s. Only one transition can be triggered in each state at any given time, so transitions in $E(s)$ compete to trigger a state change.

We can associate a clock $r_{s,\tau}$ with each enabled transition τ in a state s, showing the time remaining until τ is scheduled to occur in s. (All clock speeds $r(s, \tau)$ are set to 1.) The clock value $r_{s,\tau}$ is called the *residual lifetime* of τ in s [6]. When a transition τ^* is triggered in state s, causing a transition to state $s' = \tau'(s)$, then the lifetimes of the transitions enabled in s' are initialized as follows:

1. If $\tau \in E(s) \setminus \{\tau^*\}$, then let $r_{s',\tau} = r_{s,\tau} - r_{s,\tau^*}$.
2. If $\tau \notin E(s) \setminus \{\tau^*\}$, then $r_{s',\tau}$ is set to some value in the interval $[min\Delta(\tau), max\Delta(\tau)]$.

The type of a transition determines the general form of the interval $[min\Delta(\tau), max\Delta(\tau)]$. Event transitions can occur at any time, and thus have a lower limit of zero and an upper limit of infinity. Temporal transitions are similar to events, but can have a non-zero lower limit. An action transition represents an action taken by the run-time system, and has a finite upper limit representing the worst-case execution time for that action.

3.2. Probabilistic Extension

Traditionally, CIRCA does not reason explicitly about probabilities, but uses time bounds ($min\Delta, max\Delta$). More recently, extensions have been made to include various types of probability information [1, 18]. Younes and Musliner [18] extended the CIRCA world model by associating a probability distribution function $F(t; \tau)$ with each transition τ, giving the probability that τ will be triggered t time units after it was last enabled. We require that $F(0; \tau) = 0$ (i.e. the distribution function corresponds to a positive random variable), because no transition can be triggered before it has been enabled. A typical choice of distribution for an event transition would be an exponential distribution, and for a temporal transition a shifted exponential distribution. For an action transition one could, for example, use a uniform distribution or a truncated normal distribution. In addition we can replace S_0 with a probability distribution p_0 over S, where $p_0(s)$ is the probability that the world starts in state s. The set of possible initial states is then simply $S_0 = \{s \mid p_0(s) > 0\}$. Finally, we define transition probabilities $p(s'; s, \tau)$ expressing the probability

of the next state being s' given that τ is triggered in state s:

$$p(s'; s, \tau) = \begin{cases} 1 & \text{if } \tau(s) = s' \\ 0 & \text{otherwise} \end{cases}$$

The elements (S, p_0, p, T, E, F, r) constitute a *time-homogeneous generalized semi-Markov process* (GSMP) [6].

CIRCADIA adopts this probabilistic extension to the CIRCA world model. For example, in our current scenario the time it takes for the attacker to cause system failure after he has successfully added a new user is modeled as a uniform distribution with range of [10, 20] (see Figure 1). In this example, while all the delay-distributions happen to be uniform, they are not required to be so. In fact, since our approach is simulation-based, as will be seen in the next section, almost any distribution can be specified to govern the transition times of events and actions, as long as it is amenable to sampling.

3.3. Introducing Utilities

In probabilistic terms, the traditional CIRCA planner can only distinguish between zero and non-zero probability of reaching a set of states. With probability distribution functions available for the transitions, we can set an arbitrary threshold θ representing the highest acceptable failure probability of a plan. Setting $\theta = 0$ we revert to the old model. With $\theta > 0$, though, we can accept plans that would have otherwise been discarded. For example, it now becomes possible to have a plan with event transitions to failure states provided that the events represented by these transitions are sufficiently infrequent, or the probability of entering a state in which such events are enabled is sufficiently low. The problem of plan verification now becomes a hypothesis testing problem [18], which can be solved using the sequential testing algorithm pioneered by Wald [17].

While this extension allows for more flexibility (by accepting plans with positive but small failure probability), it fails to accommodate any tradeoffs between the severity of failure and the importance of achieving goals, or any tradeoffs among the goals. In the information security domain that we are interested in, these limitations prevent the planner from constructing defense plans that can flexibly adapt to the uncertain nature of security threats, and the changing demands of trading off information services against security level. Decision theory [10], which models uncertainty with probabilities and the costs/benefits of actions with utilities, provides an attractive answer to this challenge. In the decision-theoretic world, failures are not created the same, nor are goals. The best plan is the one that *maximizes the expected utility*.

Toward this end, we need to have a model of utility, in addition to the probabilistic models of events, transitions, and actions. Because of the goal-oriented nature of planning missions in the information security domain (and several other domains such as military planning), we define a goal-directed utility model [8] that has the following characteristics:

1. The utility function assigns a real number to a *finite-horizon plan execution path*, where the time horizon h is domain specific.

2. The utility function is a weighted sum of sub-utility functions, each of which is scaled to have range $[0, 1]$, and belongs to one of three categories: *maintenance-goal* (MG), *achievement goal (AG)*, and *repeated achievement goal* (RAG):

$$u = \sum_i w_i u_i^{MG} + \sum_j w_j u_j^{AG} + \sum_k w_k u_k^{RAG},$$

   where w_i, w_j, w_k are the weights of the corresponding sub-utility functions, which are scaled to sum to 1.

As an example, a CIRCADIA plan for running a web server may have a maintenance goal *"maintain high data throughput for as long as possible"*, an achievement goal *"complete the Perl interpreter upgrade"* and a repeated achievement goal *"perform crucial data backup every night"*. The utility function may be defined as: $u = .4u_1^{MG} + .1u_2^{AG} + .5u_3^{RAG}$. In this equation, u_1^{MG} may be defined proportional to the average data throughput, u_2^{AG} can simply be a binary function, and u_3^{RAG} may be defined as the number of successful backups performed before the time limit. Finally, the weights $.4$, $.1$, and $.5$ reflect the relative importance of the goals in the overall utility function.

Several observations are in order regarding the above formalization. First, we note that there are at least two other approaches to modeling utility in decision-theoretic planning. In the first approach (see, e.g. [7]), a plan is modeled as a sequence of actions that leads from an initial state to some final state, and the utility function is defined as a real-valued function *on the set of final states*. In this approach, only what happens *at the end* of plan execution counts. In the second approach, the utility function is defined on infinite-horizon plan execution paths via the use of a time-discounted factor [4]. We choose to define the utility function on *execution paths* (as opposed to final states) because a) CIRCADIA plans are reactive (as opposed to sequential) and b) only an execution path contains necessary information to compute MG-directed and RAG-directed sub-utilities. We choose to restrict ourselves to *finite-horizon* execution paths because available methods for computing

expected utility on infinite-horizon execution paths are typically analytical (as opposed to sampling-based), based on much simpler models of time and utilities [3]. The second observation is that this formalization is not exclusive: the model of utility can be modified based on the specifics of the actual problem. For example, we can add a deadline to an achievement goal, or a time-discounted factor to the sub-utility functions. Again, because of the sampling-based nature of our approach, this modification in general can be accommodated in the expected utility estimation method, described in the next section.

In the "ping2root" scenario, there is one maintenance goal `verbose-logging-is-off`, which has `reward` of 1. This means that the utility function u is computed over an execution path by computing the proportion of time spent in states satisfy `verbose-logging-is-off`. Finally, we need to set the utility of failed execution paths to the negative value of some large number (e.g. -10,000). The fact that this number is finite means that the system designer should make a conscious tradeoff between the achieving goals versus maintaining system safety.

3.4. Identifying Plans with Highest Expected Utilities

With the introduction of the utility model, the CIRCADIA planning problem now becomes the problem of searching for the plan with highest expected utility. The expected utility of a plan π is $E[u(X(\pi,h))]$, where $X(\pi,h)$ is the random execution path resulting from executing plan π until time horizon h. The key issue here is to compute the EUs. Since our CIRCA model corresponds to a GSMP, for which there are no known analytic methods to efficiently compute the expected utility, the only feasible approach is to use Monte Carlo sampling to approximate the EU.

Note that by definition, the utility function has range $[0,1]$, and as a consequence, the utility of a plan is a random variable with range $[0,1]$. Ideally, we would like our estimate $\tilde{u}$ of $E[u]$ to be within ϵ of the actual mean with probability of at least $1-\delta$, where ϵ and δ are small positive real numbers: $\Pr(|\tilde{u} - E[u]| > \epsilon) < \delta$. Several well-known results from statistics give upper bounds on the required number of simulation runs to ensure ϵ-precision with $(1-\delta)$-confidence. We list these results below.

1. *Chebyshev's inequality*: $\Pr(|\tilde{u}_k - E[u]| > \epsilon) \leq \frac{\sigma^2}{k\epsilon^2}$.
2. *Bernstein's inequality*: $\Pr(|\tilde{u}_k - E[u]| > \epsilon) \leq 2\exp(\frac{-k\epsilon^2}{2\sigma^2 + 2M\epsilon/3})$.
3. *Hoeffding's inequality*: $\Pr(|\tilde{u}_k - E[u]| > \epsilon) \leq 2\exp(-\frac{k\epsilon^2}{2M^2})$.

In the above inequalities, $\bar{u}_k$ denotes the mean of a k-size sample of u, σ^2 denotes the variance of u (which is at most 1/4 because u is bounded in $[0,1]$), and in Bernstein's and Hoeffding's inequalities, M is a positive number such that $|u - E[u]|$ is bounded by M almost surely (for example, we can set $M = 1$). Chebyshev's inequality is classical. Bernstein's inequality is discussed, for example in [16]. Hoeffding's inequality [9] is often used in the machine learning literature.

These inequalities translate into the following upper bounds on the required number of samples to ensure (ϵ,δ) approximation:

1. *Chebyshev's bound*: $k \geq \frac{1}{4\delta\epsilon^2}$.
2. *Bernstein's bound*: $k \geq \ln(2/\delta)(1/2\epsilon^2 + 2/3\epsilon)$.
3. *Hoeffding's bound*: $k \geq \ln(2/\delta)(2/\epsilon^2)$.

Note that Chebyshev's bound increases linearly with respect to $1/\delta$, while Bernstein's and Hoeffding's increase logarithmically. As ϵ decreases, it is not hard to see that Bernstein's bound is smaller than Hoeffding's. This leads us to adopt Bernstein's bound for small ϵ and δ, and Chebyshev's bound for larger ϵ and δ. Algorithm 1 identifies the plan with highest expected utility.

---

1. Set $current\_best\_EU = -\infty$.
2. Generate a plan A. Simulate A up to time horizon h, compute the utility of the resulting path.
3. Repeat the above step for k times, where k is the smallest among the Chebyshev's, Bernstein's, and Hoeffding's bounds, compute the average utility $\bar{u}_k$. If $\bar{u}_k > current\_best\_EU$, set $current\_best\_EU = \bar{u}_k$.
4. Go back to step 2. Continue until some stopping criterion is true (e.g. there are no more plans, or time limit is reached).

Algorithm 1. Identifying best plan using statistical guarantees.

3.5. Sequential Methods for Identifying Plans with Highest Expected Utility

The above sample upper bounds are applicable for *any* random variable with range $[0,1]$, which is important for our analysis because u is a complex function and will most likely not observe known parametric forms such as uniform or Gaussian. The downside of this generality is that these upper bounds are rather high: for $\epsilon = .01$ error margin and 95% confidence ($\delta = .05$), Bernstein's bound is

1. Set $current\_best\_EU = -\infty$.
2. Generate a plan A. Simulate A up to time horizon h, compute the utility of the resulting path.
3. Repeat 2 until one of the following occurs: (a) The number of failed execution paths so far is greater than the "rejection threshold". In this case, the plan is eliminated as one with high failure probability. Look at the failed execution paths, identify the culprits, and backjump to generate a new plan based on the culprit. (b) The number of execution paths resulting in utility less than $current\_best\_EU$ is greater than the "rejection threshold". In this case, the plan is eliminated as one with high probability of being inferior to the current best plan. Chronologically backtrack to generate a new plan. (c) The number of simulations reaches the lowest among the three (Chebyshev's, Bernstein's, and Hoeffding's) bounds. Stop the simulation for this plan and compute the average sample utility $\tilde{u}$. If $\tilde{u} > current\_best\_EU$, then set $current\_best\_EU = \tilde{u}$. Otherwise, the current plan is eliminated as being inferior to the current best plan.
4. Go back to step 2. Continue until some stopping criterion is true (e.g. there are no more plans, or plan time limit is reached).

Algorithm 2. Identify the plan with highest expected utility sequentially.

4,612 (Hoeffding's bound is 7,378, and Chebyshev's bound is 50,000). One possible way to cut down on the number of samples is to appeal to sequential analytic techniques (see e.g. [18]). Unfortunately, there is no known effective sequential method to estimate the mean of a random variable u unless parametric assumptions are made about u [5]. Note that in Step 4 in Algorithm 1, what we really are interested in is whether the current plan π is inferior to the current best plan, i.e. $E[u(\pi, h)] < current\_best\_EU$, which is clearly a hypothesis testing problem. (We can not use sequential methods to determine if the current plan is *superior to* the current best plan, since if that is the case, we will need to estimate its expected utility anyway.) If we can quickly determine, via a sequential sampling procedure that π is inferior to the current best plan, then there is no need to continue estimating the EU of π; the algorithm can move on to the next plan.

Algorithm 2 employs an heuristic, acceptance-based approach to this problem. For each sample execution of the current plan, we compute and compare the utility of the current plan to the expected utility of the current best plan. If the current plan yields lower utility for enough number of times (in a sequential analytic sense), it is deemed to be inferior and eliminated. This approach is only a heuristic because it does not take into account the magnitude of the utilities of the samples. We are working on a more rigorous approach to sequentially determine if the current plan is inferior to the current best plan.

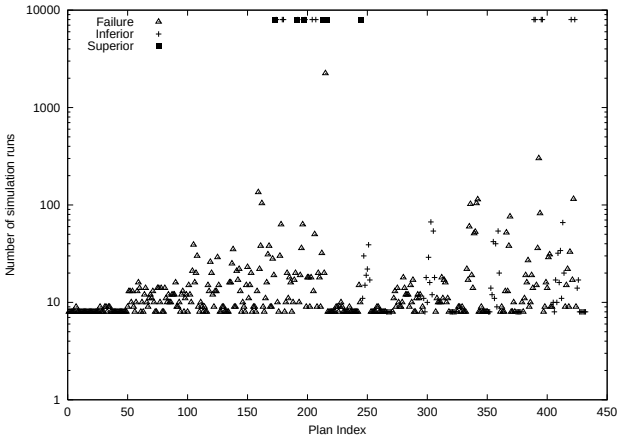

Figure 3. Many plans are determined to be failure-prone or inferior to the best-so-far with fewer than 100 simulation runs.

Note that "rejection number" is not a constant; instead it is updated after the completion of each simulation run. The exact formula to compute this number depends on several user-adjustable parameters such as "threshold probability of accepting a plan with high failure probability" (type I error) and "threshold probability of rejecting a plan with low failure probability (type II error). See Younes and Musliner [2002] for more details.

One scenario where this technique does not result in significant savings is when successive plans are of increasingly better quality. On the other hand, if the best plan is identified rather quickly (this crucially depends on the strength of the plan's heuristic, and the strength of the simulation-guided backjump routine, to be discussed later in Section 5), then from that point on we will be able to save a significant amount of time on estimating utilities, by quickly dismissing plans with high failure probability, or with high probability of being inferior to the current best. Figure 3 illustrates the savings afforded by the new algorithm in the "ping2root" problem described in Figure 1.

4. Making Tradeoffs in CIRCADIA

In Section 2, we raised the question of how to strike the right balance between two extreme solutions ("cavalier" and "paranoid") to the "ping2root" attack. With the introduction of the decision-theoretic modeling described in Section 3, we now have a principled answer to this question.

Instead of randomly deciding on being "cavalier," "paranoid," or somewhere in between, the system designer can now make a conscious effort to encode his preferences by setting the utilities of achieving goals and enduring failure. He also needs to carefully consider the probabilistic information that characterizes the transition times of events, actions, and other non-volitional transitions. Finally, he needs to set the thresholds for the sequential hypothesis tests described in Algorithm 2. While all of these efforts are intellectually demanding, their fruits can not be denied. The system designer can now sit back and let CIRCADIA do the hard work of identifying the plan that best balances between achieving the specified goals and avoiding failures. If planning time is unlimited, CIRCADIA will find that best plan simply by doing an exhaustive search through the plan space. If this is not the case (which is the more practical case), the system designer can interrupt CIRCADIA at any time and obtain the plan that is the current best candidate.

To illustrate this point, let us go back to the now familiar "ping2root" problem. We assume here that the utilities are fixed (`verbose-logging-is-off` has `reward` of 1, and `failure` has utility of -10,000). How could CIRCADIA come up with different plans like the "cavalier" and the "paranoid" ones? The answer lies in varying the transition times, or more precisely, the probability distributions of transition times of events and actions in the model. If the `new-user-added-failure-a` transition takes an amount
of time sufficient to `turn-on-verbose-logging` *and* `kill-attacker-a`, then the "cavalier" plan, depicted in Figure 2 may be the plan with highest expected utility. We can imagine this scenario to be a low-security scenario, when it is perceived that an attack is unlikely, and when an attack occurs, there is plenty of time to respond. The goal is hence to maximize the expected utility by maximizing the throughput of node A. In the military information security area, this could correspond to the DoD's INFOCON NORMAL[1] level. In contrast, if it is perceived that an attack is very likely from multiple sources (INFOCON DELTA), the system should be put on the high-security mode to react in the quickest way possible to any potential attack. This scenario will be reflected in rapid transition times for `ping2rootattempt-a`, `ping2rootsuccess-a`, `new-user-added-a`, and `new-user-added-failure-a`, and the paranoid plan having the highest expected utility. We can also imagine another scenario in between these two extreme scenarios (e.g., INFOCON ALPHA), where the plan with highest expected utility is the one that starts to react upon detecting a `ping2rootattempt-a` (see Figure 4).

[1] The five infocon levels, from the lowest to the highest are: NORMAL, ALPHA, BRAVO, CHARLIE, DELTA.

5. Some Details of the CIRCADIA Planner

The reader up to this point has been given a rather high-level overview of the CIRCADIA planner with emphasis on its decision-theoretic reasoning capabilities. Since CIRCADIA is based on CIRCA, answers to many architectural and implementation questions can be found on previous CIRCA publications. In this paper, we focus attention on two issues that are of particular importance to CIRCADIA: the *heuristic* and the *backjumper*. These two seemingly disconnected issues both have paramount importance in how determining quickly the system arrives at the plan with highest expected utility, and thus on the (anytime) performance of CIRCADIA.

CIRCA's heuristic is responsible for making a decision to assign an action to a state, and thus indirectly responsible for the *order* of the plans being generated. Presently, CIRCADIA uses the original CIRCA heuristic [14]. At any given time in the planning process, there is a set of states that have not been planned for, called *open states*. Initially, this set contains only the initial states. As planning progresses, the planner projects events and actions on this set to obtain more and more open states. The heuristic selects an open state from this set, examines the actions that are applicable to that state, and ranks them according to how likely they will be able to take the system to a state that satisfies the specified goals. This ranking is computed via an implementation of McDermott's regression-match graph [11, 14]. When there are no more open states, i.e. when every state has been planned for, we obtain a complete plan.

The backjumper, on the other hand, is responsible for handling a plan that is deemed (via simulation) to have high probability of failure. Since the heuristic does not consider failure but focuses exclusively on getting to goal states, it tends to generate "over-ambitious" plans that result in high failure probability. An illustrating example is the "cavalier plan" that ignores the threat until `new-user-added-a` (thereby trying to maximizing the time spent in `verbose-logging-is-off` states, see Figure 2). It is characteristic of the goal-oriented nature of the heuristic that the "cavalier plan" is the first plan generated by CIRCADIA. If the transition time from this state to failure is so short that there is not enough time to `turn-on-verbose-logging` *and* `kill-attacker-a`, then too many simulation traces will end in failure. In this case, the backjumper needs to examine the failure traces, identify the culprit decision that is responsible for failure (e.g., the system should `turn-on-verbose-logging` upon detecting `ping2rootattempt`, instead of doing nothing). If we imagine the search for the best plan as a depth-first search in a tree, a backjumper that correctly identifies the decision

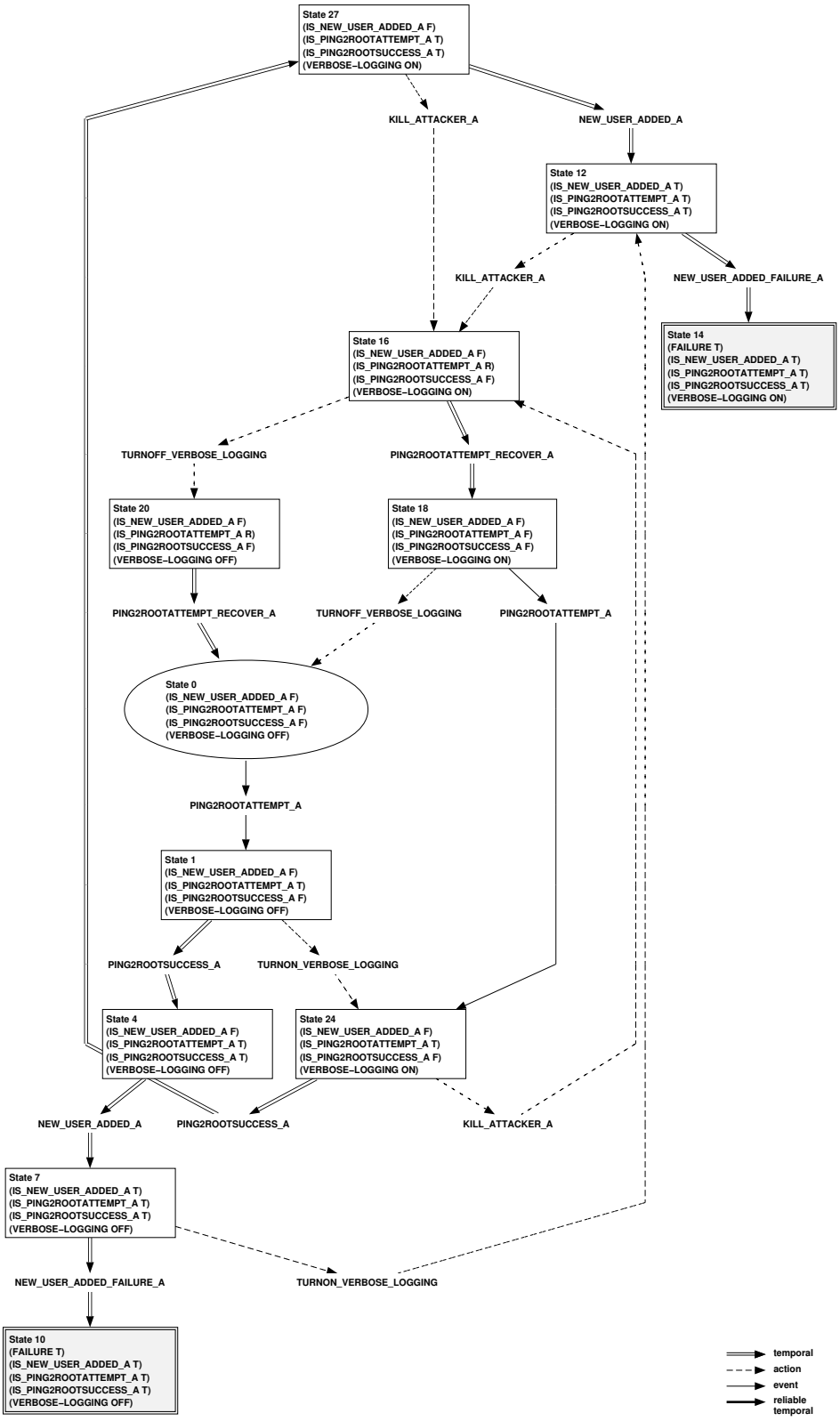

Figure 4. The "INFOCON ALPHA" plan balances security against performance.

culprit will be able to help the planner save a lot of time generating and evaluating plans in a bad branch of the tree where the better plans are located in different branches. A good backjumper will also maintain *completeness* by not letting the best plan slip out of the search net (for example, by jumping too far back and thus discarding the best plan). Currently, the backjumper in CIRCADIA is a modified version of the backjumper already present in CIRCA. We have the options to provide the backjumper with all the failure traces, the shortest failure trace, or the most common failure trace, based on which the backjumper will identify the *most recent* decision that leads to failure. A detailed discussion of the implementation of this backjumper is beyond the scope of this paper and will be reported elsewhere. Our experience experimenting with these options indicate that all of these options are complete: the planner, given sufficient time, will always find the plan with highest expected utility. Furthermore, the savings afforded by these backjumping options, i.e. the percentage of plans eliminated without evaluation, range from 39% to 51%.

While the performance of our backjumper is quite satisfactory, much work remains to be done to improve the current heuristic, as it is goal-oriented rather than EU-oriented. While constructing EU-aware heuristic is an obviously quite complex task, we could take the initial step by incorporating the ability to rank the relative importance of different goals. The next step would be to incorporate the ability to analyze the existing decision choices and simulation traces to provide a better heuristic.

Acknowledgments

This material is based upon work supported by DARPA/ITO and the Space and Naval Warfare Systems Center – San Diego under Contract No. N66001–00–C–8039. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA, the U.S. Government, or the Space and Naval Warfare Systems Center – San Diego.

References

[1] E. Atkins, E. H. Durfee, and K. G. Shin. Plan development using local probabilistic models. In *Proc. Conf. on Uncertainty in Artificial Intelligence*, pages 49–56, August 1996.

[2] R. G. Bace. *Intrusion Detection*. Macmillan Technical Publishing, 2000.

[3] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[4] R. Dearden and C. Boutilier. Abstraction and approximate decision theoretic planning. *Artificial Intelligence*, 1:219–283, 1997.

[5] M. Ghosh, N. Mukhopadhyay, and P. K. Sen. *Sequential Estimation*. Wiley Seris in Probability and Statistics, 1997.

[6] P. W. Glynn. A GSMP formalism for discrete event systems. *Proceedings of the IEEE*, 77(1):14–23, January 1989.

[7] P. Haddawy, A. Doan, and C. Kahn. Decision-theoretic refinement planning in medical decision making: Management of acute deep venous thrombosis. *Medical Decision Making*, 16(4):315–325, Oct/Dec 1996.

[8] P. Haddawy and S. Hanks. Utility models for goal-directed, decision-theoretic planners. *Computational Intelligence*, 14(3), 1998.

[9] W. Hoeffding. Probability inequalities for sum of bounded random variables. *American Statistical Association Journal*, 58:13–30, March 1993.

[10] R. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. Wiley, New York, 1976.

[11] D. McDermott. Using regression-match graph to control search in planning. *Artificial Intelligence*, 109(1-2):111–159, 1999.

[12] D. J. Musliner, E. H. Durfee, and K. G. Shin. CIRCA: a cooperative intelligent real-time control architecture. *IEEE Trans. Systems, Man, and Cybernetics*, 23(6):1561–1574, 1993.

[13] D. J. Musliner, E. H. Durfee, and K. G. Shin. World modeling for the dynamic construction of real-time control plans. *Artificial Intelligence*, 74(1):83–127, March 1995.

[14] D. J. Musliner, R. P. Goldman, M. S. Boddy, and K. D. Krebsbach. Distributed CIRCA: Guaranteeing coordinated behavior in distributed real-time domains. Technical Report SST-R97-030, Honeywell Technology Center, October 1997.

[15] D. J. Musliner, R. P. Goldman, and M. J. Pelican. Using model checking to guarantee safety in automatically-synthesized real-time controllers. In *IEEE International Conference on Robotics and Automation*, 2000.

[16] D. Pollard. *Convergence of stochastic processes*. Springer-Verlag, 1984. MR86i:60074.

[17] A. Wald. Sequential tests of statistical hypotheses. *Annals of Mathematical Statistics*, 16(2):117–186, June 1945.

[18] H. L. S. Younes and D. J. Musliner. Probabilistic plan verification through acceptance sampling. In *Proceedings of the AIPS 2002 Workshop on Planning via Model Checking*, Toulouse, France, April 2002. AAAI Press.