Singapore Management University

Institutional Knowledge at Singapore Management University

Research Collection School Of Information Systems

School of Information Systems

2-2016

Data exfiltration detection and prevention: Virtually distributed POMDPs for practically safer networks

Sara Marie MC CARTHY

Arunesh SINHA
Singapore Management University, aruneshs@smu.edu.sg

Milind TAMBE

Pratyusa MANADHATA

Citation

MC CARTHY, Sara Marie; SINHA, Arunesh; TAMBE, Milind; and MANADHATA, Pratyusa. Data exfiltration detection and prevention: Virtually distributed POMDPs for practically safer networks. (2016). *Proceedings of Decision and Game Theory for Security*. 9996,. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/4665

Data Exfiltration Detection and Prevention: Virtually Distributed POMDPs for Practically Safer Networks

Sara Marie Mc Carthy[1], Arunesh Sinha[1], Milind Tambe[1], and Pratyusa Manadhata[2]

[1] University of Southern California
saramarm@usc.edu aruneshsinha@gmail.com tambe@usc.edu
[2] Hewlett Packard Labs
pratyusa.k.manadhata@hpe.com

Abstract. We address the challenge of detecting and addressing advanced persistent threats (APTs) in a computer network, focusing in particular on the challenge of detecting data exfiltration over Domain Name System (DNS) queries, where existing detection sensors are imperfect and lead to noisy observations about the network's security state. Data exfiltration over DNS queries involves unauthorized transfer of sensitive data from an organization to a remote adversary through a DNS data tunnel to a malicious web domain. Given the noisy sensors, previous work has illustrated that standard approaches fail to satisfactorily rise to the challenge of detecting exfiltration attempts. Instead, we propose a decision-theoretic technique that sequentially plans to accumulate evidence under uncertainty while taking into account the cost of deploying such sensors. More specifically, we provide a fast scalable POMDP formulation to address the challenge, where the efficiency of the formulation is based on two key contributions: (i) we use a *virtually distributed POMDP* (VD-POMDP) formulation, motivated by previous work in distributed POMDPs with sparse interactions, where individual policies for different sub-POMDPs are planned separately but their sparse interactions are only resolved at execution time to determine the joint actions to perform; (ii) we allow for abstraction in planning for speedups, and then use a fast MILP to implement the abstraction while resolving any interactions. This allows us to determine optimal sensing strategies, leveraging information from many noisy detectors, and subject to constraints imposed by network topology, forwarding rules and performance costs on the frequency, scope and efficiency of sensing we can perform.

1 Introduction

Advanced persistent threats can be one of the most harmful attacks for any organization with a cyber presence, as well as one of the most difficult attacks to defend against. While the end goal of such attacks may be diverse, it is often the case that intent of an attack is the theft of sensitive data, threatening the loss of competitive advantage and trade secrets as well as the leaking of confidential documents, and endangerment of national security [13, 4]. These attacks are sophisticated in nature and often targeted to the vulnerabilities of a particular system. They operate quietly, over long periods of time and actively attempt to cover their tracks and remain undetected. A recent trend in these attacks has relied on *exploiting Domain Name System (DNS) queries* in order to provide channels through which exfiltration can occur [20, 8].These DNS based exfiltration techniques have been used in well-known families of malware; e.g., FrameworkPOS, which was used in the Home Depot data breach involving 56 million credit and debit card information [2].

At a high level, DNS exfiltration involves an attacker-controlled malware inside an organization's network, an external malicious domain controlled by the attacker, and a DNS server authoritative for the domain that is also controlled by the same attacker. The malware leaks sensitive data by transmitting the data via DNS queries for the domain; these queries traverse the DNS hierarchy to reach the attacker controlled DNS server. Attackers can discretely transfer small amounts of data over long periods of time disguised as legitimate user generated DNS queries. Detecting and protecting against such an attack is extremely difficult as the exfiltration attempts are often lost in the high volume of DNS query traffic and any suspicious activity will not be immediately obvious. In both academia and industry, multiple detectors have been proposed to detect DNS exfiltration. However, because of the sophisticated and covert nature of these attacks, detectors designed to protect against these kinds of attacks either often *miss attacks* or are plagued by *high false positive rates, misclassifying legitimate traffic* as suspicious, and potentially *overwhelming a network administrator* with suspicious activity alerts; these issues have been identified with machine learning based detectors [23], pattern matching based detectors [1] and information content measuring detector [19].

We focus on the problem of *rapidly determining malicious domains* that could be potentially exfiltrating data, and then *deciding whether to block traffic or not.* In our problem, the defender observes a stream of suspicious DNS based exfiltration alerts (or absence of alerts), and is tasked with inferring which of the domains being queried are malicious, and determining the best response (block traffic or not) policy. Unfortunately, as stated earlier, detectors are inherently *noisy* and each single alert does not provide a high confidence estimate about the security state. Thus, the defender needs to come up with a sequential plan of actions while dealing with uncertainty in the network and in the alerts, and must weight the cost of deploying detectors to increase their knowledge about malicious domains with the potential loss due to successful attacks as well as the cost of misclassifying legitimate network use. This problem of active sensing is common to a number of cyber security problems; here we focus on the challenge of data exfiltration over DNS queries.

There has been a large amount of work on how to deal with and make decision under uncertainty. Problems such as ours can be well modeled using Partially Observable Markov Decision Process (POMDP) to capture the dynamics of real-world sequential decision making processes, and allow us to reason about uncertainty and compute optimal policies in these types of environments. However a major drawback to these models is that they are *unable to scale* to solve any problem instances of reasonable size. In order to be successful in the cyber domain, such a models needs to be able to handle extremely large problem instances, as networks are often extremely complex, with lots of moving parts. Additionally, due to the salient nature of network states, we need to be able to make *decisions in real time* in order to observe and quickly react to a potential threat.

To address this challenge we make the following key contributions: (1) We provide a formal model of the DNS data exfiltration problem. We propose a new decision making framework using Partially Observable Markov Decision Processes (POMDPs). (2) We address the scalability issued faced when dealing with large networks by proposing a series of abstractions of the original POMDP. These include using abstract action and observation space. (3) Another step in the abstraction is a *new paradigm* for solving these models by factoring the POMDP into several sub-POMDPs and solving each individual sub-POMDP separately offline; this is motivated by previous work in distributed POMDPs with sparse interactions. We provide techniques for policy aggregation to be performed at runtime in order to combine the abstract optimal actions from each sub-POMDP to determine the final joint action. We denote this model as a *virtually distributed POMDP* (VD-POMDP). We provide conditions under which our methods are guaranteed to result in the optimal joint policy, and provide empirical evidence to show that the final policy still performs well when these conditions do not hold. (4) Finally we provide experimental evaluation of our model in a real network testbed, where we demonstrate the ability to correctly identify real attacks.

2 Background and Related Work

We split our discussion of the required background and related work for this paper along two broad categories that are covered in the two sub-sections below.

2.1 DNS Exfiltration

Sensitive data exfiltration from corporations, governments, and individuals is on the rise and has led to loss of money, reputation, privacy, and national security. For example, attackers stole 100 million credit card and debit card information via breaches at Target and Home Depot [11]; a cluster of breaches at LinkedIn, Tumblr, and other popular web services led to 642 million stolen passwords [5]; and the United States Office of Personnel Management (OPM) data breach resulted in 21.5 million records, including security clearance and fingerprint information, being stolen [27].

In the early days, exfiltration happened over well known data transfer protocols such as email, File Transfer Protocol (FTP), and Hypertext Transfer Protocol (HTTP) [13]. The seriousness of the problem has led to several "data loss prevention (DLP)" products from the security industry [24, 15] as well as academic research for monitoring these protocols [7, 10]. These solutions monitor email, FTP, and other well known protocols for sensitive data transmission by using keyword matching, regular expression matching, and supervised learning.

The increased monitoring of the protocols has forced attackers to come up with ingenious ways of data exfiltration. One such technique used very successfully in recent years is exfiltration over DNS queries [20, 1]. Since DNS is fundamental to all Internet communication, even the most security conscious organizations allow DNS queries through their firewall. As illustrated in Figure 1, an adversary establishes a malicious domain, *evil.com*, and infects a client in an organization with malware. To exfiltrate a data file, the malware breaks the file into blocks, b_1, b_2, $\cdots$, b_n, and issues a sequence of DNS queries, b_1*.evil.com*, b_2*.evil.com*, $\cdots$, b_n*.evil.com*. If their responses are not cached, the organization's DNS server will forward them to the nameserver authoritative for *evil.com*; at this point, the adversary controlling the authoritative nameserver can reconstruct the data file from the sequence of blocks.

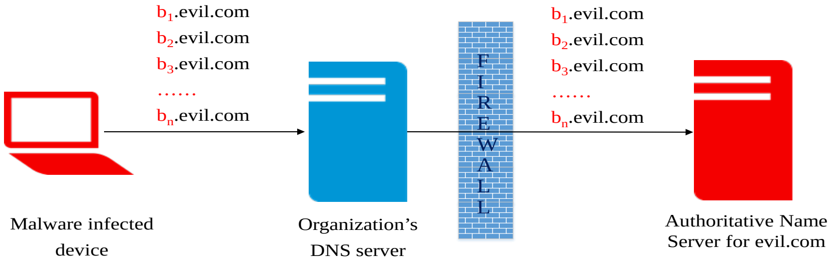


Fig. 1: Data exfiltration over DNS.

The data transmission is covert and can be accomplished by various means such as a particular sub-domain query meaning bit 1 and another sub-domain query meaning bit 0, or even the timing between queries can leak information. By compressing the data at the client, and by varying query lengths and the time interval between successive queries an adversary can adjust the bandwidth of the communication channel. The adversary could choose to transfer data as quickly as possible (long and rapid domain queries) or slowly (short queries spaced apart in time), depending on the intent behind the attack. To further hide exfiltration activity, the data blocks can be encrypted by the client before the queries are issued, and decrypted by the adversary. Further, the adversary can encode instructions within its responses to establish a two-way communication tunnel.

Hence building a reliable DNS exfiltration detector is extremely challenging. A recent work on building a detector for DNS exfiltration using measurement of information content of a DNS channel provides techniques that we use to build the low level detector in our problem setting [19]. Apart from this work in academia, there has been some work in the industry that use various heuristics to build low level detectors for DNS exfiltration [1]; examples of such heuristics are lengths of DNS queries and responses, sizes of query and response packets, entropy of DNS queries, total volume of DNS queries from a device, and total volume of DNS traffic to a domain. As far as we know, we are the first to build a cost based sequential planning tool that uses the imperfect low level detectors to determine if a domain is involved in exfiltrating data over DNS.

2.2 POMDP

There has been a large amount of work on how to deal with and make decisions in stochastic environments under uncertainty using POMDPs. However, it is known that offline POMDP solving is intractable for large problems [9, 14, 18] and given our fast response requirements an online POMDP solver is also not feasible [21]. We show empirically that our original POMDP is simply impractical to solve for even a small network of 3-4 computers. Thus, in this paper, we focus on speeding up the offline POMDP solving by performing a series of abstractions of the original POMDP. Our technique of solving the POMDP is inspired by conflict resolution techniques in solving distributed POMDP [17, 12] and distributed POMDPs with sparse interactions [25, 26]. While our VD-POMDP does not have an inherent distributed structure, we break up the original POMDP into multiple domain specific POMDPs to build a virtually distributed POMDP; this allows for scalability. However, instead of resolving conflicts at runtime or using sparse interaction (which does not exist in our split POMDPs), we modify our action space so that the actions output by each domain specific POMDP is at a higher level of abstraction. With these abstract actions from each POMDP we provide a fast technique to come up with the joint action by combining the abstract actions at execution time. The main difference with existing work on POMDP and distributed POMDP is that we reason about policies at execution time, allowing efficient“groundlevel” implementation of abstract policy recommendation from multiple virtual POMDPs. Further, the abstraction of actions is possible in our problem due to the special relation between detecting malicious domains and sensing of traffic on a network node (see Model section for details).

While, the main step in VD-POMDP compaction is based on similar ideas of factored POMDPs used in past literature [16], our approach critically differs as we do not just factor the belief state of the POMDP, but split it into multiple POMDPs per domain. Also, distinct from general distributed POMDP [6] the multiple agents (for each domain) do not share a common reward function and neither is the reward function a sum of each of their rewards.

G(N,E)	graph representing network
v_n	value of data at the n^{th} node
$v_{[d]}$	average value of the set of channels to the d^{th} domain
w_n	volume of traffic at the n^{th} node
$w_{[d]}$	total volume of the set of channels to the d^{th} domain
d	the d^{th} domain
X_d	true $\{0,1\}$ state of the d^{th} domain
M_d	estimated $\{0,1\}$ state of the d^{th} domain
X	set of all X_d random variables
$c_k =< n, \dots d >$	k^{th} channel from node n to domain d
$C_{[d]}$	subset of channels ending with the d^{th} domain
C	set of all channels
τ_k	threshold set for channel c_k
a_n	binary variable indicating if node n is sensed or not
z_k	binary variable indicating if channel c_k is sensed or not
Ω_k	$\{0,1\}$ observation on k^{th} channel
$\Omega_{[d]}$	subset of observations for channels ending with the d^{th} domain

Table 1: Notation

3 Model

The local computer network can represented as a graph $G(N, E)$, where the nodes N correspond to the set of hosts in the network, with edges E if communication between the hosts is allowed. Each node n has a particular value v_n corresponding to the value of data stored at that computer. At any point in time t each node has a variable traffic volume of requests w_n^t passing through it. We assume there are D domains, where for tractability we assume D is the the number of domains that have ever been queried for the given computer network. DNS queries made from internal nodes in the network are forwarded to special nodes, either access points or internal DNS servers, and then forwarded to external servers from these points. A channel c_k over which exfiltration can occur is then a path, starting at source node, where the query originated, traveling through several nodes in the network and finishing at a target domain d. The total number of channels is K. We use $n \in c_k$ to denote any node in the path specified by c_k.

Let X_d be a random binary variable denoting whether domain d is malicious or legitimate. We assume that a malicious domain will always be malicious, and that legitimate domains will not become compromised; this means that the state X_d does not change with time. Even though legitimate domains get compromised in practice, attackers often use new malicious domains for DNS exfiltration since an attacker needs to control both a domain and the authoritative name server for a domain to successfully carry out exfiltration. In other words, legitimate domains that get compromised are rarely used in DNS exfiltration. Hence in our model, it is reasonable to assume that domains don't change their states. We call the active sensing problem, the challenge of determining the values of X_d. In order to do this we may place detectors at nodes in the network; the state of a detector (off/on) at any node in the network is $a_n \in \{0, 1\}$. Each detector monitors all the channels passing through that particular node, i.e., all $c_k : n \in c_k$. We use the binary variable z_k to indicate if channel c_k is monitored. We can set discrete thresholds individually for each channel; lower thresholds correspond to higher sensitivity to information flow out of any particular channel.

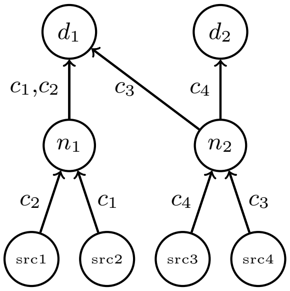


Fig. 2: Example of a network with two domains, 4 source hosts and 4 channels. Channels $c1$, $c2$, $c3$ go from sources 1, 2, and 3 to domain d_1 while channel c_4 goes from source 4 to domain d_2. We may consider the situation where we can only turn on one detector at any time step, either at node n_1 or n_2, and choose to sense on channels $\{c_1, c_2\}$ or $\{c_3,c_4\}$. We can additionally chose thresholds τ_j for each channel. Each source host has a value v_n and each node n has traffic volume w_n.

Because each channel is associated with a domain, we set a threshold τ_k for each channel. We use $|\tau|$ to denote the number of discrete threshold choices available. We then get observations in the form of alerts for each channel $\Omega_k \in \{0, 1\}$. The probability of receiving an alert for any channel is characterized by some function $\alpha(\tau_k)$ if the channel is malicious and $\beta(\tau_k)$ if the channel is legitimate. Finally, the defender classifies the state of domain d as malicious or legitimate, indicated by M_d.

3.1 The POMDP Model

Our POMDP model is a tuple (S, A, T, Ω, O, R) with state space S, action space A, state transition function T, observation space Ω, observation probabilities O and reward function R. Additionally define the average value of the channels to domain d as $v_{[d]} = \sum\limits_{n:n \in C_{[d]}} \frac{v_n}{|C_{[d]}|}$. Below we list the details of components of POMDP model. The state captures the true security state of every domain and the actions specify the thresholds for monitoring each channel, the nodes to be monitored and the decision about which domains are classified as malicious. As we assume the security state of the system does not change, the transition function is straightforward.

$$\begin{aligned} \text{States} \quad & \text{S} = \langle X_1, \dots X_D \rangle \\ \text{Actions} \quad & \text{A} = A_c \times A_n \times A_d \text{ where } \langle \tau_1, \dots \tau_K \rangle \in A_c, \langle a_1 \dots a_N \rangle \in A_n \\ & \quad \text{and } \langle M_1 \dots M_D \rangle \in A_d \\ \text{Transision} \quad & T(s', s) = \begin{cases} 1 & \text{iff } s' = s \\ 0 & \text{else} \end{cases} \end{aligned}$$

Next, we obtain an observation Ω_k for each channel, and as stated earlier for each channel the probability of an alert is given by functions α and β. We state the probability first for the observations for each domains, and then for all the observations using independence across domains.

$$\text{Observations} \qquad \Omega = \langle \Omega_1 \dots \Omega_K \rangle$$

$$\text{Observation Prob} \quad O(\Omega_{[d]}|X_d, A) = \begin{cases} \displaystyle\prod_{k:k\in C_{[d]}\wedge z_k=1} \alpha(\tau_k)^{\Omega_k}(1-\alpha(\tau_k))^{1-\Omega_k} & \text{if } X_d = 1 \\ \displaystyle\prod_{k:k\in C_{[d]}\wedge z_k=1} \beta(\tau_k)^{\Omega_k}(1-\beta(\tau_k))^{1-\Omega_k} & \text{if } X_d = 0 \\ 0 & \text{else} \end{cases}$$

$$O(\Omega|X, A) = \prod_d O(\Omega_{[d]}|X_d, A_{[d]})$$

Finally, the reward for the POMDP is given by the following equation:

$$\text{R(S,A)} = -\left(\sum_d \big(X_d(1-M_d)v_{[d]} + (1-X_d)M_d w_{[d]}\big) + \sum_n^N a_n w_n\right)$$

The reward contains two cost components: the first component has two terms for each domain that specify the penalty for mislabeling a domain and the second component is the cost of sensing over the nodes. When a malicious domain d is labeled safe then the defender pays a cost $v_{[d]}$, i.e., the average value of channels going to domain d; in the opposite mislabeling the defender pays a cost $w_{[d]}$, i.e., a cost specified by loss of all traffic going to domain d. While this POMDP model captures all relevant elements of the problem, it is not at all tractable. Consider the input variables to this model, the number of domains D, the number of nodes N and the number of channels k. The state space grows as $O(2^D)$, the action space is $O(2^N|\tau|^K 2^D)$ and the observation space is $O(2^K)$. This full formulation is exponential in all the input variables and cannot scale to larger, realistic network instances (we also show this experimentally in the Evaluation Section). In order to reduce the combinatorial nature of the observation space, action space and state space, we introduce a compact representation for the observation and action space and a factored representation for the state space that results in splitting the POMDP into multiple POMDPs.

4 POMDP Abstraction

We represent the POMDP compactly by using three transformations: (1) we use the same threshold for very channel going to the same domain and change the action space from sensing on nodes to sensing on channels, (2) reduce the observation space by noting that only the number of alerts for each domain are required and not which of the channels generated these alerts and (3) factoring the whole POMDP by domains, then solve a POMDP per domain and combine the solutions at the end. Next, we describe these transformations in details.

Abstract Actions We can reduce the action space by (1) enforcing that the same threshold is set for all channels going to the same domain and (2) by reasoning about which channels to sense over instead of which nodes to sense on. The first change reduces the action space from a $|\tau|^K$ dependance to $|\tau|^D$, where $|\tau|$ is the discretization size of the threshold for the detector. The new set of threshold actions is then $A_c = \langle \tau_1, \dots \tau_D \rangle$. The second change replaces the set of actions on nodes A_n with a set of actions on channels $A_k = \langle s_{k_{[1]}} \dots s_{k_{[D]}} \rangle$, where $s_{k_{[d]}}$ is the number of channels to be sensed out of the $|C_{[d]}|$ channels that end in domain d. This changes the action space complexity from 2^N to $|C_{[d]}|^D$. Then the action space is given by

Actions A $= A_c \times A_k \times A_d$
where $\langle \tau_1, \ldots \tau_D \rangle \in A_c$, $\langle s_{k_{[1]}} \ldots s_{k_{[D]}} \rangle \in A_k$ and $\langle M_1 \ldots M_D \rangle \in A_d$

In order to properly compute the reward we need to compute the cost of any action in A_k. To do this we need to build a lookup table mapping each action in A_k to an action in A_n, and hence obtain the cost of actions in A_k. Because we will always choose the lowest cost way to sense on a number of channels, the action of sensing a specified number of channels can be mapped to the set of nodes that minimizes the cost of sensing the specified number of channels. We can compute this using the following mixed integer linear program (MILP) mincost($\langle s_{k_{[i]}} \ldots s_{k_{[D]}} \rangle$).

$$\min_{z_k, a_n} \quad \sum_n a_n w_n \tag{1}$$

$$z_k \le \sum_{n \in c_k} a_n \quad \forall k \in \{1, \ldots, K\} \tag{2}$$

$$\sum_{c_k \in C_{[d]}} z_k \ge s_{k_{[d]}} \quad \forall d \in \{1, \ldots, D\} \tag{3}$$

$$z_k \in \{0,1\} \quad a_n \in \{0,1\} \tag{4}$$

The mincost($\langle s_{k_{[1]}} \ldots s_{k_{[D]}} \rangle$) MILP needs to be solved for every action $\langle s_{k_{[1]}} \ldots s_{k_{[D]}} \rangle \in A_k$, i.e., we need to fill in a table with $O(|C_{[d]}|^D)$ entries. If we take the example network in Figure 2, the old action space is $A_n = \{\{\emptyset\}, \{n_1\}, \{n_2\}, \{n_1, n_2\}\}$, and the new action space is $A_k = \{\{0,0\}, \{1,0\}, \{2,0\}, \{3,0\}, \{0,1\}, \{1,1\}, \{2,1\}, \{3,1\}\}$. In order to map back to the representation using nodes, we build the mapping: $\{0,0\} \rightarrow \emptyset, \{1,0\} \rightarrow \{n_1\}, \{2,0\} \rightarrow \{n_1\}, \{3,0\} \rightarrow \{n_1\}, \{0,1\} \rightarrow \{n_2\}, \{1,1\} \rightarrow \{n_1, n_2\}, \{2,1\} \rightarrow \{n_1, n_2\}, \{3,1\} \rightarrow \{n_1, n_2\}$. However, the problem of converting from number of channels to nodes (stated as mincost($\langle s_{k_{[1]}} \ldots s_{k_{[D]}} \rangle$) above) is not easy as the following theorem shows:

Theorem 1. *The problem of converting from number of channels to nodes is NP hard to approximate to any factor better than* $\ln |N|$.

Proof. We perform a strict approximation preserving reduction from the set cover problem. Consider a set cover problem. We are given a universe of m elements E and u subsets of E: U. Form a node n_u for each subset $u \in U$ and a domain d_e for each element $e \in E$. For any particular element e and any node containing that element, connect it to the domain d_e. Then, these connections, say from l nodes, defines l channels each starting from a node and ending in d_m. For any domain d choose $s_{k_{[d]}} = 1$, i.e., at least one channel needs to be sensed. It can be easily seen that for any channel c_k in this network there is a unique node it passes through: call it $n(k)$. Choose $w_n = 1$. Then, the optimization problem to be solved is the following:

$$\min_{z_k, a_n} \quad \sum_n a_n \tag{5}$$

$$z_k \le a_{n(k)} \quad \forall k \in \{1, \ldots, K\} \tag{6}$$

$$\sum_{c_k \in C_{[d]}} z_k \ge 1 \quad \forall d \in \{1, \ldots, D\} \tag{7}$$

$$z_k \in \{0,1\} \quad a_n \in \{0,1\} \tag{8}$$

First, we prove that the constraints of this optimization specify a choice of subsets (nodes) the union of which equals the set E. Since all channels going to domain d corresponds to a unique element e and at least one channel going to d is chosen (Eq. 7), this implies at least one node containing e is selected (Eq. 6). Thus, the set of nodes (hence subsets) contains all elements e.

Given, the feasible space is given by a set of subsets (nodes) the union of which produces E, the objective clearly produces the minimum number of such sets. Also, any approximate solution with guarantee α maps to an α approximate solution of the set cover problem. The theorem follows from the lack of better than $\ln n$ approximatability of set cover. □

Abstract Observations As we only reason about the state of each domain in the network, not each individual channel, we can aggregate the observation in order to reduce the observation space. Thus, instead of recording which channel generated an alert, we only record total number of alerts per domain. Given there are $|C_{[d]}|$ channels going to domain d then the observations for each domain lie in $\{0 \ldots |C_{[d]}|\}$. This observation space for each domain is then linear in the number of channels $O(|C_{[d]}|)$. The full joint observation space is exponential in the number of domains $O(|C_{[d]}|^D)$.

The set of observations is then $\Omega = \langle \Omega_1, \ldots, \Omega_D \rangle$ where $\Omega_d \in \{0 \ldots |C_{[d]}|\}$ corresponding to the number of alerts from all $|C_{[d]}|$ channels going to domain d. Because there is now multiple way for us to get this single observation, the observation probability function for each domain also needs to be modified.

$$O(\Omega_d|X_d, A) = \begin{cases} \binom{s_{k_{[d]}}}{\Omega_d} \alpha(\tau_d)^{\Omega_d} (1-\alpha(\tau_d))^{s_{k_{[d]}}-\Omega_d} & \text{if } X_d = 1 \\ \binom{s_{k_{[d]}}}{\Omega_d} \beta(\tau_d)^{\Omega_d} (1-\beta(\tau_d))^{s_{k_{[d]}}-\Omega_d} & \text{if } X_d = 0 \\ 0 & \text{else} \end{cases}$$

VD-POMDP Factored Representation Looking at both the observation probability function as well as the belief update, we can consider a factored representation of this POMDP, by factoring these by domains. If we then separate out these factored components and create a new sub-agent for each factor, so that we now have a total of D POMDP's, we can greatly reduce the state space, observation space and action space for each individual sub agent. The model for each of these individual POMDP is then given as follows.

$$\begin{array}{ll}
\text{States} & \mathrm{S} = X_d \\
\text{Actions} & \mathrm{A} = \tau_d \times \{0, \ldots, |C_{[d]}|\} \times M_d \\
\text{Transition} & T(s', s) = \begin{cases} 1 & \text{iff } s' = s \\ 0 & \text{else} \end{cases} \\
\text{Observations} & \Omega = \langle \Omega_1, \ldots, \Omega_D \rangle \text{ where } \Omega_d \in \{0, \ldots, |C_{[d]}|\} \\
 & O(\Omega_d|X_d, A) = \begin{cases} \binom{s_{k_{[d]}}}{\Omega_d} \alpha(\tau_d)^{\Omega_d} (1-\alpha(\tau_d))^{s_{k_{[d]}}-\Omega_d} & \text{if } X_d = 1 \\ \binom{s_{k_{[d]}}}{\Omega_d} \beta(\tau_d)^{\Omega_d} (1-\beta(\tau_d))^{s_{k_{[d]}}-\Omega_d} & \text{if } X_d = 0 \\ 0 & \text{else} \end{cases} \\
\text{Reward} & \mathrm{R(S,A)} = -\left(X_d(1-M_d)v_{[d]} + (1-X_d)M_d w_{[d]} + \mathrm{mincost}(s_{k_{[d]}})\right)
\end{array}$$

The complexity of the state space is reduced to $O(1)$, the action space is $O(|\tau||C_{[d]}|)$ and the observation space is $O(|C_{[d]}|)$. Table 2 shows the comparative complexities of the original POMDP

	Full POMDP		VD-POMDP
	Original	Abstract	
State	$O(2^D)$	$O(2^D)$	$O(1)$
Action	$O(2^N\|\tau\|^K 2^D)$	$O(\|C_{[d]}\|^D\|\tau\|^D 2^D)$	$O(2^{\|C_{[d]}\|}\|\tau\|)$
Observation	$O(2^K)$	$O(\|C_{[d]}\|^D)$	$O(\|C_{[d]}\|)$

Table 2: Complexities of Full and VD-POMDP models with original and compact representations.

model and the VD-POMDP model. As we use channels as actions for each domain specific POMDP, we still need to construct the lookup table to map channels as actions to nodes as actions in order to obtain the cost of each action on channels. Factoring the model in the way described above also simplifies the construction of this lookup table from actions on channels to actions on nodes, and hence computing mincost$(s_{k_{[d]}})$ can be done in a much simpler way for the VD-POMDPs. We solve a similar (MILP) as in (2)-(4) but for each VD-POMDP for domain d; thus, we only need to fill in a table with $O(|C|)$ entries, one for each of the $s_{k_{[d]}}$ actions for each domain d. The new MILP formulation is given in equations 10-12. Observe that unlike the MILP (2)-(4) used to build the lookup table for the original POMDP, this MILP is solved for a fixed domain d.

$$\min_{z_k, a_n} \quad \sum_n a_n w_n \tag{9}$$

$$z_k \leq \sum_{n \in C_k} a_n \tag{10}$$

$$\sum_{c_k \in C_{[d]}} z_k \geq s_{k_{[d]}} \tag{11}$$

$$z_k \in \{0,1\} \quad a_n \in \{0,1\} \tag{12}$$

While the above optimization is much more simpler than the corresponding optimization for the original POMDP, it is still a hard problem:

Theorem 2. *The problem of coverting the number of channels to nodes for each VD-POMDP is NP Hard.*

Proof. We reduce from the min knapsack problem. The min knapsack problem is one where the objective is to minimize the value of chosen items subject to a minimum weight W being achieved, which is a well known hard problem. Also, wlog, we can assume weights of items and W to be integers. Given a min knapsack with n items of weights w'_i and value v_i and min weight bound W form an instance of our problem with n nodes (mapped to items) and each node i having w'_i channels going directly to domain d. It can be easily seen that for any channel c_k in this network there is a unique node it passes through: call it $n(k)$. Each node i also has traffic $w_i = v_i$. Also, $s_{k_{[d]}} = W$. Then, the optimization problem being solved is

$$\min_{z_k, a_n} \quad \sum_n a_n v_n \text{ subject to } z_k \leq a_{n(k)}, \sum_{c_k \in C_{[d]}} z_k \geq W, \; z_k \in \{0,1\}, \; a_n \in \{0,1\}$$

Note that in the constraints, whenever a node is selected $a_{n(k)} = 1$ then making all w_i channels in it one makes the weight constraints less tight. Thus, any values of a_n, z_k satisfying the constraints specify a set of nodes such that the sum of its weights is $\geq W$. Coupled with the fact that the objective minimizes the values of selected nodes, the solution to this optimization is a solution for the min knapsack problem. □

Policy execution: The solutions to each of these VD-POMDP's give us an action $\langle M_d^*, \tau_d^*, s_{k_{[d]}}^* \rangle$ corresponding to a labeling of malicious or legitimate for that particular domain d, the threshold, and the desired number of channels to sense over. However, at execution time we need to turn on detectors on nodes. Thus, in order to aggregate these factored actions to determine the joint action to take at execution, we need to map the output from each POMDP back to a set of sensing actions on nodes. This can be easily accomplished by solving a single instance of the larger MILP (2)-(4) with the $s_{k_{[d]}}$ values set to $s_{k_{[d]}}^*$.

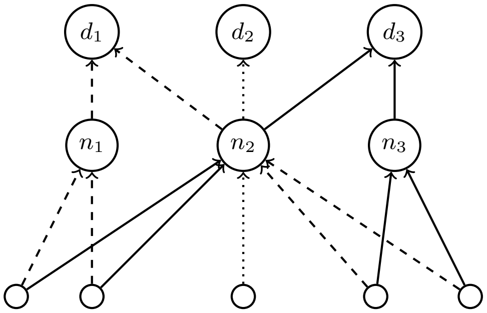


Fig. 3: Sample network with 3 domains, 3 nodes and 5 sources. The dashed lines are channels to domain d_1 the dotted line is the channel to domain d_2 and the solid lines are channels to d_3.

We *emphasize* here the importance of using the abstract channels as actions instead of nodes. The possibly alternate approach with nodes as action for each sub-POMDP and just taking union of the nodes output by each domain specific POMDP, when the channels are not disjoint, can result in over sensing. Consider the example below, where there are 4 channels going to domains d_1 and d_3 and one to d_2 and let us currently be in a belief state where the optimal action for domain d_1 and d_3 would be to sense on 2 channels out of the 4 going to each domain and the optimal action for d_2 is to sense on the one channel. Working in an action space of nodes, the VD-POMDP for d_1 would choose to sense on node n_1, the VD-POMDP for d_3 would choose to sense on n_3 as it has the lowest amount of traffic for 2 channels and the one for d_2 would choose to sense on n_2 as it is the only option. Taking the union of these would result in all the sensors being turned on. However, we can see that choosing only to sense on node n_2 satisfies the sensing requirements of all three separate VD-POMDPs.

Next, we identify a condition under which the solution from the larger MILP is optimal. In the next section, we show empirically that even when this condition is not met our approach is close to optimal.

Theorem 3. *The above described technique of aggregating solutions of the VD-POMDPs is optimal for the original POMDP iff the solution to the MILP (2)-(4) for any VD-POMPD policy action results in an equality for the constraint (3).*

Proof. First, with the standard representation the global value function given in equation 13-14, cannot generally be fully decomposed by domain due to the $R_n(a_n)$ term which couples the actions for each domain through the sending cost. The decomposition is only possible in special instances of the problem, such as if network of channels were completely disconnected. The action of selecting nodes can be partitioned by domains as $a_{n_{[d]}}$. Then, the cost associated with sensing on the nodes could be written as a sum of domain dependent terms $R_n(a_n) = \sum_d R_{n_{[d]}}(a_{n_{[d]}})$. Also, all actions (threshold, choice of nodes and decision about each domain) are now partitioned by domain, thus any action a is a combination of actions per domain a_d. Let b_d denote the belief state for domain d. The choice of nodes in this case should just be a union of the nodes chosen by each POMDP as seen from the the value function as each domain dependent component can be optimized separately.

$$V^* = \max_a \left[R(b,a) + \gamma \sum_{\Omega} P(\Omega|b,a) V^*(b,a,\Omega) \right] \tag{13}$$

$$= \max_a \left[\sum_d \Big(R_d(b_d, M_d) \Big) + R_n(a_n) + \gamma \sum_{\Omega} \prod_d P(\Omega_d | b_d, \tau_d) V^*(b,a,\Omega) \right] \tag{14}$$

$$= \max_a \left[\sum_d \Big(R_d(b_d, M_d) + R_{n_{[d]}}(a_{n_{[d]}}) \Big) + \gamma \sum_{\Omega} \prod_d P(\Omega_d | b_d, \tau_d) V^*(b,a,\Omega) \right] \tag{15}$$

$$= \max_a \left[\sum_d \Big(R_d(b_d, a_d) \Big) + \gamma \sum_{d,\Omega_d} P(\Omega_d | b_d, a_d) V_d^*(b_d, a_d, \Omega_d) \right] = \sum_d V_d^* \tag{16}$$

$$\text{where } V_d^* = \max_{a_d} \left[R_d(b_d, a_d) + \gamma \sum_{\Omega_d} P(\Omega_d | b_d, a_d) V_d^*(b_d, a_d, \Omega_d) \right] \tag{17}$$

If we instead use the compact representation of the action space, and let the actions simply be the number of channels to be sensed on and equality is obtained for the constraint (3) for any action from each domain specific POMDP (i.e., each $s_{k_{[d]}}$ can be implemented), then the value function can be decomposed by domain, because the term $R_n(a_n)$ is replaced by the $\sum_d \text{mincost}(s_{k_{[d]}})$, which can be factored by domain and does not couple the VD-POMDP's. Reconstructing the joint policy then just amounts to finding the best action from each POMDP and taking a union of these individual actions. We then just need to map back to the representation of actions on nodes, by solving the MILP (2)-(4).

$$V^* = \max_{a_d} \left[\sum_d \Big(R_d(b_d, a_d) + \text{mincost}(s_{k_{[d]}}) \Big) + \gamma \sum_{d,\Omega_d} P(\Omega_d | b_d, a_d) V_d^*(b_d, a_d, \Omega_d) \right] = \sum_d V_d^*$$

□

5 VD-POMDP Framework

Here we explain at a high level, the VD-POMDP framework as applied to the data exfiltration problem, and how it is implemented. With the VD-POMDP, entire planning model is broken up into two parts as depicted in Figure 4. The first is the offline factoring and planning, where the

POMDP is factored into several sub-agents, and each solved individually. Second is the online policy aggregation and execution, where the policies of each sub-agent are aggregated as each of them choose actions to perform.

In order to build the VD-POMDP for data exfiltration problem, we first construct the network graph, based on the topology of the actual computer network we are modeling as well as the set of domains under consideration, shown at point (a) in Figure 4. Then at (b), for each domain in our network, we construct a separate POMDP sub-agent. In order to do this we solve the MILP mincost(s_{k_d}) for each agent, in order to abstract away from the network layer and construct the compact representation of the network. At point (c) each individual POMDP agent is solved, offline, ignoring the presence of the other agents to obtaining a policy for each domain.

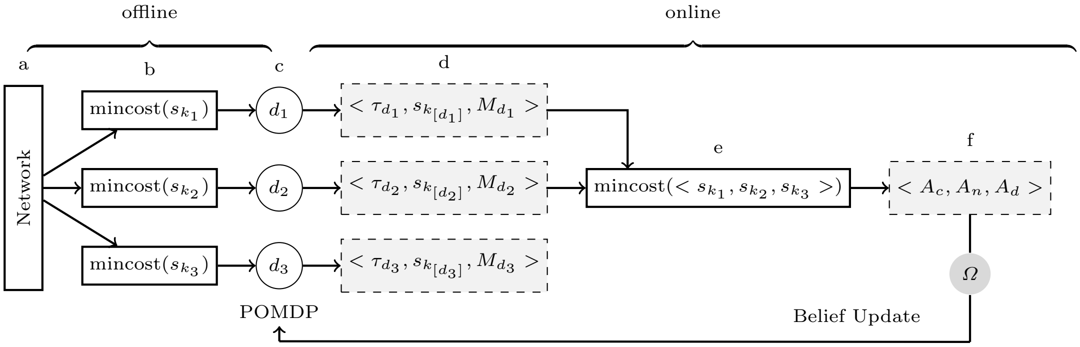

offline
online
a
b
c
d
Network
mincost(s_{k_1})
mincost(s_{k_2})
mincost(s_{k_3})
d_1
d_2
d_3
$< \tau_{d_1}, s_{k_{[d_1]}}, M_{d_1} >$
$< \tau_{d_2}, s_{k_{[d_2]}}, M_{d_2} >$
$< \tau_{d_3}, s_{k_{[d_3]}}, M_{d_3} >$
e
mincost($< s_{k_1}, s_{k_2}, s_{k_3} >$)
f
$< A_c, A_n, A_d >$
Ω
POMDP
Belief Update

Fig. 4: Flowchart for the Data Exfiltration VD-POMDP

The policies are then aggregated in an online fashion, shown at point (d) in Figure 4 to obtain a joint action (f). At each time step the agents receive observations from the network and update their beliefs individually. Each agent then presents the individual action to be performed consisting of a number of channels to be sensed on, a threshold for sensing and a classification of malicious or legitimate for their respective domain. The required number of channels for each agent is then fed into the MILP mincost($\langle s_{k_{[i]}} \ldots s_{k_{[D]}} \rangle$) to determine the set of nodes to be sensed on. The agents then again receive observations from the resulting set of detectors and iterate through this process again.

Policy aggregation is performed online as it would be infeasible to do offline policy aggregation for all but the smallest policies. If aggregation were to be performed offline, we would need to consider every possible combination of actions from each policy and then solve the MILP mincost($\langle s_{k_{[i]}} \ldots s_{k_{[D]}} \rangle$) for each of these, in order to compute the combinatorially large joint policy. Because the MILP is fast to solve, it does not result in much overhead when these joint actions are computed in an online fashion.

It is important to note here that the policy we compute is not an optimal sequence of actions, but rather a mapping of belief state to actions. This distinction is important, as it may be the case that, upon policy aggregation, there is no feasible implementation of the individual action. In such a scenario, an agent may choose an action to sense on a subset of k channels; however, given the sensing requirements of the other agents, the agent in question may actually get to sense on more channels than they had initially wanted. The agent may then end up in a belief state that they had not originally planned for, but because we are solving for the entire belief space, we still know how to behave optimally. Additionally, from Theorem 3, we know that the joint action will only be optimal if we can exactly implement each individual action, and no agent get to sense on more channels than it requests. Our policy aggregation may then result in a suboptimal joint action being taken, however, we show later in section 6, that even when the optimality condition does not hold, we can still achieve good performance.

6 Evaluation

We evaluate our model using three different metrics: runtime, performance, and robustness. We first look at the runtime scalability of VD-POMDP model, varying the size of several synthetic network as well as the number of domains and compare to the standard POMDP model. We then evaluate the performance of the VD-POMDP, measuring how quickly it can classify a set of domains as malicious or legitimate, as well as computing the accuracy of correct classifications. For small network sizes, we compare the performance of the VD-POMDP to the original model and look at the performance of the VD-POMDP on larger synthetic networks.

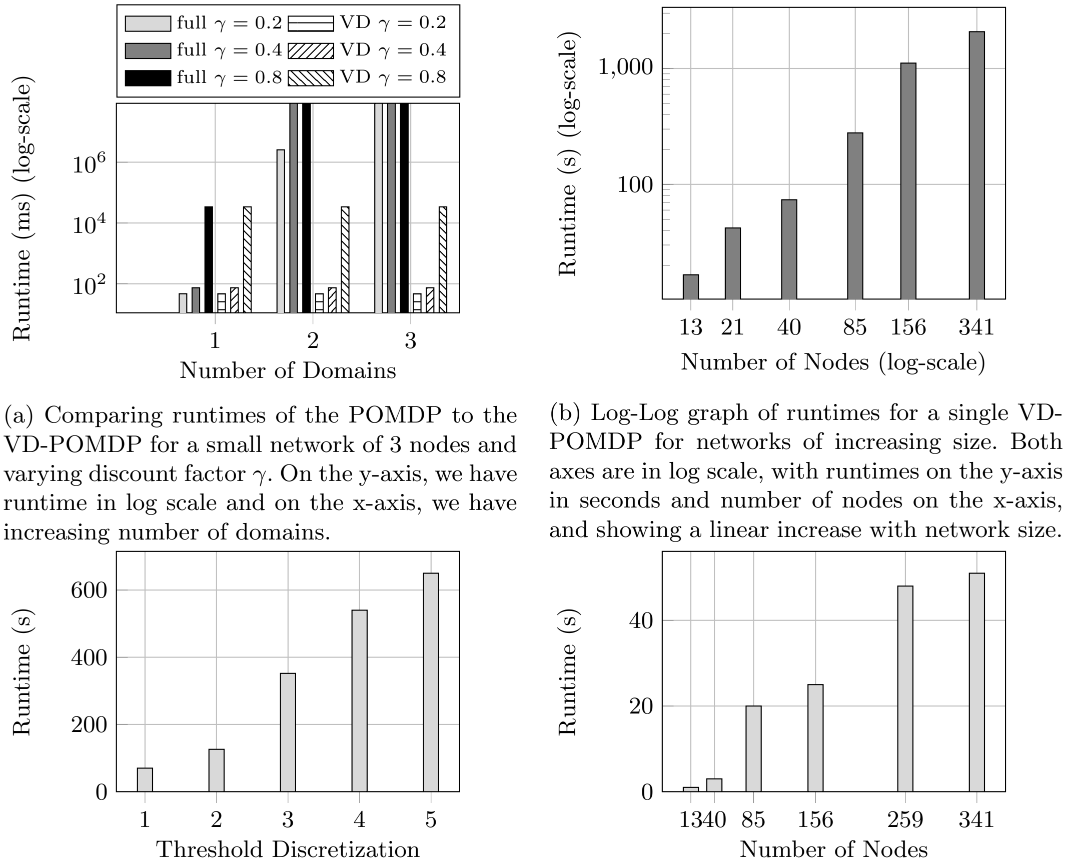
Synthetic Networks In order to test a variety of network sizes we created synthetic networks using a tree topology. Leaf nodes in the tree network correspond to source computers. Channels travel upwards from these nodes to the root of the tree; for each domain we create one such channel on each source computer. The size of the network is varied by varying the depth and branching factor of the tree.

6.1 DETER Testbed Simulation

We also evaluated the performance of our model using a real network, by running simulations on the DETER testbed. The DETER testbed provides capabilities of simulating a real computer network with virtual machines and simulating agents that perform tasks on each computer. Every agent is specified in a custom scripting language, and allows simulating attackers, defender and benign users. For our simulation we simulated legitimate DNS queries as well as launched real attacks. We performed a simple attack, by attempting to exfiltrate data from a file to a chosen malicious domain by embedding data from the file into the DNS queries. We conducted the attack using the free software Iodine [3] which allows for the creation of IP tunnels over the DNS protocol in order to generate these DNS queries. We were provided with 10 virtual machines, from which we formed a tree topology with 7 of them as host computers and sources of traffic. We then built and implemented a real time data exfiltration detector based off of the techniques proposed in [19]. The detector uses off the shelf compression algorithms like *gzip* in order to measure the information content of any channel in the network. We then set a cut off threshold for the level of allowable information content in any channel. Channels exceeding this threshold are flagged as malicious. While we chose to use this specific detector to generate observations for our model, it is important to note that any other methods for detecting exfiltration would have been equally viable.

6.2 Runtime

We first look at runtimes needed to solve the original model compared to our VD-POMDP model with increasing number of domains. Unless otherwise stated, all test used a threshold discretization of 2. We used an offline POMDP solver ZMPD [22] to compute policies; however, any solver which computes policies for the entire belief space may be used. The largest network we were able to solve for with the original model was one of only 3 nodes. For larger than 2 domains with discount factors $\gamma = -0.2$ and all cases with $\gamma = -0.4$ and $\gamma = -0.8$ the original POMDP did not finish solving in 24 hours and is shown cut off at the 24hr mark in Figure 5a. Consistent with the complexities in Table 2, in Figure 5a we see the runtimes on the y-axis increase exponentially with the number of domains on the x-axis, for the original POMDP. If the VD-POMDP models are solved in parallel, runtimes do not vary with increasing domain. If the models are solved sequentially, then we would see only a linear increase in runtime. However in the case where networks have the channels uniformly distributed among all hosts, i.e. there exists one channel from every host to every domain, then the models become identical, and it becomes only necessary to solve one of them.



(a) Comparing runtimes of the POMDP to the VD-POMDP for a small network of 3 nodes and varying discount factor γ. On the y-axis, we have runtime in log scale and on the x-axis, we have increasing number of domains.

(b) Log-Log graph of runtimes for a single VD-POMDP for networks of increasing size. Both axes are in log scale, with runtimes on the y-axis in seconds and number of nodes on the x-axis, and showing a linear increase with network size.

(c) Runtime for a single VD-POMDP with increasing action space. Test were run on a tree network with branching factor 4 and depth of 3 for a total of 85 nodes.

(d) Runtime required to build network for a single VD-POMDP. Time in seconds is shown on the y-axis and size of network in number of nodes is shown on the x-axis.

Fig. 5: Runtime results

We show the scalability of computing policies for the VD-POMDP in Figure 5b. On the y-axis, in log scale we have the runtime in seconds, and on the x-axis, also in log scale we have the number of nodes in the network, achieved by varying both the depth and branching factor of our tree network structure. We can see that there appears to be a linear scaling with the size of the network. We also show in Figure 5d the time it takes to build the network, the lookup table of costs computed by repeatedly solving (10)-(12) and pomdp files to be fed to the solver. This time corresponds to the steps (a) and (b) in Figure 4. On the y-axis we have again the runtime and on the x-axis the number of nodes in the network.

Figure 5c shows the runtime for computing the policy of a single factored agent with increasing action space. The runtime is shown on the y-axis in seconds, while the increasing action space is measured by the threshold discretization on the x-axis. We first divided the space of possible true positive and true negative rates into a number of segments equal to the discretization number. For each discretization level, we then combinatorially formed all the true positive and true negative pairs possible within that discretization number and averaged over the runtimes, in order to ensure that we were not only testing easy cases, where one choice threshold was dominant over another.

6.3 Performance

We evaluate the performance of the model by looking at the reward, the number of time steps taken to classify all domains and the accuracy of the classifications. For each test, we averaged the values over 100 simulations. Table 3 compares the performance of the original POMDP model with the VD-POMDP model. The largest tests we could run using the full POMDP were on a network of 3 nodes with a total of two domains, while solving the model with a discount factor of $\gamma = 0.2$. The VD-POMDP model performs as well in terms of accuracy and time compared to the full POMDP model. We show a normalized average reward, computed by dividing the total reward by the number of time steps taken to classify the domains to better compare the models. Since we stop the simulation after all the domains have been classified, the total reward is not the expected infinite horizon reward, so simulations which run for different amounts of time will have had the chance to accumulate different amounts of reward. The normalized reward is meant to give a better indication of what the long term average reward would be, which would be a much fairer comparison. We also looked at the VD-POMDP solved with a discount factor of $\gamma = 0.8$, where we can clearly see the benefit of longer term planning. Although this VD-POMDP takes longer to classify both domains, it has a perfect accuracy and lower normalized reward than the other two models. This shows that the model is placing more value on potential future information, by preferring to wait and collect more alerts before making a final decision. This is clearly the better choice as we see a much better accuracy. It is clear that it is necessary to be able to plan for the future to perform well in this kind of domain; it is therefore necessary to be able to solve for large planning horizons, something that we cannot do using just the original POMDP model. This demonstrates the merit of the VD-POMDP framework, as solving this problem with a simple POMDP framework is clearly infeasible.

Looking at just the VD-POMDP we test performance on a variety of larger networks in Table 4. Each of the synthetic networks are tested with 50 domains, averaged over 30 trials. The DETER network is tested with 100 domains averaged over 30 trials. For the DETER network, we used two thresholds, and determined the true and false positive rates of our detector by letting it monitor traffic at each threshold setting and observing the number of alerts obtained for each channel. We found our simple implementation of the detector had true positive rates of $\alpha(\tau 1) \simeq 0.35$, $\alpha(\tau 2) \simeq 0.45$ and true negative rates of $\beta(\tau 1) \simeq 0.8$, $\alpha(\tau 2) \simeq 0.7$, and these were the parameters used in the model for this experiment as well as all the synthetic ones. We can see that, although the

Model	Timesteps to Classify	Attack Traffic Accuracy	User Traffic Accuracy	Normalized Reward
Full POMDP $\gamma = 0.2$	11.814	0.948	0.979	-470.594
VD-POMDP $\gamma = 0.2$	11.144	0.944	0.961	-675.100
VD-POMDP $\gamma = 0.8$	29.044	1.0	1.0	-386.982

Table 3: Comparing performance of full POMDP model to factored model on a small test network of 3 nodes, with 2 domains. One domain is malicious and the other domain is legitimate.

Network	Timesteps to Classify	Attack Traffic Accuracy	User Traffic Accuracy	Normalized Reward
Synthetic 40 Nodes	4.079	1.0	1.0	-13523275.239
Synthetic 85 Nodes	3.252	1.0	1.0	-15514333.580
Synthetic 156 Nodes	3.235	1.0	1.0	-22204095.194
Synthetic 341 Nodes	3.162	1.0	1.0	-21252069.929
DETER	5.3076	1.0	0.995	-6835.588

Table 4: Performance of the factored model on larger networks.

synthetic simulations all perform extremely well, and have a perfect accuracy, the deter simulation occasionally misclassifies legitimate traffic. This is due to the uncertainty in the characterization of the detector, as network traffic is variable and may not always follow a static distribution. Observations for the synthetic experiments were drawn from the distributions that the VD-POMDP had planned for, while in the DETER experiments, traffic did not always follow the input distributions. However, even with this uncertainty, the model still performs well in this realistic network setting. A more sophisticated implementation of this detector along with a more intensive characterization would even further boost the performance.

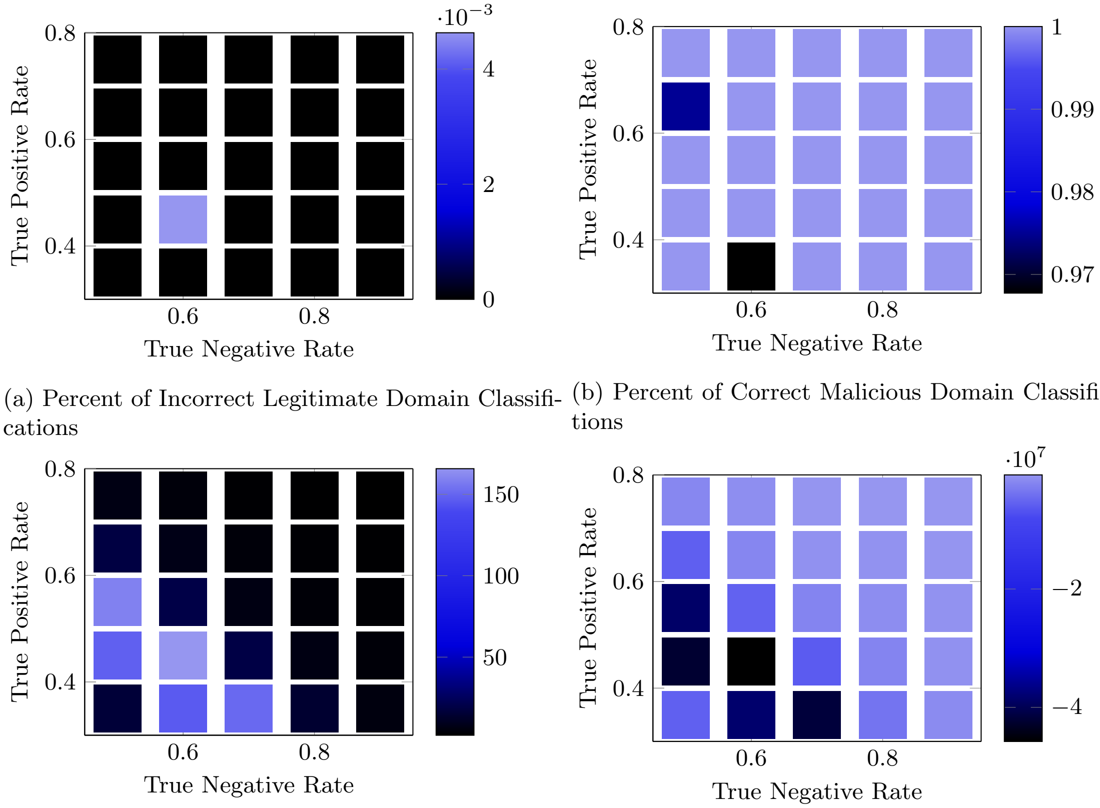
We also show an example of the diversity of actions chosen by the VD-POMDP. In Table 6 we show a trace of the actions taken by a single agent planning for a single domain. We show the number of channels chosen to sense on on, the choice of threshold, along with the classification of the domain. We also show the observations, which the number of channels that triggered alerts. The simulation ends when no more channels are to be sensed on. We can see the agent varying the number of channels as well as the threshold of the detector, as they become more and more sure that they domain is legitimate.

Time	Action			Observations
	# Channels	τ	M_d	
1	64	1	0	23
2	62	1	0	23
3	3	0	0	0
4	5	0	0	2
5	8	0	0	2
6	18	0	0	4
7	0	0	0	0

Fig. 6: Trace of a run on a network of 85 nodes of a single legitimate domain.

6.4 Robustness

Lastly, we looked at the robustness of our model to errors in the input parameter. As evidenced with the DETER experiment, the model requires known false positive and true positive rates for



(a) Percent of Incorrect Legitimate Domain Classifications

(b) Percent of Correct Malicious Domain Classifications

(c) Timesteps to Classify Domain

(d) Average Reward

Fig. 7: Testing the robustness with respect to error in the planned true positive and true negative rate.

the detector. While it may be reasonable to assume that with enough monitoring, it is possible to get accurate measures of false positive rates in a network by simply running the detector on known legitimate traffic for long periods of time, it is more difficult to characterize the true positive rates, as attacks can take many forms and exfiltration can occur over varying rates. In order to test the robustness of our model, we solved for the policy using one set of rates and then tested the model in simulation against a variety of actual rates. For our tests, the model was solved with a true negative rate of 0.8 and true positive rate of 0.55. We then drew alerts from a range of distributions for the true positive and negative rates as shown in Figures 7 on the y-axis and x-axis respectively. The metrics used to measure robustness are shown as a heat-map for each true positive, true negative pair.

In Figure 7a performance of the model was tested by looking at the percent of incorrect legitimate domain classifications ie. the percent of legitimate domains flagged as malicious. In all cases except for one, all legitimate domains were correctly flagged as non-malicious, and in one case legitimate domains were misclassified in only 0.4% of the trials. In Figure 7b the percent of correct malicious domain classifications is shown, where in all but two cases, the correct domain was always identified. Figure 7c shows the number of time steps taken to classify all the domains, while Figure 7d shows the average reward (in this case a penalty) for the the simulations. We can see that the model is robust to mischaracterization of the detectors, where the only dips in performance occur when either the detector has a low true negative rate and when the error in both the true positive and negative rates are large.

7 Conclusion and Future Work

We demonstrated the effectiveness of POMDP based planning tool in making intelligent decisions to tackle the problem of DNS based data exfiltration. These decisions were made by aggregating information from multiple noisy detectors and using sequential planning under uncertainty based reasoning. In doing so, we also proposed a new class of POMDPs called VD-POMDP that uses domain characteristics to split the POMDP into sub-POMDPs and allows for abstract actions in each sub-POMDP that can then be easily converted to a full joint action at execution time. VD-POMDP allows scaling up our approach to real world sized networks. The approach also detects attacks in near real time, thereby providing options to minimize the damage from such attacks. More generally, we believe that our approach applies to other security detection and response problems such as exfiltration over other protocols like HTTP and intrusion detection. While this work is an important step in addressing the problem of APT's in a realistic and scalable manner, we recognize that having a non-adaptive adversary is a simplification of the potentially complex interaction between attacker and defender in this environment. Building an appropriate adversary model, and considering the underlying game in this domain is a key avenue for future work in this area.

8 Acknowledgements

This research was supported by ARO Grant W911NF-15-1-0515.

References

1. Detecting DNS Tunneling. https://www.sans.org/reading-room/whitepapers/dns/detecting-dns-tunneling-34152, accessed: 2016-06-14
2. New FrameworkPOS variant exfiltrates data via DNS requests. https://blog.gdatasoftware.com/2014/10/23942-new-frameworkpos-variant-exfiltrates-data-via-dns-requests, accessed: 2016-06-14
3. Iodine (2014), http://code.kryo.se/iodine/
4. Grand theft data, data exfiltration study: Actors, tactics, and detection (2015), http://www.mcafee.com/us/resources/reports/rp-data-exfiltration.pdf
5. arstechnica: Cluster of megabreaches compromises a whopping 642 million passwords. http://arstechnica.com/security/2016/05/cluster-of-megabreaches-compromise-a-whopping-642-million-passwords/
6. Bernstein, D.S., Zilberstein, S., Immerman, N.: The complexity of decentralized control of markov decision processes. In: Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence. pp. 32–37. Morgan Kaufmann Publishers Inc. (2000)
7. Borders, K., Prakash, A.: Web tap: detecting covert web traffic. In: Proceedings of the 11th ACM conference on Computer and communications security. pp. 110–120. ACM (2004)

8. FarnHam, G.: Detecting dns tunneling. Tech. rep., SANS Institute InfoSec Reading Room (Februrary 2013)
9. Gerkey, B.P., Mataric, M.J.: Multi-robot task allocation: Analyzing the complexity and optimality of key architectures. In: Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on. vol. 3, pp. 3862–3868. IEEE (2003)
10. Hart, M., Manadhata, P., Johnson, R.: Text classification for data loss prevention. In: Proceedings of the 11th International Conference on Privacy Enhancing Technologies. PETS'11 (2011)
11. Journal, T.W.S.: Home depot's 56 million card breach bigger than target's. http://www.wsj.com/articles/home-depot-breach-bigger-than-targets-1411073571
12. Jung, H., Tambe, M.: Performance models for large scale multiagent systems: using distributed pomdp building blocks. In: Proceedings of the second international joint conference on Autonomous agents and multiagent systems. pp. 297–304. ACM (2003)
13. Labs, T.: Data exfiltration: How do threat actors steal your data? (2013), http://about-threats.trendmicro.com/cloud-content/us/ent-primers/pdf/how_do_threat_actors_steal_your_data.pdf
14. Madani, O., Hanks, S., Condon, A.: On the undecidability of probabilistic planning and infinite-horizon partially observable markov decision problems. In: Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence. pp. 541–548. AAAI '99/IAAI '99, American Association for Artificial Intelligence, Menlo Park, CA, USA (1999)
15. McAfee: Data loss prevention. http://www.mcafee.com/us/products/total-protection-for-data-loss-prevention.aspx
16. McAllester, D.A., Singh, S.: Approximate planning for factored pomdps using belief state simplification. In: Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence. pp. 409–416. Morgan Kaufmann Publishers Inc. (1999)
17. Nair, R., Varakantham, P., Tambe, M., Yokoo, M.: Networked distributed pomdps: A synthesis of distributed constraint optimization and pomdps. In: AAAI. vol. 5, pp. 133–139 (2005)
18. Papadimitriou, C.H., Tsitsiklis, J.N.: The complexity of markov decision processes. Mathematics of operations research 12(3), 441–450 (1987)
19. Paxson, V., Christodorescu, M., Javed, M., Rao, J., Sailer, R., Schales, D., Stoecklin, M.P., Thomas, K., Venema, W., Weaver, N.: Practical comprehensive bounds on surreptitious communication over dns. In: Proceedings of the 22Nd USENIX Conference on Security. pp. 17–32. SEC'13, USENIX Association, Berkeley, CA, USA (2013), http://dl.acm.org/citation.cfm?id=2534766.2534769
20. Seth Bromberger, Co-Principal Investigator, N.: Dns as a covert channel within protected networks. Tech. Rep. WP2011-01-01, National Electric Sector Cyber Security Organization (January 2011)
21. Silver, D., Veness, J.: Monte-carlo planning in large pomdps. In: Advances in neural information processing systems. pp. 2164–2172 (2010)
22. Smith, T.: Probabilistic Planning for Robotic Exploration. Ph.D. thesis, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA (July 2007)
23. Sommer, R., Paxson, V.: Outside the closed world: On using machine learning for network intrusion detection. In: Security and Privacy (SP), 2010 IEEE Symposium on. pp. 305–316. IEEE (2010)
24. Symantec: Data Loss Prevention & Protection. https://www.symantec.com/products/information-protection/data-loss-prevention
25. Varakantham, P., young Kwak, J., Taylor, M., Marecki, J., Scerri, P., Tambe, M.: Exploiting coordination locales in distributed pomdps via social model shaping (2009), http://aaai.org/ocs/index.php/ICAPS/ICAPS09/paper/view/733/1128
26. Velagapudi, P., Varakantham, P., Sycara, K., Scerri, P.: Distributed model shaping for scaling to decentralized pomdps with hundreds of agents. In: The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 3. pp. 955–962. International Foundation for Autonomous Agents and Multiagent Systems (2011)
27. Wikipedia: Office of personnel management data breach. https://en.wikipedia.org/wiki/Office_of_Personnel_Management_data_breach