Belief-Space Planning for Automated Malware Defense

Justin Svegliato, Sam Witty, Amir Houmansadr, Shlomo Zilberstein
College of Information and Computer Sciences
University of Massachusetts Amherst
{jsvegliato,switty,amir,shlomo}@cs.umass.edu

Abstract

Malware detection and response is critical to ensuring information security across a wide range of devices. There have been few attempts, however, to develop security systems that exploit the benefits of different malware detection techniques. We formally introduce an automated malware defense framework and represent it as a belief-space planning problem that optimally reduces the impact on the performance of a system. Using the framework, we then provide an example automated malware defense system for email worm detection and response. Finally, we show in simulation that the system outperforms standard security techniques that have been used in practice. The result is a novel belief-space planning approach to automated malware defense designed for robust, accurate, and efficient use in large networks of resource-constrained devices.

1 Introduction

There has been substantial work in developing malware detection techniques that detect malicious files in various contexts, such as malware injected into compromised websites [Nelms *et al.*, 2015; Tanaka *et al.*, 2017] or malware masqueraded as email attachments [Schechter *et al.*, 2004; Masud *et al.*, 2006]. Previous work has extensively studied malware detection techniques with varying performance and complexity. *Signature-based detection techniques* extract a signature from the file and compare it to the signatures of known malicious files [Faruki *et al.*, 2015; Tuvell and Venugopal, 2017]. Although signature-based detection can reliably detect simple malware, it is easily circumvented by advanced malware that obfuscates its signature through encryption and code polymorphism [You and Yim, 2010; Cesare *et al.*, 2013]. Hence, to detect obfuscated malware, *behavior-based detection techniques* examine the behavior of the file through static and dynamic analysis of its code [Tian *et al.*, 2010; Shafiq and Liu, 2017]. While behavior-based detection can more effectively detect advanced malware, it typically imposes higher computational demands. Deploying malware detection techniques across large networks of resource-constrained devices can therefore be challenging.

Despite tremendous advances in the development of malware detection techniques, there have been few attempts to build security systems that exploit the benefits of both signature-based detection and behavior-based detection. While signature-based detection offers minimal execution time requirements and low system impact, it can only reliably detect simple malware that does not employ obfuscation. On the other hand, although behavior-based detection can consistently recognize obfuscated malware, it suffers from significant execution time requirements and high system impact. As a result, given the limited resources of many devices, using both approaches in tandem with each other results in more robust, accurate, and efficient malware defense. Security systems that exploit both approaches to malware detection can thus better protect the confidentiality, integrity, and availability of private information on resourced-constrained devices.

Recent advancements in automated decision-making offer a particularly promising foundation for developing security systems that can exploit the advantages of both signature-based detection and behavior-based detection. *Partially observable Markov decision processes* (POMDPs) have recently been used in a wide range of decision-making tasks, including semi-autonomous systems [Wray *et al.*, 2016], robotic manipulation and task planning [Ruiken *et al.*, 2016], and autonomous car intersection planning [Wray *et al.*, 2017]. A POMDP provides an intuitive mathematical framework for modeling decision-making problems in which the environment is partially observable and stochastic. Such a framework is an especially good fit for security systems due to several factors. First, a POMDP inherently represents the uncertainty that a security system might have about whether or not a file is malware. Next, due to information gathering, a POMDP naturally models the malware detection techniques available to a security system. Finally, given strong theoretical guarantees, a POMDP optimally reduces the impact of malware detection techniques on the performance of a system. In short, security systems share many properties with problems for which POMDPs have been shown to be effective.

Our primary contributions are: (1) a formal definition of an automated malware defense framework, (2) a representation of the framework as a POMDP, and (3) an example automated malware defense system for email worm detection and response. Finally, we show in simulation that the system outperforms security techniques that have been used in practice.

2 Background

We begin by reviewing the formal definition of a POMDP, a decision-making framework for reasoning in partially observable, stochastic environments [Kaelbling *et al.*, 1998]. A POMDP is represented by a tuple $\langle S, A, T, R, \Omega, O \rangle$ where S is the set of states of the world, A is the set of actions of the agent, $T : S \times A \times S \to [0, 1]$ is the transition function that maps each state $s \in S$ and action $a \in A$ to the probability of ending up in state $s' \in S$, $R : S \times A \times S \to \mathbb{R}$ is the reward function that maps each state $s \in S$ and action $a \in A$ to the expected immediate reward gained in $s' \in S$, Ω is the set of observations experienced by the agent, and $O : S \times A \times \Omega \to [0, 1]$ is the observation function that maps each state $s \in S$ and action $a \in A$ to the probability of observing observation $\omega \in \Omega$.

In a POMDP, the agent does not necessarily know the true state of the world at any given time. Instead, the agent makes noisy observations that reflect its actions and the environment. Thus, to represent its uncertainty, the agent maintains a belief state $b \in B$, a probability distribution over all possible states, where B is the space of all belief states. Initially, the agent begins with an initial belief state $b_0 \in B$. After performing an action $a \in A$ and making an observation $\omega \in \Omega$, the agent updates its current belief state $b \in B$ to a new belief state $b' \in B$ given the belief state update equation

$$b'(s'|b, a, \omega) = \eta O(a, s', \omega) \sum_{s \in S} T(s, a, s') b(s),$$

where η is the normalization constant $\eta = Pr(\omega|b, s)^{-1}$.

At every time step, the agent must select an action based on its current belief state. In general, the behavior of the agent is specified by a policy $\pi : B \to A$ that maps a belief state $b \in B$ to an action $a \in A$. A policy π induces a value function $V^{\pi} : B \to \mathbb{R}$ that represents the expected cumulative reward of each belief state. An optimal policy π^* maximizes this expected cumulative reward. In other words, if the agent were to follow an optimal policy, it would perform actions that maximize its expected future reward based on its current belief state. Note that we omit many solution methods that can be used to find the optimal policy of a POMDP [Sondik, 1971; Littman, 1994; Cassandra *et al.*, 1997; Pineau *et al.*, 2003] in the interest of brevity.

Because the agent must maintain a belief state and select an optimal action at every time step, it can be computationally infeasible to use a policy over the space of belief states. In many problems, however, it is possible to encode the policy as a graph without an explicit representation of the belief state [Kaelbling *et al.*, 1998]. Such a graph is typically referred to as a *policy graph*. A policy graph is a directed graph where each node represents an action and a range of belief states and each edge represents an observation. After the initial belief state of the agent specifies the starting node, the agent follows the policy graph by interleaving two steps. The agent first performs the action indicated by the current node. The agent then makes an observation and transitions to the node associated with that observation. This process repeats until the agent reaches a terminal node in the policy graph. In practice, many solution methods use a policy graph to exploit the benefits of this more compact representation of a policy.

3 Automated Malware Defense

We now present a formal definition of an automated malware defense framework. In general, the framework classifies potentially malicious files that enter the system using a set of *threats*. The framework also performs *response actions* that can appropriately handle normal or malicious files that enter the system. Each response action is associated with a *response cost function* that describes the consequences of mishandling a file. Moreover, the framework executes *detection actions* that can gather information about potentially malicious files that enter the system through a set of *measurements*. Each detection action is associated with a *detection cost function* and an *accuracy profile* that indicates the overhead and effectiveness of that detection action respectively.

An automated malware defense framework addresses all potentially malicious files in a similar way. Suppose a file that may or may not be a threat enters the system. Initially, because the framework does not have any information about the file, it cannot evaluate whether or not that file is a threat beyond its background knowledge. Accordingly, to learn more about the file, the framework first executes a sequence of detection actions of varying costs and accuracies. After enough information about the file has been gathered, the framework then executes one of several response actions. If the framework classifies the file as a threat, it performs a response action that appropriately addresses that threat. On the other hand, if the framework classifies the file as normal, it performs a response action that treats that file normally. Once the framework has performed a response action, it can then handle other files waiting to enter the system.

Formally, an automated malware defense framework is composed of seven attributes. First, the framework has a list of all threats that can be assigned to a file. In the interest of clarity, we assume that a file can only be classified as one threat at any time. However, the framework can be extended to handle a file that may be multiple threats simultaneously with negligible overhead. We define the set of threats below.

Definition 1. *A file that enters the system can be classified by the framework as one of many* ***threats****,* $\Theta = \{\varnothing, \theta_1, \theta_2, \ldots, \theta_n\}$*, where the symbol* $\varnothing$ *denotes that the file is not a threat.*

Second, the framework has a list of all response actions that can be applied to a file. Once the framework has addressed a file using a response action, the file leaves the scope of the system: the framework can no longer apply any actions to that file and transitions to the next file waiting to enter the system. We formalize the set of response actions as follows.

Definition 2. *The system performs one of many* ***response actions****,* $\mathrm{P} = \{\rho_1, \rho_2, \ldots, \rho_n\}$*, that are used to address a file.*

Third, the framework has a function that provides the cost of applying a response action to a threat. This enables the framework to determine how effective a response action is in handling a threat. Intuitively, some response actions may be effective for one type of threat while others may be effective for another type of threat. Typically, if a response action appropriately addresses a threat, it incurs a low cost. Otherwise, the response action incurs a high cost. Note that in practice these costs reflect the constraints and preferences of the system administrators. We describe this function below.

Definition 3. *A* ***response cost function****,* $\Phi : \mathrm{P} \times \Theta \to \mathbb{R}$*, gives the expected cost of performing a response action* $\rho \in \mathrm{P}$ *against a threat* $\theta \in \Theta$*.*

Fourth, the framework has a list of all detection actions that can be applied to a file. When the framework executes a detection action against a file, it gathers information about whether or not the file is a threat. We only consider idempotent detection actions that generate the same assessment of a file at any time. As a result, because each detection action is deterministic, it is not useful to execute any detection action multiple times. We also consider detection actions that are independent of each other. However, we emphasize that the framework can be extended to use a broad range of detection actions. We define the set of detection actions as follows.

Definition 4. *The system performs one of many* ***detection actions****,* $\Delta = \{\delta_1, \delta_2, \ldots, \delta_n\}$*, that are used to gather more information about a file.*

Fifth, the framework has a list of all measurements that can be emitted by a detection action. Every measurement provides some insight about whether or not a file is a threat. For all detection actions, each measurement represents its level of certainty. If the measurement indicates a high certainty, the detection action has a high belief that the file is a threat. Otherwise, the detection action has a low belief. We provide a complete description of the set of measurements below.

Definition 5. *A detection action emits one of many* ***measurements****,* $M = \{\mu_1, \mu_2, \ldots, \mu_n\}$*, that indicate increasing levels of certainty about whether or not a file is a threat.*

Sixth, the framework has a function that provides the cost of applying a detection action to a file. This enables the framework to determine the cost incurred by a detection action. Typically, a detection action with a high cost has a high accuracy while a detection action with a low cost has a low accuracy. In general, the cost of a detection action considers execution time requirements, privacy concerns, power consumption, and other considerations of the system administrators. Broadly, we formalize this function as follows.

Definition 6. *A* ***detection cost function****,* $\Psi : \Delta \to \mathbb{R}$*, gives the expected cost of performing a detection action* $\delta \in \Delta$*.*

Note that the framework would perform all detection actions if none of the detection action incurred a cost.

Seventh, the framework has a function that provides the accuracy of a detection action. This can be viewed as a function that yields a probability distribution over the set of measurements given a detection action and a threat. Intuitively, when the file is a malicious, the detection action has a high probability of generating measurements that indicate a threat. The framework therefore should weight the assessment of each detection action based on its accuracy. We provide a formal description of this function below.

Definition 7. *An* ***accuracy profile****,* $\Lambda : \Theta \times \Delta \times M \to [0, 1]$*, gives the probability of a detection action* $\delta \in \Delta$ *generating a measurement* $\mu \in M$ *for a threat* $\theta \in \Theta$*.*

Given these attributes, it is possible to provide a complete specification of an automated malware defense framework. We provide a formal definition of the framework as follows.

Definition 8. *An* ***automated malware defense framework****,* Σ*, is represented by a tuple* $\Sigma = \langle \Theta, \mathrm{P}, \Delta, M, \Lambda, \Phi, \Psi \rangle$*, where* Θ *is the set of threats,* P *is the set of response actions,* Δ *is the set of detection actions,* M *is the set of measurements,* Λ *is the accuracy profile,* Φ *is the response cost function, and* Ψ *is the detection cost function.*

4 Belief-Space Planning

We show that an automated malware defense framework can be represented as a POMDP. Once the framework has been specified given the details of the system as well as the constraints and preferences of the system administrators, the framework can be modeled as a POMDP. This enables the framework to make decisions using the optimal policy of the POMDP. We discuss the representation of the POMDP below.

States

The model makes decisions based on several key features of the framework. First, the model must be aware of all of the threats that can be assigned to a file by the framework. Next, since each detection action does not provide any additional insight once it has been executed, the framework should only perform a detection algorithm once. The model must therefore store whether or not each detection action has been used. Finally, recall that the framework can no longer perform any additional actions once a response action has been performed. As a result, the model must only record whether or not the framework has used any of the response actions. We define the states of the model as the set $S = F_1 \times F_2 \times F_3$ where

- $F_1 = \Theta$ represents the set of threats,
- $F_2 = \{\mathit{True}, \mathit{False}\}^{|\Delta|}$ indicates whether or not each detection action δ in the set of detection actions Δ has been executed, and
- $F_3 = \{\mathit{True}, \mathit{False}\}$ indicates whether or not any response action ρ in the set of response actions P has been executed.

Note that feature F_1 is partially observable given that the framework cannot always know with certainty whether or not a file is malicious. The features F_2 and F_3 are fully observable since the framework knows which response actions or detection actions have already been performed.

Actions

The model selects actions from the set of response actions and detection actions available to the framework. If the framework has not gathered any information about the file yet, it will likely decide to perform a detection action. However, once the framework is confident about its assessment of the file, it will likely decide to perform a response action. We therefore define the actions of the model as the set $A = \mathrm{P} \cup \Delta$ where P is the set of response actions and Δ is the set of the detection actions.

Transition Function
The model requires some representation of the transition dynamics of the framework. This can be expressed as a function that maps a state and an action to a successor state. When a detection action is executed, the successor state is a state in which that detection action has been used. However, when a response action is executed, the successor state is a state that indicates whether or not any of the response actions have been used. While it is often not possible to observe whether or not the file is a threat, the successor state always corresponds to the actual threat of the file. In other words, the framework cannot change whether or not the file is a threat. Given a state $s = (f_1, f_2, f_3) \in S$ and an action $a \in A$, the transition function of the model can be expressed as the function

$$T(s,a) = \begin{cases} (f_1, f_2', f_3), & \textit{if } a \in \Delta, \\ (f_1, f_2, f_3'), & \textit{if } a \in \mathrm{P}, \end{cases}$$

where $f_1 \in F_1$ has not been updated, $f_2' \in F_2$ has been updated with respect to $f_2 \in F_2$ and $a \in A$, and $f_3' \in F_3$ has been updated with respect to $f_3 \in F_3$ and $a \in A$.

Reward Function
The model depends on some representation of the rewards of the framework. This can be expressed as a function that maps a state and an action to an expected immediate reward. Such a function is specified by the response cost function and the detection cost function of the framework. Given a state $s \in S$ and an action $a \in A$, the reward function of the model can be expressed as the function

$$R(s,a) = \begin{cases} \Phi(a), & \textit{if } a \in \Delta, \\ \Psi(s,a), & \textit{if } a \in \mathrm{P}, \end{cases}$$

where Δ is the set of the detection actions, P is the set of response actions, Φ is the detection cost function, and Ψ is the response cost function.

Observations
The model experiences a number of observations. When a detection action is performed, it emits a measurement that indicates whether or not the file is a malicious. All response actions emit an observation that signify that the file has been addressed. Thus, the observations of the model is the set $\Omega = M \cup \{\eta\}$ where M is the set of measurements and η is the termination observation.

Observation Function
The model uses some representation of the observation dynamics of the framework. This can be expressed as a function that maps a successor state and an action to a probability distribution over all observations. Recall that a detection action emits measurements based on the accuracy profile of the framework. Moreover, a response action always emits a termination observation. Given a state $s = (f_1, f_2, f_3) \in S$, an action $a \in A$, and an observation $\omega \in \Omega$, the observation function of the model can be expressed as the function

$$O(s,a,\omega) = \begin{cases} \Lambda(f_1, a), & \textit{if } a \in \Delta, \\ [\omega = \eta], & \textit{if } a \in \mathrm{P}, \end{cases}$$

where Λ is the accuracy profile, Δ is the set of detection actios, η is the termination observation, and P is the set of response actions. The operator $[\cdot]$ is Iverson bracket notation.

5 Email Worm Detection and Response

In this section, we offer an example automated malware defense system for email worm detection and response. Suppose an email server is about to forward an email to its destination in a private network. In order to determine whether or not the email has a worm, the email server can apply a combination of several malware detection techniques of varying costs and accuracies to the attachment of the email. Because each malware detection technique may or may not be confident in its evaluation of the email, it can return one of several values that indicates its level of certainty. Once the email server is confident about its assessment of the email, it can either forward the email to its destination in the network or drop the email from the network entirely. The email server should be penalized for forwarding infected emails and dropping normal emails and executing detection actions unnecessarily. The objective of the email server is therefore to execute as little of the malware detection techniques as necessary before being confident enough to forward or drop the email. While it is possible to execute all malware detection techniques, this would substantially slow down email traffic in the network and increase power consumption.

More formally, the email server can classify an email that enters the network as either an infected email or a normal email. Accordingly, there are no other threats aside from an email worm. We therefore define the set of threats below:

$$\Theta = \{\varnothing, \textit{Worm}\}.$$

To address each email that enters the network, the email server can either forward the email to its destination in the network or drop the email from the network entirely. Hence, there are two response actions that can be applied to every email. We thus define the set of response actions as follows:

$$\mathrm{P} = \{\textit{Forward}, \textit{Drop}\}.$$

In the interest of clarity, we only consider two response actions. It is possible, however, to define other response actions, such as forwarding an email with a warning, forwarding an email without the attachment, or executing the attachment on a quarantined host designed for malware detection.

In general, the email server should forward normal emails to their destinations in the network and drop infected emails from the network. This strategy is encoded by the response cost function. One example of a response cost function assigns a cost of 750 to forwarding an infected email and a cost of 50 to dropping a normal email. All other response actions incur no cost. Given a threat $\theta \in \Theta$ and a response action $\rho \in \mathrm{P}$, we define a simplified response cost function below:

$$\Phi(\theta, \rho) = \begin{cases} -750, & \textit{if } \theta = \textit{Worm and } \rho = \textit{Forward}, \\ -50, & \textit{if } \theta = \varnothing \textit{ and } \rho = \textit{Drop}, \\ 0, & \textit{otherwise}. \end{cases}$$

In practice, the response cost function encodes the preferences and constraints of the network administrator. While we provide a response cost function that demonstrates the framework, such a function is typically derived from the expected damage that can be caused by dropping normal emails and forwarding infected emails as well as other considerations.

We consider an email server with an inaccurate but fast malware detection technique as well as an accurate but slow malware detection technique. The email server can gather more information about whether or not the email is a threat by performing some sequence of detection techniques. Consequently, we define the set of detection actions as follows:

$$\Delta = \{\mathit{WeakDetect}, \mathit{StrongDetect}\}.$$

Although we do not assume a particular implementation, these detection techniques could reflect a signature-based detection technique or a behavior-based technique in practice.

All malware detection techniques can generate several measurements for emails that enter the network. Each measurement corresponds to a level of confidence based on whether or not the email is a threat. A higher confidence score indicates a higher likelihood of an email having a worm and vice versa. Hence, we define the set of measurements below:

$$M = \{\mathit{Low}, \mathit{Medium}, \mathit{High}\}.$$

Every malware detection technique is associated with some overhead. In our example, the cost of a detection technique represents a number of different factors, such as its impact on the performance of the network, any privacy violations, as well as power consumption overhead. We assume that the inaccurate detection technique has a low overhead because of low execution requirements and the accurate detection technique has a high overhead due to high execution requirements. Given a detection action $\delta \in \Delta$, we provide a simplified detection cost function as follows:

$$\Psi(\delta) = \begin{cases} -1, & \mathit{if}\ \delta = \mathit{WeakDetect}, \\ -5, & \mathit{if}\ \delta = \mathit{StrongDetect}. \end{cases}$$

As discussed earlier, the response cost function reflects the actual specifications of each detection action. In practice, it is possible to approximate the detection cost function from the expected execution time, the expected power consumption, and many other characteristics of each detection action.

Each malware detection technique offers a different degree of accuracy. If we assume that the inaccurate detection technique is solely based on the signature of the email worm, it will only perform well with known email worms. As a result, this detection technique would be less likely to emit measurements that reflect the true status of the email worm. Given the weak detection action $\delta = \mathit{WeakDetect}$, we provide an idealized accuracy profile below:

$$\Lambda(\theta, \delta, \mu) = \begin{cases} 0.4, & \mathit{if}\ \theta = \mathit{Worm\ and}\ \mu = \mathit{Low}, \\ 0.6, & \mathit{if}\ \theta = \mathit{Worm\ and}\ \mu = \mathit{High}, \\ 0.9, & \mathit{if}\ \theta = \varnothing\ \mathit{and}\ \mu = \mathit{Low}, \\ 0.1, & \mathit{if}\ \theta = \varnothing\ \mathit{and}\ \mu = \mathit{High}, \\ 0, & \mathit{otherwise}. \end{cases}$$

However, if we assume that the accurate malware detection technique analyzes the behavior of the email worm, it will remain effective with unknown or little known email worms. Hence, this detection technique would be more likely to emit measurements that reflect the true status of the email worm. Given the strong detection action $\delta = \mathit{StrongDetect}$, we offer an idealized accuracy profile as follows:

$$\Lambda(\theta, \delta, \mu) = \begin{cases} 0.1, & \mathit{if}\ \theta = \mathit{Worm\ and}\ \mu = \mathit{Low}, \\ 0.2, & \mathit{if}\ \theta = \mathit{Worm\ and}\ \mu = \mathit{Medium}, \\ 0.7, & \mathit{if}\ \theta = \mathit{Worm\ and}\ \mu = \mathit{High}, \\ 0.9, & \mathit{if}\ \theta = \varnothing\ \mathit{and}\ \mu = \mathit{Low}, \\ 0.05, & \mathit{if}\ \theta = \varnothing\ \mathit{and}\ \mu = \mathit{Medium}, \\ 0.05, & \mathit{if}\ \theta = \varnothing\ \mathit{and}\ \mu = \mathit{High}. \end{cases}$$

Note that this accuracy profile has been designed in the interest of clarity. In practice, it is possible to approximate an accuracy profile using machine learning methods.

6 Experiments

We compare the example automated malware defense system for email worm detection and response to several standard techniques that do not explicitly reason about which detection actions to perform. Each technique therefore always performs a specific set of detection actions. The set of detection actions available to each technique is as follows:

1. a *WeakDetect* action,
2. a *StrongDetect* action, and
3. a *WeakDetect* action and a *StrongDetect* action.

Note that these techniques select the response action that maximizes the expected cumulative reward given the measurements from the detection actions. As a baseline, we also consider a technique that selects a response action based on the initial belief without performing any detection actions.

All experiments run $10,000$ simulations that represent a typical instance of an email worm threat scenario where the technique must decide whether to forward or drop an email with an attachment. For the automated malware defense system, we execute a process that begins with an initial belief that indicates whether or not emails generally have worms. An email that has an attachment is then given to the process. The process then updates its belief about the email by performing a sequence of email worm detection action. After attaining a sufficient belief about the email or performing all email worm detection actions, the process can respond to the email by forwarding or dropping it. The simulation terminates once the process has responded to the email.

As discussed earlier, the automated malware defense system makes decisions based on the optimal policy graph of the POMDP. We construct the policy graph using the *Incremental Pruning* algorithm [Zhang and Liu, 1996; Cassandra *et al.*, 1997; Feng and Zilberstein, 2004]. Incremental pruning is a method that prunes the dominated alpha vectors of each action individually. Once all actions have been examined, the optimal policy graph is extracted from the remaining alpha vectors. We use a standard open source C implementation of state-of-the-art solvers, *pomdp-solve*, for the Incremental Pruning algorithm [Cassandra *et al.*, 1997]. It is also possible to use a wide range of methods to construct the policy graph, such as Sondik's One-Pass Algorithm [Sondik, 1971], the Witness algorithm [Littman, 1994], and Point-based Value Iteration [Pineau *et al.*, 2003]. Since the policy graph is already optimal, this choice does not affect our results.

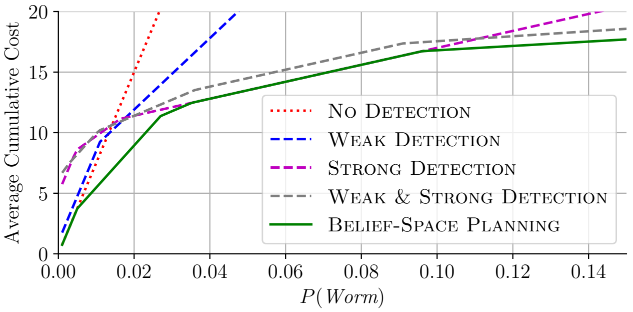

Figure 1: The average cumulative cost of our approach and the standard techniques on different threat scenarios.

Figure 1 depicts the average cumulative cost of the automated malware defense system and the four standard techniques on a wide range of threat scenarios. Each threat scenario represents a different probability of an email with a worm entering the system. For instance, when $P(Worm) = 0.14$, the system has a 14% chance of encountering an infected email. Note that our approach has the lowest average cumulative cost across every threat scenario.

Figure 2 illustrates the optimal policy graph of the POMDP used by the automated malware defense system. Each node represents an action where the *W*, *S*, *F*, and *D* nodes denote the *WeakDetect*, *StrongDetect*, *Forward*, and *Drop* actions respectively. The *F* and *D* nodes are terminal in that the system can no longer perform additional actions. Each edge represents an observation where the *L*, *M*, and *H* edges denote the *Low*, *Medium*, and *High* measurements respectively. Note that the initial belief state of the system specifies the starting node of the policy graph. The marked node is associated with a uniform initial belief state.

7 Discussion

On all threat scenarios, the automated malware defense system performs at least as well as every standard technique in Figure 1. For threat scenarios with a worm probability between roughly 1% to 4% and 10% to 15%, our approach has a significantly lower average cumulative cost than all techniques. Interestingly, for threat scenarios with a worm probability between roughly 0% to 1% and 4% to 10%, our approach has the same average cumulative cost as the technique that does not use any detection action and the technique that always uses the strong detection action respectively. Because the probability of encountering a worm cannot be known in advance, it is always beneficial to use our approach given that it performs as well if not better than all standard techniques.

The automated malware defense system makes decisions using the policy graph in Figure 2. While our approach can start with any arbitrary initial belief state, we consider the uniform belief state associated with the marked node as the starting node. The behavior of our approach is intuitive. After the system performs the *WeakDetect* action, it executes the *Drop* action if a *High* observation is emitted or executes the *StrongDetect* action if a *Low* observation is emitted. If the *StrongDetect* action emits a *Medium* or *High* observation, the system performs the *Drop* action. Otherwise, the system performs the *Forward* action. Note that the other nodes are visited given different initial belief states.

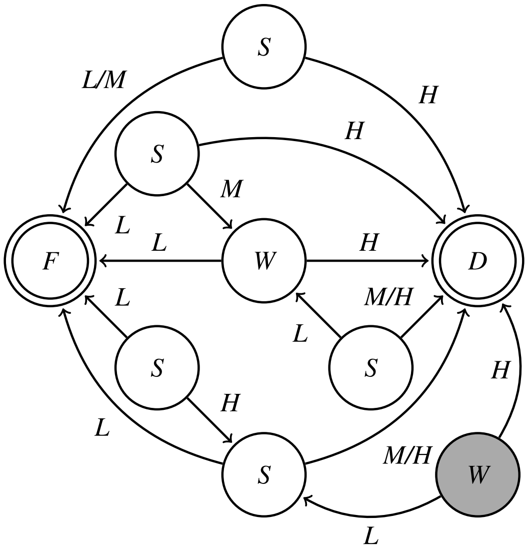

Figure 2: The optimal policy graph used by our approach.

Using belief-space planning in automated malware defense offers a wide range of benefits. First, using a POMDP obviates the need for potentially error-prone security systems that are carefully crafted by researchers and practitioners. Next, even if it is possible to develop an adequate security system by hand, the optimal policy of a POMDP guarantees the best possible trade-off between using malware detection techniques and impacting the system. Finally, because the policy graph of the optimal policy is complex even for a small number of actions and observations, it will likely become difficult—if not infeasible—to design effective security systems by hand for more realistic problems. Using belief-spacing planning therefore increases the effectiveness and reduces the overhead of security systems.

8 Conclusion

We offer a novel belief-space planning approach to automated malware defense designed for robust, accurate, and efficient use in large networks of resource-constrained devices. It offers many advantages over prevailing methods: not only does it offer more effective information security, but it also reduces the impact on the performance of a system. Finally, to show that our approach yields effective automated malware defense, we show in simulation that it outperforms standard security techniques that have been deployed in practice. Future work will conduct experiments on real systems and explore machine learning methods that can be used to generate accuracy profiles for malware detection techniques.

Acknowledgments

We thank the anonymous reviewers for their helpful comments. This work was supported in part by the National Science Foundation grants IIS-1405550 and IIS-1724101.

References

[Cassandra *et al.*, 1997] Anthony Cassandra, Michael L. Littman, and Nevin L. Zhang. Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Thirtieth Conference on Uncertainty in Artificial Intelligence*, pages 54–61. Morgan Kaufmann Publishers Inc., 1997.

[Cesare *et al.*, 2013] Silvio Cesare, Yang Xiang, and Wanlei Zhou. Malwise: An effective and efficient classification system for packed and polymorphic malware. *IEEE Transactions on Computers*, 62(6):1193–1206, 2013.

[Faruki *et al.*, 2015] Parvez Faruki, Vijay Laxmi, Ammar Bharmal, Manoj Singh Gaur, and Vijay Ganmoor. Androsimilar: Robust signature for detecting variants of android malware. *Journal of Information Security and Applications*, 22:66–80, 2015.

[Feng and Zilberstein, 2004] Zhengzhu Feng and Shlomo Zilberstein. Region-based incremental pruning for POMDPs. In *Twentieth Conference on Uncertainty in Artificial Intelligence*, pages 146–153. AUAI Press, 2004.

[Kaelbling *et al.*, 1998] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

[Littman, 1994] Michael L. Littman. The Witness algorithm: Solving partially observable Markov decision processes. *Brown University, Providence, RI*, 1994.

[Masud *et al.*, 2006] Mohammad M. Masud, Latifur Khan, and Ehab Al-Shaer. Email worm detection using Naive Bayes and support vector machine. In *International Conference on Intelligence and Security Informatics*, pages 733–734. Springer, 2006.

[Nelms *et al.*, 2015] Terry Nelms, Roberto Perdisci, Manos Antonakakis, and Mustaque Ahamad. Webwitness: Investigating, categorizing, and mitigating malware download paths. In *USENIX Security Symposium*, pages 1025–1040, 2015.

[Pineau *et al.*, 2003] Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Point-based Value Iteration: An anytime algorithm for POMDPs. In *Eighteenth International Joint Conference on Artificial Intelligence*, pages 1025–1032, 2003.

[Ruiken *et al.*, 2016] Dirk Ruiken, Tiffany Q. Liu, Takeshi Takahashi, and Roderic A. Grupen. Reconfigurable tasks in belief-space planning. In *Sixteenth International Conference on Humanoid Robots*, pages 1257–1263. IEEE, 2016.

[Schechter *et al.*, 2004] Stuart E. Schechter, Jaeyeon Jung, and Arthur W. Berger. Fast detection of scanning worm infections. In *International Workshop on Recent Advances in Intrusion Detection*, pages 59–81. Springer, 2004.

[Shafiq and Liu, 2017] Zubair Shafiq and Alex Liu. A graph theoretic approach to fast and accurate malware detection. In *IFIP Networking Conference and Workshops*, pages 1–9. IEEE, 2017.

[Sondik, 1971] Edward Jay Sondik. The optimal control of partially observable Markov processes. Technical report, Stanford University, 1971.

[Tanaka *et al.*, 2017] Yasuyuki Tanaka, Mitsuaki Akiyama, and Atsuhiro Goto. Analysis of malware download sites by focusing on time series variation of malware. *Journal of Computational Science*, 22:301–313, 2017.

[Tian *et al.*, 2010] Ronghua Tian, Rafiqul Islam, Lynn Batten, and Steve Versteeg. Differentiating malware from cleanware using behavioural analysis. In *Fifth International Conference on Malicious and Unwanted Software*, pages 23–30. IEEE, 2010.

[Tuvell and Venugopal, 2017] George Tuvell and Deepak Venugopal. Malware detection system and method for mobile platforms, 2017. US Patent 9,576,131.

[Wray *et al.*, 2016] Kyle H. Wray, Luis Pineda, and Shlomo Zilberstein. Hierarchical approach to transfer of control in semi-autonomous systems. In *Twenty-fifth International Joint Conference on Artificial Intelligence*, pages 517–523, 2016.

[Wray *et al.*, 2017] Kyle H. Wray, Stefan J. Witwicki, and Shlomo Zilberstein. Online decision-making for scalable autonomous systems. In *Twenty-sixth International Joint Conference on Artificial Intelligence*, pages 4768–4774, 2017.

[You and Yim, 2010] Ilsun You and Kangbin Yim. Malware obfuscation techniques: A brief survey. In *International Conference on Broadband, Wireless Computing, Communication and Applications*, pages 297–300. IEEE, 2010.

[Zhang and Liu, 1996] Nevin L. Zhang and Wenju Liu. Planning in stochastic domains: Problem characteristics and approximation. Technical report, Hong Kong University of Science and Technology, 1996.