Incentivizing Collaborative Breach Detection

Mridu Nanda
Duke University

Michael K. Reiter
Duke University

***Abstract*—Decoy passwords, or "honeywords," alert a site to its breach if they are ever entered in a login attempt on that site. However, an attacker can identify a user-chosen password from among the decoys, without risk of alerting the site to its breach, by performing credential stuffing, i.e., entering the stolen passwords at another site where the same user reused her password. Prior work has thus proposed that sites monitor for the entry of their honeywords at other sites. Unfortunately, it is not clear what incentives sites have to participate in this monitoring. In this paper we propose and evaluate an algorithm by which sites can exchange monitoring favors. Through a model-checking analysis, we show that using our algorithm, a site improves its ability to detect its own breach when it increases the monitoring effort it expends for other sites. We additionally quantify the impacts of various parameters on detection effectiveness and their implications for the deployment of a system to support a monitoring ecosystem. Finally, we evaluate our algorithm on a real dataset of breached credentials and provide a performance analysis that confirms its scalability and practical viability.**

1. Introduction

According to Verizon [81, Fig. 15] and IBM [41, Fig. 7], the most prevalent initial attack vector causing data breaches continues to be stolen credentials. Moreover, the sources of these stolen credentials are often themselves data breaches—over 20% of all data breaches compromise credentials [81, Fig. 24]. Data breaches are notoriously slow to be identified (207 days by one recent estimate [66]) and those stemming from stolen credentials are even harder to discover, requiring an average of 229 days after initial compromise [41, Fig. 8]. These trends will likely persist, given the ubiquity of passwords as a primary authentication mechanism [39].

Honeywords [44] seek to shrink the window between the attacker's use of breached passwords and the defender's realization that it has been breached. Honeywords are decoy passwords created by the defender and stored alongside user-chosen passwords in its credential database; login attempts using honeywords alert the site to its breach, since legitimate users do not know them. An attacker who breaches the password database and attempts to harvest its accounts can sidestep detection only if it can determine which of the passwords associated with each account is the user-chosen one. Methods to select honeywords to make this difficult for the attacker have been the subject of much research (e.g., [44], [28], [5], [24], [83], [11], [37]).

Regardless of the honeyword-generation method, however, a reliable way for an attacker to separate the user-chosen password from the honeywords is to attempt to use these passwords at another site where the same user has an account. If the user reused her password (or a similar one) there—as users often do [47], even despite password-manager support [62]—then the password that works at the remote site will almost certainly be the user-chosen password at the breached site. Subsequent honeyword-system designs (e.g., [84], [85]) have thus developed methods by which a site can remotely monitor the login attempts at other sites for entry of its honeywords. This monitoring, however, consumes nontrivial resources at sites where monitoring occurs and can exacerbate the load induced on those sites by credential-stuffing campaigns, which can reach denial-of-service volumes in some cases (e.g., [56]).

Our credential ecosystem is therefore held captive by misaligned incentives: remote monitoring of login attempts for honeyword use is *necessary* to overcome a key vulnerability of honeywords—and thus to unlock their adoption—and yet that remote monitoring requires that each site invest potentially significant resources *to protect others*. In this paper, we offer a way out. The key insight of this work is that the dependence between sites (reused passwords) that poses a risk to one site's breach-detection capability is symmetric: common users at the two sites provide accounts at each site whose honeywords can be sidestepped by stuffing credentials at the other, without the risk of alerting either site to its breach. In this paper we leverage this insight to develop a simple and scalable algorithm to support an ecosystem of sites, each self-interested, to barter monitoring favors so as to enable each site to improve its *own* breach-detection capability by increasing the amount of monitoring it performs *for others*. In doing so, we argue that this algorithm could serve as the basis for a self-sustaining breach-detection ecosystem.

The technical challenges to developing such an algorithm are several. For example, the algorithm must accommodate sites with varying resource constraints so as to not dissuade smaller sites from participating in the ecosystem. It must also support decentralized participation based solely on local information, since each site must retain control over its resource-allocation decisions and protect sensitive internal information. And most importantly, the algorithm should effectively elevate the ability of a site to detect its own breach when its credentials are stuffed elsewhere—even though sites cannot know when attacks will occur, where the attacker will stuff, or how aggressive an attacker

may be. We construct a bidding algorithm that, we show, achieves these goals and, moreover, yields risk for each site comparable to the best (myopic) bidding strategy it could adopt. We further show that decreasing risk according to our metric translates to improved security for sites, against an attacker who attempts to harvest the accounts of a previously breached site through strategic stuffing attempts at peers.

Our strategy to evaluate the efficacy of our approach begins with model-checking across a wide array of parameters controlling: user account placement at sites, site strategies, attacker aggressiveness, bidding frequency and predictability etc. Owing to the limitations of existing model-checking tools, we implemented a custom model-checker using application-specific optimizations to better scale our analysis. This revealed the key factors that drive our algorithm's near-optimal performance and resilience to strategic attackers, offering guidance for real-world deployment (e.g., preventing sites from predicting future bidders). To complement our model-checking analysis, we ground our simulation in real-world conditions using user account distributions and password reuse rates from a previously breached dataset covering $\approx$ 8000 sites and $\approx$74 million accounts. In summary the contributions of this work are:

- We design the first algorithm for collaborative breach detection by bartering monitoring resources across sites, thereby making practical honeyword adoption viable. Via model checking, we show that our algorithm enables a site to reduce its risk as effectively as any alternative strategy.
- Through a second model-checking analysis, we show that a site's risk reduction via our algorithm directly translates to stronger protection against an attacker who tries to evade detection by stuffing breached credentials at peer sites. We show the ecosystem is self-sustaining: sites are incentivized to invest in protecting *others* to boost their *own* security, creating cyclic improvements. While popular sites drive early gains, our algorithm also incentivizes unpopular sites' participation, both groups achieving comparable protection despite differing resource constraints.
- We quantify our algorithm's effectiveness using a publicly leaked breach dataset. Because model checking is computationally prohibitive at this scale, we evaluate security through simulations that incorporate user placement and password reuse rates from the dataset, grounding our analysis in real-world conditions.
- We demonstrate our algorithm's scalability to millions of collaborating sites.
- We use our model-checking analyses to inform a system design that supports a breach-detection ecosystem.

2. Related Work

2.1. Breach Detection

Breach discovery today works primarily by scanning for breached datasets across various sources, such as the dark web, where the data might be advertised for sale or simply exposed. Once a breach is discovered, breach alerting services help affected users and organizations become aware of their exposure. Specifically, compromised credential checking services (e.g., [38], [73], [54], [69]) have seen wide deployment, but do not themselves *discover* breaches. Dataset breaches that are not advertised or exposed, but instead are used directly by the attackers who breach them to harvest accounts, remain invisible to these defenses.

Honey accounts [22], [79] and honeywords are deceptive techniques to leverage an attacker's account-harvesting efforts to discover credential database breaches (indicated by honey-account or honeyword use). A honey account must be difficult to distinguish from real accounts to yield a high true detection rate, and ensuring a quantifiable rate of false detections *for database breaches* (versus another form attack on its password, e.g., online guessing) remains a challenge, especially when the password for the account is shared with another site (as in the Tripwire study [22]). True- and false-detection rates for honeywords have received considerably more study (e.g., [44], [28], [5], [24], [83], [11], [85], [37]). Some works have offered proposals to monitor for attacker efforts to reduce the true-detection rate of honeywords by first trying them in login attempts at other sites [84], [85], though none have tackled how to motivate sites to support this monitoring, which is our focus here.

2.2. Resource Allocation

Our goal is to devise a mechanism for allocating computational resources so that when a site invests resources to secure its peers (i.e., by enabling remote monitoring) it gains protection in return. Such allocations cannot be dictated by a centralized "social planner," as it would require access to private, changing preferences and constraints, and it would impose decisions about how each site should expend its own computational resources when enabling remote monitoring for peers—decisions that sites are neither willing nor able to delegate. Instead, we aim to devise a peer-to-peer mechanism that allows sites to trade computational resources.

This goal echoes prior work on P2P content distribution, notably the BitTorrent protocol [14], where agents strategically upload bandwidth to maximize their own download speed, creating an incentive-driven exchange. Most relevant to our work is Levin et al. [57], which models BitTorrent as an auction and presents *proportional response* as an alternative auction-clearing mechanism. A client using the proportional response strategy uploads bandwidth to a peer in proportion to the bandwidth previously received from that peer. Through simulations, they show that this strategy yields fairness, robustness (to Sybils and some collusion), and competitive performance.

The proportional response algorithm also underpins our P2P design. Instead of exchanging bandwidth, a site barters monitoring favors, providing proportionally more favors (within its resource constraints) to a peer from whom it receives more. While our approach builds on Levin et al., key assumptions in their setting do not hold here. They aim to optimize performance, where small slowdowns are acceptable, and treat all bandwidth contributions as interchangeable. In contrast, our focus is on security, where even

brief lapses can expose vulnerabilities, and not all favors are equally valuable—some peers are better suited to monitor others due to higher user overlap. The Tycoon project [53] also uses proportional response for resource allocation, but in a centralized setting where the auctioneer controls the resources—an assumption incompatible with our setting where sites retain control over their computational resources.

Many convergence results for proportional response dynamics exist in Fisher markets [92], [8], [12], [50], [90], where agents with fixed budgets buy fixed supplies. These models do not apply here, as we explore exchanging monitoring favors without a substitute currency or supporting infrastructure. Our setting aligns better with exchange economies, where agents trade initial resource endowments to maximize utility. Wu and Zhang [88] proved convergence when all agents value goods equally; Branzei [9] extended this to heterogeneous valuations. However, Branzei assumes simultaneous updates by all agents—a "fair exchange" of bids—which is impossible without a trusted third party [68], [77], [32], not required in our setting.

2.3. Cooperative Security

Interdependent security models, introduced by Kunreuther and Heal [51], examine how a self-interested agent's security decisions directly impact peers. These models often use graph structures, where nodes represent agents making security investments and edges represent dependencies influencing risk exposure. One class focuses on defender-defender interactions, where agents balance reducing their own risk with minimizing security investment costs through strategic interaction with peers [86], [55], [63], [43], [2]. Another class includes attacker-defender dynamics, analyzing how adversarial behavior shapes security decisions [64], [59], [36], [1], [67]. In both, the assumption is that agents can achieve their security goals independently, albeit possibly requiring increased investment. However, in our setting, cooperation is not merely an optional enhancement but a fundamental requirement for detecting breaches.

A related but distinct area is collaborative intrusion detection, where agents improve the accuracy of their own intrusion detection assessment by sharing alerts and information with peers. Prior work has explored system-level mechanisms for exchanging alerts [89], [91], [42], [30], [25]. In contrast, our work builds on existing remote monitoring mechanisms [84], [85] and focuses on the incentives of exchanging monitoring favors. Notably, GUIDEX [93] and Fung's [29] also consider incentive-driven resource sharing, but do not quantify the resulting security improvements, which is central to our approach.

3. Inter-Organization Model

We consider a collection S of $|S| = n$ *sites*, and a collection U of $|U| = \ell$ *users*. Each user has an *account* at one or more of the sites. We use $s.\mathsf{users} \subseteq U$ to denote the set of users with accounts at site $s \in S$, and $u.\mathsf{sites} \subseteq S$ to denote the set of sites at which user $u \in U$ has accounts. Naturally, $u \in s.\mathsf{users}$ if and only if $s \in u.\mathsf{sites}$. Each account is protected using a password selected by its user.

The password database at site s includes, for each account, a user-chosen password as well as a number of *honeywords*, of which the user is unaware. The honeywords for an account together with the user-chosen password are called the *sweetwords* for the account. Honeywords exist to alert s to the breach of its password database by an attacker, in that a login attempt at s using a honeyword for the attempted account is evidence of the breach of s's database. For this reason, an attacker who breaches s's password database instead stuffs the sweetwords for an account at the same user's accounts at *other* sites, in the hopes of determining the user-chosen password due to its reuse at those other sites [15], [70], [82].

To counter this threat, a site s can ask another site s' to monitor for the entry of s's honeywords in login attempts at s'. The mechanics of how this monitoring is done without exposing s's sweetwords to s', and without exposing to s any other passwords used in login attempts at s', are described in prior works [84], [85] and are not our concern here. Rather, we abstract this process as follows: s poses a *monitoring request* to s', which names an account for which logins should be monitored. If s' accepts this monitoring request, then any incorrect login attempt to the account named in the request will generate a *monitoring response* to the site s that created it. If the password used in that login attempt at s' is a honeyword for the same user's account at s, then s learns the honeyword used and can treat it as if it were attempted locally, for the sake of breach detection.

Each site s has a *monitoring capacity* $s.\mathsf{cap} \in \mathbb{N}$, which is the number of *monitoring slots* in which it can host monitoring requests. Each site is rational in the sense that desires to trade its slots for those at other sites that most effectively help detect its own database breach. To do so, each site can occasionally reallocate its slots to other sites; we refer to such a reallocation as a *bid*. That is, to make a bid, site s assigns to each other site $s' \in S \setminus \{s\}$ a number of slots $s.\mathsf{allocTo}(s') \in \mathbb{N}$ such that $\sum_{s' \in S \setminus \{s\}} s.\mathsf{allocTo}(s') \leq s.\mathsf{cap}$, and solicits $s.\mathsf{allocTo}(s')$ monitoring requests from each s'.

The focus of this paper is to develop a *bidding strategy* that rational sites can use to trade monitoring slots with peers. The proposed bidding strategy should have several desirable properties to incentivize adoption: it should reward reciprocity, so sites that contribute more monitoring capacity receive more slots in return. It should ensure fairness by allocating similar numbers of slots to similarly at-risk peers, and by including smaller or less popular sites. Crucially, the strategy must be locally computable, allowing each site to operate independently based on its own information, and scalable to enable deployment across large networks of sites.

3.1. Site Threat Model

We assume that each site is rational in wishing to use its bidding power to provide itself the best chance of detecting a breach of its own password database. Aside from manipulating its bids to accomplish this, however, we

assume sites behave correctly. For example, a site s accepts monitoring requests from s' in accordance with the allocation $s.\mathsf{allocTo}(s')$ in its bid. (Prior work [84] also discussed how s' could audit s to ensure it does so.) We believe this assumption is consistent with sites that conscientiously collaborate for their collective defense. We also assume that the exchange of a site's bid to its peer and its peer's deployment of the monitor request occur instantaneously.

Each site s knows the set of sites S with which it is collaborating, as well as its own users $s.\mathsf{users}$. However, s is not privy to the any of the bids a peer receives, nor does it know any peer's capacity. We consider two possibilities regarding how much information s has about the accounts at a peer s'. Either s knows the membership of $s.\mathsf{users} \cap s'.\mathsf{users}$, which we presume it would learn by running a private set intersection (psi) protocol [72] with s', or s knows only $|s.\mathsf{users} \cap s'.\mathsf{users}|$, which it could learn by running a PSI cardinality (psica) protocol (e.g., [48], [20], [23], [26], [19]) with s'. For simplicity, we assume that all sites have the same privacy level, $\mathsf{privLvl} \in \{\mathsf{psi}, \mathsf{psi}\}$, and uniformly use either protocol with their peers.

3.2. Risk

The allocations received from another site s' (in its bids) are valuable to site s, since they provide an opportunity to deploy that many monitoring requests to s'. But not all allocations from different peers equally enhance s's ability to detect its own breach. For example, if s and s' share no users and the attacker knows this—and we will presume it does (see §4.1)—then allocations from s' are not useful to s; in this case the attacker will never stuff sweetwords stolen from s at s', as it provides no help in harvesting accounts at s. Thus, the risk a site incurs from its peers is intimately tied to the specific mechanics of credential stuffing.

Generally, site s measures the usefulness of an allocation $s'.\mathsf{allocTo}(s)$ by the amount of defense it provides for the users that it shares with s', since those are the only users whose accounts will be stuffed at s' to harvest accounts at s. More precisely, consider an attacker stuffing f accounts at s' for users in $s.\mathsf{users} \cap s'.\mathsf{users}$ after s has deployed $k = s'.\mathsf{allocTo}(s)$ monitor requests to s'. Let $\mathsf{dodge}(s, s', k, f)$ be the event that none of the monitor requests deployed by s to s' was for one of the f users whose accounts the attacker stuffs at s'. We define a random variable $\mathbb{G}_f$, termed the *attacker gain* when stuffing these f accounts, by

$$\mathbb{G}_f = \begin{cases} f & \text{if } \mathsf{dodge}(s, s', k, f) \\ 0 & \text{otherwise} \end{cases}$$

That is, if the f stuffing attempts at s' do not overlap with the (up to) k accounts monitored by s, then the attacker gains the f accounts it stuffed. Otherwise, the attacker's stuffing attempts risked alerting s to its breach, and so we estimate the attacker's gain as nothing. We define the *risk* that s incurs from an allocation $k = s'.\mathsf{allocTo}(s)$ as

$$s.\mathsf{risk}(s', k) = \max_{0 \le f \le n'} \mathbf{E}\left(\mathbb{G}_f\right) \tag{1}$$

where $n' = |s.\mathsf{users} \cap s'.\mathsf{users}|$. Since n' of s's accounts are vulnerable to stuffing attempts at s', it will be useful to also define risk as a fraction of n', i.e.,

$$s.\mathsf{normRisk}(s', k) = \frac{s.\mathsf{risk}(s', k)}{|s'.\mathsf{users} \cap s.\mathsf{users}|} \tag{2}$$

Since $\mathbf{E}\left(\mathbb{G}_f\right) = f \times \mathbf{Pr}(\mathsf{dodge}(s, s', k, f))$, to compute Eqn. (1), we need to quantify $\mathbf{Pr}(\mathsf{dodge}(s, s', k, f))$. Doing so depends on what s knows about $s.\mathsf{users} \cap s'.\mathsf{users}$. In the psi case, i.e., where s knows the membership of $s.\mathsf{users} \cap s'.\mathsf{users}$, this probability is

$$\mathbf{Pr}(\mathsf{dodge}(s, s', k, f)) = \frac{\binom{n'-f}{\min\{n', k\}}}{\binom{n'}{\min\{n', k\}}} \tag{3}$$

assuming s deploys monitor requests to s' for $\min\{n', k\}$ accounts in $s.\mathsf{users} \cap s'.\mathsf{users}$ chosen uniformly at random. The numerator is the number of ways that s could choose to deploy requests for $\min\{n', k\}$ accounts from the $n' - f$ that the attacker does not stuff, whereas the denominator is the number of ways it could choose to deploy requests for $\min\{n', k\}$ accounts from all n'. In contrast, in the psica case where s knows only $|s.\mathsf{users} \cap s'.\mathsf{users}|$, this probability is

$$\mathbf{Pr}(\mathsf{dodge}(s, s', k, f)) = \frac{\binom{|s.\mathsf{users}|-f}{\min\{|s.\mathsf{users}|, k\}}}{\binom{|s.\mathsf{users}|}{\min\{|s.\mathsf{users}|, k\}}} \tag{4}$$

Since s does not know $s.\mathsf{users} \cap s'.\mathsf{users}$, it must deploy monitors requests for accounts chosen from $s.\mathsf{users}$. The probability is given by the number of ways s can choose accounts that the attacker does not stuff divided by the total number of ways it can choose accounts.

Whenever s receives a new bid $k = s'.\mathsf{allocTo}(s)$ from s', we assume that s deploys not only k monitors to s', but also redeploys monitors to each other site, in accordance with the bid last received from each. We quantify the risk that s incurs *per bid* r' in an auction of r bids (i.e., $1 \le r' \le r$) as

$$s.\mathsf{risk}(r') = \sum_{s' \in S \setminus \{s\}} s.\mathsf{risk}(s', k) \tag{5}$$

$$s.\mathsf{normRisk}(r') = \sum_{s' \in S \setminus \{s\}} s.\mathsf{normRisk}(s', k) \tag{6}$$

where $k = s'.\mathsf{allocTo}(s)$ is the allocation to s in bid $r'' \le r'$, where s' issued bid r'' but no subsequent bids prior to (or including) r'.

3.3. Bidding sequence

We assume that sites bid sequentially, with each bid received instantaneously by all others.We are agnostic to the mechanism that dictates the order in which sites bid. For example, a site's bid might be accepted only once it is authorized to bid by its selection through a randomized outcome on which all sites can agree (see §7).

We do not assume a specific method for determining the bidding order, but we characterize sequences based on a parameter called slack, which defines an upper bound on

difference between the most and fewest bids made by any site. When slack = ∞, there is no constraint; when slack = 1, a site may only bid again after all others have bid at least as many times.

3.4. Exhaustive bidding strategy

Our goal is to develop a *bidding strategy* for each site s, which will be our focus of §3.5. We focus on "practical" bidding strategies that are computationally efficient and leverage only the locally visible history at each site for that site to make a bid. Our goal is for the practical bidding strategy to remain competitive with an exhaustive, though impractical, strategy that always selects the "best" next move. We evaluate a practical bidding strategy by comparing the risk to a particular site s^* when all sites use it, versus when only s^* switches to an exhaustive strategy (with others remaining practical; c.f., [57]). If the difference is modest, we deem the practical strategy adequate. The exhaustive strategy is defined by the following parameters:

- s^*.cutline: When s^*.cutline = *true*, s^* is excluded from the dictated bidding sequence and instead can choose to bid when it wants (subject to the slack constraint), essentially "cutting in line". If s^*.cutline = *true* and s^* bids, then some other s must bid before s^* is allowed to bid again, lest s^* defer others' bids indefinitely. If s^*.cutline = *false*, then it must wait its turn in the bidding sequence to bid.
- s^*.foresight: This parameter is a natural number that indicates the number of forthcoming bidders that s^* can predict accurately. For example, if sites bid in a deterministic (e.g., round-robin) fashion, then s^* will generally be able to predict the full sequence of bidders in advance. It could then use this information in determining what to bid, if it is the next bidder, or, if s^*.cutline = *true* and slack allows, if it chooses to bid next.
- s^*.lookahead: This parameter is a natural number that indicates the depth beyond the next bidders predicted s^*.foresight, to which s^* analyzes all possible bidding sequences. That is, s^* can explore all possible bidding sequences of length s^*.lookahead, beginning after the s^*.foresight bidders that it knows, in order to inform its current bid or, if s^*.cutline = *true* and slack allows, its choice of whether to insert a bid. In this exploration, s^* considers every other possible bidder (subject to slack) as equiprobable. We always permit s^*.lookahead ≥ 1.

In order for the exhaustive s^* to make the "best" possible allocation looking forward s^*.foresight + s^*.lookahead steps, we equip it with access to each peer's capacities and bids, which is disallowed generally. To limit the search space, we restrict the exhaustive s^* to bidding in fixed, tunable increments; otherwise, its possible allocations would grow exponentially with capacity. Given fixed s^*.cutline, s^*.foresight, and s^*.lookahead, s^* evaluates the tree of possible outcomes based on the foreseeable s^*.foresight bidders followed by equiprobable s^*.lookahead bidders, considering all potential bidding positions if s^*.cutline = *true*. If it is s^*'s turn or s^*.cutline = *true* and bidding is optimal, s^* places the bid that minimizes s^*.risk per Eqn. (5).

3.5. Proportional-response bidding strategy

Now we propose a practical bidding strategy. We say the strategy is "practical" in that, unlike the exhaustive strategy described in §3.4, this strategy can be computed efficiently with only local information, namely a site s's own capacity, the sequence of bids s has received so far, and either $|s.\text{users} \cap s'.\text{users}|$ or $s.\text{users} \cap s'.\text{users}$ for $s' \in S \setminus \{s^*\}$ depending on the psica or psi setting, respectively.

The practical bidding strategy that we explore here is *proportional response*, shown as Alg. 1. On its first bid (P2a), s must bootstrap some information about the risk each of its peers poses to its accounts, so that it can allocate its capacity proportional to the peer providing the least risk. Since s does not know any peer's capacity, it calculates its per-peer risk assuming it will receive one slot from each peer. Since s would like to allocate resources proportional to how little risk each peer poses (rather than how much), it calculates s.baseWt(s') by first computing one minus each peer's proportion of the total risk. These values sum to $n-2$, so s normalizes by this sum to obtain s.baseWt(s') that sum to one. Finally, s uses these baseline weights to allocate a proportion of its capacity to each peer s'.

Each time s receives a new bid from s' (P1b), s incorporates this bid into the average allocation s.allocFr(s') that it has received from s', using exponential smoothing with smoothing factor s.smf. Using s.allocFr(s'), s then updates its s.avgRisk(s'). By averaging the risk incurred per peer, P1b safeguards against sites that strategically reallocate capacity to gain more slots—for example, by giving all capacity to one peer in one round and abruptly switching to another in the next. This averaging takes place only after receiving two bids (i.e., see P1a); since s''s first bid (P2a) is capacity-agnostic, s including this bid in its average calculation could falsely inflate how many slots s should expect from s' in the future.

When s bids in *response* to the slots it has received (P2b), it allocates its capacity proportionally to the site that provided the least average risk. To do so, s first calculates one minus the proportion of total average risk provided per peer. These values sum to $n-2$, so s normalizes them to obtain weights s.weight(s') that sum to one. Overall, s.weight(s') incentivizes reciprocity by rewarding peers that minimized s's risk.

Relying only on s.weight(s') to calculate s's proportional bid to s', however, could distort incentives. For example, if s received zero average risk from all peers except s', it would split s.cap evenly among the zero-risk peers, leaving no slots for s'. Such an allocation is really unfair if s' provided many slots k to s but resulted in a minuscule s.risk(s', k) due to large $|s.\text{users} \cap s'.\text{users}|$. To mitigate such situations, P2b interpolates s.weight(s') and s.baseWt(s') with the inverse of one plus the maximum s.avgRisk(s'') that s receives from any peer s''. This ensures that peers that pose similar average risk to s receive similar allocations from s (subject to the baseline risks), while still maintaining proportional responses from s when peers pose varying risks.

Algorithm 1 Proportional Response Strategy (privLvl)

P1) When site s receives an allocation k from s':

a) If this is the first or second allocation to s from s', s sets

$$s.\mathsf{allocFr}(s') \leftarrow \begin{cases} \min\{k, |s.\mathsf{users}|\} \\ \qquad \text{if privLvl = psica} \\ \min\{k, |s.\mathsf{users} \cap s'.\mathsf{users}|\} \\ \qquad \text{if privLvl = psi} \end{cases}$$

b) After the first two allocations to s from s', s updates:

$$s.\mathsf{allocFr}(s') \leftarrow \begin{cases} \left(\begin{array}{l} s.\mathsf{smf} \times \min\{k, |s.\mathsf{users}|\} \\ +\ (1 - s.\mathsf{smf}) \times s.\mathsf{allocFr}(s') \end{array}\right) \\ \qquad \text{if privLvl = psica} \\ \left(\begin{array}{l} s.\mathsf{smf} \times \min\left\{k, \left|s.\mathsf{users} \cap \right| \right\} \\ \qquad\qquad\quad s'.\mathsf{users}\right| \\ +\ (1 - s.\mathsf{smf}) \times s.\mathsf{allocFr}(s') \end{array}\right) \\ \qquad \text{if privLvl = psi} \end{cases}$$

$$s.\mathsf{avgRisk}(s') \leftarrow s.\mathsf{risk}(s', s.\mathsf{allocFr}(s'))$$

P2) When site s bids:

a) If s has not yet received an allocation from some site in $S \setminus \{s\}$, then s allocates

$$s.\mathsf{baseWt}(s') \leftarrow \left(\frac{1}{n-2}\right)\left(1 - \frac{s.\mathsf{risk}(s', 1)}{\sum_{s'' \in S \setminus \{s\}} s.\mathsf{risk}(s'', 1)}\right)$$

$$s.\mathsf{allocTo}(s') \leftarrow \lfloor s.\mathsf{cap} \times s.\mathsf{baseWt}(s') \rfloor$$

for each $s' \in S \setminus \{s\}$.

b) After s has received an allocation from every site in $S \setminus \{s\}$, s allocates

$$s.\mathsf{weight}(s') \leftarrow \left(\frac{1}{n-2}\right)\left(1 - \frac{s.\mathsf{avgRisk}(s')}{\sum_{s'' \in S \setminus \{s\}} s.\mathsf{avgRisk}(s'')}\right)$$

$$\mu \leftarrow \max_{s' \in S \setminus \{s\}} s.\mathsf{avgRisk}(s')$$

$$s.\mathsf{allocTo}(s') \leftarrow \left\lfloor s.\mathsf{cap} \times \left(\begin{array}{l} \left(\frac{1}{1+\mu}\right) \times s.\mathsf{baseWt}(s') \\ +\ \left(\frac{\mu}{1+\mu}\right) \times s.\mathsf{weight}(s') \end{array}\right) \right\rfloor$$

for each $s' \in S \setminus \{s\}$.

3.6. Evaluation

We evaluate the proportional strategy (Alg. 1) to understand whether: (1) a site can reduce its risk by increasing its capacity; (2) unpopular sites incur comparable risk to their popular peers; and (3) the proportional strategy incurs comparable risk to the exhaustive strategy. To study this, we conduct model-checking experiments over the parameter space defining the exhaustive strategy, proportional strategy, and bidding sequences. In each auction, one site—s^*—uses either the exhaustive or proportional strategy (denoted s^*_{e} or s^*_{p}, respectively) while all peers $S \setminus \{s^*\}$ always use the proportional strategy. We allow s^* to set its parameters (marked with a "$*$") independently of its peers. To examine $s^*.\mathsf{risk}$ across varying environments, we define:

- $s^*.\mathsf{pop}$: This parameter is defined over $[0, 1]$ for s^* and controls how $u.\mathsf{sites}$ is determined per $u \in U$. By varying $s^*.\mathsf{pop}$ we generate a range of user placements that model a s^* with varying popularity. Assigning each site a unique site identifier $s.\mathsf{id} \in \{1, \ldots, n\}$ such that $s^*.\mathsf{id} = 1$, we set

$$\Pr[s \in u.\mathsf{sites}] = (1 - s^*.\mathsf{pop}) \times \frac{s.\mathsf{id}}{n} + s^*.\mathsf{pop} \times \left(1 - \frac{s.\mathsf{id} - 1}{n}\right)$$

So, when $s^*.\mathsf{pop} = 1$, s^* is the most popular site: $\Pr[s^* \in u.\mathsf{sites}] = 1$, versus $\Pr[s \in u.\mathsf{sites}] = 1/n$ for the least popular site s with $s.\mathsf{id} = n$. When $s^*.\mathsf{pop} = 0$, s^* is the least popular site: $\Pr[s^* \in u.\mathsf{sites}] = 1/n$, versus $\Pr[s \in u.\mathsf{sites}] = 1$ for the most popular site s with $s.\mathsf{id} = n$. When $s^*.\mathsf{pop} = 1/2$, sites are equally popular, with $\Pr[s \in u.\mathsf{sites}] = \frac{n+1}{2n}$ per s.
- $s.\mathsf{capC}$: This parameter is the *capacity coefficient*, defined over [0,1], which determines the capacity, $s.\mathsf{cap}$ of each site $s \in S \setminus \{s^*\}$ as $s.\mathsf{cap} \leftarrow s.\mathsf{capC} \times |s.\mathsf{users}|$. This reflects the assumption that more popular sites (i.e., sites with more users) may have more resources to monitor logins for others. s^* can vary $s^*.\mathsf{capC}$ independently.

The following auctions were run with $n = 4$ sites and $\ell = 400$ users (larger experiments are reported in 4.4). We varied $s^*_{\mathsf{e}}.\mathsf{foresight}$ and $s^*_{\mathsf{e}}.\mathsf{lookahead}$ so that $s^*_{\mathsf{e}}.\mathsf{foresight} + s^*_{\mathsf{e}}.\mathsf{lookahead} \leq 3$ and $s^*_{\mathsf{e}}.\mathsf{lookahead} > 1$, and $s^*_{\mathsf{e}}.\mathsf{cutline} \in \{\mathit{true}, \mathit{false}\}$. Setting $s^*_{\mathsf{e}}.\mathsf{lookahead} \geq 1$ ensured that s^*_{e} will always consider future bidders when deciding its allocation, and setting $s^*_{\mathsf{e}}.\mathsf{foresight} + s^*_{\mathsf{e}}.\mathsf{lookahead} \leq 3$ allowed our model checker to scale efficiently (see §D.1 for more details). To implement the proportional strategy for $s \in S \setminus \{s^*\}$ we varied $s.\mathsf{smf} \in \{1, 0.75, 0.5, 0.25\}$. Additionally, for s^*_{p} we varied $s^*.\mathsf{smf} \in \{1, 0.75, 0.5, 0.25\}$.

We tested every combination of each strategy's parameters for auctions of length $r = 10$, defined as the number of bids after all $s \in S \setminus \{s^*\}$ (and potentially s^*_{p}) have completed their first bid according to P2a. For each $\mathsf{slack} \in \{1, 2, 3, \infty\}$ and $s^*_{\mathsf{e}}.\mathsf{cutline} \in \{\mathit{true}, \mathit{false}\}$, we generated 30 bidding sequences. We varied $s^*.\mathsf{pop}$, $s.\mathsf{capC}$, $s^*.\mathsf{capC}$ across five values in [0,1], and resampled each configuration 30 times. Configurations were reused across both exhaustive and proportional auctions. In total, our evaluation includes approximately 500,000,000 auctions.

Since increasing $s^*.\mathsf{pop}$ increases $|s^*.\mathsf{users}|$, and more importantly, increases at risk $|s^*.\mathsf{users} \cap s.\mathsf{users}|$ per s, we compare $s^*.\mathsf{risk}(r')$ across different $s^*.\mathsf{pop}$ values by examining $s^*.\mathsf{normRisk}(r')$ (Eqn. (6)). We use the Kruskal-Wallis H test to assess how parameter settings affect the median $s^*.\mathsf{normRisk}(r')$ and a Dunn test with Bonferroni correction determines the direction and significance of effects (p-values shown). We also report the Kruskal-Wallis effect size η^2, which measures the proportion of variance in $s^*.\mathsf{normRisk}(r')$ explained by varying a parameter [18]. Next, we summarize the key findings.

3.6.1 Sites have incentive to increase their capacity. Fig. 1 illustrates that for a fixed $s^*.\mathsf{pop}$, the distribution of $s^*.\mathsf{normRisk}(r')$ decreased ($p < 10^{-8}$) as s^* increased $s^*.\mathsf{capC}$. This underscores s^*'s incentive to increase its capacity to help *others* to reduce its *own* risk, regardless of its popularity or strategy.

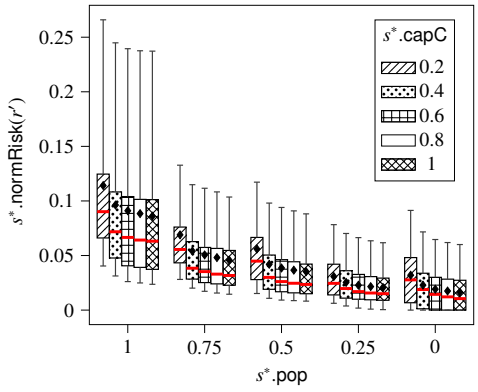


Figure 1: Distribution of s^*.normRisk(r') per s^*.pop, s^*.capC including s^*_e and s^*_p. Each box spans 25th-75th percentile; whisker spans 5th-95th percentile; diamond shows the mean; red line shows the median. s^*.normRisk(r') decreased with higher s^*.capC and lower s^*.pop.

		advFreq$_\theta$				fracAdv$_\theta$			
		slack				slack			
s^*.cutline	s^*.foresight	1	2	3	∞	1	2	3	∞
false	0	0.518	0.510	0.482	0.483	0.083	0.081	0.077	0.075
	1	0.610	0.636	0.610	0.620	0.086	0.085	0.082	0.080
true	0	0.697	0.678	0.647	0.645	0.103	0.102	0.103	0.104
	1	0.799	0.778	0.734	0.717	0.150	0.130	0.120	0.116

Figure 2: Comparison of exhaustive and proportional strategies across θ constraints defined by combinations of s^*_e.cutline, s^*_e.foresight, and slack. The left table shows advFreq$_\theta$ (Eqn. (7)) and the right table shows fracAdv$_\theta$ (Eqn. (9)). Setting $\theta \leftarrow s^*_e$.cutline = *false*, s^*_e.foresight = 0, slack = ∞ diminished advFreq$_\theta$ and fracAdv$_\theta$.

3.6.2 Unpopular sites still receive protection. Fig. 1 shows that s^*.normRisk(r') generally decreased ($p < 10^{-8}$) as s^*.pop decreased, meaning less popular sites are fairly included by their peers and receive comparable risk reduction. Of course, an unpopular s^* is penalized if its capacity is set too low, as shown in the uptick of s^*.normRisk(r') when a s^* with s^*.pop = 0 sets s^*.capC = 0.2 in Fig. 1.

3.6.3 There is risk versus privacy trade-off. We found s^* incurred less ($p < 10^{-8}$) s^*.normRisk(r') when privLvl = psi versus when privLvl = psica. This is expected, as using psi lets s^* monitor only its vulnerable users, s^*.users $\cap$ s.users, at peer s. While psi reduces s^*.normRisk(r') , it discloses cross-site account memberships, posing a privacy risk to affected users.

3.6.4 Limiting when a site can bid, and how much knowledge a site has of the bidding sequence, minimizes the risk improvement exhaustive **provides over** proportional**.** To limit the advantage of an exhaustive bidder and so reduce a proportional bidder's incentive to change strategies, we focus on parameters enforceable by global ecosystem settings: s^*_e.cutline, s^*_e.foresight, and slack. Here we identify values that minimize the advantage that s^*_e provides over s^*_p, deferring enforcement details to §7.2.

Let A denote a set of auction pairs such that $(a_p, a_e) \in A$ implies that a_p and a_e were conducted with identical user placements, site capacities, slack, s.smf, privLvl, and bidding sequence (when s^*_e.cutline = *false*) but that s^* bid according to the exhaustive strategy in a_e and according to the proportional strategy in a_p. Let $A_\theta \subseteq A$ be the subset of such pairs that satisfy some further constraints, specified as θ. For various conditions θ, we seek to quantify the fraction advFreq$_\theta$ of bids at the same index r' in auction pairs in A_θ for which s^*_e.normRisk(r') is less than (i.e., improved on) s^*_p.normRisk(r') and, in those cases, the median absolute improvement absAdv$_\theta$ and the median relative improvement fracAdv$_\theta$. Letting $a[r'] = s^*$.normRisk(r') for the r'-th bid of an r-bid auction a (i.e., $1 \le r' \le r$), these values are:

$$\text{advFreq}_\theta = \left(\frac{1}{r\,|A_\theta|}\right)\left|\left\{(a_p[r'], a_e[r']) \;\middle|\; \begin{matrix}(a_p, a_e) \in A_\theta \wedge \\ a_e[r'] < a_p[r']\end{matrix}\right\}\right| \tag{7}$$

$$\text{absAdv}_\theta = \text{med}\left\{a_p[r'] - a_e[r'] \;\middle|\; \begin{matrix}(a_p, a_e) \in A_\theta \wedge \\ a_e[r'] < a_p[r']\end{matrix}\right\} \tag{8}$$

$$\text{fracAdv}_\theta = \text{med}\left\{\frac{a_p[r'] - a_e[r']}{a_p[r']} \;\middle|\; \begin{matrix}(a_p, a_e) \in A_\theta \wedge \\ a_e[r'] < a_p[r']\end{matrix}\right\} \tag{9}$$

Among the globally enforceable parameters, s^*_e.cutline determines whether s^*_e can control the timing of its bids. We found that by disabling this choice (i.e., $\theta \leftarrow s^*_e$.cutline = *false*), we reduced advFreq$_\theta$ and reduced ($p < 10^{-8}$) fracAdv$_\theta$ per s^*_e.foresight and slack, as shown in Fig. 2.

The s^*_e.foresight parameter provides s^*_e with information about the future bidding sequence, and disabling it (i.e., $\theta \leftarrow s^*_e$.foresight = 0) reduced advFreq$_\theta$ and reduced ($p < 10^{-8}$) fracAdv$_\theta$, per s^*.cutline and slack.

The slack parameter provides s^*_e with a distribution over future bidders, and when slack is tight, s^*_e often identifies the next bidder exactly. This trend is reflected in Fig. 2, where when $\theta \leftarrow s^*_e$.cutline = *false* $\wedge$ s^*_e.foresight = 0, decreasing slack resulted in higher advFreq$_\theta$, and also raised ($p < 10^{-8}$) fracAdv$_\theta$. To mitigate this advantage, we recommend setting no restrictions on slack (i.e., slack = ∞).

In summary, our findings suggest a system deployment where a site cannot choose when to bid, cannot predict who will bid next, and otherwise faces no restrictions on possible bidding sequences. We denote these specific constraints as $\psi \leftarrow s^*$.cutline = *false* $\wedge$ s^*.foresight = 0 $\wedge$ slack = ∞. When considering the corresponding bids in A_ψ, we found advFreq$_\psi$ < 0.5, fracAdv$_\psi$ = 0.075, and

$$\text{med}\left\{\frac{a_p[r'] - a_e[r']}{a_p[r']} \;\middle|\; (a_p, a_e) \in A_\psi\right\} = 0 \tag{10}$$

This suggests that proportional provides little incentive for sites to deviate. While we cannot enforce values for the remaining s.smf, s^*_p.smf, s^*_e.lookahead, s.capC parameters, we find, assuming ψ, varying them either has negligible impact on s^*_p's risk or yields impractical gains for s^*_e.

3.6.5 Keeping track of past bids does not decrease a site's risk if it cannot choose when to bid and has limited knowledge about the next bidder. Each proportional bidder uses s.smf to track average slots received per peer and

s^*.capC	s.capC 0.2	0.4	0.6	0.8	1.0	s^*.lookahead 1	2	3
1	0.00212	0.00212	0.00173	0.00150	0.00132	0.00170	0.00200	0.00202
0.8	0.00309	0.00211	0.00200	0.00139	0.00202	0.00200	0.00203	0.00203
0.6	0.00211	0.00201	0.00134	0.00227	0.00182	0.00210	0.00210	0.00210
0.4	0.00408	0.00293	0.00295	0.00372	0.00277	0.00296	0.00296	0.00296
0.2	0.00563	0.00489	0.00653	0.00732	0.00591	0.00578	0.00582	0.00592

(a) $\psi^* \leftarrow \psi \wedge s^*.\mathsf{pop} = 1 \wedge \mathsf{privLvl} = \mathsf{psi}$ ($\mathsf{advFreq}_{\psi^*} = 0.513$)

s^*.capC	s.capC 0.2	0.4	0.6	0.8	1.0	s^*.lookahead 1	2	3
1	0.00241	0.00158	0.00113	0.00101	0.00110	0.00129	0.00129	0.00129
0.8	0.00308	0.00211	0.00127	0.00119	0.00157	0.00159	0.00159	0.00159
0.6	0.00309	0.00212	0.00170	0.00139	0.00162	0.00172	0.00172	0.00172
0.4	0.00308	0.00239	0.00170	0.00202	0.00279	0.00215	0.00212	0.00212
0.2	0.00564	0.00355	0.00423	0.00654	0.01345	0.00539	0.00540	0.00540

(b) $\psi^* \leftarrow \psi \wedge s^*.\mathsf{pop} = 0.75 \wedge \mathsf{privLvl} = \mathsf{psi}$ ($\mathsf{advFreq}_{\psi^*} = 0.442$)

Figure 3: Each cell shows absAdv_{ψ^*} with lighter cells indicating less exhaustive advantage. We omit $s^*.\mathsf{pop} < 0.75$ and $\mathsf{privLvl} = \mathsf{psica}$, due to space, though trends are similar. absAdv_{ψ^*} peaked when $s^*.\mathsf{capC} \ll s.\mathsf{capC}$, and minimally improved at higher $s^*_{\mathsf{e}}.\mathsf{lookahead}$.

guard against peers abruptly shifting allocations to maximize their own slot returns. However, under ψ constraints, we found that $s.\mathsf{smf}$ value employed by *other* sites does not affect $s^*.\mathsf{normRisk}(r')$ ($p = 0.566$). We also found that allowing s^*_{p} to set $s^*.\mathsf{smf}$ independently of its peers did not affect $s^*_{\mathsf{p}}.\mathsf{normRisk}(r')$ ($p > 1 - 10^{-8}$).

3.6.6 The costs of computing exhaustive bids render it inferior to proportional, even in the rare cases where it can reduce s^*'s risk. Fig. 3 shows that per $s^*.\mathsf{pop}$ and privLvl, s^*_{e} maximized absAdv_{ψ} when its capacity was scaled much lower than its peers' capacities (i.e., $s^*.\mathsf{capC} \ll s.\mathsf{capC}$). This suggests s^*_{e} can reduce its risk moderately relative to s^*_{p}, but only by severely restricting monitoring costs. Yet, computing the exhaustive strategy is a considerable common-case cost—far more than simply increasing $s^*.\mathsf{capC}$ to match its peers, as shown in §6.

While increasing $s^*_{\mathsf{e}}.\mathsf{lookahead}$ reduced ($p < 10^{-8}$) absAdv_{ψ}, the effect size (4.17×10^{-6}) was negligible. We attribute significance to dataset size and conclude $s^*_{\mathsf{e}}.\mathsf{lookahead}$ has little practical impact in decreasing s^*_{e}'s risk, as also reflected in Fig. 3 which shows no notable increase absAdv_{ψ} per $s^*.\mathsf{pop}$ and privLvl. Since computing optimal allocation grows exponentially in $s^*_{\mathsf{e}}.\mathsf{lookahead}$, simply raising $s^*.\mathsf{capC}$ is a more practical way to reduce risk. We conclude proportional is the preferred strategy.

4. Attacker Model

We now extend the analysis described in §3 to assess the security of the monitoring slots allocated according to the proportional strategy, against an attacker attempting to harvest a site's accounts by first stuffing sweetwords stolen from that site at peer sites. We restrict our attention to sites using the proportional strategy, since as we concluded in §3.6, ensuring that a site can neither predict next bidders ($s^*_{\mathsf{e}}.\mathsf{foresight} = 0$) nor choose when to bid ($s^*_{\mathsf{e}}.\mathsf{cutline} = \mathit{false}$) leaves a site little reason to deviate from proportional, assuming it would do so to minimize its risk.

We also focus on proportional bidders for scalability. Since the attacker explores all bids a site might make, allowing exhaustive bidders would require duplicate effort from the attacker. So, to scale our analysis, we restrict the attacker to targeting proportional bidders and infer performance against any (myopic) strategy from the close match between $s^*_{\mathsf{p}}.\mathsf{risk}$ and $s^*_{\mathsf{e}}.\mathsf{risk}$.

4.1. Attacker Threat Model

The attacker who breaches a site s^*'s password database and stuffs these credentials elsewhere is rationally motivated, in terms of wanting to harvest as many accounts as possible at s^* while dodging s^*'s monitoring efforts. Sites—in particular, s^* itself, despite having its credential database stolen—continue to behave as assumed in §3.1 and, in particular, follow proportional bidding. This threat model characterizes common practice while also highlighting the difficulty of *detecting* the passive breach of the database, since, say, behavioral modifications to s^* would provide additional features by which s^*'s administrators might detect the breach had occurred.

We make two (conservative) allowances for the attacker.

- If the attacker dodges s^*'s monitoring at s, then the attacker has successfully harvested all those user accounts it stuffed at s, and therefore at s^* (and every $s \in S \setminus \{s^*\}$). This allowance is optimistic for the attacker, but not unreasonably so, given users' tendencies to reuse the same or similar passwords across sites [15], [70], [82]. That is, once the attacker has harvested these accounts at s, it can easily harvest the same user's accounts at s^*, too [37].
- We allow the attacker to know all parameters defining the proportional strategy per $s \in S$ (i.e., slack, $s.\mathsf{smf}$, and $s.\mathsf{cap}$, as the attacker could infer these from observing bidding behavior, anyway) and to know $s.\mathsf{users}$ for every site $s \in S$ and so $u.\mathsf{sites}$ for every user $u \in U$. This also favors the attacker, though assuming otherwise seems unrealistic and would render our analysis too fragile.

4.2. Attacker's Stuffing Strategy

Much like the exhaustive s^* detailed in §3.4, the attacker a is characterized by certain parameter choices.

- $a.\mathsf{foresight}$: We parameterize the attacker with the ability to predict the next $a.\mathsf{foresight}$ bidders accurately, as well as their bids. We stress that $a.\mathsf{foresight}$ is unrelated to $s^*_{\mathsf{e}}.\mathsf{foresight}$ from the previous section, but again, in this section we consider only s^*_{p}.
- $a.\mathsf{lookahead}$: Beyond the next bidders it can predict as indicated by the $a.\mathsf{foresight}$ parameter, the attacker can examine all sequences of bidders to a depth $a.\mathsf{lookahead}$ to inform its stuffing attempt. In this exploration, the attacker considers every possible bidder (subject to slack) as equiprobable. We always permit $a.\mathsf{lookahead} \geq 1$.
- $a.\mathsf{aggression}$: We parameterize the attacker with an *aggression level*, $a.\mathsf{aggression} \in [0, 1]$. The attacker mounts only stuffing attempts that result in one minus the cumu-

lative dodge probability (from the beginning of the attack) not exceeding a.aggression.

After each bid, the attacker examines the tree of possible monitoring allocations based on the known a.foresight bidders and equiprobable bidders a.lookahead after that, for fixed values of a.foresight and a.lookahead. The attacker also generates all possible stuffing strategies across those a.foresight + a.lookahead bids consisting of unharvested users in s^*.users and that would satisfy the a.aggression constraint. From this tree, the attacker identifies the leaf that maximizes the expected number of users in s^*.users at which the attacker has captured an account at some s (and so presumably at all $s' \in S \setminus \{s\}$, as well). It then performs stuffing attempts per $s \in S \setminus \{s^*\}$ that are prescribed in the first step of the path to that leaf. The attacker continues building a.foresight+a.lookahead-depth trees until the a.aggression parameter no longer permits stuffing attempts or once the accounts of all users in s^*.users have been harvested.

We assume the attacker begins stuffing accounts only after every site has placed its first bid according to P2a. Otherwise, the attacker would trivially capture users in s^*.users at $s \in S \setminus \{s^*\}$ that have not bid any slots to s^*, mirroring the current state of the world without monitoring. The attacker's goal is to maximize s^*.cost(r'), the expected number of users in s^*.users for which the attacker can stuff some account at some $S \setminus \{s^*\}$ given the most current s^*.allocTo(s) per s. So, s^*.cost(1) is the expected number of users in s^*.users for whom the attacker stuffed an account at some site in $S \setminus \{s^*\}$ after all P2a-bids but before the first P2b-bid. The attacker remains exhaustive within its a.foresight + a.lookahead bounds, but optimizing for s^*.cost(r) would require a.foresight + a.lookahead = r, which is computationally intractable.

4.3. Evaluation

In this section, our evaluation of the attacker's strategy aims to establish: (1) s^*'s risk is a good predictor of the number of users that an attacker can harvest at the beginning of its attack, and so minimizing risk is fruitful for a site to maximize its detection ability, and (2) an attacker lacks avenues to arbitrarily increase its ability to harvest users in s^*.users. We investigate these trends via model-checking experiments with $n = 4$ sites and $\ell = 400$ users. Since we modeled only proportional bidders, we specified slack, s.smf, s^*.smf, and privLvl. We previously concluded that we need not put any restrictions on bidding order (§3.6.4) and that keeping track of past bids does not impact s^*_{p}.normRisk(r') (§3.6.5)); so, we set slack = ∞, s.smf = 1 and s^*.smf = 1. We varied privLvl $\in$ {psica, psi}.

To keep computational costs reasonable, we set a.lookahead + a.foresight ≤ 2 with a.lookahead ≥ 1, and varied a.aggression $\in [0.25, 0.5, 0.75]$. We varied s^*.pop, s.capC, and s^*.capC across five values $\in [0, 1]$, resampled each configuration 30 times, and sampled 30 bidding sequences composed of 10 attacker-eligible bids each. We define s^*.vulnUsers as users in s^*.users vulnerable to stuffing via shared credentials with some $s \in S \setminus \{s^*\}$. Under our current assumption of a password reuse rate of 1 (optimistic for the attacker), s^*.vulnUsers includes every user in s^*.users who has at least one account at $s \in S \setminus \{s^*\}$. To compare s^*.cost(r') across s^* with varying s^*.pop we normalize

$$s^*.\mathsf{normCost}(r') = \frac{s^*.\mathsf{cost}(r')}{|s^*.\mathsf{vulnUsers}|} \tag{11}$$

since higher s^*.pop implies larger $|s^*.\mathsf{vulnUsers}|$.

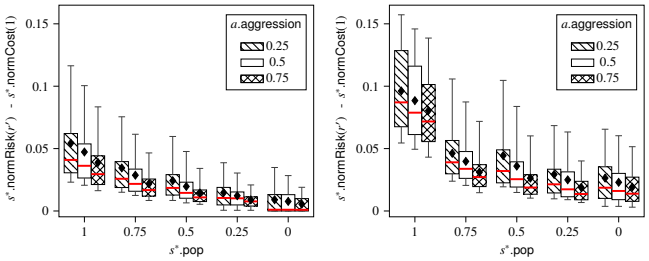
We used the Kruskal-Wallis H test and post-hoc Dunn test with Bonferroni correction to analyze the effect of various parameters on s^*.normCost(r') and s^*.normRisk(r') − s^*.normCost(r'). The latter value emphasizes the predictive accuracy of s^*.normRisk(r') for s^*.normCost(r'), with values closer to 0 indicating better predictions.

4.3.1 Risk predicts cost more accurately at early rounds of an attack. Since sites are unaware of the timing of an attack, s^*.normRisk(r') is calculated with an implicit assumption that the attack has not yet started. Consequently, once credential stuffing is underway, at bid $r' > 1$, s^*.normRisk(r') overestimated s^*.normCost(r') because the accounts of some subset of s^*.users has already been captured by the attacker. However, we found s^*.normRisk(r') to be a good predictor of s^*.normCost(1) with the median (over all auctions) of s^*.normRisk(r') − s^*.normCost(1) as 0.0222 of vulnerable users at s^*.

4.3.2 Risk predicts cost more accurately when sites know which users they share with peers. In the psi setting, s^* knows which users it shares with each peer s, which is the subset of s^*.users vulnerable to stuffing at s. In contrast, in the psica setting, s^* has strictly less information than the attacker about which of its users are vulnerable at s. This distinction between privLvl is reflected in the comparison between Fig. 4a and Fig. 4b. Specifically, for a fixed s^*.pop and a.aggression, the quantity s^*.normRisk(r') − s^*.normCost(1) was lower ($p < 10^{-8}$) when privLvl = psi than when privLvl = psica. The reduced predictive ability of s^*.normRisk(r') in the psica setting is a shortcoming that we discuss further in App. A.

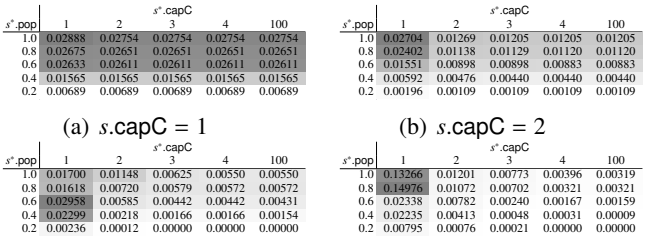
4.3.3 Risk predicts cost more accurately for less popular sites. Site s^* calculates s^*.normRisk(r') in a pairwise fashion, i.e., summing s^*.normRisk(s, k) incurred from a site s due to the allocation $k = s$.allocTo(s^*), over all $s \in S \setminus \{s^*\}$ (Eqn. (6)). Because the set of users that s^* shares with one site s can overlap with those it shares with another site s', however, calculating s^*.normRisk(r') in this way overestimates the number of its accounts that an attacker can expect to harvest at other sites. Fig. 4 shows this trend, in particular, the distribution of s^*.normRisk(r') − s^*.normCost(1) decreased ($p < 10^{-8}$) with lower s^*.pop per a.aggression, since such s^* share less users with their peers. An open question, discussed in App. A, is whether risk can be measured independently of a site's popularity.

4.3.4 Risk predicts cost more accurately when the attacker is more aggressive. Since s^* lacks information about the attacker, it calculates s^*.normRisk(r') assuming the attacker will be maximally aggressive. As



(a) psi (b) psica

Figure 4: Distribution of s^*.normRisk(r') − s^*.normCost(1) per privLvl, s^*.pop and a.aggression. Each boxplot's whiskers span the 5th-95th percentile; the diamond shows the mean; red line shows the median. s^*.normRisk(r') − s^*.normCost(1) decreased when privLvl = psi and for lower a.aggression and s^*.pop values.



s^*.capC

s^*.pop	1	2	3	4	100
1.0	0.02888	0.02754	0.02754	0.02754	0.02754
0.8	0.02675	0.02651	0.02651	0.02651	0.02651
0.6	0.02633	0.02611	0.02611	0.02611	0.02611
0.4	0.01565	0.01565	0.01565	0.01565	0.01565
0.2	0.00689	0.00689	0.00689	0.00689	0.00689

(a) s.capC = 1

s^*.capC

s^*.pop	1	2	3	4	100
1.0	0.02704	0.01269	0.01205	0.01205	0.01205
0.8	0.02402	0.01138	0.01129	0.01120	0.01120
0.6	0.01551	0.00898	0.00898	0.00883	0.00883
0.4	0.00592	0.00476	0.00440	0.00440	0.00440
0.2	0.00196	0.00109	0.00109	0.00109	0.00109

(b) s.capC = 2

s^*.capC

s^*.pop	1	2	3	4	100
1.0	0.01700	0.01148	0.00625	0.00550	0.00550
0.8	0.01618	0.00720	0.00579	0.00572	0.00572
0.6	0.02958	0.00585	0.00442	0.00442	0.00431
0.4	0.02299	0.00218	0.00166	0.00166	0.00154
0.2	0.00236	0.00012	0.00000	0.00000	0.00000

(c) s.capC = 3

s^*.capC

s^*.pop	1	2	3	4	100
1.0	0.13266	0.01201	0.00773	0.00396	0.00319
0.8	0.14976	0.01072	0.00702	0.00321	0.00321
0.6	0.02338	0.00782	0.00240	0.00167	0.00159
0.4	0.02235	0.00413	0.00048	0.00031	0.00009
0.2	0.00795	0.00076	0.00021	0.00000	0.00000

(d) s.capC = 4

Figure 5: Mean s^*.normCostCum(+10) for a.aggression = 0.75, privLvl = psi. Lighter cells show lower cost.

a result, s^*.normRisk(r') provides the best estimate for s^*.normCost(1) when a.aggression is high. Fig. 4 confirms that per privLvl and s^*.pop, the distribution of s^*.normRisk(r') − s^*.normCost(1) decreased ($p < 10^{-8}$) as a.aggression increased. Fig. 4 also shows that the effect of a.aggression on s^*.normRisk(r') − s^*.normCost(1) decreased as s^*.pop decreased, due to a decreased number of vulnerable users at an unpopular s^*.

4.3.5 The attacker's ability to capture users increases modestly with additional knowledge of the bidding sequence. We confirmed that an attacker with nonzero a.foresight can benefit ($p < 10^{-8}$) from its ability to predict the next bidder. Regardless, §7.2 outlines mechanisms to enforce a.foresight = 0, eliminating this predictive advantage. An attacker can also strategize by examining the probability distribution of all potential next bidders for a.lookahead steps ahead. However, we found that when a.foresight = 0, a.lookahead does not affect ($p = 0.265$) s^*.normCost(r'). This suggests that anticipating the future sequence of bidders does not enable the attacker to capture more users.

Since deep attacker trees, determined by a.foresight + a.lookahead, limit scalability, we use the above settings to explore larger systems in §4.4. §D.2 details the optimizations that enabled scaling in our custom model checker.

4.4. Larger systems

We now analyze the effectiveness of the attacker's strategy against proportional bidders in a scaled-up model-checking experiment with $n = 10$ sites and $\ell = 1000$ users. While previously we enabled the attacker to begin stuffing immediately after each site submitted its first bid via P2a, we now require the attacker to wait until every site has issued at least one bid under P2b. This delay was a concession to the scalability of these experiments, allowing the attacker to invest its stuffing attempts after bids fully informed by each site's responses to their peers' bids. Starting from the second P2b bid of the first bidder to issue a second P2b bid, we granted the attacker a r-bid window to maximize the expected number of users in s^*.users for whom it successfully stuffed an account at a site in $S \setminus \{s^*\}$ with a password its user reused at s^*, which we denote s^*.costCum(+r). We also define

$$s^*.\text{normCostCum}(+r) = \frac{s^*.\text{costCum}(+r)}{|s^*.\text{vulnUsers}|} \tag{12}$$

We largely adopt the same experimental setup as described in §4.3, setting slack = ∞, s.smf = 1 and s^*.smf = 1, and varying privLvl $\in$ {psica, psi} and a.aggression $\in \{0.25, 0.5, 0.75\}$. The finding in §4.3.5 and anticipated deployment described in §7.2 justify setting a.foresight = 0 and a.lookahead = 1. We vary s.capC $\in \{1, 2, 3, 4\}$, and s^*.capC $\in \{1, 2, 3, 4, 100\}$, using s^*.capC = 100 to understand s^*'s security in the limit. Like before, we vary s^*.pop between $[0, 1]$. For each configuration—defined by s^*.pop, s.capC, and s^*.capC—we sample 30 instances, along with 30 bidding sequences containing 10 eligible bids as defined above. We used the Kruskal-Wallis H test to analyze the effect of the parameters on s^*.normCostCum(+10).

4.4.1 Sites can maximize their security by increasing their capacity. Fig. 5 shows that s^* can typically reduce its s^*.normCostCum(+10) by raising its own s^*.capC above the s.capC of its peers ($p < 10^{-8}$). This not only improves its own security but nudges the ecosystem toward a higher-capacity equilibrium (i.e., higher s.capC), making peers more likely to reciprocate. We see this in Fig. 5: when s.capC = 2, every s^* is incentivized to raise its s^*.capC ≥ 3 (Fig. 5b), prompting peers to adopt s.capC = 3 in Fig. 5c. This iteration continues, as each s^* again finds it beneficial to raise s^*.capC ≥ 4, reinforcing a positive feedback loop.

Popular sites are well-positioned to drive this shift: they have greater capacity since it scales with $|s^*$.users$|$, and their slots are in higher demand due to shared users with many peers. However, when resource-constrained (e.g., Fig. 5a), a popular s^* prioritizes its popular peers, leaving unpopular peers without reciprocal support. For example, a s^* with s^*.pop = 0.2 sees its s^*.normCostCum(+10) plateau despite increasing s^*.capC. Still, a moderately popular s^* (s^*.pop ≥ 0.6) remains incentivized to increase its capacity, as it benefit from more popular peers and so can trigger cyclic reciprocation even under constraints.

4.4.2 Popular sites benefit from their peers. While popular sites may offer more slots than they receive

due to their higher monitoring capacity, they still reduce s^*.normCostCum(+10) significantly ($p < 10^{-8}$) when peers increase their capacities. Fig. 5b shows that a s^* with s^*.pop = 1 has diminished ability to reduce its s^*.normCostCum(10) beyond 0.01205, suggesting its less popular peers face resource constraints and cannot reciprocate at scale. However, when these peers increase their capacity to s.capC = 3—and Fig. 5b confirms that they are incentivized to do so—the most popular s^* benefits, as evidenced from the reduction in s^*.normCostCum(10) for s^*.pop = 1 when s^*.capC = 4 in Fig. 5c. Thus, popular sites not only drive ecosystem improvements but also benefit from peer investments that relieve capacity bottlenecks.

4.4.3 Unpopular sites receive comparable security. Fig. 5 shows that less popular sites typically incur lower s^*.normCostCum(+10) than popular ones ($p < 10^{-8}$), indicating the proportional bidding ecosystem favors unpopular sites. This is because popular sites provide surplus monitoring slots that often go to less popular peers, ensuring strong baseline security for unpopular sites.

4.4.4 psi vs. psica security outcomes. We observed that s^*.normCostCum(+10) was significantly lower ($p < 10^{-8}$) in the psi setting than in psica. This is expected: in the psi setting, a s^* that receives slots from s can deploy monitoring requests specifically for users in s^*.users ∩ s.users. In contrast, in the psica setting, the s^* might deploy requests for users outside this intersection—users the attacker knows not to stuff at s. The effect of privLvl was more pronounced at lower s.capC, since when s^* receives limited slots, strategic deployment becomes even more important.

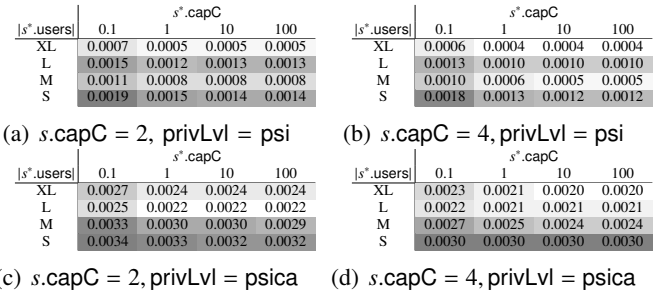
5. Data-Driven Simulations

Next, we evaluate proportional bidding using the Cit0day dataset [40] [1], which is a large credential dataset leaked from a credential-selling service. To measure password reuse precisely, we removed hashed and duplicate entries. Of the remaining entries, we retained entries belonging to the largest connected component of the user-site graph, since sites that share no users with peers trivially face no risk from credential stuffing. Post-preprocessing, the dataset contained over 108 million credential entries from 74 million users across nearly 8,000 websites. We identified users by their email addresses. For any pair of sites sharing users, we defined the password reuse rate as the ratio of the number of shared users who use the same password on both sites to the number of shared users. Reuse was very common, with the median reuse rate exceeding 0.94. We refer readers to App. B for more details on the dataset.

5.1. Attacker's Greedy Stuffing Strategy

The attacker strategy we evaluated in §4 is computationally infeasible to evaluate at the scale of this dataset. So, we instead simulated a greedy attacker that approximates optimal behavior through a series of locally optimal choices. We set a.foresight = 0 and a.lookahead = 1, giving the attacker a 10-bid stuffing window after each site placed at least one bid under P2b, following the setup in §4.3. After the r'-th (eligible) bid, the attacker emulated proportional one step ahead to estimate how many slots each $s \in S \setminus \{s^*\}$ might offer s^*. For each site, it estimated the marginal increase in expected cost from one stuffing attempt, combining its slot projection with knowledge of which users are shared across sites and empirical password-reuse rates between user-site pairs. Unlike earlier experiments that assumed perfect reuse (reuse rate of 1.0), which favored the attacker, we gave the attacker the data-driven reuse rates, which were often < 1.0.

The attacker then chose the site s with the highest marginal benefit and the user at s with the fewest accounts at other sites, i.e., $\arg\min_{u \in s.\text{users}} |u.\text{sites}|$, to preserve future stuffing opportunities. Ties in site or user selection were broken at random. The attacker repeated this process, recomputing marginal improvements to s^*.costCum($+r'$), until no further expected gain was possible.



(a) s.capC = 2, privLvl = psi

$\lvert s^*.\text{users}\rvert$	0.1	1	10	100
XL	0.0006	0.0004	0.0004	0.0004
L	0.0013	0.0010	0.0010	0.0010
M	0.0010	0.0006	0.0005	0.0005
S	0.0018	0.0013	0.0012	0.0012

(b) s.capC = 4, privLvl = psi

$\lvert s^*.\text{users}\rvert$	0.1	1	10	100
XL	0.0027	0.0024	0.0024	0.0024
L	0.0025	0.0022	0.0022	0.0022
M	0.0033	0.0030	0.0030	0.0029
S	0.0034	0.0033	0.0032	0.0032

(c) s.capC = 2, privLvl = psica

$\lvert s^*.\text{users}\rvert$	0.1	1	10	100
XL	0.0023	0.0021	0.0020	0.0020
L	0.0022	0.0021	0.0021	0.0021
M	0.0027	0.0025	0.0024	0.0024
S	0.0030	0.0030	0.0030	0.0030

(d) s.capC = 4, privLvl = psica

Figure 6: Mean s^*.normCostCum(+10) for a.aggression = 1.0. Lighter cells indicate lower s^*.normCostCum(+10).

5.2. Evaluation

We adopted the optimal parameters for proportional identified in §3.6, setting s.smf = s^*.smf = 1.0 and slack = 0, and sampled 10 bidding sequences under this setting. To evaluate security across a range of site popularities, we grouped sites into quartiles based on increasing $|s.\text{users}|$ and sampled 10 sites per quartile; "small" (S), "medium" (M), "large" (L), and "extra large" (XL). As before, we scaled capacity by user set size, setting $s.\text{cap} \leftarrow s.\text{capC} \times |s.\text{users}|$, and we varied $s.\text{capC} \in \{1, 2, 3, 4\}$ and $s^*.\text{capC} \in \{0.1, 1, 10, 100\}$. We used a wider range for s^*.capC to account for the order-of-magnitude variation in $|s.\text{users}|$; we anticipated a small s^* requiring a much higher s^*.capC to bid a comparable number of slots to its extra large peers. We varied privLvl ∈ {psi, psica}. We simulated 10 greedy attackers, which differ only in the random choices they make to break ties, and set a.aggression = 1 to simulate the worst-case scenario.

Fig. 6 summarizes the results. We omit $s.\text{capC} \in \{1, 3\}$ to conserve space, but these follow the same trends as $s.\text{capC} \in \{2, 4\}$. We report means rather than medians

1. The dataset, leaked in November 2020, has been widely reported [13], [27] and integrated into breach alerting services [40], limiting the risk of harm from this study. Following prior work [47], we anonymized email addresses and stored the data on an isolated machine to preserve privacy.

because the data is heavily skewed toward zero; many samples show attackers harvesting none of s^*.users, suggesting, in practice, sites could scale their capacities much lower than the tested s.capC, s^*.capC values. We organize our remaining findings into two parts: those that align with the conclusions in §4.4, and those that provide new nuance.

5.2.1 Validation of model-checked results. Fig. 6 confirms: (1) s^*.normCostCum(+10) was reduced with larger s^*.capC (cf., §4.4.1); (2) s^*.normCostCum(+10) for a popular s^* was lower with larger s.capC (cf., §4.4.2); (3) a less popular s^* saw s^*.normCostCum(+10) similar to more popular peers (cf., §4.4.3); and (4) s^*.normCostCum(+10) was lower with psi than with psica (cf., §4.4.4).

5.2.2 Insights beyond model-checking results. While our model-checking evaluation (§4.4) suggested that more popular sites face greater security risks (Fig. 5), Fig. 6 shows no clear link between s^*.normCostCum(+10) and popularity. This discrepancy stems in part from synthetic assumptions in the model-checking setup, where popular sites were assumed to share more users with peers. In reality, our dataset shows minimal correlation between site popularity and shared users or at-risk credentials (see Figs. 9a and 9b in App. B). This suggests that capacity planning at a site s would benefit from information beyond just its popularity, e.g., the set u.sites for each of its users $u \in s$.users. While this would be possible to compute in the psi setting, a more privacy-friendly approach would compute u.sites per $u \in s$.users while keeping u anonymous.

Moreover, comparing s^*.normCostCum(+10) for popular sites in Fig. 6a with Fig. 5b, and similarly Fig. 6b with Fig. 5d, shows generally improved outcomes in the data-driven setting. This improvement stems from two key differences: first, the use of empirical password reuse rates from the Cit0day dataset, which limits attacker success compared to the model-checking assumption of perfect reuse; and second, a more nuanced picture of cross-site user overlap, which reveals popular sites tend not to incur proportionally higher exposure through reused credentials. This suggests smaller sites may play a more active role in driving s.capC to higher equilibriums than anticipated in §4.4.1.

6. Performance

Now we evaluate the performance of our design. The time a site spends participating in a breach-detection ecosystem can be attributed to either the overhead imposed by our bidding infrastructure, or to the time of making and responding to monitoring requests according to previously developed interactive breach detection protocols [84], [85].

For the sake of this analysis, we consider the Amnesia protocol [84], and we evaluate the time to perform a single bidding step in our design. Specifically, the time incurred by a bidding step that is directly attributable to our design includes the time a site s takes to allocate its capacity in step P2b, denoted time(P2b) [2], and the time each peer $s' \in S \setminus \{s\}$ takes to update its slot allocations from s in step P1b, denoted time(P1b), accumulated over all peers—so, $(n-1) \times$ time(P1b). This bid induces additional computation on the peer sites $S \setminus \{s\}$, however, to create monitoring requests per the Amnesia protocol, to fulfill the allocation each receives in this bidding step. We denote the time to generate one monitoring request in this protocol as time(AmnesiaReqGen), and accumulate this time per monitoring slot that s allocates, i.e., s.cap $\times$ time(AmnesiaReqGen) in total. Note that this time is invariant to the actual bids or how they are computed, and so is best attributed to Amnesia itself. We report the ratio of "bidding time" to the "Amnesia time", or in other words

$$\frac{\text{time(P2b)} + (n-1) \times \text{time(P1b)}}{s.\text{cap} \times \text{time(AmnesiaReqGen)}} \tag{13}$$

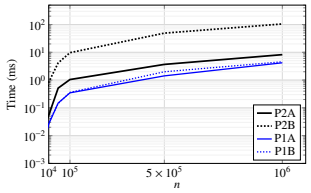
We used the implementation of the Amnesia protocol due to Wang et al. [84]. To minimize time(AmnesiaReqGen), we conservatively set the number of honeywords monitored per account to 16, the lowest number they reported, and adopted the remaining recommended parameters from their work. We implemented our proportional bidding strategy in Python, and compiled performance-critical functions such as the computation in Eqn. (1) to machine code using NUMBA. We conducted experiments on a single machine running Ubuntu 22.04.5 LTS, with an Intel Xeon Gold 6226 processor (2.7 GHz), 768 GB of RAM, and a fixed configuration of two threads.

Fig. 7a shows the values of Eqn. (13) as a function of s.cap and n. As illustrated by the very small values, the timing cost of our bidding algorithm is overwhelmed by the time for Amnesia to generate monitoring requests in response to our bids. That is, the timing costs of our proportional algorithm are a tiny fraction of the time needed to deploy monitoring requests in total. By contrast, exhaustive bidding dominates the time to generate monitoring requests; e.g., the analog of Eqn. (13) for exhaustive bidding is > 7.18 for $n = 5$ and s.cap $= 10$. For completeness, in Fig. 7b we show the times for all four steps of proportional bidding, ignoring time attributable to Amnesia.

While proportional bidding is very efficient, we highlight that bidding and monitoring-request deployment are

n \ s.cap	10^1	10^2	10^3	10^4	10^5
10^5	0.0484	0.0166	0.0135	0.0132	0.0131
10^4	0.0042	0.0014	0.0011	0.0011	0.0011
10^3	0.0013	0.0004	0.0003	0.0003	0.0003
10^2	0.0006	0.0002	0.0002	0.0002	0.0002
10^1	0.0043	0.0006	0.0002	0.0002	0.0002

(a) Ratio of bidding time to monitoring request generation time (Eqn. (13)). Values < 1.0 mean bidding is cheaper.



(b) Time to compute bids (P2a and P2b) and cumulative time for peers to process bids (P1a and P1b).

Figure 7: Performance of proportional bidding

2. A naive implementation would cause P2b to scale with sites and users; App. C shows how we reduce this to sites only.

not the most important performance costs in a breach-detection ecosystem, as they are not costs that the adversary can induce (in our threat model). In contrast, the adversary *can* induce the generation of monitoring responses by making login attempts, though our bidding algorithm plays no role in these costs; we refer the reader to Wang et al. [84, Sec. 6.5] for a discussion of these costs for Amnesia.

7. Discussion

7.1. Community Formation

Our framework relies on the formation of a community of sites that are motivated to monitor for one another, due to the tendency for users to reuse the same or similar passwords across the community. Prior work has shown that password reuse is shaped by shared characteristics among sites, such as function (e.g., shopping or email [82]), affiliation (e.g., universities [87]), geographic location [6], [61], and security posture [78], [33]. Trusted third parties could bootstrap such communities. For example, Information Sharing and Analysis Centers already coordinate threat-intelligence exchange per industry [65] and carefully vet their members [74].

A site might still prefer to confirm that it shares users with members of a community before joining, particularly if it doubts meaningful password reuse with them. Shared users could be confirmed by computing a psi/psica over user accounts, and password reuse could be estimated using tools like PassREfinder [47]. We stress, however, that the utility of our approach depends on sites joining such communities liberally; sites that do not participate in collective monitoring—the status quo today—create blind spots that attackers can exploit by stuffing stolen credentials at them.

Having joined a community, a site need only perform psi/psica computations with community members to start bidding. Presumably a site will wish to periodically update these computations, though this need not be frequent since user overlap between sites evolves slowly. Industry data from 2023 shows annual user churn rates of just 3.5–6.9% across sectors [75], suggesting that most user bases remain stable year-to-year. So, psi/psica computations for updating shared users are only needed occasionally. When they are, updateable psi schemes [58] could help minimize overhead.

7.2. Enforcing ecosystem-wide parameters

As discussed in §3, we restrict s^*.cutline = *false* and s^*.foresight = 0 to limit the risk reduction an exhaustive s^* achieves over a proportional s^*, dissuading a proportional s^* from switching strategies. To achieve this in practice, we observe a s^* cannot choose when to bid (s^*.cutline = *false*) if the next bidder is assigned outside its control, and when placing its bid, it cannot predict subsequent bidders (s^*.foresight = 0) if that bidder is assigned randomly (and only after s^* has either placed its bid or been eliminated from bidding due to its delay). These requirements can be met if the next bidder is assigned using a *randomness beacon*, which produces random values from a specified domain (in our case, the participating sites) at predictable times.

Randomness beacons are deployed with a range of trust assumptions, commonly to support blockchain applications. For example, NIST's randomness beacon [46] requires consumers of its random values to trust NIST, whereas DFINITY [35] and drand [80] offer distributed implementations using threshold signatures that prevent a small fraction of participants from biasing or learning the next random value before honest participants [60], [10]. These designs require setup of the participants with secret-shares of the signing key by a trusted party; while this requirement can be alleviated using a distributed key-generation protocol (e.g., [34], [45]), such protocols add costs and complexity. So, more recent designs avoid using threshold signing altogether (e.g., [49], [3], [17], [31], [21], [4], [16], [7]). Though even a trusted beacon might be reasonable to support collaboration among a group of sites as we propose, we are agnostic to the particular implementation used.

Enforcing a.foresight = 0, as done in sections 4.4 and 5, is a bit more complex. The reason is that an attacker could conceivably learn the assigned bidder from the randomness beacon even before the assigned bidder learns it has been chosen, resulting in a.foresight = 1. This possibility can be assumed away if the participants implement the randomness beacon among themselves and do not share their foreknowledge of the next bidder with the attacker. Alternatively, the randomness beacon could emit a cryptographic commitment to each random value, in place of the random value itself, and then transmit the random value (and anything else needed to open the commitment) to *only* the next bidder that it specifies, using a *receiver-anonymous* channel that hides the intended recipient from all but that party [71]. The selected bidder could then forward this random value with its next bid, revealing to others that it is, in fact, the chosen bidder, but eliminating the opportunity for the attacker to learn this fact before the bid is already in transit.

8. Conclusion

In this paper, we proposed a novel algorithm to incentivize the exchange of monitoring favors among sites. We systematically explored a parameter space specifying defender-defender and attacker-defender interactions and used model checking to conservatively estimate the security offered by our algorithm. We found that sites of varying popularity and resource constraints are incentivized to increase the monitoring they provide to *others* to improve their chance of detecting their *own* credential database breach. We further validated our design through simulations informed by a real breached dataset capturing realistic user overlap and password reuse patterns. We expect that, if deployed, our algorithm will enable a self sustaining credential database breach detection ecosystem, and could serve as a foundation for other cooperative security applications.

References

[1] M. Abdallah, P. Naghizadeh, A. R. Hota, T. Cason, S. Bagchi, and S. Sundaram. Behavioral and game-theoretic security investments in interdependent systems modeled by attack graphs. *IEEE Transactions on Control of Network Systems*, 7(4):1585–1596, 2020.

[2] M. Abdallah, D. Woods, P. Naghizadeh, I. Khalil, T. Cason, S. Sundaram, and S. Bagchi. Tasharok: Using mechanism design for enhancing security resource allocation in interdependent systems. In *IEEE Symposium on Security and Privacy*, pages 249–266, 2022.

[3] I. Abraham, P. Jovanovic, M. Maller, S. Meiklejogn, G. Stern, and A. Tomescu. Reaching consensus for asynchronous distributed key generation. In *40th ACM Symposium on Principles of Distributed Computing*, pages 363–373, 2021.

[4] I. Abraham, P. Jovanovic, M. Maller, S. Meiklejohn, and G. Stern. Bingo: Adaptivity and asynchrony in verifiable secret sharing and distributed key generation. In *Advances in Cryptology – CRYPTO 2023*, volume 14081 of *Lecture Notes in Computer Science*, August 2023.

[5] Akshima, D. Chang, A. Goel, S. Mishra, and S. K. Sanadhya. Generation of secure and reliable honeywords, preventing false detection. *IEEE Transactions on Dependable and Secure Computing*, 16(5):757–769, 2019.

[6] M. AlSabah, G. Oligeri, and R. Riley. Your culture is in your password: An analysis of a demographically-diverse password dataset. *Computers & Security*, 77:427–441, 2018.

[7] A. Bandarupalli, A. Bhat, S. Bagchi, A. Kate, and M. K. Reiter. Random beacons in Monte Carlo: Efficient asynchronous random beacon without threshold cryptography. In *31st ACM Conference on Computer and Communications Security*, 2024.

[8] B. Birnbaum, N. R. Devanur, and L. Xiao. Distributed algorithms via gradient descent for Fisher markets. In *12th ACM Conference on Electronic Commerce*, pages 127–136, 2011.

[9] S. Brânzei, N. Devanur, and Y. Rabani. Proportional dynamics in exchange economies. In *22nd ACM Conference on Economics and Computation*, pages 180–201, 2021.

[10] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography. In *19th ACM Symposium on Principles of Distributed Computing*, 2000.

[11] N. Chakraborty, J. Li, V. C. M. Leung, S. Mondal, Y. Pan, C. Luo, and M. Mukherjee. Honeyword-based authentication techniques for protecting passwords: A survey. *ACM Computing Surveys*, 55:1–37, 2022.

[12] Y. K. Cheung, R. Cole, and Y. Tao. Dynamics of distributed updating in Fisher markets. In *19th ACM Conference on Economics and Computation*, pages 351–368, 2018.

[13] C. Cimpanu. 23,600 hacked databases have leaked from a defunct 'data breach index' site. https://www.zdnet.com/article/23600-hacked-databases-have-leaked-from-a-defunct-data-breach-index-site/, November 2020.

[14] B. Cohen. Incentives build robustness in BitTorrent. http://bittorrent.org/bittorrentecon.pdf, May 2003.

[15] A. Das, J. Bonneau, M. Caesar, N. Borisov, and X. Wang. The tangled web of password reuse. In *21st ISOC Network and Distributed System Security Symposium*, 2014.

[16] S. Das, Z. Xiang, L. Kokoris-Kogias, and L. Ren. Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling. In *32nd USENIX Security Symposium*, August 2023.

[17] S. Das, T. Yurek, Z. Xiang, A. Miller, L. Kokoris-Kogias, and L. Ren. Practical asynchronous distributed key generation. In *43rd IEEE Symposium on Security and Privacy*, pages 2518–2534, 2022.

[18] DATAtab Team. Kruskal-Wallis-test. https://datatab.net/tutorial/kruskal-wallis-test.

[19] A. Davidson and C. Cid. An efficient toolkit for computing private set operations. In *22nd Australasian Conference on Information Security and Privacy*, volume 10343 of *Lecture Notes in Computer Science*, pages 261–278, July 2017.

[20] E. De Cristofaro, P. Gasti, and G. Tsudik. Fast and private computation of cardinality of set intersection and union. In *11th International Conference on Cryptology and Network Security*, volume 7712 of *Lecture Notes in Computer Science*, pages 218–231, 2012.

[21] L. F. de Souza, P. Kuznetsov, and A. Tonkikh. Distributed randomness from approximate agreement. In *36th International Conference on Distributed Computing*, October 2022.

[22] J. DeBlasio, S. Savage, G. M. Voelker, and A. C. Snoeren. Tripwire: Inferring internet site compromise. In *17th Internet Measurement Conference*, pages 341–354, 2017.

[23] S. K. Debnath and R. Dutta. Secure and efficient private set intersection cardinality using Bloom filter. In *18th International Conference on Information Security*, volume 9290 of *Lecture Notes in Computer Science*, pages 209–226, September 2015.

[24] A. Dionysiou, V. Vassiliades, and E. Athanasopoulos. Honeygen: generating honeywords using representation learning. In *16th ACM Symposium on Information, Computer and Communications Security*, 2021.

[25] C. Duma, M. Karresand, N. Shahmehri, and G. Caronni. A trust-aware, P2P-based overlay for intrusion detection. In *17th International Workshop on Database and Expert Systems Applications*, pages 692–697, 2006.

[26] R. Egert, M. Fischlin, D. Gens, S. Jacob, M. Senker, and J. Tillmanns. Privately computing set-union and set-intersection cardinality via Bloom filters. In *20th Australasian Conference on Information Security and Privacy*, volume 9144 of *Lecture Notes in Computer Science*, 2015.

[27] D. Endler. How much data was leaked to cybercriminals in 2020 — and what they're doing with it. https://www.forbes.com/councils/forbestechcouncil/2021/04/20/how-much-data-was-leaked-to-cybercriminals-in-2020---and-what-theyre-doing-with-it/, April 2021.

[28] I. Erguler. Achieving flatness: Selecting the honeywords from existing user passwords. *IEEE Transactions on Parallel and Distributed Systems*, 13(2), 2016.

[29] C. Fung. *Design and Management of Collaborative Intrusion Detection Networks*. PhD thesis, University of Waterloo, 2013.

[30] C. J. Fung and Q. Zhu. FACID: A trust-based collaborative decision framework for intrusion detection networks. *Ad Hoc Networks*, 53:17–31, 2016.

[31] Y. Gao, Y. Lu, Z. Lu, Q. Tang, J. Xu, and Z. Zhang. Efficient asynchronous Byzantine agreement without private setups. In *42nd IEEE International Conference on Distributed Computing Systems*, pages 246–257, July 2022.

[32] B. Garbinato and I. Rickebusch. Impossibility results on fair exchange. In *10th International Conference on Innovative Internet Community Systems*, pages 507–518, 2010.

[33] S. Gaw and E. W. Felten. Password management strategies for online accounts. In *2nd Symposium on Usable Privacy and Security*, pages 44–55, 2006.

[34] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20:51–83, 2007.

[35] T. Hanke, M. Movahedi, and D. Williams. DFINITY technology overview series, consensus system. arXiv:1805.04548 [cs.DC], 2018.

[36] A. R. Hota, A. A. Clements, S. Bagchi, and S. Sundaram. A game-theoretic framework for securing interdependent assets in networks. In *Game Theory for Security and Risk Management*, pages 157–184. Springer, 2018.

[37] Z. Huang, L. Bauer, and M. K. Reiter. The impact of exposed passwords on honeyword efficacy. In *33rd USENIX Security Symposium*, August 2024.

[38] T. Hunt. Have I been pwned? https://haveibeenpwned.com.

[39] T. Hunt. Here's why [insert thing here] is not a password killer. https://www.troyhunt.com/heres-why-insert-thing-here-is-not-a-password-killer/, 05 November 2018.

[40] T. Hunt. Inside the cit0day breach collection. https://www.troyhunt.com/inside-the-cit0day-breach-collection/, November 2020.

[41] IBM. Cost of a data breach report 2024. https://www.ibm.com/reports/data-breach, 2024.

[42] R. W. Janakiraman, M. Waldvogel, and Q. Zhang. Indra: A peer-to-peer approach to network intrusion detection and prevention. In *12th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, June 2003.

[43] L. Jiang, V. Anantharam, and J. Walrand. How bad are selfish investments in network security? *IEEE/ACM Transactions on Networking*, 19(2):549–560, 2010.

[44] A. Juels and R. L. Rivest. Honeywords: Making password-cracking detectable. In *20th ACM Conference on Computer and Communications Security*, pages 145–160, 2013.

[45] A. Kate and I. Goldberg. Distributed key generation for the Internet. In *29th IEEE International Conference on Distributed Computing Systems*, June 2009.

[46] J. Kelsey, L. T. A. N. Brandão, R. Peralta, and H. Booth. A reference for randomness beacons: Format and protocol version 2. https://doi.org/10.6028/NIST.IR.8213-draft, May 2019.

[47] J. Kim, M. Song, M. Seo, Y. Jin, and S. Shin. PassREFinder: Credential stuffing risk prediction by representing password reuse between websites on a graph. In *45th IEEE Symposium on Security and Privacy*, May 2024.

[48] L. Kissner and D. Song. Privacy-preserving set operations. In *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 241–257, August 2005.

[49] E. Kokoris-Kogias, D. Malkhi, and A. Spiegelman. Asynchronous distributed key generation for computationally secure randomness, consensus, and threshold signatures. In *27th ACM Conference on Computer and Communications Security*, pages 1751–1767, November 2020.

[50] Y. Kolumbus, M. Levy, and N. Nisan. Asynchronous proportional response dynamics: Convergence in markets with adversarial scheduling. In *37th Conference on Neural Information Processing Systems*, pages 25409–25434, 2023.

[51] H. Kunreuther and G. Heal. Interdependent security. *Journal of Risk and Uncertainty*, 26:231–249, 2003.

[52] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *International Conference on Computer Aided Verification*, 2011.

[53] K. Lai, L. Rasmusson, E. Adar, L. Zhang, and B. A. Huberman. Tycoon: An implementation of a distributed, market-based resource allocation system. *Multiagent and Grid Systems*, 1(3):169–182, 2005.

[54] K. Lauter, S. Kannepalli, K. Laine, and R. C. Moreno. Password Monitor: Safeguarding passwords in Microsoft Edge. https://www.microsoft.com/en-us/research/blog/password-monitor-safeguarding-passwords-in-microsoft-edge/, 21 January 2021.

[55] M. Lelarge and J. Bolot. A local mean field analysis of security investments in networks. In *3rd Workshop on Economics of Networked Systems*, pages 25–30, 2008.

[56] R. Lemos. Credential stuffing reaches 193 billion login attempts annually. https://www.darkreading.com/cloud-security/credential-stuffing-reaches-193-billion-login-attempts-annually, 19 May 2021.

[57] D. Levin, K. LaCurts, N. Spring, and B. Bhattacharjee. BitTorrent is an auction: Analyzing and improving BitTorrent's incentives. In *ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 243–254, 2008.

[58] G. Ling, P. Tang, and W. Qiu. Efficient updatable PSI from asymmetric PSI and PSU. Cryptology ePrint Archive, Paper 2024/1712, 2024.

[59] J. Lou, A. M. Smith, and Y. Vorobeychik. Multidefender security games. *IEEE Intelligent Systems*, 32(1):50–60, 2017.

[60] D. Malkhi and M. K. Reiter. An architecture for survivable coordination in large distributed systems. *IEEE Transactions on Knowledge and Data Engineering*, 12(2), March/April 2000.

[61] P. Mayer, J. Kirchner, and M. Volkamer. A second look at password composition policies in the wild: Comparing samples from 2010 and 2016. In *13thSymposium on Usable Privacy and Security*, pages 13–28, 2017.

[62] P. Mayer, C. W. Munyendo, M. L. Mazurek, and A. J. Aviv. Why users (don't) use password managers at a large educational institution. August 2022.

[63] R. A. Miura-Ko, B. Yolken, J. Mitchell, and N. Bambos. Security decision-making among interdependent organizations. In *21st IEEE Computer Security Foundations Symposium*, pages 66–80, 2008.

[64] K. C. Nguyen, T. Alpcan, and T. Basar. Stochastic games for security in networks with interdependent nodes. In *1st International Conference on Game Theory for Networks*, pages 697–703, 2009.

[65] National Council of ISACs. About isacs. https://www.nationalisacs.org/about-isacs, 2025.

[66] OneCloud. What is the average response time to detect a cyber breach in 2024? https://www.onecloud.com.au/resources/what-is-the-average-response-time-to-detect-a-cyber-breach-in-2024/, 4 September 2024.

[67] P. S. Oruganti, P. Naghizadeh, and Q. Ahmed. The impact of network design interventions on the security of interdependent systems. *IEEE Transactions on Control of Network Systems*, 11(1):173–184, 2023.

[68] H. Pagnia and F. C. Gärtner. On the impossibility of fair exchange without a trusted third party. Technical Report TUD-BS-1999-02, Department of Computer Science, Darmstadt University of Technology, March 1999.

[69] B. Pal, M. Islam, M. Sanusi, N. Sullivan, L. Valenta, T. Whalen, C. Wood, T. Ristenpart, and R. Chattejee. Might I get pwned: A second generation compromised credential checking service. In *31st USENIX Security Symposium*, August 2022.

[70] S. Pearman, J. Thomas, P. E. Naeini, H. Habib, L. Bauer, N. Christin, L. F. Cranor, S. Egelman, and A. Forget. Let's go in for a closer look: Observing passwords in their natural habitat. In *24th ACM Conference on Computer and Communications Security*, October 2017.

[71] A. Pfitzmann and M. Waidner. Networks without user observability. *Computers & Security*, 6(2):158–166, April 1987.

[72] B. Pinkas, T. Schneider, and M. Zohner. Scalable private set intersection based on OT extension. *ACM Transactions on Privacy and Security*, 21(2), 2018.

[73] J. Pullman, K. Thomas, and E. Bursztein. Protect your accounts from data breaches with Password Checkup. https://security.googleblog.com/2019/02/protect-your-accounts-from-data.html, 5 February 2019.

[74] REN-ISAC. Membership. https://www.ren-isac.net/membership/membertypes.html.

[75] Recurly Research. Business churn rate by industry. https://recurly.com/research/churn-rate-benchmarks/, 2024.

[76] H. Robbins. A remark on Stirling's formula. *The American Mathematical Monthly*, 62(1):26–29, 1955.

[77] T. Sandholm and X. Wang. (Im)possibility of safe exchange mechanism design. In *18th AAAI Conference on Artificial Intelligence*, pages 338–344, 2002.

[78] E. Stobert and R. Biddle. The password life cycle. *ACM Transactions on Privacy and Security*, 21(3):1–32, 2018.

[79] R. Terry. Honey accounts explained. https://www.crowdstrike.com/en-us/cybersecurity-101/identity-protection/honey-account/, 7 January 2025.

[80] The League of Entropy. drand: A distributed randomness beacon. https://drand.cloudflare.com/, 2024. Accessed: 7 December 2024.

[81] Verizon Business. Verizon 2024 data breach investigations report. https://verizon.com/dbir, 2024.

[82] C. Wang, S. T. K. Jan, H. Hu, D. Bossart, and G. Wang. The next domino to fall: Empirical analysis of user passwords across online services. In *8th ACM Conference on Data and Application Security and Privacy*, pages 196–203, March 2018.

[83] D. Wang, Y. Zou, Q. Dong, Y. Song, and X. Huang. How to attack and generate honeywords. In *43rd IEEE Symposium on Security and Privacy*, May 2022.

[84] K. C. Wang and M. K. Reiter. Using Amnesia to detect credential database breaches. In *30th USENIX Security Symposium*, August 2021.

[85] K. C. Wang and M. K. Reiter. Bernoulli honeywords. In *31st ISOC Network and Distributed System Security Symposium*, February 2024.

[86] L. Wang, S. Noel, and S. Jajodia. Minimum-cost network hardening using attack graphs. *Computer Communications*, 29(18):3812–3824, 2006.

[87] R. Wash, E. Rader, R. Berman, and Z. Wellmer. Understanding password choices: How frequently entered passwords are re-used across websites. In *12th Symposium on Usable Privacy and Security*, June 2016.

[88] F. Wu and L. Zhang. Proportional response dynamics leads to market equilibrium. In *39th ACM Symposium on Theory of Computing*, pages 354–363, 2007.

[89] Y.-S. Wu, B. Foo, Y. Mei, and S. Bagchi. Collaborative intrusion detection system (CIDS): A framework for accurate and efficient IDS. In *19th Annual Computer Security Applications Conference*, December 2003.

[90] Y. Yang, Y.-C. Lee, P.-A. Chen, and C.-C. Lin. Robustness of online proportional response in stochastic online Fisher markets: A decentralized approach. arXiv preprint arXiv:2406.00160, 2024.

[91] V. Yegneswaran, P. Barford, and S. Jha. Global intrusion detection in the DOMINO overlay system. In *11th ISOC Network and Distributed System Security Symposium*, February 2004.

[92] L. Zhang. Proportional response dynamics in the Fisher market. *Theoretical Computer Science*, 412(24):2691–2698, 2011.

[93] Q. Zhu, C. Fung, R. Boutaba, and T. Basar. GUIDEX: A game-theoretic incentive-based mechanism for intrusion detection networks. *IEEE Journal on Selected Areas in Communications*, 30(11):2220–2230, 2012.

Appendix A. Challenges in Predicting Cost

In adapting proportional response from other domains (e.g., [57], [88], [9]) to our setting, we considered many variants of Eqn. (1) and Alg. 1. Below, we summarize two of these versions and outline why did not prefer them.

- $s^*.\mathsf{normRisk}(r')$ is a less accurate predictor of $s^*.\mathsf{normCostCum}(1)$ for sites s^* that are popular (see §4.3). In this case, the overlap between $s^*.\mathsf{users} \cap s.\mathsf{users}$ and $s^*.\mathsf{users} \cap s'.\mathsf{users}$ for other sites s, s' tends to be large, meaning that computing $s^*.\mathsf{normRisk}(r')$ as a sum of $s^*.\mathsf{normRisk}(s,k)$ for $s \in S \setminus \{s^*\}$ (see Eqn. (2)) "double counts" the large number of users in that overlap. This double counting might be avoided by calculating $s^*.\mathsf{normRisk}(r')$ holistically, using the portfolio of allocations $\{s.\mathsf{allocTo}(s^*)\}_{s \in S \setminus \{s^*\}}$, versus as a simple sum of per-site contributions. While such a "portfolio" approach is possible (at least in the psi case), we nevertheless found that attributing risk to each peer individually (i.e., $s^*.\mathsf{normRisk}(s, s.\mathsf{allocTo}(s^*))$ is particularly useful because the lever available to incentivize a peer is adjusting its individual allocation; i.e., it is useful to be able to assign blame individually, so that we can incentivize each peer individually.
- We explored various other measures to predict $s^*.\mathsf{normCostCum}(1)$, besides $s^*.\mathsf{normRisk}(r')$. Most were measures of *utility*, expressed as a function of desired allocations of slots from each $s \in S \setminus \{s^*\}$ (itself computed using $|s^*.\mathsf{users} \cap s.\mathsf{users}|$) and the actual allocation of slots from s. Measures of utility that grow linearly the allocation from s suffered from the fact that incrementing or decrementing allocations tended to affect $s^*.\mathsf{normCostCum}(1)$ much more (respectively, less) if the allocation was already small (respectively, large). Nonlinear functions that we considered introduced additional tuning parameters that we found difficult to fit to the myriad other parameter settings we explored in our model-checking experiments.

An important direction for future work is therefore to refine $s^*.\mathsf{normRisk}(r')$ to better predict $s^*.\mathsf{normCostCum}(1)$ when s^* is popular. Perhaps an even greater challenge is to improve $s^*.\mathsf{normRisk}(r')$ to better predict $s^*.\mathsf{normCostCum}(1)$ in the psica setting or when $a.\mathsf{aggression}$ is low. Both scenarios involve an inherent information imbalance: in the former, s^* lacks knowledge of shared users across peers, while in the latter, it lacks insight into the attacker's aggression.

Appendix B. Cit0day Data Exploration

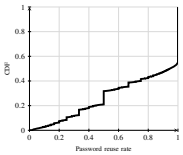
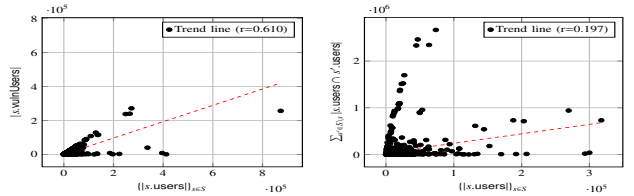


Figure 8: Password reuse rate

The processed Cit0day dataset includes 74,268,368 users across 7,914 sites and 53,241,884 unique passwords. Site sizes vary widely (see Fig. 9c). However, a site's popularity does not directly translate to greater risk. From the site's own perspective, the number of users it shares with other sites—its only observable signal of stuffing risk—correlates only weakly with site size (r = 0.197; Fig. 9b). This suggests that larger sites do not necessarily have a proportionally higher number of users with accounts elsewhere. In contrast, the actual number of vulnerable users (those who reuse passwords across sites) shows only a moderate correlation with site size (r = 0.610; Fig. 9a), indicating that overall risk does not scale directly with popularity.

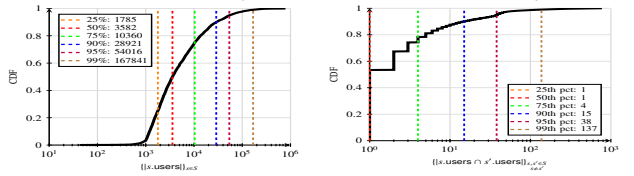
Figs. 9c and 9d show heavy-tailed distributions: most sites are small, and most site pairs share few users. Still,

password reuse is rampant among users with multiple accounts. Among the 9.3% of site pairs that share users, reuse is nearly universal; the median reuse rate in Fig. 8 is 94.5%.



(a) Total vs. capturable users. We randomly sample 1,000 sites to illustrate trends clearly.

(b) Total vs. shared users. We randomly sample 1,000 sites to illustrate trends clearly.



(c) Users per site

(d) Shared users per site

Figure 9: Exploration of Cit0day Dataset

Appendix C. Performance Optimizations

To place a bid according to P2b, s must determine $s.\mathsf{avgRisk}(s')$ from all $s' \in S \setminus \{s\}$. However, calculating $s.\mathsf{avgRisk}(s')$ involves computing the optimal number f of stuffing attempts by an attacker which maximizes Eqn. (1) assuming $k = s'.\mathsf{allocTo}(s)$. A naive implementation could iterate over all the possible values of f, which is upper-bounded by $n' = |s.\mathsf{users} \cap s'.\mathsf{users}|$. However, this will quickly be computationally prohibitive as the number of users increases. Instead, we observe Eqn. (1) is unimodal, and the optimal f is either $\lfloor (n' - \min\{n', k\})/(\min\{n', k\}+1) \rfloor$ or $\lceil (n' - \min\{n', k\})/(\min\{n', k\}+1) \rceil$ in the psi setting and either $\lfloor (|s'.\mathsf{users}| - \min\{|s'.\mathsf{users}|, k\})/(\min\{|s'.\mathsf{users}|, k\}+1) \rfloor$ or $\lceil (|s'.\mathsf{users}| - \min\{|s'.\mathsf{users}|, k\})/(\min\{|s'.\mathsf{users}|, k\}+1) \rceil$ in the psica setting. §C.1 contains the proof.

Despite reducing the number of calculations to compute Eqn. (1), evaluating $s.\mathsf{avgRisk}(s')$ still scales with $O(|s.\mathsf{users}|)$. To compute it in constant time, we use Stirling's approximation of log factorials. §C.2 provides the error terms. Combined, these two optimizations let us estimate Eqn. (1) in $O(1)$ time, implying $s.\mathsf{avgRisk}(s')$ in P2b is also calculated in $O(1)$ time. Therefore, computing a bid according to P2b only depends on the number of bids that s is receiving, which scales with the number of sites, $O(n)$.

C.1. Eqn. (1) is unimodal

We prove the psi case; psica is analogous. Eqn. (1) in the psi setting is:

$$s.\mathsf{risk}(s', k) = \max_{0 \le f \le n'} f \times \frac{\binom{n'-f}{m}}{\binom{n'}{m}} \tag{14}$$

where $m = \min\{n', k\}$. To find the maximum, we check when $\mathbf{E}\left(\mathbb{G}_{f+1}\right) > \mathbf{E}\left(\mathbb{G}_f\right)$:

$$\frac{\mathbf{E}\left(\mathbb{G}_{f+1}\right)}{\mathbf{E}\left(\mathbb{G}_f\right)} > 1 \iff \frac{f+1}{f} \cdot \frac{\binom{n'-f-1}{m}}{\binom{n'-f}{m}} > 1 \tag{15}$$

$$\iff \frac{f+1}{f} \cdot \frac{n'-f-m}{n'-f} > 1 \tag{16}$$

$$\iff f < \frac{n'-m}{m+1} \tag{17}$$

Therefore, the maximum occurs at $\lfloor (n'-m)/(m+1) \rfloor$ or $\lceil (n'-m)/(m+1) \rceil$.

C.2. Stirling approximation of log factorial error term

We derive the error bounds for approximating Eqn. (3) (psi); the derivation for Eqn. (4) (psica) is analogous. When $k > n'$ and $f = 0$, Eqn. (3) evaluates to 1. When $k > n'$ and $f > 0$, Eqn. (3) evaluates to 0. The interesting case is when $k \le n'$. We have:

$$\frac{\binom{n'-f}{k}}{\binom{n'}{k}} = \exp\begin{pmatrix} \ln((n'-f)!) + \ln((n'-k)!) \\ -\ln((n'-f-k)!) - \ln(n'!) \end{pmatrix} \tag{18}$$

Let S denote the value of Eqn. (18). Using Stirling's approximation $\ln(g!) \approx g\ln(g) - g + \frac{1}{2}\ln(2\pi g) + \varepsilon_g$ with Robbins bounds $\frac{1}{12g+1} < \varepsilon_g < \frac{1}{12g}$ [76], we can approximate Eqn. (18) by $S \cdot \exp(E_{\text{total}})$ where:

$$E_{\text{total}} = \varepsilon_{n'-f} + \varepsilon_{n'-k} - \varepsilon_{n'-f-k} - \varepsilon_{n'} \tag{19}$$

The error bounds are:

$$E^{\text{lower}}_{\text{total}} = \frac{1}{12(n'-f)+1} + \frac{1}{12(n'-k)+1} - \frac{1}{12(n'-f-k)} - \frac{1}{12n'} \tag{20}$$

$$E^{\text{upper}}_{\text{total}} = \frac{1}{12(n'-f)} + \frac{1}{12(n'-k)} - \frac{1}{12(n'-f-k)+1} - \frac{1}{12n'+1} \tag{21}$$

Therefore:

$$\frac{\binom{n'-f}{k}}{\binom{n'}{k}} \in \left[S \cdot \exp(E^{\text{lower}}_{\text{total}}), S \cdot \exp(E^{\text{upper}}_{\text{total}})\right] \tag{22}$$

Appendix D. Model Checking Implementation

In the inter-organization model (§3), s^*'s goal is to minimize its risk (Eqn. (1)). Assuming the s^*'s credential database is breached, the attacker's objective (§4) is to harvest as many users as possible by stuffing the s^* 's compromised passwords at its peers. We model each of these interactions as Markov Decision Processes (MDPs) and use

probabilistic model checking to analyze them, providing a worst-case assessment of s^*'s security.

Probabilistic model checking requires exhaustively searching the entire state space, which imposes significant computational and memory constraints. As a result, off-the-shelf tools like PRISM [52] struggle to scale to the number of sites and users in our analysis (sections 3.6, 4.3, and 4.4). To address this, we developed custom model checkers in Python with application-specific optimizations, detailed below. We validated our implementations by comparing results from small-scale experiments—limited in site and user count and excluding combinations of parameters that PRISM cannot handle—against PRISM's output.

D.1. Inter-organization Model Implementation

A state in the inter-organization model consists of (1) the bid number, (2) the average number of slots each site has received from its peers, (3) the latest bid each site has placed for s^*. If s^*.cutline = True, we additionally store the number of bids placed by $s \in S \setminus \{s^*\}$, since this determines which of s^*'s peers is slated to bid next. Additionally, if slack $< \infty$, we maintain an n-sized array that tracks how many more bids ahead each site has placed relative to its peers; each element in this array is bounded by slack.

To implement the exhaustive strategy when it bids, the s^*_e computes the allocation that minimizes s^*.risk over s^*_e.foresight + s^*_e.lookahead future steps. Our model-checker uses breadth-first search (BFS) to build a tree of this depth, with each node stores the state, as described above. Within the s^*_e.foresight depth, child nodes correspond to the next bidder specified in the bidding sequence and, if s^*_e.cutline = *true*, the s^* assuming it had not bid previously. Beyond s^*_e.foresight, nodes branch on all valid next bidders: if s^*_e.cutline = *false*, this includes every $s \in S$; if s^*_e.cutline = *true* it includes every $s \in S \setminus \{s^*\}$, and s^*_e, provided it was not the previous bidder. When s^*_e bids, nodes further branch on its possible allocations; to control state explosion, our implementation only allows s^*_e to allocate s^*.cap in fixed, tunable increments.

We cap the tree depth at s^*_e.foresight + s^*_e.lookahead to reflect the realistic assumption that s^* cannot predict bids indefinitely, and to bound memory usage during BFS. Since PRISM lacks support to easily restrict tree depth below auction length, r, we validate our model checker against PRISM only in cases where s^*_e.foresight+ s^*_e.lookahead = r.

After building the tree, we back-propagate optimal bids and cumulative risk from leaves to root, yielding s^*_e's best move. During this process, we cache the optimal rewards and policies of nodes at depths (s^*_e.foresight, s^*_e.foresight + s^*_e.lookahead] for reuse in later bids or bidding sequences, assuming all other parameters remain fixed.

D.2. Attacker Model Implementation

A state in the attacker model consists of (1) the bid number, (2) each $s \in S \setminus \{s^*\}$ allocation to s^* for that round, (3) the attacker's current stuffing strategy—i.e., which users in s^*.users are stuffed and at which $s \in S \setminus \{s^*\}$, and (4) the attacker's cumulative dodge probability.

To implement worst-case attacker, our model checker uses depth-first search (DFS) to build a tree of depth a.foresight + a.lookahead, enumerating all attacker strategies feasible within a.aggression. We use DFS, in lieu of BFS, due to memory-constraints imposed by the explosive number of attacker strategies. Within the a.foresight depth, the model checker generates attacker stuffing strategies assuming allocations corresponding to the next bidder specified in the bidding sequence. Beyond a.foresight, the model checker also branches on the possible next bidders, as described in §D.1, generating attacker strategies that satisfies a.aggression assuming the largest number of slots due to possible bids. To reduce the number of nodes generated in the tree, we cache nodes with few stuffing attempts, as these are likely to satisfy the a.aggression and thus be regenerated at multiple tree depths. For fixed $k = s^*$.allocTo(s), we also bound stuffing attempts per site using the unimodal optimization in §C.1.

Still, enumerating the attacker's stuffing strategies to depth a.foresight + a.lookahead is computationally prohibitive. To this end, we underspecify the attacker's stuffing strategy by only the number of stuffing attempts per $s \in S \setminus \{s^*\}$. Not all such underspecified strategies correspond to a valid set of uncaptured users in s^*.users, so we incrementally filter out invalid strategies as determined by a constraint solver.

We backpropagate from tree leaves to root, to compute the optimal underspecified strategy—per-site stuffing counts—over the a.foresight +a.lookahead depth. To concretize this strategy with valid users, we retroactively search for a subset of uncaptured users in s^*.users that satisfies the strategy. To narrow this recursive search, we observe an attacker prioritizes stuffing users with fewer accounts to preserve future stuffing options.

Even enumerating underspecified strategies becomes computationally prohibitive as the number of sites and users grows, since suboptimal per-site stuffing counts (at a given depth) may yield high payoff later if s^*.allocTo(s) decreases at deeper levels. However, when a.foresight+a.lookahead = 1, as in §4.4, the attacker's goal is to immediately maximize expected user captures. In this case, our model checker iterates over site indices to find the maximal underspecified per-site stuffing strategy, subject to §C.1 and a.aggression. Since this maximal strategy may not correspond to a valid subset of uncaptured users in s^*.users, we track both (1) the best valid strategy found so far and (2) invalid strategies that yield higher payoff but currently violate constraints. These invalid strategies act as upper bounds that may later become valid. If no such invalid strategies remain, we conclude the optimal per-site stuffing strategy has been found.