Reducing Attack Surface by Learning Adversarial Bag of Tricks.

Jonathan Goohs[1,2], Ray Mier[1,2], Paul Deist[1,2], and William Casey[2]

[1]United States Navy
[2]United States Naval Academy

May 31, 2022

Abstract

Network defense fundamentally involves decision-making to minimize hazards under resource constraints. One clear example is minimizing the attack surface by patching or neutralizing hazardous software vulnerabilities. We define a normative decision theory for optimal patch planning by developing the mathematical problem and game theoretic scenario for defending a network against strategic adversaries. First, we evaluate the static problem, where the network control state is fixed, and hazard scores (such as CVSS) are assumed accurate. The *defenders* optimal *patch plan* will maximize the expected hazard removed. We show that the static problem is a constrained optimization problem reducing to WEIGHTED KNAPSACK, and optimal patch plans are efficiently calculated with *binary programming*. The dynamic problem is constructed as a sequence of static problems connected by reachability conditions of the network control graph, where each node's control states are determined by player actions *flipping a node* as described in network Flip-it Games. Several non-intuitive but insightful mathematical observations are detailed for the dynamic case. For example, while each network state (of control) yields the best defense (solved locally), the patchwork of local solutions is shown to be highly discontinuous and sensitive to both network topology and adversarial actions. We further investigate how the model provides a foundation for defenders to improve patch planning by learning the attack preferences of adversaries. We summarize the mathematical insights for the patch planning problem. We conclude by discussing network design elements that may further the dual objectives of defending networks while learning adversarial attack patterns.

1 Introduction

Vulnerabilities are fault conditions in software that attackers can exploit[1]. Vulnerability patching is the security action of defenders to remove or neutralize known software vulnerabilities so attackers cannot exploit them.

Because patching is resource-intensive and costly, the amount of patching a defender can apply is subject to a resource constraint, often far less than the total capacity needed to address the myriad vulnerabilities a network can present. Notwithstanding the problem of vulnerability identification (or inventory) and technical capacity for remediation, the resource limitation gives rise to a considerable planning problem that is the focus of this work. The network defender must select which vulnerabilities to patch and in what order to minimize *attack surface* thereby preventing the attacker from exploiting a defended network. Additionally, attackers do not announce their intentions, preferences, or plans, so defenders often plan security actions with zero or partial information about the vulnerabilities attackers may target.

Hazard scores for vulnerabilities have been suggested to assist defensive efforts, the most well-known being the Common Vulnerability Scoring System (CVSS). CVSS measures a range of exploitation hazards stemming from each known vulnerability. The higher the score, the greater the risk that adversaries could leverage the vulnerability to exploit a defended network. As such, scoring systems, including CVSS, enable a normative decision-making approach well suited for expected utility optimization. Under the assumption that the attacker's preference is proportional to the hazard score, we show that scores can be used to minimize the attacker's expected rewards. Calling upon the mathematics of constrained optimization for both definition and solution, we define a constrained optimization problem for the static case describing the local scenario (when network control states are fixed) between a single attacker and a network defender. The optimization problem reduces to Weighted Knapsack. Weighted Knapsack problems are solved (usually efficiently) with binary programming, a variant of integer programming to render a policy or *patch plan* that specifies an actionable sequence of patches to apply and the resources used to do so.

With the constrained optimization problem made concrete, we gain insight into the mathematical nature of the dynamic problem where defense goals for a network can structurally change due to an attack, i.e., a network Flip-It Game [33, 25]. We consider the Flip-It game to model the state of control for a network asset. In Flip-It, each time step is a zero-sum value whose ownership resolves to the last attacker (or player who most recently flipped its control state). Additionally, control states are unknown to players, who pay to audit/attack a node to check/take ownership. Viewed alongside Flip-It games, patching is a defensive action to modulate critical parameters of Flip-It game, amplifying the cost and attenuating the likelihood of a successful attack upon a node - thus minimizing

[1] A cyber attack would exploit a vulnerability to gain control of a network node, breach privacy, steal digital assets, disrupt a critical service, and so on.

expected loss in zero-sum games. To formalize the dynamic problem, we introduce a control graph whose nodes represent a state of control for the network and edges describe reachability through a single action (for example, a payer attack can flip a server node's control state). The dynamic problem considers all possible sequences of static games defined by reachability for the subjacent control graph. Several non-intuitive consequences are highlighted. Not only can structural goals of patch planning change, but changes can be highly discontinuous and even invert the gains of prior defense actions. To our knowledge, this study is the first to explore the stability properties of patching strategies in the dynamic case. From the dynamic case, we discuss implications for the security problem at large and turn to the notion of learning attack strategies. Since cyber attackers are likely to have preference structures determining which vulnerability to exploit and in which order, we ask if the normative theory can be enhanced by a learning process that actively improves hazard estimates as experience with attackers reveals their preferences. We provide a foundation for learning methods that balance the two aims of learning adversarial preferences and optimizing defenses.

2 Background and Motivation

Strategic resourcing of cyber defense is a critical problem for cyber security [28]. Software vulnerabilities are one of the key sources of concern, as attackers can deploy exploits against them. With the increasing amount of vulnerabilities and devices connected to the internet, nearly every business has some version of the described resource allocation problem required to patch vulnerabilities. The threat is more apparent for some organizations, such as large technology companies. Other businesses, for example, a mom-and-pop shop with five computers on their network, may know very little about the cyber defense resource allocation problem or that it even exists. No matter the scale of the organization, it is crucial that defining a network security framework is imperative in the current cyber domain.

A free and open system known as the CVSS (Common Vulnerability Scoring System) provides a comprehensive assessment of known vulnerabilities for IT assets on a scale of 1-10 [22]. The CVSS provides an excellent resource for organizations to understand how serious a vulnerability may be on their networks and point to actionable mitigations such as security patches that remove the exposure. However, CVSS also provides cyber adversaries with a detailed description of what vulnerabilities could be used to exploit an organization if it is not capable of patching all known vulnerabilities. CVSS scores introduce another interesting dynamic as their descriptions also can provide an explanation of how network defenders are patching their vulnerabilities, which can aid cyber adversaries in their following exploits. From an organizational perspective, listing the full description of a vulnerability may not be desired, and so CVSS scores can sometimes be viewed and are often criticized as incomplete [30, 32].

The CVSS score value is broken up by three sets of metrics: *base*, *temporal*,

and *environmental* [2]. The base metric set is broken down into the exploitability of the vulnerability and the impact the vulnerability would have on a virtual environment if exploited. The temporal set of metrics deals with how valid the vulnerability report is and how it can be patched over time. Environmental metrics mainly focus on the basic principles of cybersecurity: Confidentiality, Integrity, and Availability[2].

The notion of scanning and patching vulnerabilities has ushered in many commercialization and open source software efforts with various programs in the space. Some of the scanners include Nessus vulnerability scanner [6], OpenVAS vulnerability scanner [9], SolarWinds Security Event Manager [12], ManageEngine Vulnerability Manager [5], Paessler PRTG Network Monitor[10], Nexpose network risk monitoring tool [8], Tripwire IP360 Agent-Based Vulnerability Management [13], Intruder IO [4], ImmuniWeb AI vulnerability platform [3], Netsparker [7], PortSwigger [11], and Acunetix [1].

The notion of hazard score refinement has prior art. Gallon and Bascau in [23] used CVSS scores with attack graphs to re-calculate a damage score. This broadened the notion of hazard as a function that can be refined based on network attributes and policies. The dynamic case where network states can be affected by an attacker was left as an open problem. Here we extend the prior art by filling in the important and realistic question of how defense resourcing should be adjusted dynamically when nodes of the network are either unwillingly or unwittingly ceded to attackers. This is done by considering the mathematical objectives for minimizing hazards as they depend dynamically on the control states of the network.

The dynamic scenario of changing control states has been modeled game theoretically with FlipIt games [33]. FlipIt considers a network node as a resource that any player can attack; while attacking has a cost, the benefit is owning the resource whose reward is proportional to the time it is owned. Players have no free information as to the actual machine state but can either audit or attack to determine the control state temporarily. In this paper, we outline the normative decision theory for defense in a network version of FlipIt [25], and illustrate the mathematical challenges which arise.

Liu et al. considered how forensic evidence can reveal attack graphs in [26], indicating learning from prior attacks. The authors used probabilistic attack graphs and detailed how the inference can be improved as evidence is gained; however, recommendations for defensive re-prioritization were not made. Attack graph recovery has been further considered in [16] Other deceptive technologies such as honey pots have been considered for learning attack graphs [15, 14, 21] and optimal network hardening by use of attack graphs and deception in [21]. While some defensive measures have been considered, the subject of how to resource patch planning efforts is left open. Our contribution of analyzing the mathematical structure of optimal patch planning can provide a sound foundation for the dual objectives of prescribing policy and learning from attackers by leveraging prior art.

3 Methods and Materials

3.1 The Static Problem

Let $\mathcal{M} = \{m_1, m_2, \ldots m_\omega\}$ be a set of network nodes (i.e., machines or components) controlled by the defender who commissioned, powers, and maintains the nodes. The network topology is defined by single hop access from node to node and is given by the edges $\mathcal{E} \subset \mathcal{M} \times \mathcal{M}$.

Inherent to each machine is an *attack surface* that is a set of vulnerabilities that an attacker could potentially use to take control. The simplest game model has two players, a defender and an attacker, who play for a zero-sum (for each node for each unit of time). The Attacker selects a vulnerability on the defender machine, and the attack is successful if the vulnerability remains unpatched. The attack fails if the defender had patched the vulnerability before the attack. If the attack succeeds, the attacker receives (the defender loses) the node value until the defender can take it back by cleaning up the machine. If the attack fails, the defender remains in control of the node.

Inventory: Let V be a finite (but potentially large) set of commonly known vulnerabilities (or ports of attack). Note that the vulnerabilities may refer to any component of hardware or software, service, or procedure; therefore, we model them abstractly as a set and defer treatment of their complex dependencies for later. Associated to each machine is a set of commonly known vulnerabilities[2] yielding a function:

$$\mathcal{V} : \mathcal{M} \to 2^V : m_j \to \mathcal{V}_j,$$

with $\mathcal{V}_j = \mathcal{V}(m_j)$ being a discrete list of vulnerabilities afflicting node (machine or component) m_j.

Hazards: Associated with each vulnerability is a hazard and cost structure for remediation. Let

$$h : V \to \mathcal{R}^+ : v_i \to h_i,$$

with h_i a measure of hazard associated with vulnerability v_i. Scoring systems such as CVSS are based on complex considerations of multiple factors but are designed to measure a notion of hazard.

Cost Structure for vulnerability patching: Remediation of vulnerability v_i will have a simple linear cost structure:

$$c_i(n) = \begin{cases} 0 \text{ if } n = 0, \\ b_i + m_i n \text{ for } n > 0. \end{cases}$$

The cost scales with n, the number of patches applied across the network.

Attack Surface: For a network, the attack surface will refer to the set of machine nodes and services that are reachable to an attacker. These are most commonly thought of as servers (e.g., web servers, DNS servers, VPN servers,

[2]Here it is possible to model information, as differing agents may have different knowledge of vulnerabilities, to keep it simple we restrict ourselves to work only with commonly known vulnerabilities such as vulnerability lists that USCert widely tracks

and firewalls) that interface a private network to the World Wide Web. The vulnerabilities present on the attack surface are the main ports of entry for an attacker. The informational asymmetry is most critical on the attack surface. While the defender privately knows the true state of each vulnerability (i.e., its presence and patch level), the attacker privately knows their attack capabilities (i.e., which attacks they are likely to use). The notion of readability can be modeled by an indicator function: $\mathcal{X} : \mathcal{M} \to \{0, 1\} : m_i \to \mathcal{X}_i$ With

$$\mathcal{X}_i = \begin{cases} 1 \text{ if } m_i \text{ is reachable to an attacker} \\ 0 \text{ otherwise} \end{cases}$$

In the static problem, we can assume the entire network is visible to the attacker, or without loss of generality, the essential problem is played only over the reachable network.

Attacker Abstraction: Here, we make explicit the assumption underlying the hazard functions and how it relates to the attacker's goal. This assumption will later be helpful when we assume a more realistic proposition that attacker preferences are unknown by the defender and therefore must be learned. *Attacker Assumption I: The attacker will select only from known vulnerabilities, with a frequency in proportion to a commonly known hazard score. Further, we can assume the attacker will select a reachable computer to attack with uniform random probability. The attacker is successful only when they select an unpatched vulnerability.* Assumption I is overly simplistic but is useful for analysis. The more general case is when the *true* hazard scores are the private information of the attacker, and the defender can construct of incremental hazard improvement scheme aimed to converge to that of its attacker's private and withheld preferences.

3.1.1 Static Problem and Optimal Patch Planning

The goal of resource-limited defense is to remove a maximal amount of hazard from the network while satisfying resource constraints.

Letting variables ξ_{ij} be zero/one variables to indicate whether machine j has vulnerability i, we can aggregate the network hazards along the axis of both nodes and vulnerabilities. The node component of network hazard is: $\sigma_j = \sum_{i=1}^{N} \xi_{ij} h_i$ with the sum running over the vulnerability index i. The vulnerability contribution to network hazard is: $\gamma_i = h_i n_i$ where $n_i = \sum_{j=1}^{M} \xi_{ij}$ with j ranging over the machine index and thus aggregating the number of vulnerability instances for v_i found on the network.

The zero-one matrix ξ are parameters of the specific problem instance having a total network hazard, $Z = \sum_{j=1}^{M} \sigma_j = \sum_{i=1}^{N} \gamma_i$. A number of nodes M, number of vulnerabilities N and two cost parameters per vulnerability, as well as a zero-one matrix $N \times M$ such as viewed in table 1 represent the parameters for all such problem instances.

Decision Space: Letting ζ_{ij} (for i ranging over the vulnerability index and j ranging over the node index) be the zero/one decision variables determining

patch	cost	haz	vul	m_1	m_2	m_3	$\dots$	m_j	$\dots$	m_M	v tot
b_1	m_1	h_1	v_1	ξ_{11}	ξ_{12}	ξ_{13}	$\dots$	ξ_{1j}	$\dots$	ξ_{1M}	γ_1
b_2	m_2	h_2	v_2	ξ_{21}	ξ_{22}	ξ_{23}	$\dots$	ξ_{2j}	$\dots$	ξ_{2M}	γ_2
b_3	m_3	h_3	v_3	ξ_{31}	ξ_{32}	ξ_{33}	$\dots$	ξ_{3j}	$\dots$	ξ_{3M}	γ_3
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\iddots$	$\vdots$	$\vdots$
b_i	m_i	h_i	v_i	ξ_{i1}	ξ_{i2}	ξ_{i3}	$\dots$	ξ_{ij}	$\dots$	ξ_{iM}	γ_i
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\iddots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
b_N	m_N	h_N	v_N	ξ_{N1}	ξ_{N2}	ξ_{N3}	$\dots$	ξ_{Nj}	$\dots$	ξ_{NM}	γ_N
node totals				σ_1	σ_2	σ_3	$\dots$	σ_j	$\dots$	σ_n	Z

Table 1: Network hazard Z as the aggregate of hazards arising from each vulnerability instance (summing rows in the last column), or arisen from each machine (summing columns in the last row).

whether or not to patch[3] vulnerability v_i on machine m_j. Letting $x_i = \sum_{j=1}^{M} \zeta_{ij}$, a hazard reduction of $x_i h_i$ can be achieved with cost $c_i(x_i)$

Letting $\mathbf{x}$ be the selection of how many instances of each vulnerability to patch, we can now formulate the constrained optimization problem as:

$$\max_{\mathbf{x}} \Phi(\mathbf{x}) \quad \text{with} \quad \Phi(\mathbf{x}) = \sum_{i=1}^{N} h_i x_i. \qquad \text{(Objective)}$$

Said differently, the inner product between *the hazard load vector* $\mathbf{h}$ and the decision vector $\mathbf{x}$. This maximization is subject to the constraint:

$$\sum_{i=1}^{N} c_i(x_i) \leq C \qquad \text{(Resource Limit)}$$

.

We refer to the arg-max $\mathbf{x}$ as the *optimal selection* given the constraint C and this can determine (possibly non-uniquely) *a patch plan* in the form of zero-one matrix ζ_{ij}. An optimal selection can be seen to satisfy: $0 \leq x_k \leq n_k$.

Computational Solution The problem of determining an optimal selection can be reduced to weighted knapsack problems solved with integer programming. Equivalently the problem of determining an optimal patch plan is an instance of binary programming (a special case of integer programming). Our reduction to weighted knapsack by the Method of Lagrange is found in the appendix. Briefly, the method of Lagrange reformulates constrained optimization into unconstrained optimization problems by using additional slack variables that are necessarily zero when the objective function is optimized.

[3]Notice the notation allows fixing a nonexistent vulnerability $\zeta_{ij} = 1$ when $\xi_{ij} = 0$ but the goal to reduce hazard should avoid any such assignment as a non-productive use of resources. Therefore we keep the notation primitive and simple.

The weighted knapsack problem curtails a general NP-Complete decision problem but admits to pseudo-polynomial time heuristic algorithms and can often be solved efficiently for random instances [31, 17, 27]. In the implementation, we use the Julia programming language, and the JuMP [20] module for optimization and linear programming.

3.2 Dynamic Problem

Cyber attacks are characterized by unwilling and often unknown loss of control of services or network nodes. Attacks are achieved by taking unauthorized control of a network node, thereby altering the network control state. We represent network control states by a temporal function Γ that assigns nodes to agent players. To focus on the essential problem, we assume that only cyberattacks cause changes to the network control state Γ, ignoring for now other types of updates, e.g., voluntary transfer of control, leasing, licensing, containers, virtualization, and so forth.

3.2.1 Dynamic Networks and Control

To describe the dynamic problem, we will assume a temporal domain comprised of regular time steps: $T = \{t_0 + kh : k \in \{0, 1, 2, \cdots\}\}$, The only parameters for the time domain is an initial reference time t_0 and unit time step $h = \Delta t$. Time step k will refer to $t_0 + kh$.

Let $\mathcal{M} = \{m_1, m_2 \dots m_\omega\}$ be a set of nodes. The set of edges at time step k will be denoted by: $\mathcal{E}_k$, with $\mathcal{E}_k \subset \mathcal{M} \times \mathcal{M}$. The time-indexed edge sets provide a snapshot of node connectivity; this snapshot captures the momentary network topology (including all possible single-hop communications in the network, whether they are utilized or not). The dynamic network is denoted by: $\mathcal{N}_k = \langle \mathcal{M}, \mathcal{E}_k \rangle$. Without any loss of generality, the essential problem can be expressed with a network of fixed size (Note 0 in table 1) as machines *introduced into* and machines *removed from* the network can be represented with appropriate isolating edge modifications[4].

Within the network, the usual notion of neighborhood applies and can also be indexed by time; that is, for any $m_j \in \mathcal{M}$: $\mathcal{N}_{jk} = \{m' \in \mathcal{M} : (m, m') \in \mathcal{E}_k\}$.

As in the static case, V is the set of commonly known vulnerabilities. The state of vulnerability for any particular node/service/component is extended to a spatial-temporal function having domain of $\mathcal{M} \times T$ (node/services, time step) and range the subsets of V:

$$\mathcal{V} : \mathcal{M} \times T \to 2^V : (m_j, k) \to V_{jk},$$

with V_{jk} enumerating the active vulnerability instances for machine j at time step k.

Control of nodes: Within the dynamic scenario we consider a set of agents $\mathcal{A} = \{a_1, a_2, \cdots a_\nu\}$, and control function $\Gamma : \mathcal{M} \times T \to \mathcal{A}$ that provides the

[4] Without loss of generality, an isolated machine can be linked into the network at any time and can be isolated away from the network topology at a later time.

state of control for each network node, $\Gamma(m_j, t_k) = a$, if node j is controlled by agent a at time step k. We will use the notation convention that $\Gamma_{jk} = i$ when $\Gamma(m_j, t_k) = a_i$. We note in table 1 (Note 2) that the dynamic nature of the game allows for control states to change or flip as in FlipIt game [33, 25].

Interior, Boundary, and Contested Zones: At each time step, each stakeholder controls a portion of the network denoted by $\Psi_{ek} := \{j : \Gamma_{jk} = e\}$. The interior of control at time step k can be defined as:

$$\Psi^{\circ}_{ek} = \{m_j \in \mathcal{M} : \mathcal{N}_{jk} \subset \Psi_{ek}\}.$$

The interior is the set of nodes that are not immediate neighbors to nodes owned by other agents. The boundary or *Contested zone* at time step k is: $\Delta\Psi_{ek} := \Psi_{ek} \setminus \Psi^{\circ}_{ek}$. Consistent with the notation introduced in the static problem $\mathcal{X}_{jk}$ are indicators for reachable nodes at time step k, we will have: $\mathcal{X}_{jk} = 1 \Leftrightarrow m_j \in \Delta\Psi^k_e$ for any e (ranging over the index for agents).

Notice that when control states change or flip, this directly affects both interior and boundary regions of control. As such, how to best prioritize defense for the dynamic problem, where the control for nodes can change, naturally arises as an important question.

Attacker Abstraction: Again, we make explicit the assumption underlying the hazard function and its relation to attacker actions.

Attacker Assumption II: An attack occurs at a boundary node chosen uniform randomly. An attack is successful only when the vulnerability attacked remains unpatched. A successful attack changes the control state of the network, setting the attacker to be the controlling agent of the attacked node. An unsuccessful attack leaves the control state unchanged. Additionally, the attacker will select a vulnerability to attack in proportion to the hazard score. Further, we will continue by assuming the network topology remains fixed.

We extend the indicator functions along the temporal axis, let:

$$\xi_{ijk} = \begin{cases} 1 \text{ if } v_i \in V_{jk} \\ 0 \text{ o.w.} \end{cases}$$

The node component of network hazard is extended to a temporal function: $\sigma_{jk} = \sum_{i=1}^{N} \xi_{ijk} h_i$ with the sum running over the vulnerability index i. The vulnerability contribution to network hazard is likewise extended: $\gamma_{ik} = h_i n_{ik}$ where $n_{ik} = \sum_{j=1}^{M} \xi_{ijk}$ with j ranging over the machine index to aggregate the number of vulnerability instances for v_i found on the network.

The zero-one tensor ξ summarizes the distribution of vulnerabilities and determines the total network hazard at time k: $Z_k = \sum_{j=1}^{M} \sigma_{jk} = \sum_{i=1}^{N} \gamma_{ik}$. The dynamic network problem scenario is summarized in table 2.

3.2.2 Dynamic Problem and Optimal Patch Planning

In the dynamic problem, the defender will have a fixed budget constraint for each time period. The goal can then be to plan a sequence of patch plans to remove a maximal amount of hazard. Hazard removed will be measured in

patch cost		haz	vul	m_1	m_2	m_3	$\dots$	m_j	$\dots$	m_M	v tot
b_1	m_1	h_1	v_1	ξ_{11k}	ξ_{12k}	ξ_{13k}	$\dots$	ξ_{1jk}	$\dots$	ξ_{1Mk}	γ_{1k}
b_2	m_2	h_2	v_2	ξ_{21k}	ξ_{22k}	ξ_{23k}	$\dots$	ξ_{2jk}	$\dots$	ξ_{2Mk}	γ_{2k}
b_3	m_3	h_3	v_3	ξ_{31k}	ξ_{32k}	ξ_{33k}	$\dots$	ξ_{3jk}	$\dots$	ξ_{3Mk}	γ_{3k}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\iddots$	$\vdots$	$\vdots$
b_i	m_i	h_i	v_i	ξ_{i2k}	ξ_{i2k}	ξ_{i3k}	$\dots$	ξ_{ijk}	$\dots$	ξ_{iMk}	γ_{ik}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\iddots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
b_N	m_N	h_N	v_N	ξ_{N1k}	ξ_{N2k}	ξ_{N3k}	$\dots$	ξ_{Njk}	$\dots$	ξ_{NMk}	γ_{Nk}
node totals				σ_{1k}	σ_{2k}	σ_{3k}	$\dots$	σ_{jk}	$\dots$	σ_{nk}	Z_k
control state				Γ_{1k}	Γ_{2k}	Γ_{3k}	$\dots$	Γ_{jk}	$\dots$	Γ_{Mk}	

Table 2: Network Hazard in the dynamic case is a sequence of reachable static problems indexed by time step k. The machine control states reflects the agent controller will depend on time (last row), and this can dramatically alter the optimization goal for the problem instance at time k depending on who onws what.

expectation (averaged over all possible control state transitions) over a finite (i.e., time steps $k \in [0, T]$) or infinite horizon (time steps $k \in [0, \infty)$). We use the standard Markov Decision Process reward sums, which in general applies a geometric sum discount factor to future rewards. Letting $R(k)$ be the reward at time step k (i.e., the hazard removed), and T the horizon, the general objective is to maximize a discounted sum of the form:

$$\sum_{k=1}^{T} \delta^k R(k),$$

where δ is known as the *discount factor*.

The objective can be viewed as a sequence of static problems with complex dependencies on prior control states, which in turn depend on prior player actions. The best future actions may generally be quite sensitive to player actions. The set (and hence search) of possible outcomes grows exponentially in time steps. We will only state the constrained optimization function from the perspective of the defender agent.

Decision Space: Let ζ_{ijk} be the zero/one tensor or decision variables that determine whether or not vulnerability v_i is patched on machine m_j at time point k. We will assume vulnerabilities do not re-emerge, thereby removing the vulnerability at time k will ensure its removal at all time points larger than k (within the horizon $[k, T]$) regardless of control changes to the node.

Let $x_{ik} = \sum_{j=1}^{M} \zeta_{ijk}$ be the number of instances of v_i patched at time k, we have:

$$R(k) = \sum_{i=1}^{M} x_{ik} h_i.$$

The discounted hazard removed up to T is:

$$\sum_{k=1}^{T} \delta^k R(k).$$

Let c_{ik} denote $c_i(x_{ik})$, or the aggregate investment to remediate v_i at time step k. Further let S be a service interval ($S << T$) where a resource limit places capacity C_S on the resources expended over the service interval. We denote the time difference operator as Δ, for example $\Delta c_{ik} = c_{i(k+1)} - c_{ik}$ and $\Delta^S c_{ik} = c_{i(k+S)} - c_{ik}$

The constrained optimization problem is:

$$\max_{\zeta} \Phi(\zeta) \quad \text{with} \quad \Phi(\zeta) = \sum_{k=1}^{T} \delta^k \sum_{j=1}^{M} \sum_{i=1}^{N} h_i \zeta_{ijk} \qquad \text{(Objective)}$$

Subject to the constraints that for each k:

$$\sum_{i=1}^{N} \Delta^S c_{ik} \leq C_S \quad \text{for} \quad k \in [0, S, 2S, \ldots (\lfloor T/S \rfloor) S] \qquad \text{(Service Resource Limit)}$$

We refer to the arg-max ζ as the *optimal patch plan.*

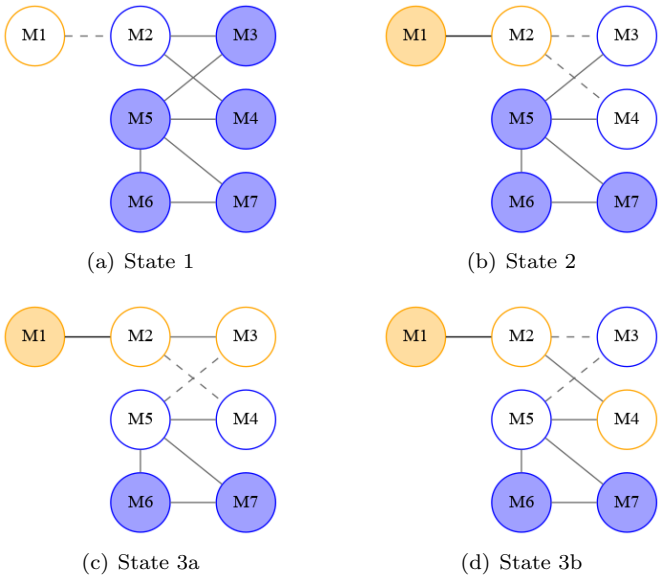
Computational solution: Optimization for this problem with heuristic methods is planned in our future work. The problem contains several interesting possibilities, including branch and bound, memoization, and finite-horizon dynamic programming for solving various substructures. Ultimately we are interested in constructing effective heuristic methods which can iteratively improve expected rewards. We further anticipate the role of learning adversarial preference to play a significant part.

3.2.3 An Example Dynamic Problem

To illustrate some of the dynamic problem's mathematical properties, we construct an artificially small but instructive example. We consider seven nodes, five vulnerabilities, and two agents, and illustrate (in figure 1) several states of a network's control graph reachable by attacker actions. Note that temporally state 1 can transition to state 2 when agent 2 takes control of m_2, from state 2 parallel possible transformations are shown, $3a$ is achieved when agent 2 takes m_3 while $3b$ is achieved when agent 2 takes m_4. Additionally, table 3 clarifies the state of node vulnerability and boundary for each of these states. Finally, table 4 summarizes the associated constrained optimization problem for each defending agent for each state described.

The example helps to illuminate both the essential aspects and challenges which the dynamic problem engenders. In particular *discontinuity in time* and *variation in future states.*

Temporal Discontinuity: just by changing the boundary in a single step from state 1 to state 2, vastly different numerical optimization problem results.



(a) State 1 (b) State 2

(c) State 3a (d) State 3b

Figure 1: Dynamic States of Control for a Network Graph. Two agent controllers (orange and blue) are shown in several states. Nodes are drawn as circles, the color of the node boundary will indicate the node's controller, the interior of a node is colored if the node is within the player's interior of control, otherwise it is a boundary node. The contested zone are boundary nodes where the essential static game is played. The diagram illustrates four distinct states of control, a control state can transition by flipping a node in the contested zone, dotted lines illustrate the possible routes of attack. State 1 can be transformed to state 2 by the orange agent flipping control of $M2$. State 2 can be transformed to state $3a$ or state $3b$ if orange agent attacks $M3$ or $M4$ respectively.

This discontinuity can be fairly dramatic; for example, the blue player goes from considering two vulnerabilities to five (see table 4). Further, a hazard function that places twice the hazard on v_3, v_4, v_5 vs v_1, v_2 could potentially shuttle any ongoing efforts to address or patch the security problems from state 1, as something more than twice as important arises from the transition.

Variation of Future States: Notice also that there can be a great discrepancy between possible next states. This suggests the complexity of planning for the future, even for a single step, may entail vastly differing patch plans depending entirely upon an adversary's steps. In our example the hazard (see table 4) for v_3 activated in state $3a$ and v_5 activated in state $3b$ could be equally high and dwarf all other concerns.

Additionally, similar discontinuities and variations arise from an unpatched vulnerability being subsumed by other agents when control of a node is flipped. Even more curious is the possibility that an unpatched vulnerability could pro-

Node	V_1	V_2	V_3	V_4	V_5	$\Delta\Psi^1$	$\Delta\Psi^2$	$\Delta\Psi^{3a}$	$\Delta\Psi^{3b}$
M_1	1	0	0	1	0	1	0	0	0
M_2	0	1	1	0	0	1	0	0	0
M_3	1	1	0	0	1	0	1	1	1
M_4	0	1	1	1	0	0	1	1	1
M_5	1	0	0	1	0	0	0	0	1

Table 3: dynamic boundaries affect the loading coefficients for the static problem

time step	Player-1(blue)	Player-2(orange)
$k = 1$	$\max(v_2x_2 + v_3x_3) : c_2x_2 + c_3x_3 < C_1$	$\max(v_1x_1 + v_4x_4) : c_1x_1 + c_1x_1 < C_2$
$k = 2$	$\max(v_1x_1 + 2v_2x_2 + v_3x_3 + v_4x_4 + v_5x_5)$ $: c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4 + c_5x_5 < C_1$	$\max(v_2x_2 + v_3x_3) : c_2x_2 + c_3x_3 < C_2$
$k = 3a$	$\max(v_1x_1 + v_2x_2 + v_3x_3 + 2v_4x_4)$ $: c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4 < C_1$	$\max(v_1x_1 + 2v_2x_2 + v_3x_3 + v_5x_5)$ $: c_1x_1 + c_2x_2 + c_3x_3 + c_5x_5 < C_2$
$k = 3b$	$\max(2v_1x_1 + v_2x_2 + v_4x_4 + v_5x_5)$ $: c_1x_1 + c_2x_2 + c_4x_4 + c_5x_5 < C_1$	$\max(2v_2x_2 + 2v_3x_3 + v_4x_4)$ $: c_2x_2 + c_3x_3 + c_4x_4 < C_2$

Table 4: The dynamic sequence of static problems.

vide a bridge back to restoring an agent's control for a node taken. This suggests a novel variation of Flip-It game where the means to flipping a node can be added or removed as a new type of action. While in typical networks, physical control may obviate these possibilities, the prospect of ad hoc networks may enable such strategies as conceivably rewarding. Still, the effects of prior patches/investments in network nodes whose control states are fluid can engender the possibility of transferred and shared utility.

The geometry of the static problem variation (as above) are likely to indicate the following tradeoff. Clonal nodes reduce variation and discontinuities in the structure of defending a network but can be efficiently attacked by attackers if they exploit specific vulnerabilities. On the other hand, diverse and Non-clonal nodes introduce more significant variation and discontinuities but are less likely to be efficiently exploited by attackers.

3.3 Learning an Adversary's Bag of Tricks

Node vulnerabilities can be exploited by an attacker to flip control of a system. To do so an attacker must select an available vulnerability which remains unpatched, and further must be capable of exploiting it. There are many vulnerabilities and since developing capable exploits for any vulnerability is costly, attackers will naturally reuse exploits and express pointed preferences to use vulnerabilities they have used in the past.

Our next goal involves iterative learning better hazard scores based on a specific attacker. To model the attacker, we will use a preference function. In

a game such as network FlipIt, the preference function would be private information to the attacker but could be learned by other players during repeated interactions. We consider the defense problem to incrementally improve hazard scores as the adversaries' preference distribution can be learned. This learning objective can improve expected hazard reduction with specific re-weighting of hazards in ongoing patch planning problems.

Let $\mathcal{F}^N$ be the set of preference functions over N objects (i.e., vulnerabilities $|V| = N$). We will have $f \in \mathcal{F}^N$ if $f : V \to \mathbb{R}^+ : i \to f(i)$ thereby assigns non-negative preference values to the set of commonly known vulnerabilities would be a member of $\mathcal{F}^N$. Let $\mathcal{P}^N$ be the space of discrete probability vectors for N events, that is for $p \in \mathcal{P}^N$ we will have $p_i \geq 0$ for all $i \in [1, \ldots, N]$ and $\sum_{i=1}^{N} p_i = 1$. We define the probability projector: $\mathcal{P} : \mathcal{F}^N \to \mathcal{P}^N : f \to \mu = \mathcal{P}(f)$ with $\mu_i = \frac{f(i)}{\sum_{j=1}^{N} f(j)}$.

Learning Attacker Preference The single attacker learning problem will posit a single attacker with constant preference function $f_A \in \mathcal{F}^N$. The learning goal, starting from any hazard score $h_0 \in \mathcal{F}^N$, is to derive an update procedure for the hazard estimator: $U : h_k \to h_{k+1}$ that converges to f_A in the limit as i gets large and the defender has more interactions with an adversary. A notion of revelation is required to enable the learning procedure. Defenders often need to forensically investigate attacks when restoring control to a compromised node. A forensic investigation can reveal the initial vulnerability exploited [26].

As more experience is gained, the attacker's distribution can be estimated with increased precision by forming an empirical distribution of counts revealed up to time step k. In the appendix, we provide the standard boosting algorithm. Under the hypothesis that the attacker preference is distributed as a Multinomial distribution, the boosting algorithm forms a converging sequence of maximum likelihood estimation $\rho^{(k)}$ that will converge to $\mathcal{P}(f)$ when f is the attacker preference function. Further, $\rho^{(0)}$ can be initialized to any hazard score, such as defaulting to CVSS scores. Details of boosting are provided in appendix 3.

4 Results

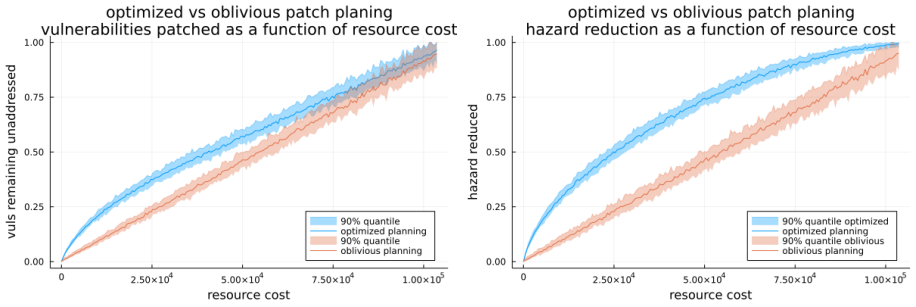
4.1 Problem I: Patch Planning for the Static Problem

We will refer to our method as *strategic or optimized patching*. To demonstrate the effect of strategic patching, we generate a testbed of virtual patch planning problems. The generation algorithm will have fixed parameters, including the number of nodes and vulnerabilities. A random uniform distribution will determine the number of vulnerabilities per node and select which vulnerabilities are assigned to each node. Then, using the generated problems, we employ our strategic patching method and contrast its outcomes with that of *oblivious patching* where the choice of which vulnerability to patch is arbitrary (and selected uniform randomly). The result is a statistical summary of outcomes measured as counts of vulnerabilities removed and associated hazard reduction.

By letting one network parameter take on a range of values while others stay fixed, we can illustrate a *response operator curves* (ROC) to contrast the two methods.

In Figure 2(a) and 2(b) we demonstrate the ROC curve for variable constraint budget. The fixed parameters of the generated networks are: $(M, V) = (400, 600)$; that is, four hundred nodes and six hundred distinct vulnerabilities. The fixed distributions for the networks are that: hazard scores for each vulnerability are distributed discrete uniform[5] randomly over $[6, 100]$, costs structure of each vulnerability will have $b_i = 0$, and m_i generated discrete uniform over $[10, 100]$, the number of vulnerabilities per node is distributed discrete uniform randomly over $[0, 10]$.

The overall constraint on resourcing to eliminate hazard is the independent variable (the *abscissa*), and will take on two hundred equal-distant points in the interval $[0, 100000]$, for each point we generate 40 sample networks and measure the outcomes as the number of vulnerabilities removed 2(a), and hazard removed 2(b). The outcome statistics (90% quantile and median) are plotted as the dependent variable (the *ordinate*) for both Strategic patching (blue) and Oblivious patching (orange). As is expected, the strategic patching is clearly more efficient at reducing the number of vulnerabilities. The efficiency generally depends on the distributions (vulnerability hazard, vulnerability cost, vulnerabilities per node); however, the strategic patching should be no worse than the oblivious method (matching performance only in trivial cases for the network distributions above). One feature of the strategic patching which is clearly apparent in 2(a) is the immediate boost in efficacy with a small budget; while this tappers off as the budget increases, its clear that a sizable area between the two graphs highlights the importance of a strategic approach, in particular when budgets are extremely constrained, this is when efficiencies are most pronounced when a strategic approach is used.



(a) vuls removal (b) hazard removal

Figure 2: Operational characteristics of strategic patching vs oblivious patching

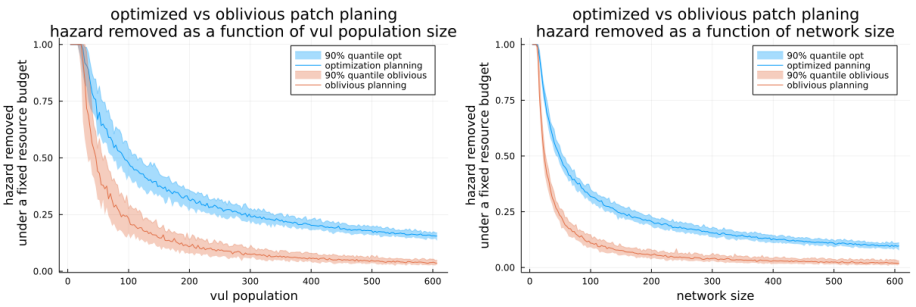
The resulting efficiencies are premised on the assumption that hazard scores

[5]The discrete uniform random variable over an interval $[a, b]$ will take on integer values within the range $[a, b]$.

are meaningful. Thus the result also underscores the importance of accurate measures for hazards in a scoring system such as CVSS. Further, one can view oblivious patching as the extreme of three different scenarios: 1) oblivious patching to extremely accurate hazard scores, 2) Oblivious patching to extremely noisy hazard scores, and finally, 3) Strategic patching to extremely noisy hazard scores.

Next, in figure 2(b) we show for the same study as in figure 2(a), the reduction of hazard score, for which we can see strategic patching is shown to be more efficient than before. The efficiency of hazard removal will depend on the relationship between hazard and cost, and the strategic patching method is particularly efficient in the presence of vulnerabilities with high hazards having low patching costs. While we are not familiar with studies on such a relation, we note that the goal of many automated patch procedures are aimed to decrease the cost to amplify the effect of strategic patch planning.

Asymptotic efficiences: Next, we fix the resource capacity and explore the efficacy of the method in the limit of larger problems: population size of vulnerabilities in 3(a), and network size (measured in the number of nodes) in figure 3(b). For each, the use of strategic patching outperforms oblivious patching by sizable fractions. The scenarios closely approximate that of figure 2(a) and 2(b), In both scenarios the resource budget is fixed to 50000 units, in figure 3(a) the number of vulnerabilities ranges from 4 to 600 while $M = 400$ remains fixed, and in figure 3(b) the number of nodes varies from 4 to 1200 while $V = 400$ remains fixed.



(a) Large number of Vuls

(b) Large number of nodes

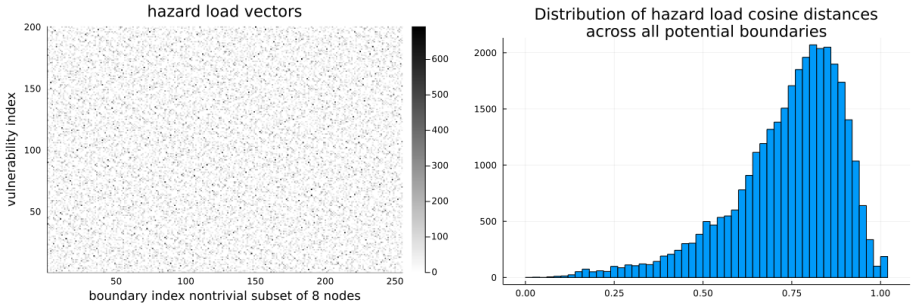
Figure 3: Asymptotic efficiency of strategic patch planning vs oblivious as the defense problem scales. (a) in number of vulnerabilities, and (b) number of nodes.

Folding boundary, and the Dynamic Problem: Below we computationally evaluate the discontinuity of the static problem over various boundaries of network control.

To illustrate this, we generate a problem with eight nodes and 200 vulnerabilities, each of which draws a hazard score from a fixed a discrete uniform distribution over $[6, 100]$; similarly, patch cost is discrete uniform over $[10, 100]$

and Vulnerabilities per node is Discrete uniform over $[1, 40]$. We consider all non-trivial boundaries as the 255 nonempty subsets of the eight nodes. For each, we calculate the hazard load vector and view these as a heat map in figure 4(a) below. Next, to consider the discontinuity inherent to protecting various subsets (e.g., boundaries of network), we consider all pairwise similarities between all pairs of hazard load vectors to produce a histogram showing bulk problem similarity in figure 1(a). Not surprisingly since all bounded (bounded by largest hazard) vectors are in the positive orathant of $\mathbb{R}^{200}$ a mode of the distribution arises and it can readily be seen that solving the static problem for several subsets can be rather dissimilar but not orthogonal either, thus suggesting heuristic possibilities for the dynamic problem.

In figure 4(a) we view



(a) hazard loads (b) orthogonality

Figure 4: Strategic patching of sub-problems will inherit some structural relation but also display wide variation. In (a) the hazard load vectors for all 255 subsets of 8 nodes are shown, the pairwise similarities are shown as a histogram in (b), on the abscissa zero represents orthogonality while one represents identical resulting patch plans.

This observation brings up a design problem with two notable extremes: 1) Clonal design where all nodes are the same delivers the benefit that a single patch plan may need little revision as an attacker flips the state of control in various network nodes. The drawback is that a single exploit may be efficiently used by an adversary to flip the entire network. Thus when an attacker finds a way in, they can quickly take control of the entire network. 2) Diversity where each node has a different set of vulnerabilities due to differing configured software systems. The benefit of such a design is that it increases the burden for the attacker, who must manage a larger set of exploits; the drawback is that the defense problem becomes sensitive to the network state of control.

Learning Adversarial Bag of Tricks: The learning algorithm outlined in appendix 3 is shown to be capable of rapid convergence to an attacker's exploit preference. In figure 5 we use a random set of scores for a set of $V = 200$ vulnerabilities, and we initialize a random adversary preference function by using a Dirichlet distribution (V, α), where hyperparameter $\alpha < 1.0$ places

the majority of the measure on a select few options. We also use a threshold $\tau = 0.002$ to trim the support of the Dirichlet distribution; this replaces any probability values lower than τ with zero and reflects the realistic possibility that attackers will not utilize a vulnerability they are not familiar with. The outcome of the threshold procedure is then normalized to obtain ϕ a probability vector p over the vulnerability options. The learning objective for defense is to estimate ϕ, and we measure the estimation accuracy by the cosign distance (or angle between the estimate and actual). Learning occurs from observations, and each window plotted on the abscissa will contain a number of observations; these observations were taken over each window and were used to update the hazard scores according to our outlined boosting method. As figure 5 indicates, an adversarial preference can be learned rather robustly. In this case, a distribution over 58 of 200 vulnerabilities can be learned rapidly having cosine similarity of .8 at 50 observations and .9 near 75 observations.

An important factor for the learning rate is the size of the repertoire (of vulnerabilities) that an adversary can attack. This is easiest to understand by considering limiting cases: An adversary that knows only one attack and an adversary that knows all attacks and can leverage a uniform random selection or Dirichlet$(200, 1.0)$. Realistically it's likely that most adversaries acting in cyberspace, who are themselves resource bound, will need to select a few vulnerabilities to learn and effectively exploit.

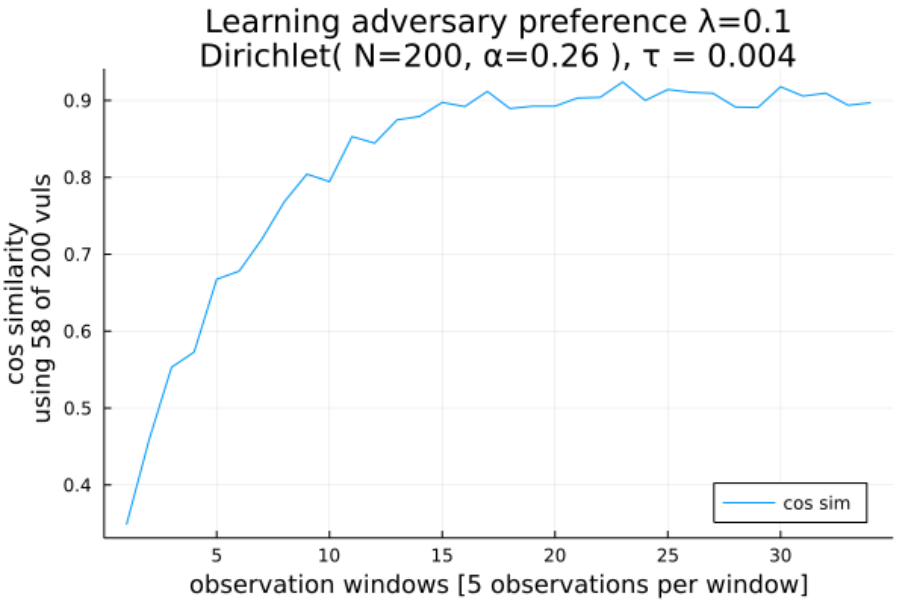


Figure 5: Learning an adversarial preference is assisted by sparsity

4.2 summary

When we compare two patching strategies (strategic and oblivious) for a realistic size data set, we observe that strategic patching (prescribed by our algorithm) removes vulnerabilities and hazard levels at least as efficiently as oblivious patching. Hazard scores accuracy is essential. Inaccurate hazard scores with strategic patch planning may as well be implemented with oblivious patch planning to obtain the same expected hazard reduction. This further underscores the importance of learning adversarial preferences used by attackers. In particular, hazard removal can be highly efficient for organizations with smaller resource budgets. Efficiencies can be amplified by vulnerabilities characterized by high hazard levels and low patch costs. Thus any engineering or management efforts that improve the efficacy of patching are helpful. Strategic patching outcomes express asymptotically efficiency well above that of oblivious patching as the problem scales in the number of vulnerabilities or the number of nodes. Additionally, in the dynamic problem, strategic patch planning can vary greatly as the network control state graph changes. We also note that design elements such as topology and diverse vulnerability configurations inherent to the network become important factors. Learning the preferences of a single adversary facilitates a dynamic approach where hazard scores can be initialized to the CVSS scoring system and refined to match an adversaries preference in a relatively short time.

5 Discussion and Conclusion

Strategic Patching: The benefits of strategic patching as a solution to the Static problem have several clear implications. First, the accuracy of hazard scores is important; when hazards are accurate strategic patching (while always better than or as good as an oblivious patching strategy) yields dramatic efficiencies for low resource organizations. The return on additional resourcing is marginally less for organizations that have high levels of resources. In our study, oblivious planning may also be seen as a proxy for poorly scored or poorly understood vulnerability hazards. Additionally, the discrepancy between oblivious and strategic patching helps to emphasize the importance of refining measures of hazard for particular attackers and highlights the value of having or gaining such information.

Dynamic Problem: The dynamic problem presents both challenges and opportunities, not only for heuristic solutions to our stated optimization problem but also for network design, where designers can potentially modulate or channel strategic defensive actions to greater efficiencies. We observe that system software has largely been designed and fielded as clonal copies, which helps reduce patch costs but also efficiencies for the attacker. Biological systems leverage diverse strategies as a natural defense; however, for software and patch management, such benefits have rarely been achieved due to the increased cost of patch maintenance required. Given the recent concerns placed on cyberattacks,

it may be worth reconsidering these network design factors as security requirements become more critical. Additionally, the role of deceptive technologies such as honeypots could play a role in assisting learning and defending deeper network components or critical assets. The search for optimization strategies for strategic patch planning in the dynamic problem is not yet complete. While our model helps to expose mathematical intuition, we plan to aim at the bigger problem in future work.

Learning an attacker's strategy: While we demonstrate the possibility of learning a single attacker strategy, the problem is broadened and more realistic by considering a normative decision theory for patch planning under the assumption of multiple attackers. Attackers aware of the defense algorithm would do better to form fake maneuvers which are aimed to conceal their true preferences, thus calling on information asymmetrical games [19] to better model dynamic attackers. But also in defense/evasion games, strategies can enter into an evolutionary dynamic known as antagonistic chase [18], which mirrors behaviors identified in adversarial machine learning [24]. Notwithstanding the usual difficulty of recovering attacker strategies, attack graph recovery techniques [26], deceptive honeypots, as well as advanced behavioral monitors may help to improve learning rates for the defense. Still, antagonistic chases imply a limited shelf life for both defense and offensive strategy as an organizing principle.

6 Conclusion

A model for normative decision-making for patch planning is presented. The static problem has a solution space over patch plans, and we show the optimal solution is the plan that removes the largest hazard, thereby denying a maximal expected utility for the attacker. We provide a reduction to Knapsack problem and constrained optimization algorithm to solve patch planning instances. We further define the dynamic patch-plan optimization problem and how it can be synthesized from a set of static problems stitched together by the reachability of an underlying network control graph. The dynamic graph reveals interesting challenges, including discontinuity of solution when the network boundary changes slightly; additionally, our results indicate the possibility of heuristics to be considered in future work. Noting that attacker preferences may differ from score systems (such as CVSS), we provide an iterative hazard score refinement that can start from CVSS and converge to that of an attacker. The model we present adds mathematical concreteness to defense management and its dual goals to both learn what to defend while resourcing defense patch efforts to prevent attacker exploitation.

Appendix 1: Solution for Static Problem

We provide the code (written in the Julia language) which formalizing the problem of generating patch planning problem instances. All parameters are described in code comments in Figure 6. The code can be used to repetitively generate a large number of problem instances so *strategic patching* can be compared to *oblivious patching*.

Next in figure 7 we show how patch planning problem instances (generated from code in 6) can be re-coded or reduced to instances of a weighted knapsack constrained optimization problem.

Finally in figure 8 as we provide the high level binary programming routine which takes as input weighted knapsack instances and calculates a set of binary values that optimize the weighted knapsack.

Our patch planning algorithm first reduces the network problem instance to weighted knapsack and then applies the weighted knapsack solver.

Problem Instance Generation

```
function problem_instance(M, V, V_score_dist, V_patch_cost_dist, V_per_M_dist ;
V_freq_hyper_parameters=1, verbose = true )
    # given
    #   1) a number of machines M,
    #   2) a number of vulnerabilities V,
    #   3) V_score_dist (a discrete distribution)
    #   4) V_patch_cost_dist (a discrete distribution)
    #   5) V_per_M_dist (a discrete distribution)
    # This function will return a concrete problem instance (set of machines with
vulnerabilities.  )  The problem instances can be solved in terms of hazard reduction.
    # Output:
    #   1) Machines: UnitRange{Int64}  -- lists the machines
    #   2) Vuls:  UnitRange{Int64}  -- lists the vuls
    #   3) VScore:  Vector{Int64} (alias for Array{Int64, 1})  -- scores the vuls
    #   4) Vpatch_cost: Vector{Int64} (alias for Array{Int64, 1})  -- cost per vul: m
    #   5) Map:  Vector{Vector{Int64}} (alias for Array{Array{Int64, 1}, 1})
    #            -- is a vector of vector's one per machine, each is lists act vuls.
    Machines = 1:M
    Vuls = 1:V
    Vscore = rand( V_score_dist, V );
    Vpatch_cost = rand( V_patch_cost_dist, V );
    V_freq_dist_dist = Dirichlet( V, V_freq_hyper_parameters )
    V_freq_dist = Categorical( rand( V_freq_dist_dist ));
    Map = [ rand( V_freq_dist, rand( V_per_M_dist)) for k in Machines ]
    return[ Machines, Vuls, Vscore, Vpatch_cost, Map ]
end
```

Figure 6: Problem Instance Generation

```
function translate_to_weighted_knapsack(M,V,VS,VPC,Map;VPS=missing,netBDY =missing)
    # Given
    #   1) M (Vector of integer ids for nodes/machines)
    #   2) V (Vector of integer ids for vulnerabilities)
    #   3) VS (Vector of floating values scoring the hazard for each vulnerability)
    #   4) VPC (Vector of floating values which are costs for fixing each
vulnerability)
    #   4) Map (Vector of vectors that map the vulnerabilities (by id) to machines)
    # This function will calculate:
    #   1) the hazard load vector whose inner-product with decision vars is
    #          will comprise the objective function for the static problem.
    #   2) the cost loads vector whose inner-product with the decision vars must
    #          not exceed the resource constraint.
    #   other values returned are affected by the optional inputs but provide a means
    #          to observe the outcome of an applied patch plan.
    #
    if ismissing( VPS )
        VPS = [ 0 for v in V]
    end
    if ismissing( netBDY )
        netBDY = [ 1 for m in M]
    end
    counts =  [ ((VPS[v] == 0 ) ? [ sum(Map[m] .== v ) for m in M ]'*netBDY : 0 ) for
v in V ]
    hazards = VS .* counts; # the hazard load vector
    costs   = VPC .* counts;
    return [[ hazards, costs ], [sum(counts), sum(hazards), sum(costs) ]]
end
```

Figure 7: Reduction to weighted knapsack is little more than calculating hazard and cost load for decision variables.

```
function weighted_knapsack(benefit, weight, capacity; verbose = false)
    @assert( length( benefit ) == length( weight ) )
    model = Model(GLPK.Optimizer)
    @variable(model, x[1:length( benefit )], Bin)
    # Objective: maximize benefits
    @objective(model, Max, benefit' * x)
    # Constraint: can carry all
    @constraint(model, weight' * x <= capacity)
    # Solve problem using MIP solver
    optimize!(model)
    reduction = objective_value(model)
    soln = [value( x[i] ) for i in 1:length(benefit)]
    utilization = weight' * soln
    if verbose
        println("Objective is: ", reduction)
        println("capacity used: ", utilization)
        println("Solution is:")
        for i in 1:length(benefit )
            print("x[$i] = ", value(x[i]))
            println(", p[$i]/w[$i] = ", benefit[i] / weight[i])
        end
    end
    return [reduction, utilization, soln ]
end
```

Figure 8: Constrained optimization code.

Appendix 3: Learning by boosting

We presume a count of vulnerabilities attacked at time k as E_k so that $E_k(i)$ will be the number of times v_i is attacked in the interval $[0, k]$. Boosting algorithm such as those used to estimate the multi-armed bandit problem, can be utilized to efficiently estimate f_A with desirable properties. Let h_0 be an initial hazard scoring, and $p_0 = \mathcal{P}(h_0)$. Also let $q_k = \mathcal{P}(E_k)$. letting $p_{k+1} = (1-\lambda)p_k + \lambda q_k$ and telescoping the update function provides:

$$\begin{aligned} p_{k+1} &= (1-\lambda)p_k + \lambda q_k \\ &= (1-\lambda)\left((1-\lambda)p_{k-1} + \lambda q_{k-1}\right) + \lambda q_k \\ &= (1-\lambda)\left((1-\lambda)\left((1-\lambda)p_{k-2} + \lambda q_{k-2}\right) + \lambda q_{k-1}\right) + \lambda q_k \\ &= \dots \\ &= (1-\lambda)^{j+1}(p_{k-j}) + \sum_{e=1}^{j}(1-\lambda)^e \lambda q_{k-e} + \lambda q_k \end{aligned}$$

Letting $j = k$:

$$\begin{aligned} p_{k+1} &= (1-\lambda)^{k+1}p_0 + \lambda(\sum_{e=1}^{k}(1-\lambda)^e q_{k-e} + q_k) \\ &= (1-\lambda)^k p_0 + \lambda \sum_{e=0}^{k-1}(1-\lambda)^e q_{k-e} \qquad (*) \end{aligned}$$

If the attacker randomly selects target vulnerabilities in proportion to their preference function f_A, then E_k is distributed $\textsc{Multinomial}(\mathcal{P}(f_A), k)$, therefore $\mathbb{E}[E_k(i)] = k * \frac{f_A(i)}{\sum_\nu f_A(\nu)}$, and $\mathbb{V}[E_k(i)] = k\frac{f_A(i)}{\sum_\nu f_A(\nu)}\left(1 - \frac{f_A(i)}{\sum_\nu f_A(\nu)}\right)$.

To show the boosting technique can achieve the learning goal we need to show $p_{k+1} \to \mathcal{P}(f_A)$ in expectation, while the rate of convergence is determined by a central limit theorem for Multinationals [29].

Noting that $\mathbb{E}[q_k(i)] = \frac{1}{k}\mathbb{E}[E_k(i)] = \frac{f_A(i)}{\sum_\nu f_A(\nu)}$, we can calculate the expecta-

tion for $p_{k+1}(i)$ as:

$$\begin{aligned}
\mathbb{E}[p_{k+1}(i)] &= \mathbb{E}[(1-\lambda)^k p_0(i) + \lambda \sum_{e=0}^{k-1} (1-\lambda)^e q_{k-e}(i)] \\
&= (1-\lambda)^k p_0(i) + \lambda \sum_{e=0}^{k-1} (1-\lambda)^e \mathbb{E}[q_{k-e}(i)] \\
&= (1-\lambda)^k p_0(i) + \frac{f_A(i)}{\sum_\nu f_A(\nu)} \lambda \sum_{e=0}^{k-1} (1-\lambda)^e \\
&= (1-\lambda)^k p_0(i) + \frac{f_A(i)}{\sum_\nu f_A(\nu)} (1-(1-\lambda)^k)
\end{aligned}$$

For any $\lambda \in (0,1)$ convergence in expectation with exponential rate for any initial hazard scores (p_0).

The central limit theorem for multinomials can provide tighter bounds on rate of convergence in probability for our estimate, however we indicate here the interesting possibility that the rate of convergence for the distribution could also be modulated by the adversarial selection of distribution. The antagonistic chase (such as detailed in [18]) that could result would play out between attacker and defender within a space of Dirichlet distributions over the vulnerability indices. There are many interesting open questions related to evolutionary strategies.

References

[1] Acunetix Web Application Security Testing invicti. https://www.acunetix.com. Accessed: 2022-02-28.

[2] CVSS first. https://www.first.org/cvss/. Accessed: 2022-02-28.

[3] Immuniweb AI Vulnerability Scanner immuniweb. https://www.immuniweb.com. Accessed: 2022-02-28.

[4] Intruder intruder. https://www.intruder.io. Accessed: 2022-02-28.

[5] ManageEngine Vulnerability Manager Plus manageengine. www.manageengine.com/vulnerability-management. Accessed: 2022-02-28.

[6] Nessus tenable. https://www.tenable.com/products/nessus/nessus-professional. Accessed: 2022-02-28.

[7] Netsparker netsparker. https://www.netsparker.com/. Accessed: 2022-02-28.

[8] Nexpose Vulnerability Scanner rapid7. https://www.rapid7.com/products/nexpose/. Accessed: 2022-02-28.

[9] Open Vulnerability Assessment Scanner solarwinds. https://www.openvas.org/. Accessed: 2022-02-28.

[10] Paessler PRTG Network Monitor paessler - the monitoring experts. https://www.paessler.com/network_monitoring_tool. Accessed: 2022-02-28.

[11] PortSwigger portswigger. https://portswigger.net. Accessed: 2022-02-28.

[12] Solarwinds Security Event Manager solarwinds. www.solarwinds.com/security-event-manager. Accessed: 2022-02-28.

[13] Tripwire IP360 tripwire. https://www.tripwire.com/products/tripwire-ip360/. Accessed: 2022-02-28.

[14] Ahmed H Anwar and Charles Kamhoua. Game theory on attack graph for cyber deception. In *International Conference on Decision and Game Theory for Security*, pages 445–456. Springer, 2020.

[15] Ahmed H Anwar, Charles Kamhoua, and Nandi Leslie. Honeypot allocation over attack graphs in cyber deception games. In *2020 International Conference on Computing, Networking and Communications (ICNC)*, pages 502–506. IEEE, 2020.

[16] Martín Barrère, Rodrigo Vieira Steiner, Rabih Mohsen, and Emil C Lupu. Tracking the bad guys: An efficient forensic methodology to trace multi-step attacks using core attack graphs. In *2017 13th International Conference on Network and Service Management (CNSM)*, pages 1–7. IEEE, 2017.

[17] L Caccetta and Araya Kulanoot. Computational aspects of hard knapsack problems. *Nonlinear Analysis*, 47(8):5547–5558, 2001.

[18] William Casey, Steven E Massey, and Bud Mishra. How signalling games explain mimicry at many levels: from viral epidemiology to human sociology. *Journal of the Royal Society Interface*, 18(175):20200689, 2021.

[19] William Casey, Jose A Morales, Thomson Nguyen, Jonathan Spring, Rhiannon Weaver, Evan Wright, Leigh Metcalf, and Bud Mishra. Cyber security via signaling games: Toward a science of cyber security. In *International Conference on Distributed Computing and Internet Technology*, pages 34–42. Springer, 2014.

[20] Iain Dunning, Joey Huchette, and Miles Lubin. Jump: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

[21] Karel Durkota, Viliam Lisỳ, Branislav Bošanskỳ, and Christopher Kiekintveld. Optimal network security hardening using attack graph games. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[22] Bascou Jean-Jacques Gallon, Laurent. Using cvss in attack graphs. pages 59–66, 2011.

[23] Laurent Gallon and Jean Jacques Bascou. Using cvss in attack graphs. In *2011 Sixth International Conference on Availability, Reliability and Security*, pages 59–66. IEEE, 2011.

[24] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and J Doug Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pages 43–58, 2011.

[25] Aron Laszka, Gabor Horvath, Mark Felegyhazi, and Levente Buttyán. Flipthem: Modeling targeted attacks with flipit for multiple resources. In *International Conference on Decision and Game Theory for Security*, pages 175–194. Springer, 2014.

[26] Changwei Liu, Anoop Singhal, and Duminda Wijesekera. Mapping evidence graphs to attack graphs. In *2012 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 121–126. IEEE, 2012.

[27] Silvano Martello, David Pisinger, and Paolo Toth. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management science*, 45(3):414–424, 1999.

[28] Dale McMorrow. Science of cyber-security. Technical report, MITRE CORP MCLEAN VA JASON PROGRAM OFFICE, 2010.

[29] Carl Morris. Central limit theorems for multinomial sums. *The Annals of Statistics*, pages 165–188, 1975.

[30] Nuthan Munaiah and Andrew Meneely. Vulnerability severity scoring and bounties: Why the disconnect? In *Proceedings of the 2nd International Workshop on Software Analytics*, pages 8–14, 2016.

[31] David Pisinger. Where are the hard knapsack problems? *Computers & Operations Research*, 32(9):2271–2284, 2005.

[32] Jonathan Spring, Eric Hatleback, Allen Householder, Art Manion, and Deana Shick. Time to change the cvss? *IEEE Security & Privacy*, 19(2):74–78, 2021.

[33] Marten Van Dijk, Ari Juels, Alina Oprea, and Ronald L Rivest. Flipit: The game of "stealthy takeover". *Journal of Cryptology*, 26(4):655–713, 2013.