JOURNAL OF CYBERSECURITY

Journal of Cybersecurity, 1(1), 2015, 37–51
doi: 10.1093/cybsec/tyv002
Advance Access Publication Date: 5 November 2015
Research Article

Research Article

Optimizing time allocation for network defence

Tristan Caulfield[1,*] and Andrew Fielder[2]

[1]Department of Computer Science, University College London and [2]Institute for Security Science and Technology, Imperial College London

*Corresponding author: Department of Computer Science, University College London, Gower Street, London WC1E 6BT, United Kingdom. Tel: +44 (0)20 7679 3672; Fax: +44 (0)20 7387 1397; E-mail: t.caulfield@ucl.ac.uk

Received 16 May 2015; revised 19 September 2015; accepted 28 September 2015

Abstract

The presence of unpatched, exploitable vulnerabilities in software is a prerequisite for many forms of cyberattack. Because of the almost inevitable discovery of a vulnerability and creation of an exploit for all types of software, multiple layers of security are usually used to protect vital systems from compromise. Accordingly, attackers seeking to access protected systems must circumvent all of these layers. Resource- and budget-constrained defenders must choose when to execute actions such as patching, monitoring and cleaning infected systems in order to best protect their networks. Similarly, attackers must also decide when to attempt to penetrate a system and which exploit to use when doing so. We present an approach to modelling computer networks and vulnerabilities that can be used to find the optimal allocation of time to different system defence tasks. The vulnerabilities, state of the system and actions by the attacker and defender are used to build partially observable stochastic games. These games capture the uncertainty about the current state of the system and the uncertainty about the future. The solution to these games is a policy, which indicates the optimal actions to take for a given belief about the current state of the system. We demonstrate this approach using several different network configurations and types of player. We consider a trade-off for the system administrator, where they must allocate their time to performing either security-related tasks or performing other required non-security tasks. The results presented highlight that, with the requirement for other tasks to be performed, following the optimal policy means spending time on only the most essential security-related tasks, while the majority of time is spent on non-security tasks.

Key words: stochastic games; network defence; patching.

Introduction

System administrators or security managers have the challenging task of maintaining an organization's defence of its networks against attack. These networks consist of a number of machines, running various pieces of software, which provide the organization's business services. Vulnerabilities are almost inevitably discovered for every piece of software, and these vulnerabilities can be exploited by attackers to gain access to a system.

To defend against these attacks, system administrators can take a number of actions, including monitoring the systems and network for intruders, applying patches to the systems to stop possible exploits, and restoring systems that have been compromised. However, each of these actions takes time—a limited resource—and system administrators often have other tasks to complete in addition to those related to security. While system administrators must ensure the security of their systems, they are unable to neglect these other tasks, which presents the requirement for them to efficiently distribute their time between these two kinds of tasks.

In this article, we present an approach to calculating optimal time allocation policies for network defence. That is, for a given network of systems and a belief about the current state of the network, what are the optimal actions to spend time on?

Our approach allows us to capture different network configurations and types of attackers and defenders. These are then used to create a partially observable stochastic game (POSG) which is solved to find an approximate solution to the optimal policy which indicates the optimal action to take for any given belief about the current state. We demonstrate the approach using different scenarios

which have different network configurations, players, time constraints and costs.

The remainder of this article is divided as follows. Section 2 gives an overview of the aspects of cybersecurity relevant to the rest of the article, and gives an introduction to our approach. This is followed by Section 3, which is a discussion of related work. Section 4 contains a brief introduction to the methods used model problems about calculating the optimal actions to take, and presents our approach to solving them. Then, Section 5 describes how we represent the specific problem of time allocation for network defence and gives details about the implementation. Section 6 presents the different scenarios we use to demonstrate this approach and Section 7 presents the results of these scenarios. Next, Section 8 contains a discussion about our approach and its applicability, and finally, we conclude with Section 9.

Background

In this section, we present an overview of the aspects of cybersecurity relevant to the rest of the article and give the foundations for our model.

Underlying the model presented here is the concept of the vulnerability lifecycle, which describes the process in which vulnerabilities are created, discovered, exploited and patched within a piece of software. Figure 1 show the various stages of the lifecycle. A vulnerability is created in a piece of code due to a mistake during software development.

What happens next depends on who discovers the vulnerability. It may be discovered by attackers, who could develop an exploit for it, or by security researchers or the software company itself, who might publically disclose the existence of the vulnerability or quickly release a patch. Exploits can still be developed for vulnerabilities that have been disclosed or patched. It is also possible that an exploit discovered by attackers may go undisclosed and unpatched for a significant amount of time.

Once an exploit is developed, a system running the vulnerable software can potentially be compromised by attackers until a patch is created and then applied to the system. In Fig. 1, the grey boxes indicate states where systems are vulnerable to the exploit.

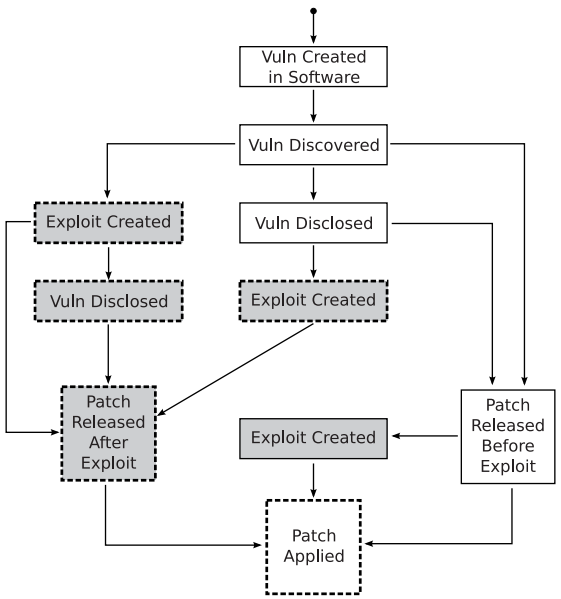

Figure 1. States in the vulnerability lifecycle. The grey states are ones where a system can be compromised by this vulnerability. The states with dashed borders are the states we model in this article.

In this article, we consider a subset of the states of the vulnerability lifecyle. These are shown with dotted borders in Fig. 1. We only model the states where an exploit has been created, before or after disclosure; we ignore the states where an exploit has not been created, or where a patch is released before an exploit is developed.

Systems usually run multiple pieces of software and are typically connected to other devices. Therefore, we consider a network that consists of any number of interconnected devices in a hierarchical structure. The network consists of a single external source, which may be connected to any number of devices, and is the location from which each attack is launched. Each different device contains different information, which can be of differing value to defenders and attackers. The connections of devices link the external network to the devices containing data. Attacks occur in the network by exploiting a vulnerability in a piece of software at one of the connected nodes in the network, with each newly compromised device giving the attacker access to data or other network devices.

We can further express the network diagram in terms of a directed attack graph with vulnerabilities applied. An example of this can be seen in Fig. 2.

In the example, we can see that while the initial stages of an attack can compromise either Device 1 or 2, the rest of the attack must compromise both Devices 3 and 4 to succeed. In this case, we consider a state of vulnerabilities in the network such that Device 5, has no currently unpatched software vulnerabilities, which means that while it is connected in the network and the data store is available to be reached by it, there is no current logical connection in the attack graph for the external attacker to be able to compromise. If a vulnerability was to be discovered in a piece of software running on Device 5, then there would then exists an attack path between Devices 3 and 5, which would give the attacker a method to gain access.

Alternatively, if the defender were to patch vulnerability 6 on Device 4, then the attacker would have no viable attack paths, since there would be no devices that are connected directly to the data source that have a vulnerability for the attacker to exploit in order to perform a successful attack. In this case, an attacker would then have to wait for a new vulnerability to be found and exploitable on either Device 3 or 4 for them to be able to launch a fully successful attack.

Player strategies

The model presented in this work considers the interaction between two players, a defender and an attacker. The defender is the lead of a team of system administrators, who have the task of maintaining the security of the system. The attacker attempts to breach the defences of the system in order to exfiltrate some critical data.

We consider the attacker to be a well-resourced, determined player. In this way, we consider that the attacker will not give up until they have managed to successfully steal the data held on the system they are attacking.

In this model, we consider that each of the players has some finite resources to perform actions at each stage of the game. At each stage, the players must make the best possible move based on the information they have about the state of the network and the vulnerabilities.

For the defender, this represents an allocation of time to dedicate certain members of the system administration team to a number of different tasks. The defender is constrained by the length of time that each task takes and the total number of hours available among all members of the administration team.

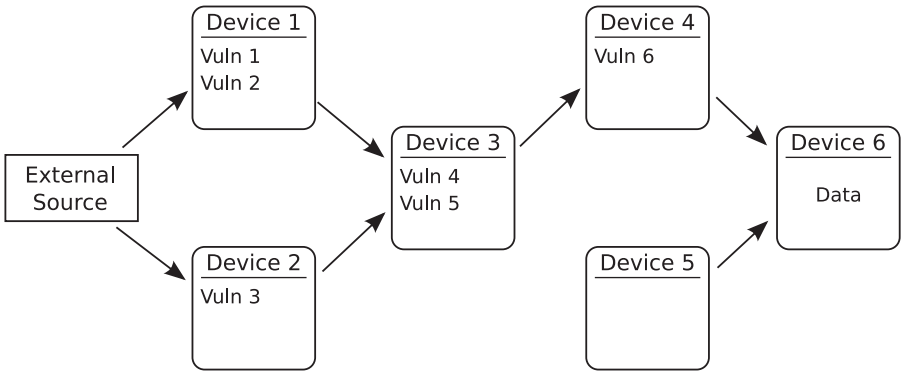

Figure 2. A sample attack graph.

Unlike the defender, who is constrained by the time the system administrators are available to work, the attacker has no such constraint. However, we consider that the attacker is able to perform only a single action in their turn.

Defender's actions

A system administrator can be expected on a day to day to basis perform one of three tasks related to security, monitor, patch and recover. However, we consider that a system administrator is not just assigned to security tasks, but have a variety of tasks that must be performed to maintain the performance of the system. As such, we factor in an expected level of maintenance that should be performed on the system or other non-security related tasks.

Monitor

Monitoring is the action the defender uses to discover about the state of the network. A defender has no intuitive knowledge about the state of the system, in so far that they are unable to tell if a device has been compromised. The defender can choose to monitor the activity of devices to see if there has been any activity that would cause them to believe that the device has been compromised.

Patch

Should a vulnerability exist in a piece of software, and a patch has been developed for the vulnerability, then the defender can choose to take some time to apply the patch. When the defender chooses to patch the system, the defender removes the option for the attacker to use an exploit for that vulnerability. This means that should the attacker attempt to attack via that exploit, they would fail to compromise the device.

Recover

Even if a patch has been developed for a vulnerability, if a device susceptible to that vulnerability has become compromised, then the attacker is still able to propagate an attack to the next step. In order for the defender to regain control of the device, they need to recover the system to a state where they are confident that it is not compromised.

It should be noted that recovering a device before applying a patch is not effective as the exploit the attacker used to initially compromise the device still exists and is still a viable attack. Additionally, if there is another vulnerability that can be exploited on the same device, patching one will not necessarily stop the attacker from compromising the device.

Unrelated tasks

We consider that outside of the security tasks, a system administrator has other tasks related to maintaining the system that need to be performed. As such, the model defines the concept of an unrelated task, which stands for any of these other tasks.

Other tasks are specified in this model, to represent that when a network is under attack, then the resources of the defender are more likely to be focused on the defence of the network as opposed to regular tasks. However under normal conditions, allocating all the available administrator time to security will mean that other tasks would go unfulfilled, and would constitute a waste of resources. This is such that we define a cost to the defender for allocating too few resources to unrelated tasks, such that under normal operation the defender would not benefit from allocating more administrator time to defence over system performance and maintenance.

Attacker's actions

The attacker in this model has only two different actions. They can do nothing, or they can launch an attack using an available exploit.

Exploit

Once an exploit has been developed, the attacker can act to use the exploit to advance through the system towards the data that they aim to steal. In this model, an attacker may only use a single attack at time, even if there are multiple exploits available.

No action

The attacker can choose to take no action on their turn, which in turn requires no use of resources by the attacker. This action will typically be taken by the attacker when they do not want to use an exploit or have no remaining usable exploits.

Related work

One of the critical components in the model provided in this work is the vulnerability lifecycle—the representation of the development exploitation and patching of software vulnerabilities. Schneier [1] provides an initial look at the 'window of exposure' that is central to the concept of the vulnerability lifecycle. Frei *et al.* [2] provide a more formal overview of the vulnerability lifecycle, considering predominantly the risk and the groups that know specifically of the risk. They specifically look at risk exposure, looking at the gap between exposure of the vulnerability and patch availability in more than 14 000 published vulnerabilities.

The work by Beres *et al.* [3] expands on the framework for the vulnerability lifecycle, considering additional steps for completeness. While the work presents all of the stages of the lifecycle it focuses on the 'risk exposure window', which is considered the time between public disclosure of the vulnerability and an organization considering that the vulnerability is no longer a risk.

Ioannidis *et al.* [4] develop a model examining the trade-offs present in the deployment of software patches. The authors present a model for finding the optimal frequency for the deployment of patches, considering the costly nature of implementing patches, weighed against the security risks of not applying them. This work lays the foundations for understanding the trade-offs in acting proactively by applying patches against delaying deployment and increasing the risk of being vulnerable to an attack and having to defend reactively.

Arora *et al.* [5] discuss the notion of a socially optimal planner, who acts to optimize the behaviours of vendors developing patches for vulnerabilities. This acts critically with the vulnerability lifecycle presented in this work to reduce the probability of transition to a state where there is a disclosed vulnerability with no available patch, which is very detrimental to the defender's ability to effectively defend their system. Further work by Arora identifies other pressures that may influence their ability and willingness to release patches efficiently, such as competition [6] and influence of disclosure [7].

Attack graphs are a prominent method of representing the process by which attackers attempt to break into a system. A number of different forms of attack graph have been created focusing on different methods to represent the attack. One form of attack graph representation is a privilege-based model, such as those presented by Dacier *et al.* [8], in which the states of the model represent a set of privileges that the attacker has gained with the steps of the attack taken to that point. Ortalo *et al.* [9] extend the work to consider an experimental setting consisting of major Unix vulnerabilities.

A more common representation of the atomic states in an attack graph is to represent each stage as a security property violation. The fundamentals of security property violation based attack graphs were outlined by Phillips and Swiler [10], where each stage represents an event that occurs which is required for the compromise of the device. This approach stems from the initial Insecurity Flow Model presented by Moskowitz and Kane [11], where they consider the spread of an insecurity from an attacker's initial location to a store of data to be protected.

Attack graphs are commonly deployed to perform a security analysis on a network, implementing the structures of networks and attacks in order to better understand weaknesses within the given space [12–16].

One of the main implementations of Bayesian attack graphs, is to provide cost-benefit analysis on the adoption of cybersecurity solutions, as in [17–19]. This has been further expanded to consider complete risk assessments of systems in order to more accurately represent the problem of optimal investments in cyber defences [20–22].

In work by Dantu *et al.* [23], the authors define a model of risk management using attack graphs with behavioural qualities to create risk management strategies. This work considers the difference in attacker profile and preferences and models the qualities that the attacker has creating a detailed assessment of the risks associated with different classes of attacker. The work was extended to predict attacker type based on certain network actions undertaken by an attacker [24, 25].

A similar approach has also been developed by Wang *et al.* [26], where the authors apply the analysis to the problem of intrusion detection systems.

One of the major advancements in reasoning about how to analyse attack graphs has been the development of Bayesian Attack Graphs [27], where Bayesian inference is applied to the properties of the graph. Liu and Man [28] present an overview of how to perform network vulnerability assessment using Bayesian networks. Poolsappasit *et al.* [29] provide an empirical study of a Bayesian network for risk management.

A further approach is to use attack trees [30], where the nodes in the tree represent the steps of a possible attack. In order to compromise a system, a path from a set of required leaf nodes to the root node must exist. The set of required nodes is given by conditions between branches traversing up a tree.

Moore [31] provided a security analysis of a fictitious company using an attack tree-based system. The Authors create attack trees from the network and perform an analysis on the resulting trees.

Within the field of games and cybersecurity, one of the key works was undertaken by Lye and Wing [32]. The authors represent the interactions between an attacker and a defender as a general-sum stochastic game. The work presents an example case study of how the decision-making process, as represented by stochastic games, can be applied to cybersecurity. However, the model does not capture one of the key operations of system administrators in a cybersecurity scenario, which is to maintain defensive properties of a network in advance of an attack.

Work by Fielder *et al.* [33] presents a model for assigning system administrator time to security-related tasks using a game-theoretic approach. The approach weights the value of the data available from each device in a network, where the allocation of a limited number of administrators are assigned to ensure that the most at risk or valuable targets were given more attention.

Calculating optimal policy

The goal of this work is to figure out the best way to allocate time between different defensive tasks. The optimal allocation at any given moment depends on the state of the system: if a computer has been compromised, it is probably better to spend time fixing it than doing something else. Markov Decision Processes (MDPs) describe situations where the future is uncertain and an agent wishes to maximize its future reward (or minimize future loss). Solving a MDP gives a policy, which describes the optimal action to take for any given state.

However, it is often impossible to know the exact state of the system so the best action to take is unclear. These situations can be described by Partially Observable Markov Decision Processes (POMDPs), which extend MDPs to account for uncertainty about state. Solving these gives a policy that describes the optimal action to take for a given belief about the current state: if it is unlikely that the machine has been compromised, it is better to do something else.

However, we want to know what the best action to take is when there is also an attacker making decisions about what to do. Both MDPs and POMDPs have game-theoretic counterparts that allow for multiple players: stochastic games and POSGs respectively. It is the latter of these that we use in our approach.

In this section we give a brief overview of MDPs, stochastic games, POMDPs and POSGs. We then introduce a version of POSGs where all players share the same observations and describe how we implemented a solver for this class of problem.

MDPs

A MDP is a tuple $(S, A, P.(\cdot,\cdot), R.(\cdot,\cdot), \gamma)$ where

- S is a finite set of states.
- A is a finite set of actions, A_s are the actions available from state s.
- $P_a(s,s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$ is the transition probability of moving to state s' from state s with action a.
- $R_a(s,s')$ is the immediate reward (or cost) received after a transition from s to s'.
- $\gamma \in [0,1]$ is the discount factor.

The goal is to find a policy, in the form of a function $\pi(s)$ that specifies the action that should be taken in state s. The policy should maximize the discounted rewards over an infinite horizon:

$$\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1}) \text{ where } a_t = \pi(s_t). \tag{1}$$

We can define the value of each state, $V(s)$ recursively:

$$V_{n+1}(s) = \max_a \left[\sum_{s'} P_a(s,s')(R_a(s,s') + \gamma V_n(s')) \right]. \tag{2}$$

This equation, which separates the current reward from future rewards, is known as the Bellman equation. The optimal value function, V^* can now be calculated: starting with an initial (arbitrary) set of values V_0, iterate and compute V_{n+1} for all states until convergence, where $V_{n+1} = V_n$.

The policy is then calculated by finding the maximizing action for each state:

$$\pi(s) = \arg\max_a \left[\sum_{s'} P_a(s,s')(R_a(s,s') + \gamma V(s')) \right]. \tag{3}$$

The optimal policy, π^*, is the policy for the optimal value function, V^*, and specifies the action that maximizes the discounted reward over all future states.

If the reward (or cost) is independent of the new state, these can be re-written as

$$V_{n+1}(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} P_a(s,s') V_n(s') \right] \tag{4}$$

and

$$\pi(s) = \arg\max_a \left[R(s,a) + \gamma \sum_{s'} P_a(s,s') V(s') \right], \tag{5}$$

where $R(s, a)$ is now the reward (or cost) for performing action a in state s.

Stochastic games

The method for solving stochastic games is similar and was first given by Shapley [34]. For games of more than one player, the actions of the other players must be taken into account. Instead of simply updating the values as the transition-probability weighted averages of the next states, as in MDPs, the values of the next states are used to construct a simple matrix game $G_s(V)$, and the value of V is updated by solving this matrix game for each state.

Naturally, V now gives values for each of the players in the game, and each player has their own actions: a represents the combined actions of all the players.

Instead of selecting the action that maximizes the future value, the actions of the other players must be taken into account. For each state $s \in S$, construct the matrix game

$$G_s(V_n) = \left[g_{a \in A_s} : R(s,a) + \gamma \sum_{s'} P_a(s,s') V_n(s') \right]. \tag{6}$$

Then update the value of V by solving the matrix game:

$$V_{n+1}(s) = \text{NE}[G_s(V_n)], \tag{7}$$

where NE is the expected value of playing the matrix game's Nash equilibrium strategy. By iteratively updating V until convergence, as with a MDP, the optimal value functions and actions for each player can be calculated.

POMDPs

POMDPs are an extension of MDPs that allow for uncertainty about the current state. Like normal MDPs, there can be uncertainty about which state will follow an action, but in POMDPs, the subsequent state is not directly known. Instead, it must be inferred from observations.

A POMDP extends a MDP with a set of observations, Ω, and a set of observation probabilities O. As in a MDP, after every action $a \in A$, the state switches from $s \in S$ to a new state $s' \in S$, according to the transition probabilities $P_a(s,s')$, and the agent receives a reward $R_a(s,s')$. Then, the agent receives an observation $o \in \Omega$ which gives information about the new state s' according to the observation probabilities: $O(o|s,a)$.

POMDPs can be converted into belief-MDPs, where the state becomes a probability distribution b over the states S in the original POMDP and the probability of being in state s is given by $b(s)$. This is now a fully observable MDP because the state (the probability distribution over states) is always known.

After making an action a and getting an observation o, the belief about the new state is updated using Bayes's rule:

$$b_o^a(s') = \frac{O(o|s',a)}{\Pr(o|a,b)} \sum_{s \in S} P_a(s,s') b(s) \tag{8}$$

normalized by $Pr(o|a,b) = \sum_{s' \in S} O(o|s',a) \sum s \in S P_a(s,s') b(s)$.

The Bellman equation for POMDPs is:

$$V_{n+1}(b) = \max_{a \in A} \left[\sum_{s \in S} R(s,a) b(s) + \gamma \sum_{o \in \Omega} Pr(o|a,b) V_n(b_a^o) \right], \tag{9}$$

where the optimal value function, V^*, is found when $V_{n+1} = V_n$ for all belief points.

A policy for POMDPs is now a function $\pi(b)$ over the continuous set of probability distributions over S. The optimal policy function π^* specifies the optimal action to take for each belief b.

Approximate solutions to POMDPs

The value functions of POMDPs can be approximated by iteration, similar to normal MDPs. Here we give a brief explanation of how POMDPs can be approximated, and of the PERSEUS algorithm [35] for computing approximate solutions. A much more detailed description of POMDPs and of the algorithm is available in [35].

The value function for a POMDP is piecewise linear and convex (PWLC) for finite horizon problems [36] and a PWLC function can approximate the value function of an infinite horizon problem arbitrarily well.

This means that we can divide the belief space into regions, partitioned by a set of vectors. For iteration n, the value function V_n is composed of a set of vectors $\{\alpha_n^i\}, i = 1, \ldots, |V_n|$. The value of a belief point b is determined by the vector that maximizes that region of the belief space:

$$V_n(b) = \max_{\{\alpha_n^i\}_i} b \cdot \alpha_n^i. \quad (10)$$

Each vector has an associated action, which is the action that maximizes the discounted future reward. To decide which action to take a certain belief point, find the maximizing vector for that point, and then take the associated action.

To perform an iteration, the vectors for each belief point are updated using the backup operator, $\alpha_{n+1}^b = \mathbf{backup}(b)$, which calculates the vector from the next iteration that maximizes b:

$$\mathbf{backup}(b) = \arg\max_{\{g_a^b\}a\in A} b \cdot g_a^b, \text{ where} \quad (11)$$

$$g_a^b = r_a + \gamma \sum_o \arg\max_{\{g_{a,o}^i\}_i} b \cdot g_{a,o}^i, \text{ and} \quad (12)$$

$$g_{a,o}^i(s) = \sum_{s'} O(o|s',a) Pr(s'|s,a) \alpha_n^i(s'). \quad (13)$$

The challenge of POMDPs is finding the smallest set of belief points that are needed to calculate each of the vectors in the value function. Exact solutions must find all of these vectors, and for complicated problems the required number of belief points makes computing a solution intractable. Algorithms for approximate solutions attempt to limit the number of belief points, making computating a solution more feasible.

The PERSEUS algorithm [35] is one such algorithm, which uses a fixed set of (randomly sampled) belief points, and attempts to minimize the number of vectors used to approximate the value function.

Initially, a set B of belief points is sampled by randomly taking actions and exploring the belief space of the problem. Then, the initial value function V_0 is set to a single vector. For each iteration, the algorithm improves the value function using the backup stage listed in Fig. 3. This randomly selects belief points and calculates the new vector for that point; if the vector also increases the value of other belief points those points are then excluded from this iteration. Then, if there are any remaining belief points, one of those is selected and the process repeats. This means that the value of all of the belief points will increase, but the number of vectors tends to remain smaller.

The number of vectors used to approximate the value function increases over time, as the approximation becomes more accurate. As the number of vectors increases, so does the time it takes to compute each iteration. The iteration continues until some arbitrary condition, such as a rate of convergence or the time taken, is met.

Approximate solutions to POSGs

POSGs are an extension of stochastic games that allow, similar to POMDPs, uncertainty about state. However, because POSGs have multiple players, they are much more complicated. In the standard formulation, each player can have different rewards, actions and observations. It is this last item that brings a lot of the complexity: because each player receives different observations, each player potentially has different beliefs about the state. In order to account for this, each player must therefore also maintain beliefs about the other players.

POSGs are an active area of research, and presently, it is possible only to solve finite-horizon problems to a limited depth. Finding optimal allocations of time to network defence tasks can potentially be a much more complicated task in terms of the number of states, actions and observations than those currently studied for POSGs. In order to make it computationally tractable, and to allow us to compute approximate solutions for infinite-horizon games, we use a simplified version of the POSG. In this class of POSGs, both players receive the same observations and so share the same belief about the state of the world. This has some implications for the results, which will be discussed later in Section 8, but the simplification allows the problems to be solved in a manner similar to POMDPs.

To do this, we implemented the PERSEUS algorithm in the Julia programming language [37], and then modified it to find approximate solutions to this class of POSGs.

The changes add separate rewards and actions for each player. While each player has their own actions, the transition probabilities are based on the joint actions of all players. A value function $V_p(b)$ is estimated for each player $p \in P$. For each iteration, the value of $V_{p,n+1}$ is calculated for each player using the same randomized, point-based strategy as in the PERSEUS algorithm.

The major difference is in the backup stage: $\alpha = \mathbf{backup}(b)$ no longer selects the α vector that maximizes the value of b, but, similar to the approach for normal stochastic games described in Section 4.2, returns the vector corresponding to the Nash equilibrium value.

If the equilibrium is a pure-strategy equilibrium then the vector returned corresponds to the joint equilibrium action of the players. If the equilibrium is a mixed-strategy equilibrium, then the vector is the average of all of the vectors corresponding to the joint actions that make up the equilibrium, weighted by their probability of being played.

In PERSEUS, each vector has an associated action. Here, the equilibrium probabilities of playing each action are used instead. Thus, a policy $\pi_p(b)$ for player p at belief point b specifies the probabilities with which to perform each action.

Experimental setup

To find a policy for time allocation, it is first necessary to create a POSG that describes the network, vulnerabilities, players and costs. Each game is described using matrices that represent the transition probabilities from state to state for each action, matrices that represent the probabilities of seeing each different observation from each state after each action and matrices containing the reward values players receive when moving to each state after each given action.

1. Set $V_{n+1} = \emptyset$. Initialise $\tilde{B}$ to B.
2. Sample a belief point b uniformly at random from $\tilde{B}$ and compute $\alpha = \mathbf{backup}(b)$.
3. If $b \cdot \alpha \geq V_n(b)$ then add α to V_{n+1}, otherwise add $\alpha' = \arg\max_{\{\alpha_n^i\}_i} b \cdot \alpha_n^i$ to V_{n+1}.
4. Compute $\tilde{B} = \{b \in B : V_{n+1}(b) < V_n(b)\}$.
5. If $\tilde{B} = \emptyset$ then stop, else go to 2.

Figure 3. The backup stage of the PERSEUS algorithm (from [35]).

We want to be able to find policies over a wide range of possible configurations and parameters, so we implemented code that takes a description of the network, vulnerabilities, and players and automatically creates the matrices required for the POSG solver.

States

The network is composed of a number of nodes, which represent devices on the network. Each node has the following properties:

- Whether or not it is externally attackable.
- The other nodes from which it can be attacked.
- The vulnerability lifecycles that affect the node.
- The cost to the defender if the node is compromised.
- The reward to the attacker if the node is compromised.
- The cost to recover the node.
- The probability of an attack on the node being successful.
- The probability of discovering the node is compromised by an unknown exploit when monitoring.

The vulnerabilities represent the lifecycle of vulnerabilities for a particular piece of software. Each vulnerability also has several properties:

- Exploits developed per year.
- Mean time to disclosure.
- Mean time to patch availability.
- Probability of a new exploit being undisclosed.

The number of states in a POMDP or POSG has an impact on the length of time it takes to compute a policy; with too many states, the computation can become intractable. We therefore want to express the essential parts of the vulnerability lifecycle and the state of the system (whether or not it is compromised) using as few states as possible. Figure 4 shows the states used to represent each vulnerability for a node. This shows the transitions that occur as a result of the vulnerability lifecycle (the dotted lines in the figure), and the transitions that are caused by an action by the players (the solid lines). The states from the vulnerability lifecycle used here are those shown with dotted borders in Fig. 1; this is not the whole lifecycle, but captures the most important parts: known and unknown (undisclosed) exploits, as well as patching.

The transition probabilities for the vulnerability lifecycle are determined by the properties of the vulnerability: the number of exploits per year, the mean time before disclosure and patches, and the probability of an exploit being undisclosed (unknown). The transition probabilities for actions are determined by properties of the node: the probability of a successful attack, and the probability of monitoring. Naturally, these depend on the current state, too; for example, attacking when there is no vulnerability has zero probability of success.

The states also capture whether or not the system has been compromised. These are the shaded states in Fig. 4. The attacker's attack action can move to a compromised state, and the defenders recover action can restore the system to an uncompromised state. The defender's monitor action changes the state from unknown and compromised to disclosed and compromised. The patch action changes the state from patch available to done, if the system is not compromised, and to patched if the system is compromised.

When a vulnerability has been patched, and the system is not compromised by that vulnerability, the state transitions back from Done to Inactive, allowing a new exploit to be developed. To model multiple possible exploits at the same time, multiple vulnerabilities must be used for each piece of software modelled.

Each vulnerability represented in the game requires each of the states in the figure. To create a model with multiple nodes and multiple vulnerabilities, the transition matrices for each vulnerability are created and are then combined together using the Kronecker product. This results in large matrices that represent the transition probabilities for all the different combinations of states from all of the vulnerabilities.

Actions and observations

For each problem below, the system administrators have a set amount of time which they can allocate to tasks. There are three defensive tasks, monitor, patch, and recover, and another task that represents spending time doing other things. The three defensive tasks are actually per node: you can monitor node 1, or patch node 2, for example.

Each of these actions has a time requirement. For the experiments below, monitoring takes one hour, and patching and recovering take two. In terms of the POSG, an action is actually any combination of these per-node actions (and of doing other tasks) that adds up to the total number of hours available. For example, one action is to spend all of the time doing other tasks. Another action, assuming there are four total hours available, is to monitor two-nodes and patch a third. However, each node can only be in an

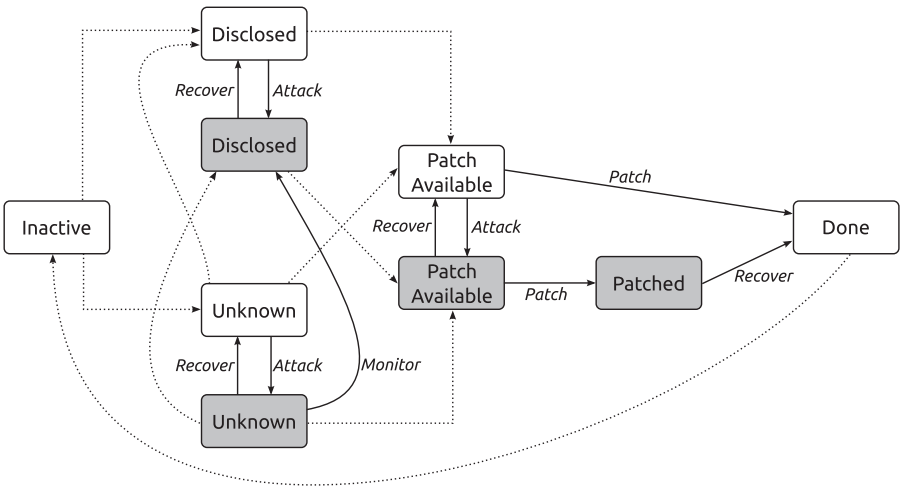

Figure 4. States, actions and transitions for a vulnerability. Solid lines show transitions that are caused by an action by either the attacker or the defender. Dotted lines show transitions that occur automatically, as part of the vulnerability lifecycle. States where the system has been compromised are shown filled in.

action once: it is invalid to monitor and patch the same node in a single action. This constraint makes it possible to calculate the transition probabilities.

The actions for the attacker are simpler: they can do nothing, or attack one of the nodes using a specific vulnerability. The attacker can only perform one action each turn.

After each action, the players receive an observation. There are seven basic observations:

1. Disclosed.
2. Patch Available.
3. Disclosed—Bad.
4. Patch Available—Bad.
5. Patched—Bad.
6. Done.
7. Nothing.

The probabilities with which the players receive the observations depends on the state and the action taken. With this information, they update their beliefs about the current state of the environment.

The Bad observations are only received when the defender uses the monitor action. The other observations occur with probability 1, when in the corresponding state. The observations are per-vulnerability. When more than one node or vulnerability is used, the full set of observations reflect all possible combinations of observations from the individual vulnerabilities.

There are no observations for the inactive and unknown states, because there is supposed to be uncertainty about whether or not an unknown exploit exists.

Because the attacker and defender share observations in the reduced version of POSGs that we use, there are no observations informing the attacker about the success of attacks as this would also inform the defender.

Implementation details

The matrices describing the problem can be extremely large, requiring a lot of memory. As the number of vectors grows, the amount of memory and CPU time required to compute the new vectors for the next iteration grow considerably. Larger problems (using 4 nodes) consumed more than 32 GB of RAM after a small number of iterations.

The solution we found to this was to parallelize the computation of the backup across a number of nodes in a cluster. Since the algorithm is written in Julia [37], this was relatively easy to implement. The backup calculations and values are spread across the worker nodes, meaning the per-node memory requirements are reasonable. The main node controls the others and computes the Nash equilibria using the values returned from the worker nodes. We wrote a wrapper around the equilibria solver described in [38], allowing it to be called directly and quickly from the Julia code.

For the three-node problems described below, this distributed approach made computation feasible. However, calculating a reasonable number of iterations still took several days, using 20–25 worker nodes. The initial iterations, with a low number of vectors, were relatively quick. Later iterations took upwards of one hour each to compute. Figure 5 shows the number of vectors used to represent the value functions for the attacker and defender in the two-node problem, averaged over 20 runs. The three-node problems have similar growth, although the number of vectors for the defender's value function tends to stay lower for longer.

Examples

In this section, we present a study of a number of different scenarios to highlight certain aspects of the model. First, we present a two-node model to identify how the optimal strategy is developed. We then compare a number of configurations of three-node networks, where we are looking at how the difference in certain parameters affects the generation and execution of the optimal strategies.

Two-node model

The initial case presents a simple network layout where we represent an outward facing server and main internal server of an organization. The outward facing device does not contain sensitive information, whereas the main internal server contains data that are considered to be sensitive in nature and valuable to a determined attacker.

We consider that the cost of compromise for the defender at the external node and internal node to be 100 and 100 000, respectively. For the purposes of this simulation, the determined attacker we consider is only interested in the data held on the internal server, giving values of 0 and 100 for successfully compromising the two devices.

For the purposes of all the simulations presented in this article, we consider that the defender has a time cost of 2 units to perform the actions of Patching and Recovery, and a cost of 1 unit for monitoring. The attacker has a uniform cost of 1 unit to perform any action. Additionally, we consider that the act of recovery for the

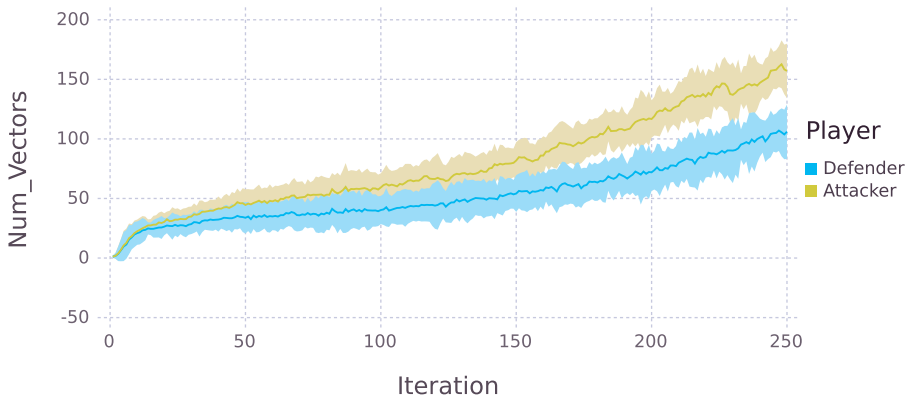

Figure 5. Number of vectors approximating the value functions of the attacker and defender at each iteration. Average over 20 runs, with 5–95% confidence intervals.

defender has some financial cost, which in our configuration is given a value of 40.

The results in all of the tables and graphs are obtained by simulating the policy that was calculated for that scenario. A policy is simulated by selecting an initial starting state and belief over the states, and then performing the action indicated by the policy. There is a policy for both the attacker and the defender, and their joint actions are used to determine the transitions from state to state of the underlying system. Table 7 shows the average profit and average allocation of hours to tasks for 10 000 simulation runs starting from random initial states. Each simulation lasts for 365 steps, representing the policy's performance over a year. Table 8 shows the same figures, but always from the same starting state, where the system is initially uncompromised. The first table gives an idea of the performance of the policy over all states, while the second gives a perhaps more realistic estimate of the expected loss: it is likely that the system is uncompromised when you start following the policy.

We have tested the simulation on this simple two-node configuration, in order to clearly identify the behaviour of the defender. We can see in Fig. 6 that the schedule generated reduces the expected loss of the organization to a minimum in more than 25% of cases. There is then a fairly even distribution of loss with a spike in cases at the highest levels of loss. This seems to indicate that the schedule is either highly effective in the scenario or provides little to no defence at all.

The details of the schedule, presented in Table 7, indicate that frequently the defender chooses to perform unrelated task, and this indicates that the defender believes that the system is in a secure state and that there is no reason to act to protect the network. Second to this, the defender recovers both nodes 1 and 2, with more emphasis placed on node 1, since all viable attacks must first compromise that node, and this is why we see some monitoring on that node, although the frequency of this action is very low.

Three-node model

An advancement of the two-node network, introduces an interim firewall that is placed between the outward facing server and the main internal server. This is depicted in Fig. 7.

In this case, we consider a defender that is concerned primarily with securing the data that is held on the main internal server similar to the two-node case presented before. However with the inclusion of the firewall, the defender has some loss associated with the firewall becoming compromised.

In this case, we represent what we term a data driven attacker, whose primary interest lies in the data held on the defender's main internal server. As such the attacker gets no reward from compromising either of the external two devices and only gets a reward for a successful compromise of the main internal server. The payoffs for this general case are given in Table 1.

Number of administrators

One of the key comparisons that we want to make is how the number of hours available to the defender. In this scenario, we are interested in how the strategies for the defender change if we increase the number of hours available to the defender.

If we consider that the allocation defined in the initial three-node case is representative of the actions available to two system administrators across a day, then we are first interested in what an increase in personnel looks like in terms of the optimal strategies. In this case, we consider the organization to have an additional system administrator, increasing the available allocation for the system administrators time from 4 to 6 units.

Alternative players

In addition to the number of hours, we need to consider that there might be different kinds of requirements that impact the requirements of the players. In our model, this is represented by a change in the payoffs for the players.

The first alternative player we consider is a defender who is in a highly regulated industry, where even a small breach of defence is met with heavy fines from regulators or representing the risk of potential legal action. This increases the loss for the defender across the whole of the system.

In this first case, we consider that the attacker is the same data-driven attacker that was represented in the previous cases. The payoffs for this case are shown in Table 2.

We considered earlier defenders that are heavily invested in the value of the data that is held on their internal systems, and we now want to consider a smaller SME type entity. We consider that while the defender in this case has a reason to defend their data, the value

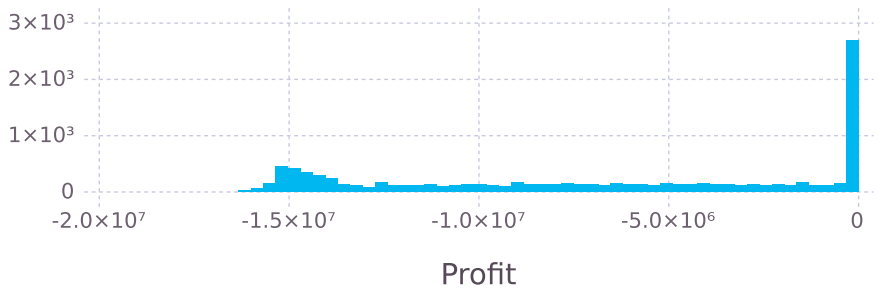

Figure 6. Histogram of profit from 10 000 simulations of the two-node policy.

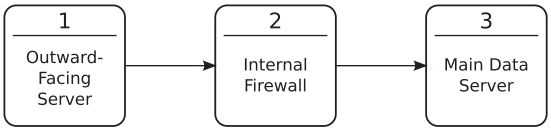

Figure 7. Three-node network diagram.

Table 1. Payoffs for a three-node case for a general use case

Player	Node 1	Node 2	Node 3
General Case Defender	10	100	100 000
Data-Driven Attacker	0	0	100

of lost data from the internal servers is not as severe as for the previous defender. The payoffs are given in Table 3.

As with the Highly-Regulated Defender, the attacker in the SME case is considered to be a data-driven attacker that we have been comparing against to this point.

So far we have only considered a single kind of attacker, one who is interested only in the data that the defender has on their internal servers. In this case, we consider an attacker that is interested in their reputation, as such the attacker gets a large reward for compromising the outward facing server. However, the attacker gets a diminished reward for compromising the internal network, because the data held have little value to the attacker. The payoffs for this case are shown in Table 4.

In this case, we consider the defender to be the defender that we initially considered of the three-node model.

Alternative network configurations

So far we have considered only a simply connected network; however, most networks are highly interconnected. The final cases we present in this work represent potential sub net configurations of a larger network.

In the first alternative network configuration, we consider a set-up with two outer nodes both connected to a single internal node. In this configuration, we identify that one of the outer devices is more prone to software vulnerabilities, increasing the likelihood that a new vulnerability will be discovered in that node. This configuration is shown in Fig. 8.

In this case, we consider that the defender should put more emphasis on protecting the more vulnerable external node, while minimizing the risk that the internal node is compromised by other means. These payoffs are given in Table 5.

The second network configuration we consider a single outer node that connects to two inner nodes. Both of the internal nodes in this case are more valuable than the outer node, but one of the internal nodes is worth considerably more than the other. This is shown in Fig. 9.

It is likely that the attacker will focus on the highly valuable node, so we expect that the defender will put a much higher emphasis on ensuring that the more valuable target is not compromised. The payoffs are shown in Table 6.

Table 2. Payoffs for a three-node case with a Highly-Regulated Defender

Player	Node 1	Node 2	Node 3
Highly-Regulated Defender	20 000	20 000	100 000
Data-Driven Attacker	0	0	100

Table 3. Payoffs for a three-node case with an SME like entity

Player	Node 1	Node 2	Node 3
Highly-Regulated Defender	10	100	10 000
Data Driven Attacker	0	0	100

Table 4. Payoffs for a three-node case with a reputation-driven attacker

Player	Node 1	Node 2	Node 3
General Case Defender	10	100	100 000
Data-Driven Attacker	80	10	50

Results

The following section details the results obtained from the experiments above, first comparing the results of various similar three-node networks along with a look at the behaviour of alternative network layouts. To do this, we first need to analyse the results for the three-node case. The results presented are based on 10 000 runs from a randomized initial network vulnerability state. The strategy space for each of the

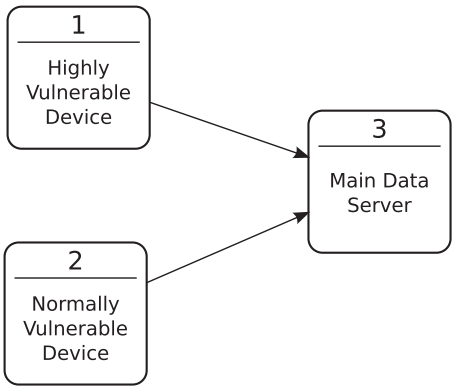

Figure 8. Diagram of the three-node network for the simulation with a single more vulnerable node.

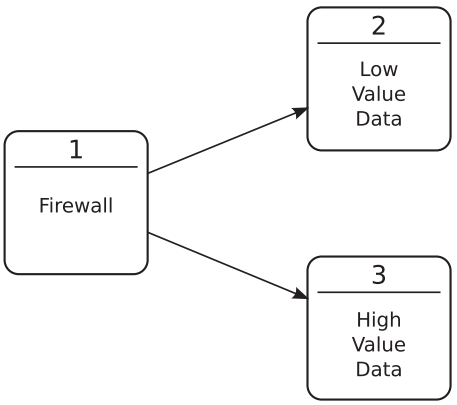

Figure 9. Diagram of the three-node network for the simulation with a single more valuable node.

Table 5. Payoffs for a three-node case with a single highly vulnerable node

Player	Node 1	Node 2	Node 3
General Case Defender	10	10	100 000
Data-Driven Attacker	0	0	100

Table 6. Payoffs for a three-node case with a single highly valuable node

Player	Node 1	Node 2	Node 3
General Case Defender	10	1000	100 000
Data-Driven Attacker	0	20	100

different experiments can be seen in Table 7, with all the graphs presented relating to these solutions. We present the results generated when the strategies are played from a clean system state in Table 8.

These tables and graphs represent the performance of the policy for each scenario, but do not actually describe what the policy is. They give an indication of which actions the policy suggests, and how well it prevents loss, but the actual policy itself is hard to express. The policy associates actions with regions of belief-space; the results here naturally depend on which regions of belief space are reached during the simulations. However, by simulating each policy a large number of times, it is likely that the results give a fairly accurate depiction of the performance of a policy.

One of the first things we can see in comparison to the two-node model is how much lower the expected damage is in the three-node model. The addition of an extra layer of security between the attacker's initial position and the data, makes it more difficult for the attacker to succeed in breaching the system to steal the data. While the attacker does get an interim payoff for breaching the second layer, this is not in line with the payoff the attacker gets for a two-stage breach in the two-node model.

From Fig. 10, we can see in general that the majority of test cases give an expected loss of close to 0. In each of the cases, we can attribute these initial low values with strategies where the attacker has not been successful in breaching the defences of the system beyond the outermost layer and that much of the loss is associated with penalty for not performing other related tasks. These low level results account for more than 45% of cases sampled, where the remaining 55% is spread with among the remaining cases with nearly 20% of the samples in cases where there are only minor breaches to the system.

We can see from Table 7 that the defender places a higher emphasis on node 1 than any of the other nodes. This is to be expected, since we know that there is a single attack path that must pass through this node, therefore if this node is kept in a stable state then the defender has to be concerned less about the more valuable components becoming compromised. This is particularly noticeable in terms of recovery, and the defender significantly reduces the amount of actions taken in recovering the node further into the system.

Number of administrators

When we increase the number of administrators, we see that as expected we get a reduction in the expected damage across the system. We see that there is an increase in monitoring and recovery at node 3 and a reduction in recovery at nodes 1 and 2 and monitoring at

Table 7. The profit and hours allocated to each task over a whole year, averaged over 10 000 simulations using random start states

Name	Avg Profit	Monitor 1	Monitor 2	Monitor 3	Patch 1	Patch 2	Patch 3	Recover 1	Recover 2	Recover 3	Other	% Other
Two-Node	−661.14	4.33	0.28	—	0.52	1.51	—	314.02	205.23	—	934.11	63
Three-Node	−79.56	18.18	5.33	5.48	4.72	1.13	1.33	170.02	86.54	28.99	1138.28	78
More Time	−50.63	17.94	0.00	17.42	0.00	1.79	0.63	60.15	34.15	38.93	2018.99	92
Regulated	−136.83	1.14	0.98	0.00	0.78	1.80	3.23	92.26	57.25	18.13	1284.44	88
SME	−1.49	31.04	1.54	1.97	0.22	7.71	1.59	65.77	53.06	19.57	1277.52	87
Reputation	−19.86	24.41	29.75	26.53	0.29	2.09	0.00	161.48	114.01	4.61	1096.82	75
High-Vuln	−537.99	0.25	0.00	0.25	1.32	1.76	1.89	157.99	14.23	145.56	1136.76	77
High-Value	−559.50	4.87	0.09	4.83	0.94	0.00	0.09	161.43	132.28	152.92	1002.57	69

The final column gives the percentage of time allocated to non-security tasks.

Table 8. The profit and hours allocated to each task over a whole year, averaged over 10 000 simulations, starting from an initially-uncompromised state

Name	Avg Profit	Monitor 1	Monitor 2	Monitor 3	Patch 1	Patch 2	Patch 3	Recover 1	Recover 2	Recover 3	Other	% Other
Two-Node	−312.52	8.92	0.39	—	0.51	1.30	—	193.53	76.34	—	1179.01	81
Three-Node	−39.66	4.58	0.00	0.01	3.59	0.56	0.84	109.52	46.00	10.07	1284.82	88
More Time	−18.11	18.38	0.00	18.17	0.00	1.06	0.65	24.94	12.01	10.61	2104.18	95
Regulated	−65.71	0.57	0.48	0.00	0.28	0.90	1.86	43.30	28.16	6.37	1378.07	94
SME	−0.56	12.27	0.11	0.29	0.06	2.31	0.88	29.69	23.11	4.58	1386.70	95
Reputation	−3.84	7.61	30.15	9.04	0.06	1.06	0.00	108.53	39.43	0.54	1263.58	87
High-Vuln	−453.88	0.19	0.00	0.19	0.54	1.09	0.99	192.03	7.83	110.72	1146.42	78
High-Value	−381.89	7.17	0.24	3.53	0.65	0.00	0.24	160.34	85.41	100.16	1102.25	75

The final column gives the percentage of time allocated to non-security tasks.

Figure 10. Histogram of profit from 10 000 simulations of the basic three-node policy.

node 2. It should be noted that many of the additional hours available are spent performing other non-security related tasks, with almost double the number of hours being spent on this over the 4 unit model. This shows that there is some inefficiency to having too many administrators or that in general there are not enough effective security tasks to warrant the loss from not performing the required daily tasks.

In terms of distribution of loss, we can see in Fig. 11 that in comparison to the 4 unit model, the 3 unit model has a much larger distribution in the lowest damage bracket, with more than 65% of simulations causing a loss in this range. We can also see that there is a far smaller spread of the loss.

Alternative players

In each of the first two cases below, we are interested more in the change in strategy generated by the defender in relation to the above case, since many of the scenarios the structure of the payoffs for the defender makes the expected loss much less important in the comparison.

In the case of the Highly-Regulated Defender, we naturally see the expected loss to be much higher than that of the initial defender, since they lose a large amount even from a minor breach.

In terms of distribution of damages, we see in Fig. 12 that there is a smoother gradient to the distribution. We see a relatively smooth gradient from the lowest damage levels down to the extremely high values of damage. The gradient is likely to be more exaggerated than for the original defender, based on the increase in damage from minor breaches.

Of all cases tested, this is the defender that places the most emphasis on patching Node 3. However, it seems uncharacteristic of the defender to also place a high emphasis on non-security related activities when the ratio of loss from a failure to perform unrelated activities to a breach is so much higher. This is in contrast to the SME Defender, where a similar amount of emphasis is placed on performing unrelated tasks. Both of these cases put considerably more effort into these unrelated security tasks than the original defender.

For the SME Defender, we see that their average expected loss is much lower than that of the original defender, which again is to be expected given their lower losses from being breached.

We see a very similar trend in Fig. 13 for the SME defender as we did for the Highly-Regulated Defender; however, we see the trend because the value ranges presented are smaller than those associated with the Highly Regulated Defender. This likely gives a more accurate description of results at the low damage end, where we will still see that while the majority of these cases still see no breach, there are still minor breaches that occur that have next to no impact on the organization.

Figure 11. Histogram of profit from 10 000 simulations of the three-node policy, with additional time available to allocate.

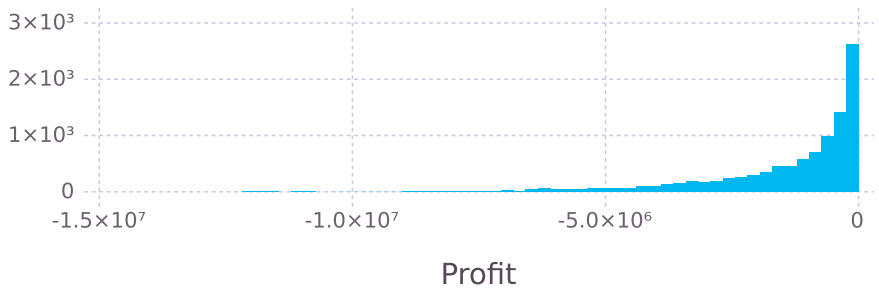

Figure 12. Histogram of profit from 10 000 simulations of the policy from the highly-regulated scenario.

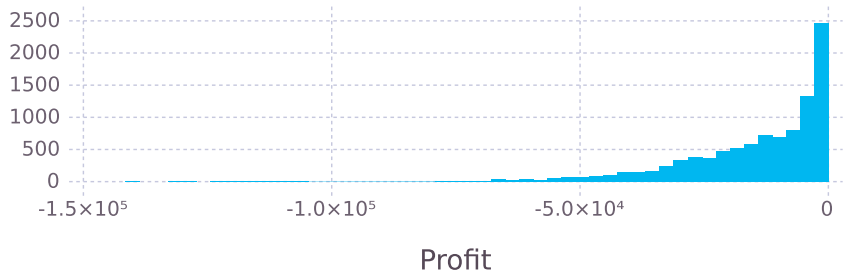

Figure 13. Histogram of profit from 10 000 simulations of the policy from the SME scenario.

The final alternative player we considered was the Reputation Driven attacker. In this case, we can more directly compare the payoffs of the defender since in this case only the payoff of the attacker have changed from the initial three node case.

We can clearly see in Fig. 14 that with a different attacker, the defender is able to minimize the damage of more than 90% of attacks. This is achieved, because the defender is placing a heavy emphasis on recovering the state of the first two-nodes, noting that the attacker is not gaining as much by attacking deeper. This recovery ensures that the key data storage of the company is rarely at risk of attack. It should be noted that while the defender is seen as generally secure in this case, we can expect a number of minor breaches that while not impactful to the defender will yield the attacker a high reward.

Alternative network configurations

For the different network designs, we need to take their behaviour in isolation, since the results are not directly comparable to the initial three-node network. This is given that we know that the defence for the most valuable node is weaker.

We can see that for a single highly vulnerable node in Table 7 that in comparison to the other nodes almost no consideration is given by the defender to monitoring and recovery of the less vulnerable outer node. This is in line with the prediction, where the defender has to focus attention on the most vulnerable or valuable parts of the network. Although the results show that an equal emphasis is placed on patching across all nodes.

For the more valuable node, we see that the defender chooses to recover all of the nodes with a high frequency. However, monitoring is only performed on the outer node and the more valuable node. The solution for this scenario features almost no patching at all, and this is possibly due to the ineffectual nature of patching once an attack is in progress and the relative short chain of attack. This indicates that the defender is more concerned about protecting the valuable node and while will recover the less valuable inner node, they are not willing to invest more resources in protecting it as along with the prediction the attacker is less willing to attack that node.

One thing we can see about both of these cases, is that unlike the initial three node network, where emphasis was placed primarily at the external node, in both these cases a higher level of emphasis is placed on the inner most node. These results are more consistent with a two-node model, where there is less layers of security protecting the most valuable data. It should be noted that both alternative network designs performed better than the tested two-node network case under the same conditions, where the added component is enough to cause some change in attacker strategy from a simple two-node network, which then causes less damage to the organization as the attack effort is spread over more targets, some of which are easier to protect and others that are less valuable.

In both Figs 15 and 16, we see that the distribution of expected loss is more closely associated with a two node network. In these

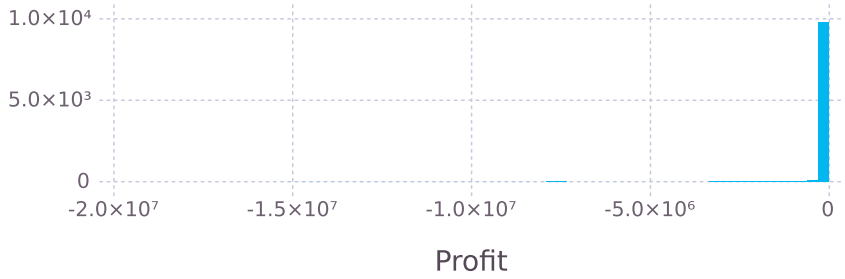

Figure 14. Histogram of profit from 10 000 simulations of the policy from the reputation-driven attacker scenario.

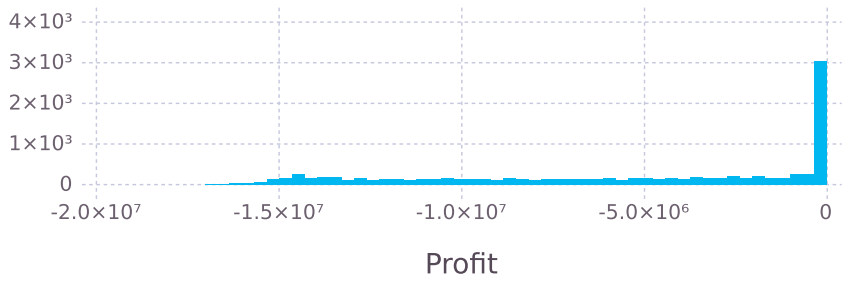

Figure 15. Histogram of profit from 10 000 simulations of the policy from the highly-vulnerable node scenario.

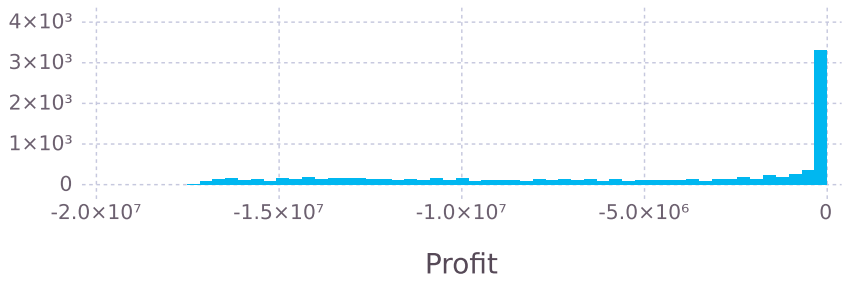

Figure 16. Histogram of profit from 10 000 simulations of the policy from the high-value node scenario.

cases, we see that after the initial spike of completely defended scenarios a fairly even distribution of scenarios representing a fairly even distribution of attacks. In both cases, only a little more than 30% of cases successfully repel all serious attacks.

Discussion

Across all of the different configurations, we find that there is an emphasis placed on performing non-security related tasks, where many of the solutions propose spending between 75% and 90% of time on these tasks. In the case where there is more time available for tasks because of the inclusion of an additional system administrator, the percentage of time spent on non-security related tasks exceeds 90%. There are two considerations for why non-security related tasks dominate the distribution of system administrator time: the cost of ignoring non-security tasks and the lack of available effective security tasks.

We see throughout all of the results that outside of non-security related tasks, the most amount of time taken by the system administrators is spent on recovery. A large part of this is due to the fact that there is no action that allows the system managers to mitigate against an exploit. If there is an exploit without a patch, recovering a system removes the attacker, but does not prevent immediate re-compromise. Thus, the manager is forced to recover multiple times before a patch arrives. Given that the current simulated environment does not allow for workarounds for the lack of official patch, recovery on compromise provides the most efficient method for reducing the amount of damage, since it means that the system will be returned to a normal state, requiring the attacker to spend resources attacking again. Future work will consider the possibility of alternatives to official patches, which should reduce the emphasis on repeated recovery as the most effective defence prior to the release of an official patch.

In this work, we placed a large emphasis on performing tasks that are not related to security, penalizing the defender for spending time on security. While, for some organizations, this high emphasis on non-security related tasks may be critical, there are others that would not be impacted so greatly. With this in mind, we consider that this is something that we would like to further address, identifying scenarios for lower importance system tasks. Part of this is that here we consider that each of the administrators is hired as a general system administrator rather than a specified security administrator. It could be the case that a dedicated security manager would not be penalized in the same way, opting to perform security related tasks over un-related tasks.

While a reduction in emphasis on non-security related tasks through lower penalties may reduce the necessity for them to be performed, it does not reduce the amount of time that there is a viable security related activity to be performed. While patches can only be applied on creation, both monitoring and recovery can be performed at any time. However, recovery is only a viable option in the case that there is belief about a compromise, since it makes little sense to spend limited resources on recovery of a system that is not believed to be compromised. So provided the option to patch is taken as soon as it is available, and that recovery is performed when considered necessary, which would provide the most optimal reduction in expected damage, the trade-off then exists between choosing either monitoring or unrelated tasks. More generically, the trade-off is between possible every day security activities and non-security related tasks.

It is here that the magnitude of the penalty for not performing other tasks becomes most relevant. With a penalty that is greater than the benefit of every day security tasks, the optimal policy will always favour the loss minimizing strategy, which is to generally perform non-security related tasks unless a key action such as recovery and patching is available.

Looking at the results, following the policy results in minimal loss to the organization in most cases. In the cases where the loss is greater, the vulnerability lifecycle has produced a series of vulnerabilities that cause the system to be weak to attack, which persists because patches for those vulnerabilities take a long time to arrive.

The issue with patch prominence is also reflected in the low values that are seen for patching across all the results in Table 7, since there are potentially too few patches being generated on average to consider patching as a high priority. We aim to expand on our work to consider this in the future, such that we can consider a custom fix or workaround for the vulnerability. This custom fix would eliminate the vulnerability at a high cost to the organization; however since it would not come from the software vendors, there is a probability that it would be unsuccessful. In addition to this, we also want to consider things like enforced regulations and policies derived from the management of an organization.

It is also interesting to note in many of the network configurations the greater amount of time spent on monitoring the node that is externally attackable. The emphasis on monitoring Node 1 relates to the most simple way for a system to be able to verify if they can be attacked. This is due predominantly to the design of the test network. In the case of the simple three node network that is connected in a linear manner, the attacker must launch an attack from Node 1, so in order for any attack to be successful, it must pass through that location. This is being used as an effective shortcut for the defender to understand the potential state of the network without investing additional resources. Additionally, we see that in the High Vulnerability case, where there are two nodes that can be attacked from an external source, the prevalence of monitoring on the outermost nodes is reduced to near zero levels. In this case, the information gained from monitoring the external nodes is not as useful to the defender. As a result, the High Vulnerability case more frequently recovers both the highly vulnerable external node and the valuable internal node.

The use of shared observations between the attacker and the defender is a limitation of this approach. In reality, an attacker would likely have more knowledge than the defender about undisclosed exploits and about whether or not a device on the network is currently compromised. However, including such knowledge into the model would have changed the type of structure required to represent the problem and made the computation intractable. The actual impact of this on the results is likely to be small: the attacker would probably be more likely to launch some attacks when the defender still assigns a very low probability to the existence of an unknown exploit, and the defender would probably monitor slightly more to compensate.

Conclusions and future work

We have presented a new model for thinking about network defence that captures the decisions network administrators need to make about how to allocate their time.

We represent the problem as a POSG, which can be solved to give a policy that describes the optimal action to take for any particular belief about the current state of the system. We used a number of examples to illustrate the approach, and showed how different types of attacker or defender and different network configurations result in different policies.

There is a trade-off between the security and non-security activities of system administrators, and we have shown that it is important to optimize the time to perform security tasks only when they are required. Further work will look at the impact that the cost of ignoring non-security tasks has on the decision to perform other tasks over security related tasks, including a study of the effect of frequency of patch availability on the optimality of security decision making.

We have looked at a relatively simple network model, consisting of only two or three nodes, an advancement of this work would be to consider a larger model. This approach is successful for problems of this size, but larger problems have exponentially higher computational and memory requirements. In order to do this we will need to look at improving the efficiency of the approaches we have used, by using newer techniques for solving POMDPs or finding more compact ways to represent the problem.

Funding

This work is funded by EPSRC grant EP/K006517/1, "Productive Security", and EPSRC grant EP/K005790/1, "Games and Abstraction".

Conflict of interest statement. None declared.

References

1. Schneier B. Managed security monitoring: closing the window of exposure, 2000.
2. Frei S, May M, Fiedler U *et al*. Large-scale vulnerability analysis. In: *Proceedings of the 2006 SIGCOMM Workshop on Large-Scale Attack Defense*. ACM, 2006, 131–8.
3. Beres Y, Griffin J, Shiu S *et al*. Analysing the performance of security solutions to reduce vulnerability exposure window. In: *Computer Security Applications Conference, 2008. ACSAC 2008. Annual*. IEEE, 2008, 33–42.
4. Ioannidis C, Pym D, Williams J. Information security trade-offs and optimal patching policies. *Eur J Oper Res* 2012;**216**:434–44.
5. Arora A, Telang R, Xu H. Optimal policy for software vulnerability disclosure, arbeitsbericht der h. john heinz iii school of public policy and management, 2004.
6. Arora A, Forman C, Nandkumar A *et al*. Competition and patching of security vulnerabilities: an empirical analysis. *Informat Economics Policy* 2010;**22**:164–77.
7. Arora A, Krishnan R, Telang R *et al*. An empirical analysis of software vendors' patch release behavior: impact of vulnerability disclosure. *Informat Syst Res* 2010;**21**:115–32.
8. Dacier M, Deswarte Y, Kaâniche M. Quantitative assessment of operational security: Models and tools. In: Katsikas SK and Gritzalis D. (eds), *Information Systems Security*. London: Chapman & Hall, 1996, 179–86.
9. Ortalo R, Deswarte Y, Kaâniche M. Experimenting with quantitative evaluation tools for monitoring operational security. *Software Engineering, IEEE Transactions* 1999;**25**:633–50.
10. Phillips C, Swiler LP. A graph-based system for network-vulnerability analysis. In: *Proceedings of the 1998 Workshop on New Security Paradigms*. ACM, 1998, 71–79.
11. Moskowitz IS, Kang MH. An insecurity flow model. In: *Proceedings of the 1997 Workshop on New Security Paradigms*. ACM, 1998, 61–74.
12. Dawkins J, Hale J. A systematic approach to multi-stage network attack analysis. In: *Information Assurance Workshop, 2004. Second IEEE International*. IEEE, 2004, 48–56.
13. Frigault M, Wang L, Singhal A *et al*. Measuring network security using dynamic Bayesian network. In: *Proceedings of the 4th ACM Workshop on Quality of Protection*. ACM, 2008, 23–30.
14. Ray I, Poolsapassit N. Using attack trees to identify malicious attacks from authorized insiders. In: *Computer Security–ESORICS 2005*. Springer, 2005, 231–46.
15. Saha D. Extending logical attack graphs for efficient vulnerability analysis. In: *Proceedings of the 15th ACM Conference on Computer and Communications Security*. ACM, 2008, 63–74.
16. Wang L, Singhal A, Jajodia S. Measuring the overall security of network configurations using attack graphs. In: *Data and Applications Security XXI*. Springer, 2007, 98–112.
17. Arora A, Hall D, Piato CA *et al*. Measuring the risk-based value of it security solutions. *IT Professional* 2004;**6**:35–42.
18. Butler SA. Security attribute evaluation method: a cost-benefit approach. In: *Proceedings of the 24th International Conference on Software Engineering*. ACM, 2002, 232–40.
19. Noel S, Jajodia S, O'Berry B *et al*. Efficient minimum-cost network hardening via exploit dependency graphs. In: *Computer Security Applications Conference, 2003. 19th Annual*. IEEE, 2003, 86–95.
20. Butler SA, Fischbeck P. Multi-attribute risk assessment. In: *Symposium on Requirements Engineering for Information Security*, 2002.
21. Noel S, Jajodia S. Optimal ids sensor placement and alert prioritization using attack graphs. *J Network Syst Manage* 2008; **16**:259–75.
22. Wang L, Noel S, Jajodia S. Minimum-cost network hardening using attack graphs. *Computer Commun* 2006;**29**:3812–24.
23. Dantu R, Loper K, Kolan P. Risk management using behavior based attack graphs. In: *Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on*, Vol. 1. IEEE, 2004, 445–9.
24. Dantu R, Kolan P, Akl R *et al*. Classification of attributes and behavior in risk management using bayesian networks. In: *Intelligence and Security Informatics, 2007 IEEE*. IEEE, 2007, 71–74.
25. Dantu R, Kolan P, Cangussu J. Network risk management using attacker profiling. *Security Commun Networks* 2009;**2**:83–96.
26. Wang L, Liu A, Jajodia S. Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts. *Computer Commun* 2006;**29**:2917–33.
27. Xie P, Li JH, Ou X *et al*. Using bayesian networks for cyber security analysis. In: *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference* IEEE, 2010, 211–20.
28. Liu Y, Man H. Network vulnerability assessment using bayesian networks. In: *Defense and Security*. International Society for Optics and Photonics, 2005, 61–71.
29. Poolsappasit N, Dewri R, Ray I. Dynamic security risk management using bayesian attack graphs. *Dependable and Secure Computing, IEEE Transactions* 2012;**9**:61–74.
30. Dewri R, Poolsappasit N, Ray I *et al*. Optimal security hardening using multi-objective optimization on attack tree models of networks. In: *Proceedings of the 14th ACM Conference on Computer and Communications Security*. ACM, 2007, 204–13.
31. Moore AP, Ellison RJ, Linger RC. Attack modeling for information security and survivability. Technical report, DTIC Document, 2001.
32. Lye K-W, Wing JM. Game strategies in network security. *Int J Informat Security* 2005;**4**:71–86.
33. Fielder A, Panaousis E, Malacaria P *et al*. Game theory meets information security management. In: *ICT Systems Security and Privacy Protection*. Springer, 2014, 15–29.
34. Shapley LS. Stochastic games. *Proc Natl Acad Sci* 1953; **39**:1095–100.
35. Spaan MTJ, Vlassis N. Perseus: randomized point-based value iteration for pomdps. *J Artif Int Res* 2005;**24**:195–220.
36. Smallwood RD, Sondik EJ. The optimal control of partially observable processes over a finite horizon. *Oper Res* 1973;**21**:1071–88.
37. The Julia language. http://julialang.org/ (20 October 2015, date last accessed)
38. Govindan S, Wilson R. A global Newton method to compute Nash equilibria. *J Economic Theory* 2003;**110**:65–86.