Agent-Based Decision Support for Moving Target Technology Deployment

Ben Priest, Era Vuksani, Neal Wagner, Brady Tello, Kevin M. Carter and William W. Streilein
MIT Lincoln Laboratory
244 Wood Street
Lexington, MA 02420, USA
{benjamin.priest, era, neal.wagner, brady.tello, kevin.carter, wws}@ll.mit.edu

ABSTRACT
Moving target (MT) technologies seek to protect cyber systems by making them less homogenous, less static, and less deterministic in order to increase the complexity required for a successful cyber attack. While such technologies provide a promising avenue for defense, they are often associated with significant performance costs. Therefore, it is necessary to investigate the tradeoffs between the security and performance of a candidate MT technique before deploying it in a live setting. However, testing the effectiveness and usage costs of an MT technique on a live network is a costly and difficult process. This paper presents an agent-based approach for simulating an operational network and measuring the impact of security policies that incorporate MT technologies. It captures a computer network with mission-critical and non-mission-critical users and traffic, cyber attacks sent by malicious actors, and an MT technique to be evaluated. The agent-based system produces metrics that measure the effects of attacks and security policies on overall and mission-specific network operations and allows for the evaluation of multiple candidate MT policies without having to test these policies on a live network. Furthermore, it serves to guide the search for optimal policies that maximize the protection provided by the MT technology, while minimizing its performance overhead. We demonstrate the model via a case study that evaluates a particular MT technology and provides decision support for setting the optimal policy associated with its deployment.

Categories and Subject Descriptors
K.6.5 [Management of Computing and Information Systems]: Security and Protection; I.6.8 [Simulation and Modeling]: Types of Simulation—Monte Carlo

General Terms
Security, Simulation, Measurement, Experimentation

Keywords
Agent-based modeling, moving target, security metrics

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...$15.00.

1. INTRODUCTION
Commercial and governmental entities world-wide have long since accepted a net-centric model for storing, organizing, and accessing internal information and fostering intra- and inter-organizational communications. Practically all entities operating in both the public and private sectors, including banking, finance, public utilities, manufacturing, law enforcement, and medical services depend upon secure computer networks in order to continue to function properly. These agencies are under constant threat of cyber attack by malicious actors seeking to steal information, disrupt operations, or establish a quiet foothold on the network. Indeed, recently documented attacks against the International Monetary Fund, Lockheed Martin, Citibank, Google, and Sony underlie the severity of the problem [12]. Due to the sheer volume of different types of attacks and the ever-changing nature of the attack surface, as yet no silver bullet defensive strategy has presented itself. Thus, network administrators tasked with defending a network must decide which strategies, technologies, and policies are best suited for protecting their particular network from anticipated cyber threats.

Moving target (MT) techniques are a class of security technologies that seek to protect cyber systems by making them less homogenous, less static, and less deterministic in order to increase the complexity required for a successful cyber attack. However, these gains do not come for free, as MT techniques are associated with performance costs [24]. Thus, it is necessary to investigate the tradeoffs between security and performance of a MT technique before deploying it in a live setting.

When deploying a security technology designed to mitigate a particular type of malicious activity, a network administrator must make a number of critical decisions, such as on which hosts to deploy the technology, how to configure the deployments, and other decisions dependent on the nature of the technology. This collection of decisions is called a security policy. The particular choice of policy may greatly affect both the network’s security posture and its operating overhead.

Policy decisions for even one security technology are often complicated, and may have unintended adverse consequences for the network. However, the robust evaluation of different policies is usually costly and impractical. DARPA and others use large-scale network emulation experiments to select the best security policies regarding their infrastructure. These “cyber ranges” typically involve hundreds or thousands of machines and dozens of operators [26]. While very useful for gathering information and assessing security policies, operating a cyber range is much too costly for all but the most large and security-focused of organizations.

We propose an agent-based modeling method for evaluating MT techniques for multiple relevant attack and defense scenarios with
significantly reduced infrastructure, manpower, and time-costs. Agent-based models focus on the interaction of computational models of autonomous agents with each other and their environment. System behavior in such models is decentralized, where emergent phenomena are dependent solely upon low-level agent behaviors. Consequently, agent-based modeling and simulation is a preferred tool in the study of complex systems [5]. Our method is intended to capture interactions between attackers, defensive systems, benign hosts, and network operations and serves to economically evaluate multiple MT policy configurations and their effects on the security posture and overhead of the network. Furthermore, we utilize Monte Carlo experiments to collect metrics that support the evaluation of the different policies.

In this paper, we present evidence that this agent-based modeling system can provide decision support for the deployment of MT technologies by way of a case study evaluating a particular MT capability. This case study evaluates a technology that utilizes a multi-compiler technique (MCP) to defend against attacks aimed at a particular piece of software. Multi-compiler techniques generate new and diverse versions of a software application in order to increase the complexity faced by an attacker that seeks to leverage knowledge of an application’s binary structure to build exploits. The technique is intended to detect when an attacker attempts to exfiltrate information about the application binary in use on the system. When it detects an information leak in an outgoing communication, it uses a multi-compiler to create a new binary for the software. This new binary presents a different attack surface that is intended to mitigate an attack predicated on information leaked about the old binary.

While the technique can mitigate attacks, its operation incurs a significant computational overhead, so deploying it on every machine and setting it issue a new binary after every communication with the outside world is not feasible. Thus, an optimal deployment and usage policy must be established. We utilize an agent-based simulation environment to model MCP, including its associated usage costs, on a simulated network and evaluate its performance for different policy settings. These policies are parametrized using data-driven methods to estimate the overhead associated with a real implementation of MCP. The main contribution of this paper is to provide an agent-based simulation approach to support optimal deployment of MT technology at the network scale and to demonstrate this approach via a case study that evaluates a particular multi-compiler MT technique and recommends optimal policies for its deployment.

The paper proceeds as follows: Section 2 provides a description of the related work in network security modeling and simulation. Section 3 describes the MCP technique being modeled. Section 4 describes the network simulation model, Section 5 describes security and performance metrics generated by the model, and finally, Sections 6 and 7 demonstrate the model via a case study and provide analysis and conclusions, respectively.

2. RELATED WORK

There exist numerous recent studies investigating various models appropriate for network intrusion detection. Some recent examples include [2, 3, 4, 7, 9, 10, 14, 13, 21, 23, 28, 27, 29, 30]. While several of these studies employ simulation to explore the efficacy of proposed models for intrusion detection, the focus of these studies is on network situational awareness rather than network simulation.

[1] and [18] provide agent-based simulation studies that are focused on investigative network modeling rather than network situational awareness. [1] employs agent-based simulation to analyze a modeled network with respect to a specific network security protocol in order to automatically determine protocol compliance. [18] provides an agent-based model that integrates the OMNet++ network simulation software package with network libraries INET Framework and ReaSE to investigate cooperative botnet attacks and corresponding defenses.


MT defense is a broad area of research that covers cyber defenses across many computing domains. MT defenses protect cyber systems by introducing diversity and dynamism, increasing the complexity of the attack surface and thereby thwarting attacks. MT defenses can be categorized into five categories: dynamic runtime environments, dynamic software, dynamic data, dynamic platforms, and dynamic networks [15].

Dynamic software is an MT defense that dynamically changes application code to thwart attacks that rely on the details of a specific binary. Automated software diversity and multi-compilers are dynamic software techniques that introduce diversity in compiled binaries at compilation time to combat software monoculture. [22] uses a multi-compiler to distribute diverse binaries across a network to reduce the number of machines that a single attack can compromise. [23] mitigates buffer overflows by maintaining diverse binaries for the same application, which vote on each incoming request.

3. MT MANAGEMENT TECHNIQUE

In this paper, we focus on a particular hypothetical dynamic software MT technique that uses a multi-compiler to dynamically introduce software diversity to thwart attacks. We build an agent-based simulation system to evaluate the technology and its feasibility at a network scale. In so doing, we develop strategies for deployment prior to investing time and energy into implementing the technology.

We refer to the MT technique, introduced in Section 1, as MCP. MCP is a hypothetical management system for maintaining software diversity that determines when the managed machine should generate a new binary for a critical application and swap it out for the current one. MCP is similar to multi-compiler techniques described in the literature, such as in [22] and [25]. However, it differs from the aforementioned studies by dynamically swapping between multiple different binaries for its managed application on a single device. Specifically, MCP specifies that hosts should deploy a new binary when it believes that an attacker could have obtained information about the aforementioned binary. The MCP technique was selected as the case study for this paper because it is being evaluated as a part of a larger project which aims to develop and build new multi-compiler technology prototypes.

An attacker can discover and exfiltrate binary details by utilizing an exploit that causes the targeted device to send communications to the attacker that leak information about the binary. Thus, to maintain perfect defensive posture, the defender would need to swap binaries every time the device sends an outgoing communication. In practice this approach may not be feasible as generating and swapping binaries introduces a significant computational overhead. Hence, we need a management strategy for MCP that determines
when and how to execute binary swaps.

It is possible to detect an exfiltration of a known binary by searching outgoing communications for matching byte sequences. Hence, MCP is a system that intercepts outgoing communications and searches them for binary leaks. If one is found, MCP instructs the host to swap to a new binary. The specific costs of running MCP will depend upon how it is implemented and how it is configured for deployment. For this paper, we assume a particular implementation of MCP that is associated with a constant cost both for searching an outgoing communication and for swapping binaries once a leak is detected.

However, for even the most efficient search algorithm, searching all outgoing communications will also introduce significant overhead on the system. Therefore, searching every outgoing communication may not be advisable. This paper examines how to determine the best search settings for deploying MCP before investing in the process of implementing it on a network scale. As discussed in Section 1, the approach utilizes an agent-based simulation model to model MCP and relevant attacks at the network scale and investigate multiple MCP settings and attack scenarios in order to guide the search for optimal settings appropriate for MCP deployment on a live network.

4. NETWORK MODEL

Our agent-based model is concerned with network-level activity and uses an abstract representation of network devices that focuses only on internal device details that are relevant to MCP and attacks associated with such devices. In our model, there are two domains in which agents can exist: the internal and external domains. The internal domain represents the network to be modeled, while the external domain represents the external internet. Three principal types of agents are included: users, attackers, and defenders. We will refer to the settings defining the users and attackers as the scenario. A scenario is a model of a real network and its interaction with the outside world. Additionally, we will refer to the settings defining the defenders as the security policy, or more succinctly the policy. The focus of this paper will be to decide how best to set this policy based upon its impact on the network for a given simulated scenario.

User and attacker agents can communicate with one another, while the defender can monitor these communications and take appropriate action. Communications are abstract representations of a packet stream, containing a source and a destination, as well as notional data. Attackers can send a communication that contains an exploit to a defender, while benign users send communications that may leak details about their application binary to the attacker.

We use the term “scenario” to encapsulate the parameters that define the environment in which users and attackers interact. We separate the scenario into three components, the benign ($S_B$), mission ($S_M$), and attacker ($S_A$) scenarios. We assume a constant network latency, $t_c$, as part of the scenario definition. The whole scenario can be described with the tuple

$$S = (S_B, S_M, S_A, t_c).$$  \hspace{1cm} (1)

Meanwhile, as stated above, we use the term “policy”, denoted $P$, to refer to the parameters that define defender behavior. An experiment, $E$, is described completely by a scenario-policy pair.

For the purposes of the model, agents operate in a discrete timing environment where actions take a certain number of time units to complete. Parameters for the scenario and policy that measure time are expressed in these time units.

4.1 Benign Scenario

Users are benign agents that model a particular machine that is connected to the internet and exist in both the internal and external networks. Internal users are the users who work on the enterprise or “home” network to be protected by the defender. External users are benign entities that exist outside of the network that model machines not protected by the defender. In either case, a user’s primary function is to send and receive communications, generating traffic on the home network. Internal and external users can send, receive, and reply to communications, as depicted in Figure 1.

A benign scenario will involve $N_I$ internal users and $N_E$ external users. These users draw communication inter-emission times from an exponential distribution with a fixed parameter $\lambda_u$. Users also respond immediately to incoming benign communications with probability $P_R$. The benign scenario may be described by the tuple

$$S_B = (N_I, N_E, \lambda_u, P_R).$$  \hspace{1cm} (2)

4.2 Mission Scenario

In addition to background communication traffic, some internal users execute specific, inter-dependent, and timely tasks. We call this series of tasks the mission. Missions model a critical task chain, whose execution is the main purpose for the network. Missions are time-sensitive, therefore any delay incurred to a mission’s completion is undesired. Users who execute the mission reside in the home network.

We implemented the specific mission depicted in Figure 2. The mission involves three internal users and three database servers that exist on six different home network devices. Mission User (MU) 1 checks Database 1 for outstanding task requests. If found, it will perform the task and push the completed task to Database 2. MU 2 will pull down this completed task from Database 2 and operate on it. When finished, MU 2 will communicate the completed task to MU 3, who will finish the task and push it to Database 3, which stores the finished mission.

The home network includes $N_M$ mission teams, meaning that there are $6 \times N_M$ mission devices. Mission users require a fixed amount of uninterrupted time, $t_M$, to perform their tasks before they can communicate with the next mission user or database in the chain. Normal operations, such as sending and receiving benign traffic can occur during this time, but suffering or mitigating a compromise will delay the mission. Additionally, mission users that check for mission updates do so with a period of $t_C$. The mission scenario is given by a tuple

![Figure 1: User layout](image)


\[ S_M = (N_M, t_M, t_C). \]  

(3)

### 4.3 Attacker Scenario

Attackers reside in the external domain. For the purposes of this paper we assume that they are capable of initiating binary disclosure campaigns, which allow them to gain a foothold on a targeted home network device as noted by the procedure outlined in Figure 3. When an attacker campaign succeeds, the target device is compromised. A user on a compromised device cannot perform communications or mission operations until the compromise is cleansed by the defender.

An attacker’s goal is to disrupt operations on the home network by attacking internal users’ devices. We assume that attackers do not have previous knowledge regarding the home network, which leads to information-seeking campaigns. By sending initial exploit communications to a device, an attacker can gain insight into the software binaries that exist on a home user’s device. The attacker can then craft an attack against one of those binaries. Successfully transmitting this attack will allow the attacker to compromise the targeted device and disrupt its operations, unless the binary has already been swapped with a new binary by the defender.

The attacker scenario involves \( N_A \) attackers. These attackers each initiate a new binary disclosure campaign against a randomly selected target from the internal users, which has an exponential distribution with a fixed parameter \( \lambda_a \). The attacker scenario may be described by the tuple

\[ S_A = (N_A, \lambda_a). \]  

(4)

### 4.4 Defender Policy

A defender is a model of the security policy applied to an internal user. The defender has three capabilities: cleansing the user’s device when it is compromised, inspecting outgoing communications for binary information leaks, and swapping the user device’s software binary when it detects a leak in an outgoing communication. The latter two capabilities are used to model MCP. Each defender capability has an associated time cost.

The MCP defense model is depicted in Figure 4. The attacker sends an initial attack communication to the user, which causes the user to respond with a communication, leaking information about the user’s binary. MCP intercepts this outgoing communication and must make a decision based on its policy configuration to either inspect the communication for a leak or let it pass through uninspected. If MCP does not decide to inspect, then the leak reaches the attacker who can then utilize this information to create and send a second attack communication that will compromise the user’s device. If MCP decides to inspect and consequently detects the leak, it then generates a new version of the binary that replaces the user’s current version. It then allows the communication to continue to its destination (the attacker) unchanged. The attacker, upon receiving the communication containing the leak will create and send the second communication that is intended to compromise the user’s device but will be unsuccessful as the user’s binary has been swapped for a new binary and the leaked information is no longer applicable.

The MCP defense policy defines defenders for internal users that inspect an outgoing communication with probability \( P_I \). Depending upon the implementation of the inspection capability, we assume that the defender inspection detects binary information leaked in an outgoing communication with probability \( P_D \) and has a probability of falsely detecting a binary leak in a benign communication, \( P_{FP} \).

Inspection of an outgoing communication takes some amount of time, \( t_S \). If a leak is detected, real or not, the defender will proceed to swap the user’s binary, which delays the user’s tasks for a fixed amount of time, \( t_B \).

In a real system, \( P_D \), \( P_{FP} \), and \( t_S \) will not be independent: for any given algorithm looking for binary leaks, increasing \( P_D \) and decreasing \( P_{FP} \) will increase computational overhead, or \( t_S \). \( t_B \) on the other hand is the cost to create and swap the existing binary, which is independent of the search algorithm used for inspection.

If a binary disclosure attack is not detected by the defender and the user becomes compromised, the defender can cleanse the user, which also delays the user’s tasks for a fixed duration, \( t_E \). In practice, \( t_E \) will be much greater than \( t_B \), since the time to perform a binary swap will in general be much less than the time needed to restore a compromised machine to a safe state. If possible, \( t_S \), \( t_B \), and \( t_E \) should be set relative to \( t_i, t_C, t_M \) using data-driven methods that utilize ground truth data in order to allow the simulation model to accurately capture MCP execution as it would occur in a real setting. We describe a policy, \( P \) as a tuple of parameters of the given form

\[ P = (P_I, P_D, P_{FP}, t_S, t_B, t_E). \]  

(5)

### 5. SIMULATION METRICS

As discussed in Section 4.3, our attack model assumes that the attacker’s goal is to disrupt network operations. Thus, attacker activity is intended to negatively impact the target network’s security posture by compromising machines. A consequence of this behav-
ior is that the attacker can reduce the network’s performance by forcing the network to devote resources to fight it.

A decision support process will be of limited utility unless it produces results that are both measurable and actionable. We must be able to compare different policy simulations to select those with preferable qualities. As previously discussed, network users can be in a number of different states, depending on their interactions with the network. For instance, at any given point, users are either compromised, undergoing defensive mitigations, or free to perform normal tasks. We developed an experimental methodology allowing us to monitor various metrics, accounting for different features of users’ state over the course of the experiment. We gauge the effectiveness of a given defensive policy by examining the tradeoffs between metrics measuring the performance, the impact on the network’s ability to function efficiently, and those measuring the network’s security posture, the defender’s ability to mitigate attacks on the network.

5.1 Performance metrics

As described in the previous section, internal users can experience downtime either because an attacker has compromised their machine, which must be cleansed, or because MCP is swapping the binary on their machine after detecting an outgoing binary leak. In either case, the user is unable to perform tasks, decreasing productivity.

In a particular simulation run, the home network has $N$ users and the simulation is executed for some total amount of time given by $t$ time units. A particular user, $i$, may or may not experience some amount of overhead over the course of the simulation. We can independently measure the overhead due to attacker exploits, $d_{E}^{(i)}$, and the overhead due to policy, $d_{P}^{(i)}$, and state the total overhead for a user $i$ as $d_{E}^{(i)} + d_{P}^{(i)}$, due to the mutual exclusivity of the two overheads. We can normalize the effective overall impact by taking the complement of $d_{E}^{(i)} + d_{P}^{(i)}$ for user $i$ and dividing it by $t$, the total time for the simulation, which gives us an index between 0 and 1. A value near zero means that the user is heavily impacted, a loss for the policy, while a value closer to 1 means that the user is virtually unaffected, a win for the policy. We can then average these values across all users. We formalize this system performance index metric as follows:

$$\text{System Performance Index} = \frac{\sum_{i=1}^{N} t - (d_{E}^{(i)} + d_{P}^{(i)})}{N}. \quad (6)$$

As discussed in Section 4.1, some users execute a multi-user mission and each mission user is relevant only at specific stages of the mission. Thus, it would not be sufficient to simply average the user impacts defined above for mission users to determine the overall impact to the mission.

To illustrate this concept consider the following example. Mission user MU 1 from Figure 2 holds the responsibility of handling the first stage of the modeled mission. If this user’s device is attacked, either successfully or unsuccessfully, while the user is executing its mission, then the mission is affected. However, if the same device is attacked after the user has finished his task on the mission, then that mission is not affected even though the user’s device is impacted by the attack. Therefore, a calculation that simply averages the impacts for mission users over the course of a simulation run leads to an inaccurate measure of overall mission impact.

In order to measure the specific impact on a mission, we need to introduce different measures. For each instance of the mission, $j$, we record $d_{E}^{(j)}$ and $d_{P}^{(j)}$, the amounts of attacker and policy downtime, respectively, experienced by any of the mission users while they are in the process of working on a mission-related task. If the ideal length of an unimpeded mission is $l$, and the observed length of the mission is $t^{(j)}$, then $t^{(j)} = l + d_{E}^{(j)} + d_{P}^{(j)} + \epsilon$, where $\epsilon$ is a small noise parameter introduced by mission users’ need to wait for updates to mission databases. $\epsilon$ is therefore a function of $t_{C}$, the mission server check delay, $t_{t}$, the network latency, and a random seed that is used to generate stochasticity in the experiment.

We can normalize this mission metric in the same manner as the system metric by taking the complement of the downtime with respect to the total runtime of the mission. We report the average for the $M$ total missions run over the course of the experiment. The mission performance metric is thus given by

$$\text{Mission Performance Index} = \frac{\sum_{j=1}^{M} t^{(j)} - (d_{E}^{(j)} + d_{P}^{(j)})}{M}. \quad (7)$$

Additionally, the actual mission delays due to attack ($d_{E}^{(j)}$), defense ($d_{P}^{(j)}$), and both attack and defense ($d^{(j)} = d_{E}^{(j)} + d_{P}^{(j)}$) are interesting as well. This is because there is an ideal length of an unimpeded mission, $t = 5 * t_{t} + 3 * t_{M}$, where $t_{t}$ is the network latency and $t_{M}$ is the time required for a mission actor to perform its task. In execution, even without any attacks, this time will be perturbed by a small noise parameter $\epsilon$ resulting from mission users’ need to wait for updates to mission databases. Thus, delay for any mission $j$ is $t^{(j)} = t^{(j)} - (t + \epsilon)$. Since $\epsilon$ will tend to be very small, this means that we can compare $d_{E}^{(j)}$ and $d_{P}^{(j)}$ for any missions $j$ and $k$, even if they are from different experiments with different total simulation runtimes, so long as they have the same mission scenario and network latency $t_{t}$. Thus, we also track statistics of the total mission delay as a metric of performance.

5.2 Security metrics
As described in section 4.4, a defender can either thwart an attack or fail to prevent it, resulting in costly clean up. An obvious metric that evaluates the security posture of each user is the user’s total number of compromises by an attacker campaign. Of course, this measure is not meaningful unless it is expressed either as a rate, i.e. the number of compromises per unit time, or as a ratio of compromises to the total number of attacks due to the differences in total simulation runtime across different experiments. Since we evaluate our model in contexts that can have varying attacker activity levels, we select the latter option.

If we run an $N$-user simulation where user $i$ is attacked $a^{(i)}$ times, resulting in $c^{(i)}$ successful compromises, we can report a normalized security index $\hat{S}^{(i)}$ for user $i$. This index has the same form as the impact metrics described in the previous section. We can apply an identical chain of reasoning as detailed in the same form as the impact metrics described in the previous section. For the purpose of making decisions for an existing network, this system. Furthermore, we present a use-case for making policy decisions using the system.

As described in section 4.4, in order to make policy decisions using an agent-based modeling approach, we must choose a scenario. For the purpose of making decisions for an existing network, this scenario should match the activity on the network as closely as possible because the effects of the policy will be heavily dependent upon the scenario to which it is applied. For instance, a scenario that models a relatively inactive attacker will have a different optimal policy setting than one that models a relatively active attacker. Furthermore, the proportions of the time costs for various actions will affect the metrics collected during the experiment. For instance, the ratio of settings for $t_E$, the time cost of being compromised, and $t_B$, the time cost of cycling binaries, affect the performance indexes of equations 6 and 7 significantly. A relatively high $t_B$ will increase the cost of swapping binaries, which will in turn affect performance index computations associated with high values of $P_I$.

We performed simulations using commodity hardware to calibrate relative unit time values for all time cost parameters of the simulation model, with the exception of mission time cost parameters $t_M$ and $t_C$. After calibration of the non-mission time cost parameters, these parameter values were used to set the mission time cost parameters. $t_M$ was set to be 40% larger than $t_P$, the time cost associated with cleansing a compromised device, and $t_C$ was set to be on the same order as $2 \times t_L$, the round-trip time for a communication. We give the values specified in the scenario description below.

We executed several simulated experiments using the model described in Section 4. All experiments are executed using the same benign scenario. The benign scenario is given by the tuple $S_B = \langle 250, 100, 10, 0.5 \rangle$, which specifies 250 internal and 100 external users. Benign users issue new communications probabilistically following a Poisson process [17] with an exponential inter-arrival time with a mean of 10 time units. Upon receiving a benign communication, a benign user issues a reply communication to the source of the initial communication with probability of 0.5.

The mission scenario used for all experiments is given by the tuple $S_M = \langle 40, 5000, 30 \rangle$ which specifies 40 mission groups operating in parallel inside the home network. These mission groups are mutually exclusive, so a user can be involved with at most one mission. Thus, 240 of the 250 internal benign devices are involved in a mission group as either a mission user or a mission database. Mission users require 5000 time units to perform their work before they can initiate the next step in the mission chain. If a mission user is interrupted by an attack or defensive mitigation, it picks up where it left off once its defender has recovered it to a safe state. Thus, if a user’s device is repeatedly compromised or is continuously mitigating attacks, the user may not complete its mission task. Additionally, mission users that wait for updates to mission databases check these databases every 30 time units.

We explore several different attacker scenarios in our experiments. Table 1 summarizes the parameterizations of these attacker scenarios. We used attacker scenario $S_{A_1}$ in our main experiment described below. In this scenario, there are three attackers. These attackers initiate new attack campaigns against a random internal benign user as a Poisson process with an exponential inter-arrival time with a mean of 15 time units. For the scenario $S$ defined by $S_{A_1}$ and the above benign and mission scenarios, the simulation parameter settings used imply that attacker traffic accounts for $\sim 0.12\%$ of the total communication traffic observed by a single home network device.

A policy is a tuple $P = \langle P_I, P_D, P_{FP}, t_S, t_B, t_E \rangle$, as described in Section 4. The probability of inspection, $P_I$, is a configurable feature of the policy, but will have dramatic effects on a simulated scenario, since it determines how often the defender operates. We examine policy tuple $P = \langle P_I, 1, 0.1, 4, 30, 3000 \rangle$, and explore different values for $P_I$ varying $P_I$ between 0.1 and 1.0. As dis-

<table>
<thead>
<tr>
<th>Scenario</th>
<th>$N_A$</th>
<th>$\lambda_a$</th>
<th>Attack Traffic Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_{A_1}$</td>
<td>3</td>
<td>15</td>
<td>~0.12%</td>
</tr>
<tr>
<td>$S_{A_2}$</td>
<td>6</td>
<td>15</td>
<td>~0.18%</td>
</tr>
<tr>
<td>$S_{A_3}$</td>
<td>12</td>
<td>15</td>
<td>~0.29%</td>
</tr>
<tr>
<td>$S_{A_4}$</td>
<td>1</td>
<td>30</td>
<td>~0.06%</td>
</tr>
</tbody>
</table>

Table 1: Attacker Scenarios

$P_D$ and $P_{FP}$. We implemented and tested a Bloom filter search algorithm to calibrate $t_S$. A Bloom filter is a probabilistic data structure that can be used to test whether an element is part of a set [6]. Here, a Bloom filter is used to search outgoing communications and detect if they include any leaked information pertaining to a known application binary. This type of search does not allow the possibility of a false negative but does allow false positives. A particular configuration for a Bloom filter specifies the filter’s false positive rate, carrying with it an associated search cost. Bloom filters configured for lower false positive rates have higher associated search costs relative to those configured for higher false positive rates. We used a Bloom filter configured with a false positive rate, $P_{FP}$, of 0.1 to calibrate $t_S$ for all experiments described in this paper. As Bloom filter search does not allow false negatives, $P_D$ is set to 1 for all experiments as well.
discussed above, we calibrated $t_S = 4$ based upon our choices of $P_B = 1$ and $P_{FP} = 0.1$ and with respect to the rest of the time cost parameters in the scenario. We tuned $t_B = 30$ and $t_E$ similarly. Our full scenario description is given by $S = \langle S_B, S_A, S_M, 10 \rangle$, setting network latency to 10 time units.

The experiments below are the results of Monte Carlo simulations, each of which were run for 200 iterations. Here, an iteration is defined as the completion of a single mission cycle from start to finish. We evaluate the simulations utilizing the metrics described in Section 5. Figures 5—8 give experimental results for attacker scenario $S_A$ for ten values of $P_I$ between 0.1 and 1.0 varied in increments of 0.1.

Figure 5 plots results for the system security index versus the system performance index. In the figure, plotted points are labeled by the values of $P_I$ used in the experiment that generated the resultant point. The plot displayed here provides insights into the tradeoffs between performance and security for all of values of $P_I$ evaluated. Notably, there is an turning point at the point associated with $P_I = 0.8$ at which the performance index starts to decrease as $P_I$ increases. This indicates that past this point, increasing MCP’s probability of inspection actually results in more aggregate computational overhead than it mitigates downtime due to compromise. This result is critical for network administrators responsible for initiating policy. He or she needs to take into account how tolerant the enterprise is to compromise when making the decision to set $P_I$.

In Figure 5 plotted points representing runs with $P_I = 0.8$, 0.9, and 1.0 are traced by a line representing the Pareto Efficient Frontier or Pareto front for the two objectives of system security and system performance. The plotted points associated with these values of $P_I$ dominate all other plotted points, that is they contain values for one objective (either security or performance) that cannot be improved upon without causing a corresponding degradation of the other objective [11]. Thus, while it may be best from a system performance perspective to select $P_I = 0.8$ in order maximize performance, an organization may prefer to sacrifice some level of performance to increase system security to some minimum level. It is here that the security versus performance values associated with the Pareto front can provide support for the decision-making process. For this experiment from a system-level perspective $P_I = 0.9$ and 1.0 may also be considered as they both improve upon the security observed for $P_I = 0.8$ albeit while sacrificing some level of system performance.

However, system-level metrics do not tell the whole story. If the mission is a critical component, then we must also take the effects on the mission into account. Figure 6 plots the mission security index versus the mission performance index. The plot displayed in this graph is very similar to the one in Figure 5. There is a similar turning point where the performance index begins decreasing for larger values of $P_I$, this time at the point associated with $P_I = 0.4$. In the figure, the Pareto front is traced by a line connecting points associated with $P_I = 0.4$, 0.5, 0.8, and 1.0 with respect to the two objectives of mission security and mission performance. Thus, these settings for $P_I$ dominate all other $P_I$ settings and may be used to support policy decisions made by network administrators. As can be seen in the figure, $P_I = 0.4$ is the optimal setting from a mission performance perspective but the other three settings for $P_I$ that make up the remainder of the Pareto optimal set may be selected as well depending on the organizational tolerance to compromise of the mission.

Notably, the curve in Figure 6 contains more jitter than does Figure 5. Specifically, the Pareto front in Figure 6 excludes the points where $P_I = 0.6$, 0.7, and 0.9. There is no a priori reason why the performance and security indices should not be monotonic past the inflection point at $P_I = 0.4$. This phenomenon can be explained by the fact that the attack surface of the mission shifts over the mission’s duration, as it moves between stages and different actors become relevant. The discussion of the mission-oriented metrics in Section 5.1 and the mission description in Section 4.2 give more details about how the mission’s attack surface shifts as it executes.

Although the results in Figure 6 are jittery, the broad result that performance suffers for values of $P_I$, the probability of inspection, above 0.4 is a viable conclusion from the plot. A mission’s shifting attack surface provides a rudimentary level of moving target defense prior to any further defenses. Thus, it makes sense that missions would be more sensitive to passive overhead introduced by MCP than stationary devices, validating the different Pareto fronts observed in Figures 5 and 6.

The results described above for security versus performance from the mission perspective are corroborated by the results in Figure 7. This figure plots the mean total mission delay ($d_I$), as well as the mean mission delay due to MCP overhead ($d_{FP}$) and the mean mission delay due to attacker compromise ($d_E$). Note that $d_P$ increases as $d_E$ decreases, but that the relationship is not linear. While $d_E$ decreases significantly up to $P_I = 0.4$, its rate of decrease slows for greater values. Meanwhile, $d_P$ increases linearly as a function of $P_I$ as would be anticipated, since it is function of $P_{FP}$, $P_I$, $t_C$, $t_B$, and the total amount of traffic passing through the user. $P_{FP}$’s being greater than zero means that as $P_I$ increases, a greater por-
tion of benign traffic will be misidentified as binary leaks, meaning that mitigation overhead will increase. This means that for values of $P_I$ greater than 0.4, $\hat{d}$ actually increases albeit at a slower rate than it decreases for values of $P_I$ ranging from 0.1 to 0.4.

Figure 8 gives detailed statistics of the delay encountered by missions under each of the policies under review. It superimposes the $\hat{d}$ curve from Figure 7 onto box and whisker plots displaying the distribution of mission delays for each value of $P_I$. It is interesting to note that the box and whisker plots compress for increasing values of $P_I$. This is because random effects becomes less relevant as $P_I$ increases. Even for missions that are relatively undefended, it is possible that due to random chance they may not be targeted during their execution. Additionally, missions with few defenses are more likely than missions with many defenses to be compromised many times over the course of their execution. So, as $P_I$ increases, the certain cost of defense increases while the uncertain cost of attacks decreases, leading to the convergence of the box and whisker plots that we observe in the figure.

So, what is the recommendation for this system? For an attacker who is executing availability attacks, $P_I = 0.4$ is the optimal policy for MCP in terms of mission performance. Further increases to $P_I$ will increase the attack mitigation rate but will also increase the mitigation overhead at a faster rate. Although $P_I$ to 0.4 allows for optimal mission execution, other settings for $P_I$ might be preferred depending on an organization’s perspective on cyber security and its tolerance for security risk. Additionally, the two perspectives (system and mission) discussed above may be weighted depending on the value placed upon each of these by the organization. For some organizations, only the mission is important while for others both the mission and the system are of similar importance. For the scenario examined above, two settings for $P_I$, 0.8 and 1.0, occur in the Pareto optimal set for both system and mission perspectives. For an organization that values both of these perspectives, either of these settings can represent a reasonable tradeoff between both mission and system perspectives as well as between security and performance.

It is important to note that these recommendations are only valid for the given scenario. For a different network environment or a different attacker scenario, the system may recommend a different policy setting. To illustrate this we examine three additional attacker scenarios that are summarized in table 1. These additional scenarios are given by $S_{A_2}$, $S_{A_3}$, and $S_{A_4}$ in the table. Note that $S_{A_2}$ and $S_{A_3}$ model more active attackers than $S_{A_1}$, as can be seen in the table by their greater attack density (i.e. their greater ratio of attacker traffic to total traffic) while $S_{A_4}$ models a much less active attacker than $S_{A_1}$.

Figure 9 plots the average mission delay for additional scenarios $S_{A_2}$ and $S_{A_3}$ as well as the first scenario, $S_{A_1}$. Note that while $S_{A_1}$ has a turning point in its curve where the mean mission delay begins to increase, the mean mission delay curves for $S_{A_2}$ and $S_{A_3}$ are consistently decreasing as $P_I$ increases. This indicates that, although these policies do incur more mitigation overhead as $P_I$ increases as described above, this increase is outweighed by the decrease in overhead due to compromise.

Finally, we consider a scenario with a relatively inactive attacker, $S_{A_4}$. As given in table 1, $S_{A_4}$ models an attacker that generates approximately 0.06% of the total traffic seen on the network. For this experiment we examine settings for $P_I$ that start at 0.0 rather than 0.1 as done for the previous experiments. The reason for this change is due to the relative inactivity of the attacker: a setting of $P_I = 0.0$ effectively turns off MCP defense which, for higher levels of attack, translates to significant mission delay due to compromise that essentially prevents any missions from completing. However, for lower levels of attack, missions may still complete even when no defense is present.

Figure 10 plots the mean mission delay of the corresponding experiment investigating $S_{A_4}$. The resulting curve indicates that the delay due to the attacker is very quickly outweighed by the overhead incurred by the policy. From the figure, mission delay is min-
The policy to setting is actually more costly than simply suffering the occasional delay. Meanwhile, the attacker model stresses the importance of setting the scenario to match the real network and the real threat as closely as possible. For an attacker modeled by $S_A$, we concluded that the ideal policy setting from a mission performance perspective is $P_t = 0.4$ while other reasonable policy settings that balance between security and performance and mission and system perspectives are $P_t = 0.8$ and $1.0$. However, the attacker models $S_A$ and $S_A^I$ suggest a policy setting of $P_t = 1.0$. In this case, when the network is under relatively heavy attack, it may be tempting to swap binaries every time a user communicates with the outside world rather than search for binary leaks in outgoing communications as suggested by the MCP defense. However, a simulation modeling this policy would result in missions that never complete due to the persistent downtime incurred by the normal operations of a highly active (perhaps overactive) defense. Thus, MCP remains a necessary technology for managing this binary-swapping mitigation, even when under heavy attack. Meanwhile, the attacker model $S_A^I$ represents so little threat that deploying MCP with anything other than a very low inspection probability causes more delay. So overhead due to running MCP at anything higher than a minimal $P_t$ setting is actually more detrimental to the network and the mission than simply allowing the occasional availability attack, especially since such attacks occur infrequently. Additionally, turning off MCP completely (mission delay associated with $P_t = 0$ in the figure) results in a rather steep rise in mission delay. This indicates that the $S_A^I$ scenario, despite its relatively low attacker activity level, is still active enough to warrant deployment of the MCP defense, albeit at a minimal setting.

These important observations emphasize our earlier statement stressing the importance of setting the scenario to match the real network and the real threat as closely as possible. For an attacker modeled by $S_A$, we concluded that the ideal policy setting from a mission performance perspective is $P_t = 0.4$ while other reasonable policy settings that balance between security and performance and mission and system perspectives are $P_t = 0.8$ and $1.0$. However, the attacker models $S_A$ and $S_A^I$ suggest a policy setting of $P_t = 1.0$. In this case, when the network is under relatively heavy attack, it may be tempting to swap binaries every time a user communicates with the outside world rather than search for binary leaks in outgoing communications as suggested by the MCP defense. However, a simulation modeling this policy would result in missions that never complete due to the persistent downtime incurred by the normal operations of a highly active (perhaps overactive) defense. Thus, MCP remains a necessary technology for managing this binary-swapping mitigation, even when under heavy attack. Meanwhile, the attacker model $S_A^I$ represents so little threat that deploying MCP with anything other than a very low inspection setting is actually more costly than simply suffering the occasional compromise which arises quite infrequently. In this case, setting the policy to $P_t = 0.1$ represents a reasonable choice especially from a mission performance perspective.

### 7. CONCLUSIONS

This paper presents an agent-based simulation model for evaluating MT techniques at the network scale. It is intended for use by network administrators to evaluate candidate MT policy configurations for a given network environment at low cost before selecting an appropriate configuration to be deployed. The model captures interactions between attackers, MT defensive systems, and network operations and measures their effects on the security posture and performance overhead of the network.

The model is demonstrated via a case study in which a MCP MT technology is evaluated in a representative network environment for multiple policy configurations and attack scenarios. Experimental results show the relative tradeoffs between security and performance for the various policies investigated and recommend optimal policies for MCP deployment. This case study illustrates the model’s ability to provide decision support for optimal deployment of MT technologies at the network scale without having to incur the relatively large expense of deploying multiple candidate MT policies on a live network.

Planned future work for the system is focused on developing adaptive models for both the attacker and the defender that are intended to capture an attacker’s ability to observe and learn from its interactions with the defender and adapt its attack accordingly and vice versa. Other interesting areas of potential work include the consideration of new attacker models such as an attacker whose goal is to use stealthy techniques to persist and/or to exfiltrate critical network data and the consideration of different implementations for the search algorithm underlying MCP’s inspection of outgoing communications for binary leaks.

### 8. ACKNOWLEDGEMENTS

This work is sponsored by the Department of Defense under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the authors and are not necessarily endorsed by the United States Government.

### 9. REFERENCES

Intelligence and Computational Intelligence. Third International Conference (AICI 2011), 2011.


