A co-design adaptive defense scheme with bounded security damages against Heartbleed-like attacks

Zhisheng Hu, Ping Chen*, *Member, IEEE*, Minghui Zhu, *Member, IEEE* and Peng Liu, *Member, IEEE*

***Abstract*—This paper proposes a co-design adaptive defense scheme against a class of zero-day buffer over-read attacks that follow unknown stationary probability distributions. In particular, the co-design scheme integrates an improved UCB algorithm and a customized server. The improved UCB algorithm adaptively allocates guard pages on a heap based on induced damage of the guard pages so as to minimize the accumulated damage over time. The security damages of the improved UCB algorithm are proven to be always below a temporal bound without knowing which attack is launched when the buffer allocation follows a certain stationary probability distribution. Then an efficient server modification is introduced to randomly allocate buffers. Moreover, the damages of our scheme asymptotically converge to those of the optimal defense policy where the launched attacks and their distributions are known in advance. Further, the co-design scheme is evaluated with several real-world Heartbleed attacks. The experiment results demonstrate the validity of the upper bound and show that the adaptive defense is effective against all the attacks of interest with runtime overheads as low as 5%.**

***Index Terms*—Adaptive Defense, Buffer Over-read Attacks, Reinforcement Learning**

I. Introduction

BUFFER over-read attacks [1]–[3] are unauthorized reads which go beyond the boundaries of vulnerable buffers. Many buffer over-read attacks are zero-day attacks since the locations of vulnerable buffers are usually unknown to security analysts. Also, buffer over-read attacks are difficult to detect because they do not tamper any content in the memory. As a result, zero-day buffer over-read attacks can produce severe damages to servers. In August 2014, 4.5 million patient records are stolen from Community Health System by manipulating Heartbleed [4]. And according to [5], [6], 24-55% of HTTPS servers in the Alexa Top 1 Million were initially vulnerable to Heartbleed, allowing theft of the servers' private keys and users' session cookies and passwords.

A variety of techniques have been proposed to deal with buffer over-read attacks. The techniques can be classified into two categories according to their requirements of the knowledge of vulnerabilities. The first category of techniques need to identify the vulnerabilities; i.e., the locations of vulnerable buffers, and then generate tailored defenses. One typical example is patching [7]. However, the process of patching is very time-consuming. According to the report from Google Project Zero team [8], across all vendors, it takes 15 days on average to patch a vulnerability that is being used in active attacks. Complementary defenses should be deployed to mitigate zero-day attacks and before any patches are generated. The second category of techniques do not need to identify the vulnerabilities in advance. Instead, they deploy memory safety retrofitting or preliminary defense actions; e.g., guard pages, to increase the difficulty for the attacker to succeed. For instance, DieHard [9] applies randomization and replication to heap allocation. Large distances between buffers make it more difficult for the attacker to over-read valid data. HeapTherapy [10] places inaccessible guard pages randomly throughout the heap space to prevent over-reads accessing undesired area. The second category of techniques can partially mitigate zero-day buffer over-read attacks. In this paper, we focus on zero-day over-read attacks and the second category of techniques.

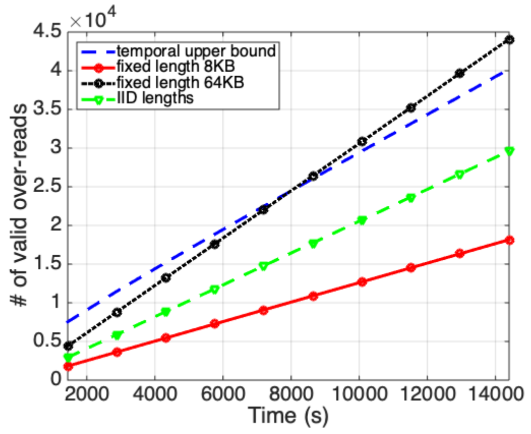


Fig. 1. The security damages under the protection of DieHard against different Heartbleed attacks.

As mentioned, buffer over-read attacks do not tamper the content in the memory. Therefore, security analysts usually do not know when the attacks start or end. However, we can evaluate the defense techniques afterwards by analyzing their temporal security damages. Fig. 1 shows the temporal security damages (the x axis indicates the time after the attacks are launched) under the protection of DieHard in terms of the number of valid over-reads. Here valid over-reads refer to the Heartbleed requests that read valid buffer content. DieHard is

This work was supported in part by ARO W911NF-13-1-0421 (MURI), NSA H98230-15-1-0289, NSF ECCS-1710859, NSF CNS-1814679, NSF CNS-2019340 and NSF CNS-1422594.

* corresponding author.

Z. Hu is with Baidu Security, Sunnyvale, CA 94089. E-mail: zhishenghu@baidu.com.

P. Chen is with Institute for Big Data, Fudan University Shanghai, China 200433. E-mail: pchen@fudan.edu.cn.

M. Zhu is with the School of Electrical Engineering and Computer Science, Pennsylvania State University, University Park, PA 16802. E-mail: muz16@psu.edu.

P. Liu is with the College of Information Sciences and Technology, Pennsylvania State University, University Park, PA 16802. E-mail: pliu@ist.psu.edu.

able to mitigate a particular Heartbleed attack [11] where each Heartbleed request has a fixed over-read length as 8KB (the red solid line with circle markers). However, when the over-read lengths are slightly modified, the security damages are much more severe. This is because DieHard allocates multiple miniheaps for the buffers with fixed size and the largest distance between two buffers is also fixed. If the attacker over-reads more than the largest distance, valid data objects can be read. In particular, the green dashed line with triangle markers represents a mutated attack where the over-read lengths of Heartbleed requests are independent and identically distributed (IID) and their distributions are uniform over 4KB to 64KB. The black dotted line with circle markers represents another mutated attack which fixes the over-read length to 64KB for all Heartbleed requests. It can be seen that the security damages after 10,000 seconds increase from 12,500 to 21,000 to 31,000 valid over-reads for three attacks. Similarly, the mutant attacks significantly increase the damages at other time instances. In real world, the attacker can easily possess a larger set of attack scripts by modifying the existing attack scripts as shown in the example. Since the adopted attack script is unknown, it is difficult for the defender to guarantee the security damages of its defenses.

The above observation motivates a question: Can we design a defense under which its temporal security damages are always below a temporal bound; e.g., the blue dashed line in Fig. 1, against a class of zero-day buffer over-read attack scripts without knowing which script is actually used by the attacker? In this paper, we only focus on a class of mutants of Heartbleed attacks and refer the class of attacks as Heartbleed-like attacks. The Heartbleed-like attacks have two salient features. First, the attacker probes victim systems with a large number of requests during a relatively long period. Second, the attack actions of each attack are IID random variables. In addition, the Heartbleed-like attacks only read contiguous buffers. The Heartbleed-like attacks include the original Heartbleed attack [12], [13] as a special case. In particular, the original Heartbleed attack typically requires hundreds of thousands of requests and could last hours. And the original Heartbleed attack adopts fixed over-read lengths, a special probability distribution. To the authors' best knowledge, the question remains open. The problem is challenging because 1. the locations of the vulnerable buffers are unknown; 2. the attack actions; e.g., over-read lengths, are unknown.

To address the above two challenges, we propose a new co-design scheme which integrates an improved UCB (Upper Confidence Bound [14]) algorithm, named UCB-Z (UCB zero-day) algorithm, and allocator customization. The defender uses guard pages; i.e., the inaccessible area on the heap, as the preliminary defense actions. When an over-read request touches a deployed guard page, the defender can observe a segmentation fault that indicates the over-read request is blocked. Deploying guard pages induces availability damage of the server. Then the defender follows the UCB-Z algorithm to deploy guard pages at certain locations and reduce security damages.

In particular, the UCB-Z algorithm proceeds in defense cycles. At the beginning of each defense cycle, the defender subtracts the number of guard pages from the number of segmentation faults to quantify the instant damage mitigated by the action deployed in the last defense cycle. The algorithm then maintains the empirical average mitigated damage for each defense action over time. Then the defender, on the one hand, exploits the defense action with the highest average mitigated damage, and on the other hand, explores other defense actions. The defender balances between the exploitation and exploration to minimize accumulated damage over time. To implement the algorithm, the defender only needs to access induced segmentation faults after the actions are taken but is not required to know the cause of the segmentation faults; e.g., the portion of over-read memory. This information is hard to gather when defending zero-day over-read attacks. Compared to the classic UCB algorithm [14], the UCB-Z algorithm takes many issues in real-world servers (e.g., communication errors or transmission delays between the defender and the server) into account. As mentioned, the attack can send a large number of Heartbleed requests within a relatively long time. Repeated interactions between the attack and the server provide the UCB-Z algorithm sufficient information and time to observe, learn and adapt.

We prove that the UCB-Z algorithm provides a temporal damage upper bound which asymptotically converges to the damage of the optimal defense policy even when observed segmentation faults are subject to delays and errors. This upper bound is valid for the class of attacks when the attack and buffer allocation follow certain stationary probability distributions, respectively. The random allocation of buffers is motivated by the feature that some existing allocators; e.g., HeapAlloc [15], [16], allocate part of buffers at random addresses on the heap. However, the randomness of heap object allocation is limited by memory alignment enforced in operating systems [17]. For example, HeapAlloc always allocates buffers larger than 512KB into a separate heap block. Therefore, directly implementing the UCB-Z algorithm on the current servers cannot guarantee the damage bound. Another issue is that segmentation faults brought by guard pages usually crash the servers. These two issues necessitate server modification. Our customized allocator provides two properties. First, all buffers, including vulnerable buffers, are allocated on the heap by a given fixed distribution. Second, the allocator is equipped with a signal handler to guarantee that the system can operate normally without being interrupted by segmentation faults.

Contribution. In this paper, we develop a co-design scheme which integrates the UCB-Z algorithm and a customized allocator to defend against the Heartbleed-like attacks. We prove that the security damages of UCB-Z are always below a temporal bound without knowing which attack in the class is launched. Moreover, the explicit form of the upper bound is derived and its dependency on the unknown distribution of over-read lengths is highlighted. The upper bound also indicates that the security damages of UCB-Z asymptotically converge to those of the optimal defense policy where the distribution of over-read lengths is known in advance. We introduce an efficient server modification to randomly allocate

Notation	Meaning
a,$\mathcal{A}$, Θ	attack action, attacker's action space, probability distributions space over $\mathcal{A}$
$\mathbb{P}_\theta \in \Theta$	a specific Heartbleed-like attack: a probability distribution of over-read lengths
p_i, H, L, M	page, heap, size upper bound of buffers, number upper bound of alive buffers
$\vec{SA}$, $\mathbb{P}_{SA}$, Ω_{SA}	starting addresses of buffers, probability distribution of starting addresses, space of all possible starting addresses
d, $\mathcal{D}$	defense action, defense action space
$u(d,a,\vec{SA})$, $c(d)$, $u(d,a,\vec{SA})$	effectiveness function, cost function, utility function
$h(t)$, $\mathcal{T}_i$	the number of the utility values received at t, utility delay
$\epsilon(t)$, ϵ	utility error, utility error upper bound
$\vec{u'}(t)$	received utility values at defense cycle t
φ, $D_\varphi(\mathbb{P}_\theta, N)$, $\eta(t)$	defense algorithm, induced security dam-age up to defense cycle N, information set of the defender
$\mathbf{1}_{\{\Pi\}}$, $T_d(t)$	indicator function, number of times defense action d has been chosen by defense cycle $t-1$
$\bar{\mu}'_d(t)$	*empirical average utility* of defense action d by the end of defense cycle $t-1$
$I_d(t)$	*upper confidence index* of defense action d at the beginning of defense cycle t
$BR(\mathbb{P}_\theta)$, $BR_{2\epsilon}(\mathbb{P}_\theta)$	best response to the attack $\mathbb{P}_\theta$, 2ϵ best response
μ_{min}	the smallest expected utility value when the defender adopts the best response
$\Delta(\mathbb{P}_\theta, d)$	sub-optimality of defense action a against attack $\mathbb{P}_\theta$
Δ_{min}	the smallest sub-optimality when the defender adopts the 2ϵ best response

TABLE I
SYMBOLS AND NOTATIONS.

buffers and handle segmentation faults. We evaluate the co-design scheme with several real-world Heartbleed attacks. The experiment results demonstrate the validity of the upper bound and show that the adaptive defense is effective against all the attacks of interest with runtime overheads as low as 5%.

The remainder of the paper is organized as follows. Section II introduces the attack model and formulates the security problem of interest as an optimal sequential decision-making problem. Section III gives an overview of our adaptive defense's design and provides the analytical results of the algorithm. Section IV describes the implementation issues of our adaptive defense. Section V presents the evaluation results. Lastly, the paper is concluded in Section VI.

II. Background and Problem Statement

In this section, we introduce the background of Heartbleed attack and the formal statement of the problem of interest. For convenient reference, the symbols and notations used in this paper are summarized in Table I.

A. Heartbleed-like Attacks

The Transport Layer Security (TLS) protocol maintains communication links between a client and a server by allowing the client to send Heartbleed requests to the server. As shown in Figure 2, each Heartbleed request message consists of a "message type", a "payload's length", a "payload" and a "padding". When a server receives a Heartbleed request, it stores the "payload" in a heap buffer, and then replies to the client with the "payload". Because of missing bound checking in the handling of the Heartbleed request, the server replies the message to the attacker based on the "payload's length" information in the Heartbleed request, regardless how long the "payload" actually is. So the message returned by the server may include not only the heap buffer storing "payload", but also whatever else in the subsequent memory chunk. The buffer storing "payload" is referred to as the vulnerable buffer. The attack keeps sending the victim server system with a large amount of requests at a constant frequency to steal sensitive information.

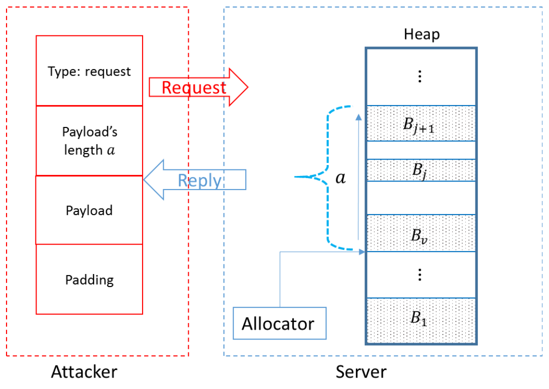


Fig. 2. A pair of Heartbleed request and reply.

B. Problem Statement

We formally state the security problem of interest depicted in Fig. 3. In particular, an attacker keeps sending requests at a constant frequency to a victim server, and a defender periodically adjusts defense actions on the basis of feedbacks to achieve damage upper bound. The uniform time interval between two defense adjustments is denoted as a defense cycle. In what follows, we first introduce the models of the attacker, server and defender. Afterwards, we introduce the notion of utility to quantify the security damages of the server. Then we formulate the security problem to upper bound the aggregate damage over a defense horizon.

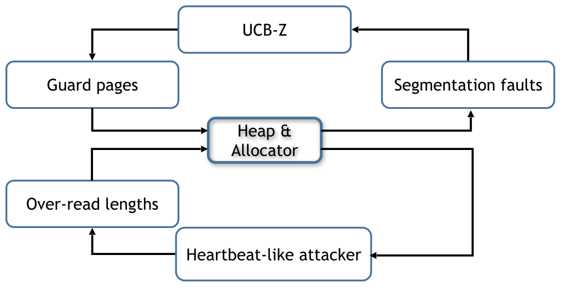


Fig. 3. The interactions among the attacker, heap and defender.

1) Attack Model: In this paper, we focus on a class of mutant Heartbleed attacks that are extended from the original Heartbleed attack. This class of attacks are characterized by an extended ability that the over-read lengths in each attack are selected by an unknown (to the defender) but fixed probability distribution. This extension describes some possible features of unknown Heartbleed-like attacks and includes the Heartbleed attack as a special case. To increase the probability of reading sensitive information without being detected, the attacker would like to change its attack actions. Using predetermined probability distributions is a reasonable and implementation-friendly choice for the attacker to adjust its actions in real world. For example, the Heartbleed attack tools in [18] can build Heartbeat requests with random payload's lengths.

We study the security problem from defender's point of view. Each attack action a is a vector of over-read lengths of the Heartbleed requests sent in a defense cycle. It induces the action space $\mathcal{A}$ for the attacker. In addition, the space of all probability distributions over $\mathcal{A}$ is denoted by Θ and each attack is defined as a probability distribution $\mathbb{P}_\theta \in \Theta$. Notice that the original Heartbleed adopts fixed over-read lengths, which is included in Θ.

2) Server: Heartbleed-like attacks impact the heap and allocator in the server. Heap is the portion of memory where dynamically allocated buffers reside [19]. The heap is partitioned into pages; i.e., $H \triangleq \{p_1 \cdots, p_{\mathcal{M}}\}$, where p_1 is the page with the lowest address. Based on our observation [20], the sizes of all buffers on the heap are uniformly upper bounded, where the upper bound is denoted by L. Also based on our observation [20], the numbers of alive buffers during different defense cycles are uniformly upper bounded, where the upper bound is denoted by M. All the buffers are allocated by a modified allocator at random addresses on the heap with a uniform probability distribution $\mathbb{P}_{SA}$. Here uniform allocation refers to that all the buffers are uniformly allocated over all the possible permutations of M addresses drawn from the heap. All the possible permutations of M addresses consist of the space of the starting address Ω_{SA}. For ease of presentation, it is assumed that the numbers of buffers at different defense cycles are constant M. If not, zeroed buffers are appended. The starting addresses of the buffers are denoted by a random vector $\vec{SA}$ taking values in Ω_{SA}.

The above uniform allocation is an extension of partial random starting addresses; i.e., part of buffers are allocated at random addresses, in existing allocators such as HeapAlloc [16]. We will show that the uniform allocation introduces a mathematical property that is necessary to derive the damage upper bound in Section III-B. And the modified allocator can be achieved by the efficient sever modification shown in Section III-C.

3) Defense Action: Some preliminary defenses can increase the difficulty of reading valid data or block Heartbleed requests without knowing the vulnerable buffers. In this paper, the defender uses guard pages as preliminary defenses. The defender is equipped with a set of defense actions denoted by $\mathcal{D}$, where each defense action $d = [p_{g_1}, \cdots, p_{g_K}]$ is to insert a certain number of guard pages at certain locations on the heap. Guard pages do not belong to any buffers, and thus normal write or read operations do not touch these guard pages. However, Heartbleed requests may read beyond buffer boundaries and touch the guard pages. If so, segmentation faults are triggered [21], [22]. Intuitively, if guard pages do not incur any cost, then the best solution is placing guard pages after all buffers. However, The study of Dynamic Buffer Overflow Containment (DYBOC) [23] shows that the cost is mainly induced by deploying guard pages. In fact, there are a lot of small size buffers in a process. If we set a guard page for every buffer, the overhead will be unacceptable.

4) Utility: When facing buffer over-read attacks, security analysts usually prefer detections of the attacks. Here detections refer to processed information; e.g., what sensitive data is read and how many read bytes are out of a buffer, that can confirm the impacts of the attacks. However, in our security problem, such information is unavailable during vulnerability windows since we focus on zero-day over-read attacks. As a result, we only require the defender to access feedbacks such as raw alerts in intrusion detection systems [24] or segmentation faults. Feedbacks can indicate the joint effect of the defense and attack and can be directly observed by the defender.

Given any pair of defense and attack actions, the server returns the number of segmentation faults as feedbacks to the defender. Formally, feedback is defined as utility in the form of $u(d, a, \vec{SA}) = W_r r(d, a, \vec{SA}) - W_c c(d)$, where $r(d, a, \vec{SA})$ is the number of segmentation faults to quantify the effectiveness and $c(d)$ is the number of guard pages to quantify the cost. As mentioned, the overhead result from DYBOC justifies that the number of guard pages can be used to quantify the cost of defense action d. The cost can be viewed as part of the availability damage of the server. And the constant weights W_r and W_c are chosen according to the preference of the defender on security and efficiency. The utility value $u(t) = u(d(t), a(t), \vec{SA}(t))$ quantifies the mitigated damage at a particular defense cycle t. The utility values are uniformly bounded, and there are u^- and u^+ such that $\forall t$, $u(t) \in [u^-, u^+]$. Additionally, the defender knows the bounds.

Recall that the attack action a and the starting addresses of the buffers $\vec{SA}$ are two independent random vectors. Hence for each d, $r(d, a, \vec{SA})$ is a discrete random variable derived from a and $\vec{SA}$, which is formally described as the following property. **Property M-1**: For each random vector attack a with probability distribution $\mathbb{P}_\theta$ and the starting addresses $\vec{SA}$ with probability distribution $\mathbb{P}_{SA}$, the induced utility for each defense $d \in \mathcal{D}$ is a discrete random variable with expected value $\mathbb{E}_{\mathbb{P}_{\mathbb{P}_\theta}, \mathbb{P}_{SA}} u(d, a, \vec{SA}) \triangleq \sum_{(a,\vec{SA}) \in \mathcal{A} \times \Omega_{SA}} u(d, a, \vec{SA}) \mathbb{P}_\theta(a) \mathbb{P}_{SA}(\vec{SA})$.

When the server load is heavy, the server might not be able to generate the utility value at the end of the defense cycle. Besides, the communication between the defender and the server may be affected by transmission delays [25]. Therefore, at the end of defense cycle t, the defender may receive utility values of previous defense cycles instead of the current one $u(t)$. More formally, for any defense cycle t, the defender receives several utility values from the server which are denoted by the vector $\vec{u}(t) = \begin{bmatrix} u(t-\mathcal{T}_1) & u(t-\mathcal{T}_2) & \cdots & u(t-\mathcal{T}_{h(t)}) \end{bmatrix}^T$, where $\mathcal{T}_1, \cdots, \mathcal{T}_{h(t)}$ are multiples of defense cycles, representing the delays, and $h(t)$ is the nonnegative number of the utility values received by the defender at t. The defender only receives the utility value of each defense cycle once. The utility delays are uniformly upper bounded by $\mathcal{T}$; i.e., $\forall t, \forall i \in \{1, \cdots, h(t)\}, \mathcal{T}_i \leq \mathcal{T}$.

Since the segmentation faults could be triggered by operations other than Heartbleed requests, the received utility values could be corrupted by errors. We denote the utility error for each defense cycle t as $\epsilon(t)$. The utility errors are uniformly upper bounded by ϵ; i.e., $\forall t, |\epsilon(t)| \leq \epsilon$. Then for defense cycle t, the defender receives several utility values with errors from the server which are denoted by vector $\vec{u}'(t) = \begin{bmatrix} u'(t-\mathcal{T}_1) & \cdots & u'(t-\mathcal{T}_{h(t)}) \end{bmatrix}^T = \begin{bmatrix} u(t-\mathcal{T}_1)+\epsilon(t-\mathcal{T}_1) & \cdots & u(t-\mathcal{T}_{h(t)})+\epsilon(t-\mathcal{T}_{h(t)}) \end{bmatrix}^T$.

5) Problem Formulation: Given an attack $\mathbb{P}_\theta \in \Theta$ and a defense algorithm φ, the induced security damage up to defense cycle N is defined as $D_\varphi(\mathbb{P}_\theta, N) \triangleq -\sum_{t=1}^{N}\sum_{j=1}^{h(t)} \mathbb{E}\left(u(t-\mathcal{T}_j)+\epsilon(t-\mathcal{T}_j)\right)$. Here φ is a mapping from defender's information set $\eta(t)$ to its defense action; i.e., $d(t) = \varphi(\eta(t))$, where $\eta(t) \triangleq \{\vec{u}'(1), \cdots, \vec{u}'(t-1), d(1), \cdots, d(t-1)\}$ denotes the information set of the defender at the beginning of defense cycle t. In this paper, we aim to design a defense algorithm φ so as to tightly upper bound the temporal security damage $D_\varphi(\mathbb{P}_\theta, N)$ against all possible attacks in Θ.

III. Defense Design

In this section, we first propose a defense algorithm, namely UCB-Z, to address the security problem formulated in Section II-B5. Then we prove that the damage of the UCB-Z algorithm has a temporal upper bound. At last, we provide the design of server modification to meet the randomness property of buffer allocation.

A. UCB-Z Algorithm

Our defense is deployed before details of zero-day buffer over-read attacks; e.g., locations of vulnerable buffers and attack actions, are revealed. The limited knowledge about zero-day attacks prevents the defender from choosing effective defenses. However, the long-running feature of the Heartbleed-like attacks; i.e., the attacks probe the victim server with a large amount of requests, provides opportunities for the defender to evaluate the cost-effectiveness of previously taken defense actions based on received utility values, identify latent patterns of ongoing zero-day attacks, and adjust defense actions to minimize security damages. This key observation motivates us to extend UCB algorithms in stochastic Multi-armed Bandit problems [26], [27] which are particularly well suited to problems where the decision-maker interacts with partially unknown and stationary random environments, but can evaluate their actions via observed feedbacks, and gradually identify optimal actions during the course of repeated interactions with the environments.

Next, we would like to extend UCB algorithms to the UCB-Z algorithm which can solve the problem in Section II-B5. Before that, let us introduce a set of notations which are needed in the algorithm.

- $\mathbf{1}_{\{\Pi\}}$ is an indicator function: $\mathbf{1}_{\{\Pi\}} = 1$ if Π is true and $\mathbf{1}_{\{\Pi\}} = 0$ otherwise.
- $T_d(t) = \sum_{\tau=1}^{t-1} \mathbf{1}_{\{d(\tau)=d\}}$ is the number of times defense action d has been chosen by defense cycle $t-1$.
- $\forall d \in \mathcal{D}$, $\bar{\mu}'_d(t) = \frac{\sum_{\tau=1}^{t-1}\sum_{i=1}^{h(\tau)} \left(u'(\tau-\mathcal{T}_i)\mathbf{1}_{\{d(\tau-\mathcal{T}_i)=d\}}\right)}{\sum_{\tau=1}^{t-1}\sum_{i=1}^{h(\tau)} \mathbf{1}_{\{d(\tau-\mathcal{T}_i)=d\}}}$ represents the *empirical average utility* of defense action d by the end of defense cycle $t-1$.
- $\forall d \in \mathcal{D}$, $I_d(t) = \left(\bar{\mu}'_d(t) + (u^+ - u^-)\sqrt{\frac{2\ln(t)}{T_d(t)}}\right)$ represents the *upper confidence index* of defense action d at the beginning of defense cycle t.

Algorithm 1: The UCB-Z Algorithm
```
1  Initialization: for $d \in \mathcal{D}$ do
2      $T_d(1) = 0$;
3      $\bar{\mu}'_d(1) = 0$;
4  Loop: for $t = 1; t \leq N; t++$ do
5      for $d \in \mathcal{D}$ do
6          if $T_d(t) == 0$ then
7              $I_d(t) = +\infty$;
8          else
9              $I_d(t) = \bar{\mu}'_d(t) + (u^+ - u^-)\sqrt{\frac{2\ln(t)}{T_d(t)}}$;
10     $d(t) = \arg\max_{d \in \mathcal{D}} I_d(t)$;
11     $T_{d(t)}(t+1) = T_{d(t)}(t) + 1$;
12     for $d \in \mathcal{D} \setminus \{d(t)\}$ do
13         $T_d(t+1) = T_d(t)$
14     Defender receives $\vec{u}'(t)$;
15     for $d \in \mathcal{D}$ do
16         $\bar{\mu}'_d(t+1) = \frac{\sum_{\tau=1}^{t}\sum_{i=1}^{h(\tau)} (u'(\tau-\mathcal{T}_i)\mathbf{1}_{\{d(\tau-\mathcal{T}_i)=d\}})}{\sum_{\tau=1}^{t}\sum_{i=1}^{h(\tau)} \mathbf{1}_{\{d(\tau-\mathcal{T}_i)=d\}}}$;
```

The main idea of the UCB-Z algorithm is described as follows. By Property M-1, the true cost-effectiveness of a defense action can be reflected by its expected utility value. By the law of large number, it is known that the empirical average utility value converges to the expected utility value when the defense action is chosen infinite often. The UCB-Z algorithm uses empirical average utility value (the first term of $I_d(t)$) to evaluate how well a defense action works. On the other hand, a penalty term (the second term of $I_d(t)$) is added to ensure that each defense action is chosen infinite often. In particular, the penalty term for a particular defense action d keeps increasing and dominates the $I_d(t)$ if the action is not chosen for a long time. As a result, defense action d is chosen again when $I_d(t)$ becomes largest among all defense actions.

The pseudocode in Algorithm 1 formalizes the above idea. In particular, at the beginning of defense cycle t, the defender updates the upper confidence index $I_d(t)$ of each defense action (Line 5 to 9): For the actions that have never been chosen, the algorithm sets their indices to be positive infinity[1] (Line 6 to 7). For the actions that have been chosen, the defender updates their indices based on their perturbed empirical average utility values (Line 8 to 9). Then the defender executes action $d(t)$ with the largest index (Line 10) and updates the number of times each defense action has been chosen (Line 11 to 13). At the end of defense cycle t, the defender receives the utility values $\vec{u}'(t)$ (Line 14) and then updates the empirical average utility values of all defense actions. (Line 15 to 16).

B. Mathematical Analysis

In this section, we will prove that the UCB-Z algorithm can provide a temporal damage upper bound even when the received utility values are subject to finite delays and errors if the induced utility values follow **Property M-1**.

The following theorem states that the temporal security damages caused by the UCB-Z algorithm are always below the derived upper bound no matter which unknown distribution the attacker follows. Before the theorem statement, let us introduce several notations first. We define the best response of the defender as $BR(\mathbb{P}_\theta) \triangleq \arg\max_{d\in\mathcal{D}} \mathbb{E}_{\mathbb{P}_{\mathbb{P}_\theta},\mathbb{P}_{SA}} u(d,a,\vec{SA})$. Let $\mu_{\min} \triangleq \min_{\mathbb{P}_\theta\in\Theta} \mathbb{E}_{\mathbb{P}_{\mathbb{P}_\theta},\mathbb{P}_{SA}} u(BR(\mathbb{P}_\theta),a,\vec{SA})$; i.e., the smallest expected utility value when the defender adopts the best response, and let $\mu_{\max} \triangleq \max_{\mathbb{P}_\theta\in\Theta} \mathbb{E}_{\mathbb{P}_{\mathbb{P}_\theta},\mathbb{P}_{SA}} u(BR(\mathbb{P}_\theta),a,\vec{SA})$; i.e., the largest expected utility value when the defender adopts the best response. We also define $\Delta(\mathbb{P}_\theta,d) \triangleq \mathbb{E}_{\mathbb{P}_{\mathbb{P}_\theta},\mathbb{P}_{SA}} u(BR(\mathbb{P}_\theta),a,\vec{SA}) - \mathbb{E}_{\mathbb{P}_{\mathbb{P}_\theta},\mathbb{P}_{SA}} u(d,a,\vec{SA}) \geq 0$ which quantifies the sub-optimality of defense action $d \in \mathcal{D}$ against attack $\mathbb{P}_\theta$, and $BR_{2\epsilon}(\mathbb{P}_\theta) \triangleq \arg\min_{d\in\{d'\in\mathcal{D}|\Delta(\mathbb{P}_\theta,d')>2\epsilon\}} \Delta(\mathbb{P}_\theta,d)$ is the 2ϵ best response; i.e., the best defense action whose expected utility value is at least 2ϵ smaller than that of the best response. Then $\Delta_{min} \triangleq \min_{\mathbb{P}_\theta\in\Theta}\{\Delta(\mathbb{P}_\theta, BR_{2\epsilon}(\mathbb{P}_\theta))\}$ is the smallest sub-optimality when the defender adopts the 2ϵ best response.

Theorem 1. *Under the* **Property M-1**, *for any attack* $\mathbb{P}_\theta \in \Theta$, *the aggregate expected damage of the UCB-Z algorithm after* N *defense cycles is upper bounded in the following way:*

$$\begin{aligned}
&D_{UCB-Z}(\mathbb{P}_\theta, N)\\
&\leq -N\mu_{\min} + \mathcal{T}\mu_{\max} + 3|\mathcal{D}|(N-\mathcal{T})\epsilon\\
&+ 8(u^+ - u^-)^2|\mathcal{D}|\frac{(u^+ - u^- + \epsilon)\ln(N-\mathcal{T}+1)}{(\Delta_{min}-2\epsilon)^2}\\
&+ \left(1+\frac{\pi^2}{3}\right)|\mathcal{D}|(u^+ - u^- + \epsilon).
\end{aligned}$$

The proof of Theorem 1 is given in Appendix A. Theorem 1 formally shows that the UCB-Z algorithm has a security guarantee against any attack in Θ. This guarantee does not require the defender to know which attack distribution is used in advance. More importantly, the theorem indicates that, as the defense horizon increases, the security damage of the UCB-Z algorithm gets closer to that of the optimal policy which knows the attack distribution. We evaluate the asymptotic optimality in Section V-B and give the formal justification of the asymptotic optimality in Appendix A-C.

[1]In implementation, their indices are set to be a number much larger than others'.

C. Server Modification

The upper bound in Theorem 1 is valid when **Property M-1** is true. To satisfy this property, we will modify the server such that all the buffers are allocated at random addresses on the heap. This server modification does not incur much performance efficiency loss. And the implementation details will be elaborated in Section IV.

In particular, we would like to modify the server so that the allocator in the server could allocate all buffers, including vulnerable buffers, on the heap uniformly. To achieve this requirement, we divide the heap into a fixed number of chunks where each chunk occupies L pages of memory (L is determined by the largest buffer). At the beginning of a defense cycle, the allocator retrieves all the buffers from the heap, randomly chooses M chunks, and then distributes old and newly incoming buffers one by one into M chunks. If the number of buffers in the defense cycle is less than M, zeroed buffers are appended. The uniform distribution allocation is used in many real-world allocators such as HeapAlloc [15], [16]. If a buffer is freed during one defense cycle, its chunk will not be assigned to other buffers in this cycle. In addition, if a buffer is freed and reallocated during one defense cycle, it will be treated as a newly incoming buffer and allocated to a different chunk.

Recall that guard pages are inaccessible area on the heap and can trigger segmentation faults. And segmentation faults will lead to process crashes by default (no handle). So we need to make guard pages difficult to be accessed by normal operations and enable the server to continue work when segmentation faults are triggered. In particular, an additional page, which is referred as to guard page candidate, is appended after each chunk. This extra page will not be used to store any buffer content. Therefore segmentation faults are only triggered when invalid access to guard page candidates takes place. We also implement a signal handler which will skip the specific load/store instruction to avoid process crashes when segmentation faults are triggered. The implementation details will be illustrated in Section IV.

IV. Memory Allocator Modification

To allocate all buffers, including vulnerable buffers, on the heap by the aforementioned distribution, we define a specific heap object structure and provide heap management, including initialization, object allocation and object deallocation. In addition, a signal handler is inserted to the target program to guarantee that the system can adaptively defend against the attack without interruption. Based on our evaluation (see Section V), the memory allocator introduces low runtime overhead and acceptable memory overhead.

A. Heap Object Structure

For each heap object, an additional 4KB page (*guard page candidate*) is allocated immediately after the object. Note the

4KB page can be accessed before it is set to *guard page*; e.g., `mprotect` in Linux. At the beginning of each defense cycle, a defense action (some guard page candidates are set as guard pages) will be selected by the UCB-Z algorithm. While at the end of a defense cycle, guard pages will be set back to guard page candidates.

B. Initialization

During the initialization phase, a large area of free memory is obtained from the system using `mmap`, which will be treated as the heap by our allocator. The heap is divided into a number of chunks where each chunk occupies a fixed size memory, including a 4KB guard page candidate. Instead of maintaining metadata in each chunk, the allocator maintains a bitmap, which indicates the status of each chunk: allocated (marked as 1) and freed (marked as 0). In our current implementation, the bitmap is a buffer allocated on the heap. To protect the bitmap from being over-read or over-written by the attacker, guard pages are set before and after the bitmap. To protect the bitmap from being accessed by discrete reads or writes by the attacker, the bitmap is allocated to another location on the heap every defense cycle. As such, it is non-trivial for the attacker to successfully locate the bitmap.

C. Object Allocation

When a memory allocation request arrives, the allocator first checks whether the request is asking for a large object (larger than 16KB). If so, a method named `allocateLargeObject` will complete the task by using `mmap`. The address is stored in a table for our allocator to deallocate it. Otherwise, the allocator processes the request with method `allocateSmallObject`. This method first randomly selects a freed heap chunk (the one that has not been occupied), then it returns the address of that chunk to the request, and updates the bitmap by resetting the bit for the allocated heap chunk to be 1. This method only consumes constant time.

D. Object Deallocation

To deallocate an object, the allocator checks whether the request is for a large object (larger than 16KB). If so, `freeLargeObject` will complete the task. Otherwise, it invokes a method named `freeSmallObject`, which simply finds the corresponding bit of the heap chunk in the bitmap, but does not set the bit to 0 until the end of a defense cycle. This method also consumes constant time.

E. Segmentation Fault Handler

To guarantee that the system can adaptively defend against the attack without interruption, a segmentation fault handler is added in the target program; e.g., Apache-2.2.14. When a segmentation fault signal is received, the signal handler will skip the specific load/store instruction; i.e., `F3 A5 rep movs dword ptr [edi],dword ptr [esi]` in memcpy, and make the web server continuously run. It is worth mentioning that reading uninitialized memory can be intentional (for example, entropy creation) or accidental (for example, the result of a programming error). The former should not be performed by benign users, as undefined behavior provides no guarantees of introducing entropy and will quite possibly have opposite effect. The latter can be exploited by attackers to read the contents of memory to identify memory locations that can subsequently be used to bypass defenses like address space layout randomization or leak other secrets [28]. When uninitialized reads happen, the handler will catch the exception and trap into the handler function. The handler function will find the position of the instruction following the copy function and return to that instruction. For example, the attacker leverages the `memcpy` function to do the uninitialized reads. The instruction next to `memcpy` is `printf`, the segmentation fault handler will skip the data copy operation and invoke `printf` directly.

V. Evaluation

There are two attractive features of our co-design defense scheme. First, it can provide upper bound on security damage against the Heartbleed-like attacks without knowing which attack is launched. Second, the customized server has low runtime overhead and acceptable memory overhead. We are going to use real-world experiments to validate the two features in this section.

A. Experiment Environment

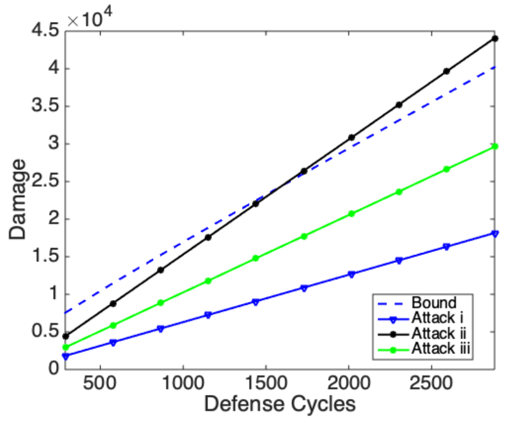
The experiments are conducted on an Intel i7-4770 processor (8M Cache, up to 3.90 GHz) with 8GB physical RAM running Fedora Core Release 8 with Linux kernel version 2.6.23.1. We use Apache (2.2.14) with OpenSSL (1.0.1c) as the vulnerable web server. We customize a memory allocator according to Section III-C and implement the UCB-Z algorithm. In order to test the effectiveness of data protection under Heartbleed-like attacks, we mimic benign users to access the web server. Each benign user's request contains the HTTP headers, which contain valid data such as the login credentials and a cookie. In the following experiments, the length of a defense cycle is 5 seconds, and during one defense cycle, the Heartbleed requests start from the vulnerable buffer `pl` on the heap. Based on our observation, the largest buffer allocation in Apache is 3,672 bytes long, which spans less than 4 pages; i.e., $L = 4$, and the largest numbers of buffers existing in Apache is less than 3,000; i.e., $M = 3,000$. On the attacker side, we launch three long-running Heartbleed-like attack scripts based on an open source attack [11]. Each attack script launches 10 Heartbleed requests per defense cycle and lasts for 4 hours. The over-read lengths for each attack script are IID random variables varying from 2 to 16 pages based on some predetermined distribution shown as follows.

- *Attack i*: The attacker sends all requests with 2-page over-read length.
- *Attack ii*: The attacker sends all requests with 16-page over-read length.
- *Attack iii*: The over-read length of each request is an IID random variable that takes value from $\{2, 4, 6, 8, 10, 12, 14, 16\}$ pages with equal probability.

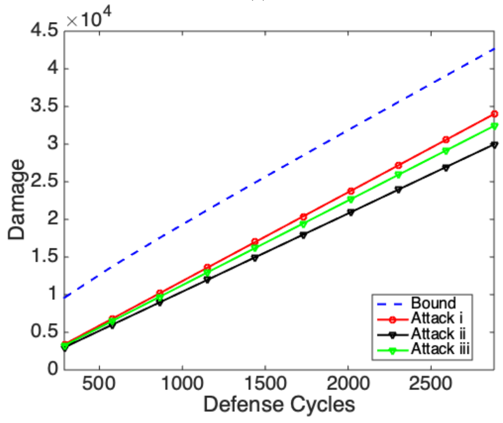
This class of Heartbleed attack scripts include the original Heartbleed attacks as a special cases where fixed over-read lengths are chosen with probability one.

B. Effectiveness & Upper Bound

We first show that our co-design defense scheme is effective against three real-world Heartbleed-like attacks by comparing the temporal security damages with those of the peer techniques after all the three attacks are launched. We choose DieHard and HeapTherapy, which are the best Heartbleed defenses we have found so far. Then we show that the security damage upper bound is valid against the attacks.



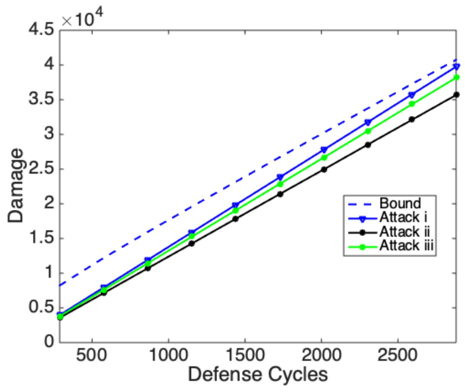
(a)



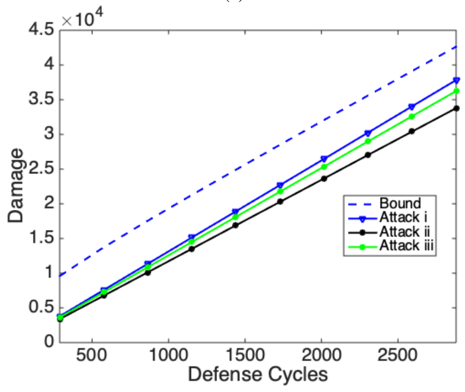
(b)

Fig. 4. (a) compares the temporal damages of DieHard under three attacks with the mathematical upper bound and (b) compares the temporal damages of the UCB-Z algorithm under the same attacks with the mathematical upper bound.

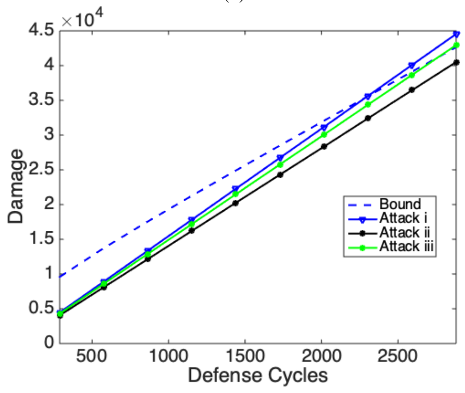
Comparison with DieHard. It should be noticed that our adaptive defense focuses on memory over-read but DieHard does not. In particular, DieHard applies randomization and replication to heap allocation and provides fault-tolerance. DieHard increases the chances of turning malicious memory errors into benign errors. DieHard allocates multiple miniheaps for the buffers with one size. In addition, due to overprovisioning (by a factor of F), the biggest distance between two buffers are at most $F * N * S$, where N is the number of allocated buffers with the same size, and S is the size of each buffer in the miniheap. Large distances between buffers prevent out-of-bounds reads from achieving valid data.



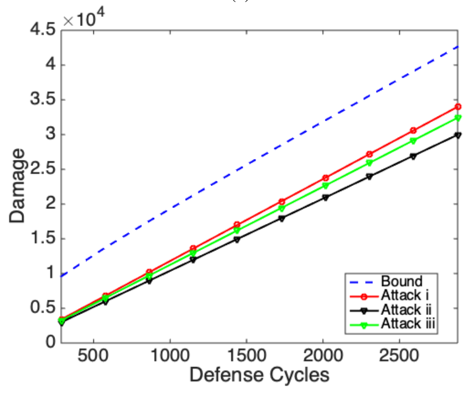
(a)



(b)



(c)



(d)

Fig. 5. (a) - (c) compares the temporal damages of HeapTherapy under three attacks with the mathematical upper bound when $P_m = 0.1$, $P_m = 0.13$ and $P_m = 0.15$ respectively; (d) compares the temporal damages of the UCB-Z algorithm under the same attacks with the mathematical upper bound.

Since DieHard does not have guard pages, it will not trigger segmentation faults. Then we slightly change the utility function so we can get measurements for DieHard. In particular, we use the number of failed over-read requests; i.e., the Heartbleed requests that cannot read any valid data, in each defense cycle as the reward. And we use the overprovisioning factor F as the cost for DieHard since it introduces redundant memory consumption. It is worth mentioning that although the attacker cannot distinguish leaked data from random data, they can use pattern match to find sensitive data/objects. DieHard can fill the memory with some invalid data/objects, which may mislead the attacker from getting some accurate data/object. However, the attacker can still get suspicious sensitive data/object and use them to try to attack.

The comparison results are shown in Fig. 4. The UCB-Z works well for all three attacks and never violates the upper bound in Theorem 1 while DieHard only works well for *Attack i*. Under our testing environment, N is 663 on average and the default value of F is 2. So the biggest distance between two buffers is 2*663*4=5,304 bytes. Once the attacker reads over 5,304 bytes, it will always read the valid data. For *Attack ii* and *Attack iii*, there are high probabilities that the requests read over 5,304 bytes. Therefore the performance of DieHard can drop dramatically with slightly changed attacks. On the other hand, Fig. 4 - (b) validates that the damages of UCB-Z algorithm are always below the damage bound derived from Theorem 1 under different attacks.

Comparison with HeapTherapy. HeapTherapy can handle heap over-read attacks by placing inaccessible guard pages randomly throughout the heap space. In particular, HeapTherapy randomly chooses a portion of buffers, which is denoted by a tuning parameter P_m, to add guard pages. A larger P_m offers a higher chance to detect any single buffer over-reads, but also incurs a higher overhead. And P_m is determined by the user before the program is run and kept fixed all the time. Since HeapTherapy also deploys guard pages and could trigger segmentation faults, we use the same utility function in Section II-B4 and also compare the damages of HeapTherapy with that of the UCB-Z algorithm. In this comparison, we use the same three Heartbleed attacks; i.e., *Attack i*, *Attack ii* and *Attack iii*. For HeapTherapy, we test different P_m values and compare their aggregate utility values with the upper bound in Theorem 1. The results are shown in Fig. 5. Similar to DieHard, HeapTherapy only works well for *Attack ii*. In fact, the short over-read lengths in *Attack i* and *Attack iii* could reduce the chance of HeapTherapy to detect Heartbleed requests.

TABLE II
DEFENSE COMPARISON

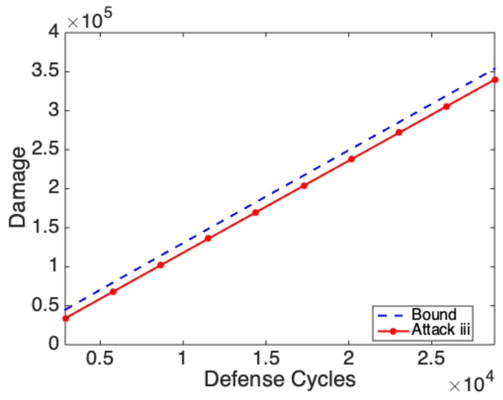
Our Method	ASLR	Diehard	CFG	
Yes	Yes	No	Yes	Performance
Yes	No	No	No	Complete

Discussion. The state-of-art memory protection-based defenses can be categorized into the following three classes: 1. Memory randomization like ASLR; 2. Memory object protection like Diehard and HeapTherapy; 3. Control Flow Object Integrate Checking like CFI. We summarize the comparison in Table II. Here performance means that the defense performance overhead is larger than 5%; and complete means that the defense can be circumvented. The comparisons between our adaptive defense with other defenses do not mean that they are less superior. From Figs 4 and 5, we can see that both DieHard and HeapTherapy provide an effective defense against their specific attack respectively. In fact, if the defender knows which attack is launched by the attacker, it can choose the defense technique that is effective; e.g., DieHard for *Attack i* or HeapTherapy for *Attack ii*. The defender can expect better performance by tailoring the parameters in their defense technique; e.g., the overprovisioning factor F in DieHard, according to the launched attack. However, when facing the attacker with a class of zero-day attacks, the defender does not know which attack is launched. And the unknown attacks, even slightly deviated attacks, can result in much worse security damage than the defense techniques expect. The comparison results in this paper are used to validate the correctness of the upper bound in Theorem 1 and provide an insightful aspect to assess the security brought by a defense system; i.e., the worst-case security guarantee quantified by the temporal damage upper bound.

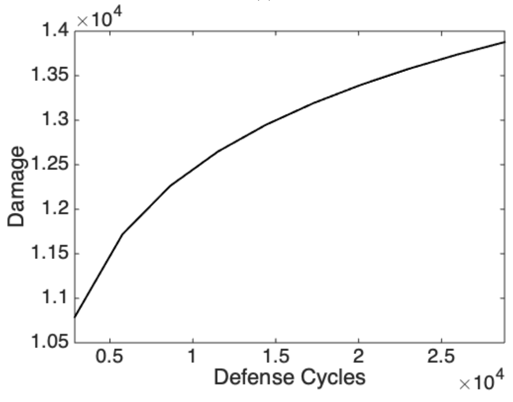
Comparison with the Optimal Policy. We further discuss how the upper bound can reflect the UCB-Z algorithm has a good worst-case security guarantee. Recall that the best response of the defender with respect to a given attack is the defense action with the highest expected utility value (defined in Section III-B). The optimal defense strategy is to stick to the best response if the defender knows the utility distributions. Here we want to compare the UCB-Z with the optimal policy with respect to *Attack iii* if the defender knows the utility distributions. Fig. 6 compares the temporal damage of the optimal policy with the upper bound in Theorem 1 when the server is under *Attack ii*. The difference between the temporal damage of the optimal policy and the upper bound increases but the difference increases slow down as time goes by. In addition, the difference is approximately 5% the size of the damage of the optimal policy after 28,800 defense cycles. As time goes by, the difference gets dominated by the temporal damage. The result indicates that upper bound is asymptotically tight in the sense that the asymptotic damage of the UCB-Z algorithm gets closer to that of the optimal policy.

C. Performance Overhead

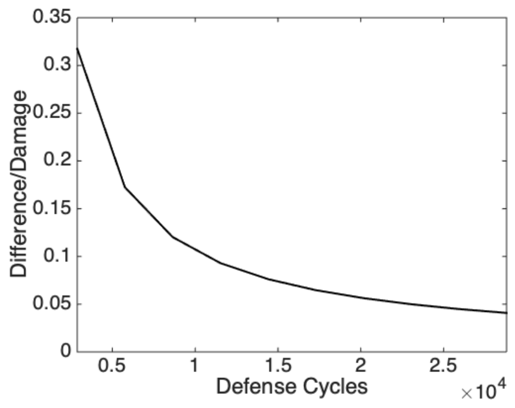
Runtime Overhead. Our allocator has three main jobs: selecting guard pages, allocating buffers and deallocating buffers. The allocation and deallocation take constant time. There are two most time-consuming parts: selecting the guard pages based on the UCB-Z algorithm at the beginning of each defense cycle and calculating the segmentation faults at the end of each defense cycle. The runtime overhead heavily depends on the length of a defense cycle. If the length is longer, the runtime overhead will be lower; otherwise, the runtime overhead will be higher. As such, we select three different defense cycles, 5 seconds, 10 seconds and 60 seconds, respectively.



(a)



(b)



(c)

Fig. 6. (a) compares the temporal damage of the optimal policy under *Attack iii* with the mathematical upper bound, (b) shows the difference between the temporal damage of the optimal policy and the mathematical upper bound and (c) presents the proportion of the difference to the damage of the optimal policy.

To evaluate the runtime overhead in a real-world environment, we mimic the benign users by using Apache benchmark (ab) to simulate 50,000,000 HTTP GET requests. Meanwhile, we also mimic the malicious users by launching the same Heartbleed script for 2 hours.

We compile OpenSSL-1.0.1c and httpd-2.2.14 with our allocator and GNU Libc [29] allocator, respectively. For the compiled Apache web server, we mimic both the benign and malicious users from two different physical machines. Then we use Apache benchmark (ab) on the third physical machine to evaluate the runtime overhead of web server. The result is shown in Table III. We can see when a defense cycle increases from 5s to 60s, the runtime overhead decreases sharply from 27.4% to 5.0%. We further evaluate the overhead brought by the segmentation fault handler. Each skip operation performed by the segmentation fault handler takes near 160 μs. One attack request costs 1.5 s on average. In a 30-seconds-defense cycle, the additional overhead brought by the segmentation fault handler is about 3.2 ms (less than 0.02%).

TABLE III
RUNTIME OVERHEAD

GNU Libc Allocator	Modified System	
	Defense Cycle	Overhead
3521.0 req/s	60s	3345.4 req/s (5.0%)
	10s	2889.8 req/s (17.9%)
	5s	2554.7 req/s (27.4%)

Memory Overhead. For each buffer, the modified allocator allocates 5 pages (including 1 guard page candidate). It will inevitably introduce memory overhead. We compare the memory used by Apache with GNU Libc Allocator and the one with modified memory allocator. For each executed Apache, we snapshot the used memory and read the `VmRSS` value of `/proc/[pid]/status` 10 times. Since Apache is a multi-process program, we write a script to summarize all the `VmRSS` from all the Apache processes. In order to simulate the real-world workload, we use command `ab -n 10000 -c 100 https://ip/index.html` to simulate 10,000 requests with 100 requests at a time. Apache with GNU Libc Allocator uses 258.540MB RAM on average. Apache with the modified allocator uses 773.620MB RAM on average. Our modified allocator introduces no overhead in kernel space, but 109.2% overhead in user space.

VI. Conclusion

This paper proposes a co-design adaptive defense scheme against Heartbleed-like attacks. In particular, the UCB-Z algorithm is proposed to guide the defender to allocate guard pages on a heap. The security damages of the UCB-Z algorithm are proven to be always below an upper bound without knowing which attack is launched. In addition, a concrete server modification is proposed in an Apache server to support the random buffer allocation. The experiment results show that the co-design adaptive defense scheme has an asymptotically tight damage upper bound against real-world Heartbleed-like attacks. The runtime overhead of the co-design scheme is as low as 5%.

References

[1] MITRE, "CWE-126: Buffer over-read," https://cwe.mitre.org/data/definitions/126.html, year=2006, month = Jul,.
[2] CVE-2009-2523, "License logging server heap overflow vulnerability," Jul 2009, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-2523.
[3] NIST, "Heap-based buffer over-read in Nagios Core 3.5.1," Dec 2013, https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-7108.
[4] S. Frizell, "Report: Devastating Heartbleed flaw was used in hospital hack," Oct 2014, https://time.com/3148773/report-devastating-heartbleed-flaw-was-used-in-hospital-hack/.

[5] Z. Durumeric, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, J. Beekman, M. Payer, and V. Paxson, "The matter of Heartbleed," in *Proceedings of the 2014 Conference on Internet Measurement Conference (IMC '14)*, Vancouver, BC, Canada, Nov 2014, pp. 475–488.

[6] S. Gujrathi, "Heartbleed bug: An openssl heartbeat vulnerability," *International Journal of Computer Science and and Engineering*, vol. 2, no. 5, pp. 61–64, 2014.

[7] J. Juhl, "Applying patches to the Linux kernel," Aug 2005, https://www.kernel.org/doc/html/v4.18/process/applying-patches.html.

[8] B. Hawkes, "0day "in the wild"," May 2019, https://googleprojectzero.blogspot.com/p/0day.html.

[9] E. D. Berger and B. G. Zorn, "Diehard: Probabilistic memory safety for unsafe languages," *ACM SIGPLAN Notices*, vol. 41, no. 6, pp. 158–168, 2006.

[10] Q. Zeng, M. Zhao, and P. Liu, "Heaptherapy: An efficient end-to-end solution against heap buffer overflows," in *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '15)*, Rio de Janeiro, Brazil, June 2015, pp. 485 – 496.

[11] T. Lee, "Heartbleed (cve-2014-0160) test & exploit python script," 2014, https://gist.github.com/eelsivart/10174134.

[12] CVE-2014-0160, "Heartbleed bug," Apr 2014, https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-0160.

[13] FireEye, "Attackers exploit the heartbleed openssl vulnerability to circumvent multi-factor authentication on vpns," Apr 2014, https://www.fireeye.com/blog/threat-research/2014/04/attackers-exploit-heartbleed-openssl-vulnerability.html.

[14] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[15] O. Whitehouse, "An analysis of address space layout randomization on Windows Vista," *Symantec advanced threat research*, pp. 1–14, 2007.

[16] Microsoft, "Heapalloc function," https://docs.microsoft.com/en-us/windows/desktop/api/heapapi/nf-heapapi-heapalloc, Dec 2018.

[17] Y. Ding, T. Wei, T. Wang, Z. Liang, and W. Zou, "Heap taichi: Exploiting memory allocation granularity in heap-spraying attacks," in *Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC '10)*, Austin, Texas, USA, Dec 2010, pp. 327–336.

[18] E. O. Stangvik, "Heartbleed-tools," Jun 2014, https://github.com/einaros/heartbleed-tools.

[19] R. Bryant, O. David Richard, and O. David Richard, *Computer systems: A programmer's perspective*. Prentice Hall, 2003, vol. 2.

[20] W. Gloger, "Ptmalloc," May 2006, http://www.malloc.de/en/index.html.

[21] Linux, "mprotect(2) - linux man page," Apr 2020, http://linux.die.net/man/2/mprotect.

[22] P. Silberman and R. Johnson, "A comparison of buffer overflow prevention implementations and weaknesses," Aug 2004, https://www.blackhat.com/presentations/bh-usa-04/bh-us-04-silberman/bh-us-04-silberman-paper.pdf.

[23] S. Sidiroglou, G. Giovanidis, and A. D. Keromytis, "A dynamic mechanism for recovering from buffer overflow attacks," in *Proceedings of the 8th International Conference on Information Security (ISC '05)*, Singapore, Sep 2005, pp. 1–15.

[24] E. Zambon and D. Bolzoni, "Network intrusion detection systems: False positive reduction through anomaly detection," Jul 2006, http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Zambon.pdf.

[25] J. B. Zurschmeide, *IRIX advanced site and server administration guide*. Silicon Graphics, 1994, vol. 1.

[26] V. Kuleshov and D. Precup, "Algorithms for multi-armed bandit problems," *arXiv preprint*, vol. abs/1402.6028, 2014.

[27] T. L. Lai and H. Robbins, "Asymptotically efficient adaptive allocation rules," *Advances in applied mathematics*, vol. 6, no. 1, pp. 4–22, 1985.

[28] R. Seacord, "Understanding uninitialized reads," Feb 2017, https://www.nccgroup.com/us/about-us/newsroom-and-events/blog/2017/february/understanding-uninitialized-reads/.

[29] GNU.org, "The GNU C library (glibc)," Aug 2020, https://www.gnu.org/software/libc/.

[30] M. Habib, C. McDiarmid, J. Ramirez-Alfonsin, and B. Reed, *Probabilistic methods for algorithmic discrete mathematics*. Springer Science & Business Media, 2013, vol. 16.

[31] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, pp. 79–86, 1951.

Appendix A

This section gives the proof and implications of Theorem 1.To prove Theorem 1, we introduce the optimal defense strategy and prove that the gap between the UCB-Z algorithm and the optimal strategy is upper bounded.

Since the random vector $\vec{SA}$ follows the specific uniform distribution introduced in Section II-B2 all the time, for ease of presentation, we will refer $\mathbb{E}_{\mathbb{P}_{\mathbb{P}_\theta},\mathbb{P}_{SA}} u(d,a,\vec{SA})$ as to $\mu(\mathbb{P}_\theta,d)$ and $\mathbb{E}_{\mathbb{P}_{\mathbb{P}_\theta},\mathbb{P}_{SA}}$ as to $\mathbb{E}$ in the rest of the paper.

Recall that the optimal defense action with respect to a given attack is the one with the highest *expected* utility value. The optimal defense strategy, denote φ^*, is to stick to the optimal action if the defender knows the utility distributions. However, the defender does not know the true expected utility values for its defense actions *a priori* because it is facing zero-day attacks. This prevents the defender from applying the optimal strategy. Then for a particular attack $\mathbb{P}_\theta$, we introduce *Regret* to quantify the gap between the defender's actual aggregate expected damage by following some strategy φ and the aggregate expected damage of the optimal strategy:

$$\begin{aligned}
&R_\varphi(\mathbb{P}_\theta,N) \triangleq D_\varphi(\mathbb{P}_\theta,N) - D_{\varphi^*}(\mathbb{P}_\theta,N)\\
&= N\mu(\mathbb{P}_\theta,BR(\mathbb{P}_\theta)) - \sum_{t=1}^{N}\sum_{j=1}^{h(t)} \mathbb{E}\left(u(t-\mathcal{T}_j)+\epsilon(t-\mathcal{T}_j)\right).
\end{aligned} \tag{1}$$

If the defender chooses defense actions by following the UCB-Z algorithm, the regret cannot be larger than the derived upper bound no matter which set of distributions the utility values follow. This is formalized in Lemma 1.

Lemma 1. *Under the* **Property M-1***, for any attack $\mathbb{P}_\theta \in \Theta$, the regret of the UCB-Z algorithm after N defense cycles is upper bounded in the following way:*

$$\begin{aligned}
&R_{UCB-Z}(\mathbb{P}_\theta,N)\\
&\le \mathcal{T}\mu(\mathbb{P}_\theta,BR(\mathbb{P}_\theta)) + \sum_{d:0<\Delta(\mathbb{P}_\theta,d)\le 2\epsilon} 3(N-\mathcal{T})\epsilon\\
&\quad + 8(u^+-u^-)^2 \sum_{d:\Delta(\mathbb{P}_\theta,d)>2\epsilon} \frac{(\Delta(\mathbb{P}_\theta,d)+\epsilon)\ln(N-\mathcal{T}+1)}{(\Delta(\mathbb{P}_\theta,d)-2\epsilon)^2}\\
&+ \left(1+\frac{\pi^2}{3}\right) \sum_{d:\Delta(\mathbb{P}_\theta,d)>2\epsilon} (\Delta(\mathbb{P}_\theta,d)+\epsilon).
\end{aligned}$$

Before we prove Lemma 1, we first introduce Lemma 2 to provide bounds on how a random variable deviates from its expected value.

Lemma 2. *(McDiarmid's inequality [30]). Suppose $X_1,X_2,\cdots,X_n$ are independent random variables all taking values in set $\mathcal{X}$, and let $f:\mathcal{X}^n\to\mathbb{R}$ be a function of $X_1,X_2,\cdots,X_n$ that satisfies for all $x_1,\cdots,x_n,x_i'\in\mathcal{X}$, $|f(x_1,\ldots,x_n) - f(x_1,\ldots,x_{i-1},\hat{x}_i,x_{i+1},\ldots,x_n)| \le c_i$, where $1\le i\le n$ and c_i is any positive real number. Then for any $\varepsilon>0$, $Pr\{\pm f \mp \mathbb{E}f \ge \varepsilon\} \le \exp\left(\frac{-2\varepsilon^2}{\sum_{i=1}^n c_i^2}\right)$.*

This lemma is proved as the Theorem 3.1 in [30].

A. Proof of Lemma 1

Proof. The proof extends that in the paper [14] to further include the utility delays and errors. For any $d\in\mathcal{D}$, define

$\bar{\mu}_d(t) \triangleq \frac{\sum_{\tau=1}^{t-1}\sum_{i=1}^{h(\tau)}(u(\tau-\mathcal{T}_i)\mathbf{1}_{\{d(\tau-\mathcal{T}_i)=d\}})}{\sum_{\tau=1}^{t-1}\sum_{i=1}^{h(\tau)}\mathbf{1}_{\{d(\tau-\mathcal{T}_i)=d\}}}$ is the *empirical average error-free utility* the defender receives by choosing defense action d by the end of defense cycle $t-1$.

According to the definition of *Regret*, we can present the *Regret* for UCB-Z after N defense cycles as follows:

$$R_{UCB-Z} = \sum_{t=1}^{N}\mu(\mathbb{P}_\theta, BR(\mathbb{P}_\theta)) - \sum_{t=1}^{N}\sum_{j=1}^{h(t)}\mathbb{E}\left(u(t-\mathcal{T}_j)+\epsilon(t-\mathcal{T}_j)\right).$$

There are at most $\mathcal{T}$ utility values that will be missed because of the delays. And since the utility errors are uniformly bounded by ϵ, we have $\forall t, \epsilon(t) \geq -\epsilon$. Then:

$$\begin{aligned} &R_{UCB-Z} \\ &\leq N\mu(\mathbb{P}_\theta, BR(\mathbb{P}_\theta)) - \sum_{t=1}^{N-\mathcal{T}}\mathbb{E}\left(u(d(t),a(t),\vec{SA}(t))+\epsilon(t)\right) \\ &\leq N\mu(\mathbb{P}_\theta, BR(\mathbb{P}_\theta)) - \sum_{t=1}^{N-\mathcal{T}}\left(\mathbb{E}u(d(t),a(t),\vec{SA}(t))-\epsilon\right) \\ &= N\mu(\mathbb{P}_\theta, BR(\mathbb{P}_\theta)) - \sum_{t=1}^{N-\mathcal{T}}(\mu(\mathbb{P}_\theta,d(t))-\epsilon). \end{aligned}$$

We define how many times defense action d was chosen during the $N-\mathcal{T}$ defense cycles as a random variable $T_d(N-\mathcal{T}+1)$. Then we have $\sum_{t=1}^{N-\mathcal{T}}\mu(\mathbb{P}_\theta,d(t)) = \sum_{d\in\mathcal{D}}\mu(\mathbb{P}_\theta,d)T_d(N-\mathcal{T}+1)$. So alternatively, we can also represent *Regret* as follows:

$$\begin{aligned} &R_{UCB-Z} \leq \\ &\mathcal{T}\mu(\mathbb{P}_\theta, BR(\mathbb{P}_\theta)) + \sum_{t=1}^{N-\mathcal{T}}(\mu(\mathbb{P}_\theta, BR(\mathbb{P}_\theta)) - \mu(\mathbb{P}_\theta,d(t))+\epsilon) \\ &= \mathcal{T}\mu(\mathbb{P}_\theta, BR(\mathbb{P}_\theta)) \\ &+ \sum_{d:\Delta(\mathbb{P}_\theta,d)>2\epsilon}(\Delta(\mathbb{P}_\theta,d)+\epsilon)\mathbb{E}T_d(N-\mathcal{T}+1) \\ &+ \sum_{d:0<\Delta(\mathbb{P}_\theta,d)\leq 2\epsilon}(\Delta(\mathbb{P}_\theta,d)+\epsilon)\mathbb{E}T_d(N-\mathcal{T}+1). \end{aligned} \tag{2}$$

To prove this theorem, we need to identify an upper bound of $\mathbb{E}T_d(N-\mathcal{T}+1)$. Note that in the inequality (2), we separate the defense actions into two cases: **Case 1, defense actions $d\in\mathcal{D}$ satisfy that $\Delta(\mathbb{P}_\theta,d)>2\epsilon$; Case 2, defense actions $d\in\mathcal{D}$ satisfy that $\Delta(\mathbb{P}_\theta,d)\leq 2\epsilon$.**

Case 1, defense actions $d\in\mathcal{D}$ satisfy $\Delta(\mathbb{P}_\theta,d)>2\epsilon$.

At defense cycle t, let $b_d(t) = \lceil (u^+-u^-)^2\frac{8\ln t}{(\Delta(\mathbb{P}_\theta,d)-2\epsilon)^2}\rceil$. We consider two temporal phases with respect to action d at defense cycle t. The first phase is when $T_d(t)\leq b_d(t)$. And the second phase is when $T_d(t)\geq b_d(t)$.

If a non-optimal defense action d stays in the first phase for the whole time; i.e., $\forall 1\leq t\leq N-\mathcal{T}+1$, $\mathbb{E}T_d(t)\leq b_d(t)$, the regret brought by choosing this non-optimal defense action is small. To upper bound the regret, we focus on when a non-optimal defense action d achieves the second phase. We want to prove that in the second phase, the defender selects action d only a very few times in expectation. Therefore, the regret brought by choosing a non-optimal defense action even more than $b_d(t)$ times is small. Note for the second phase, we have

$$\mu(\mathbb{P}_\theta, BR(\mathbb{P}_\theta)) - \mu(\mathbb{P}_\theta,d) \geq 2(u^+-u^-)\sqrt{\frac{2\ln(t)}{T_d(t)}}+2\epsilon. \tag{3}$$

This inequality holds because $T_d(t)\geq b_d(t) = \lceil (u^+-u^-)^2\frac{8\ln t}{(\Delta(\mathbb{P}_\theta,d)-2\epsilon)^2}\rceil$ and since $\Delta(\mathbb{P}_\theta,d)>2\epsilon$. Then $2(u^+-u^-)\sqrt{\frac{2\ln(t)}{T_d(t)}}+2\epsilon \leq 2(u^+-u^-)\sqrt{\frac{2\ln(t)}{b_d(t)}}+2\epsilon = 2\sqrt{\frac{2\ln t}{\frac{8\ln t}{(\Delta(\mathbb{P}_\theta,d)-2\epsilon)^2}}}+2\epsilon = \Delta(\mathbb{P}_\theta,d)-2\epsilon+2\epsilon = \Delta(\mathbb{P}_\theta,d) = \mu(\mathbb{P}_\theta, BR(\mathbb{P}_\theta))-\mu(\mathbb{P}_\theta,d)$.

And for any non-optimal defense actions $d\in\mathcal{D}$ such that $\Delta(\mathbb{P}_\theta,d)>2\epsilon$, $\mathbb{E}T_d(N-\mathcal{T}+1)$ is upper bounded as follows:

$$\begin{aligned} &\mathbb{E}T_d(N-\mathcal{T}+1) \leq b_d(N-\mathcal{T}+1) + \sum_{t=b_d(N-\mathcal{T}+1)}^{N-\mathcal{T}}\mathbb{E}\mathbf{1}_{\{d(t)=d\}} \\ &= b_d(N-\mathcal{T}+1) + \sum_{t=b_d(N-\mathcal{T}+1)}^{N-\mathcal{T}}Pr\{d(t)=d\}. \end{aligned}$$

Based on the UCB-Z algorithm, the defender may choose d when there exists a pair of $T_d(t), T_{d'}(t)$ for any $d'\in\mathcal{D}$ such that $\bar{\mu}'_d(t)+(u^+-u^-)\sqrt{\frac{2\ln(t)}{T_d(t)}} \geq \bar{\mu}'_{d'}(t)+(u^+-u^-)\sqrt{\frac{2\ln(t)}{T_{d'}(t)}}$. Note that $\bar{\mu}_{d'}(t)-\epsilon \leq \bar{\mu}'_{d'}(t) \leq \bar{\mu}_{d'}(t)+\epsilon$ for all $d'\in\mathcal{D}$. Therefore we have for all $d'\in\mathcal{D}$,

$$\begin{aligned} &Pr\{d(t)=d\} \\ &\leq \sum_{T_d(t)=b_d(t)}^{t}\sum_{T'_d(t)=1}^{t}Pr\left\{\bar{\mu}_d(t)-\epsilon+(u^+-u^-)\sqrt{\frac{2\ln(t)}{T_d(t)}} \geq \right. \\ &\left. \bar{\mu}_{d'}(t)+\epsilon+(u^+-u^-)\sqrt{\frac{2\ln(t)}{T_{d'}(t)}}\right\}. \end{aligned}$$

We consider the defense action induced by the best response $BR(\mathbb{P}_\theta)$ and we have,

$$\begin{aligned} &Pr\{d(t)=d\} \leq \\ &\sum_{T_d(t)=b_d(t)}^{t}\sum_{T_{BR(\mathbb{P}_\theta)}=1}^{t}Pr\left\{\bar{\mu}_d(t)-\epsilon+(u^+-u^-)\sqrt{\frac{2\ln(t)}{T_d(t)}} \geq \right. \\ &\left. \bar{\mu}_{BR(\mathbb{P}_\theta)}(t)+\epsilon+(u^+-u^-)\sqrt{\frac{2\ln(t)}{T_{BR(\mathbb{P}_\theta)}(t)}}\right\}. \end{aligned} \tag{4}$$

To ensure $\bar{\mu}_d(t)-\epsilon+(u^+-u^-)\sqrt{\frac{2\ln(t)}{T_d(t)}} \geq \bar{\mu}_{BR(\mathbb{P}_\theta)}(t)+\epsilon+(u^+-u^-)\sqrt{\frac{2\ln(t)}{T_{BR(\mathbb{P}_\theta)}(t)}}$, one of the following two inequalities must be true:

$$\bar{\mu}_{BR(\mathbb{P}_\theta)}(t) \leq \mu(\mathbb{P}_\theta, BR(\mathbb{P}_\theta)) - (u^+-u^-)\sqrt{\frac{2\ln(t)}{T_{BR(\mathbb{P}_\theta)}(t)}} \tag{5}$$

$$\bar{\mu}_d(t) \geq \mu(\mathbb{P}_\theta,d)+(u^+-u^-)\sqrt{\frac{2\ln(t)}{T_d(t)}}. \tag{6}$$

If (5) and (6) are both false, then

$$\begin{aligned}&\bar{\mu}'_{BR(\mathbb{P}_\theta)}(t) + (u^+ - u^-)\sqrt{\frac{2\ln(t)}{T_{BR(\mathbb{P}_\theta)}(t)}}\\ &\geq \bar{\mu}_{BR(\mathbb{P}_\theta)}(t) - \epsilon + (u^+ - u^-)\sqrt{\frac{2\ln(t)}{T_{BR(\mathbb{P}_\theta)}(t)}}\\ &> \mu(\mathbb{P}_\theta, BR(\mathbb{P}_\theta)) - \epsilon = \mu(\mathbb{P}_\theta, d) + \Delta(\mathbb{P}_\theta, d) - \epsilon \quad (7)\\ &\geq \mu(\mathbb{P}_\theta, d) + 2(u^+ - u^-)\sqrt{\frac{2\ln(t)}{T_d(t)}} + 2\epsilon - \epsilon \quad (8)\\ &> \bar{\mu}'_d(t) + (u^+ - u^-)\sqrt{\frac{2\ln(t)}{T_d(t)}}. \quad (9)\end{aligned}$$

The inequality (7) holds when (5) is false, and inequality (8) holds because of inequality (3). When (6) is false, we can reach inequality (9), which contradicts with $I_d(t) \geq I_{BR(\mathbb{P}_\theta)}(t)$.

For any $d \in \mathcal{D}$ which achieves second phase and satisfies $\Delta(\mathbb{P}_\theta, d) > 2\epsilon$, let X_i be the error-free utility when d is chosen for the $i-th$ time and let $f(x_1, \cdots, x_{T_d(t)}) = \frac{1}{T_d(t)} \sum_{i=1}^{T_d(t)} (X_i)$. Then we get $f = \bar{\mu}_d(t)$ and $\mathbb{E}f = \mu(\mathbb{P}_\theta, d)$. Note that $X_i \in [u^-, u^+]$ for all $1 \leq i \leq T_d(t)$. Then we have $|f(x_1, \ldots, x_{T_d(t)}) - f(x_1, \ldots, x_{i-1}, \hat{x}_i, x_{i+1}, \ldots, x_{T_d(t)})| \leq \frac{1}{T_d(t)}(u^+ - u^-)$.

And by Lemma 2, the probability that inequality (5) is true is upper bounded as follows:

$$Pr\{(4)\} \leq \exp\left(\frac{-2\left((u^+ - u^-)\sqrt{\frac{2\ln(t)}{T_{BR(\mathbb{P}_\theta)}(t)}}\right)^2 T_{BR(\mathbb{P}_\theta)}(t)}{(u^+ - u^-)^2}\right) = t^{-4}. \quad (10)$$

Similarly, the probability that inequality (6) is true can also be upper bounded:

$$Pr\{(5)\} \leq t^{-4}. \quad (11)$$

By inequality (4), for any $d \in \mathcal{D}$ such that $\Delta(\mathbb{P}_\theta, d) > 2\epsilon$,

$$\begin{aligned}&\mathbb{E}T_d(N - \mathcal{T} + 1)\\ &\leq b_d(N - \mathcal{T} + 1) + \sum_{t=b_d(N-\mathcal{T}+1)}^{N-\mathcal{T}} Pr\{d(t) = d\}\\ &\leq b_d(N - \mathcal{T} + 1)\\ &+ \sum_{t=b_d(N-\mathcal{T}+1)}^{N-\mathcal{T}} \sum_{T_d(t)=b_d(t)}^{t} \sum_{T_{BR(\mathbb{P}_\theta)}(t)=1}^{t} \left(Pr\{(4)\} + Pr\{(5)\}\right).\end{aligned}$$

And by inequalities (10) and (11), we have,

$$\begin{aligned}&\mathbb{E}T_d(N - \mathcal{T} + 1)\\ &\leq b_d(N - \mathcal{T} + 1) + \sum_{t=b_d(N-\mathcal{T}+1)}^{\infty} \sum_{T_d(t)=b_d(t)}^{t} \sum_{T_{BR(\mathbb{P}_\theta)}(t)=1}^{t} 2t^{-4}\\ &\leq b_d(N - \mathcal{T} + 1) + \sum_{t=b_d(N-\mathcal{T}+1)}^{\infty} 2t^{-2}\\ &\leq b_d(N - \mathcal{T} + 1) + \sum_{t=1}^{\infty} 2t^{-2}\\ &\leq b_d(N - \mathcal{T} + 1) + \frac{\pi^2}{3}\\ &\leq (u^+ - u^-)^2 \frac{8\ln(N - \mathcal{T} + 1)}{(\Delta(\mathbb{P}_\theta, d) - 2\epsilon)^2} + 1 + \frac{\pi^2}{3}.\end{aligned}$$

Case 2, defense actions $d \in \mathcal{D}$ satisfy $\Delta(\mathbb{P}_\theta, d) \leq 2\epsilon$. The number of times that this kind of actions are chosen can be uniformly upper bounded as follows:

$$\mathbb{E}T_d(N - \mathcal{T} + 1) \leq N - \mathcal{T}. \quad (12)$$

Inequality (12) holds because no actions can be chosen more than the $N - \mathcal{T}$ times during $N - \mathcal{T}$ defense cycles.

Then the inequality (2) about the *Regret* becomes:

$$\begin{aligned}&R_{UCB-Z}(\mathbb{P}_\theta, N)\\ &\leq \mathcal{T}\mu(\mathbb{P}_\theta, BR(\mathbb{P}_\theta)) + \sum_{d:0<\Delta(\mathbb{P}_\theta,d)\leq 2\epsilon} 3(N - \mathcal{T})\epsilon\\ &\quad + 8(u^+ - u^-)^2 \sum_{d:\Delta(\mathbb{P}_\theta,d)>2\epsilon} \frac{(\Delta(\mathbb{P}_\theta, d) + \epsilon)\ln(N - \mathcal{T} + 1)}{(\Delta(\mathbb{P}_\theta, d) - 2\epsilon)^2}\\ &\quad + \left(1 + \frac{\pi^2}{3}\right) \sum_{d:\Delta(\mathbb{P}_\theta,d)>2\epsilon} (\Delta(\mathbb{P}_\theta, d) + \epsilon).\end{aligned}$$

□

B. Proof of Theorem 1

Proof. With Lemma 1, we can easily show that the aggregate expected damage of the UCB-Z algorithm is upper bounded by the regret upper bound plus the aggregate expected damage of the optimal policy against the worst attack in Θ.

By the definition of regret in equation (1), we know $D_{UCB-Z}(\mathbb{P}_\theta, N) = D_{\varphi^*}(\mathbb{P}_\theta, N) + R_{UCB-Z}(\mathbb{P}_\theta, N)$. And with bounded utility values and limited number of defense actions, we can easily get $\sum_{d:\Delta(\mathbb{P}_\theta,d)>2\epsilon} (\Delta(\mathbb{P}_\theta, d) + \epsilon) \leq |\mathcal{D}|(u^+ - u^- + \epsilon)$ for any attack. Combine this with Lemma 1, we can achieve Theorem 1. □

C. Implications

The upper bound stated in Theorem 1 consists of five terms. The first term is the least damage by applying the optimal defense strategy against all attacks in Θ. The second term $\mathcal{T}\mu_{\max}$ is the regret caused by the delays of utility reports, and this term is a constant. The third item is the regret brought by choosing the suboptimal defense actions whose expected utility values are ϵ-close to the optimal one's. If $\epsilon = 0$, this term is 0. The fourth term $8(u^+ - u^-)^2|\mathcal{D}|\frac{(u^+ - u^- + \epsilon)\ln(N - \mathcal{T} + 1)}{(\Delta_{min} - 2\epsilon)^2}$ increases logarithmically in N. The fifth term is the regret brought by choosing the suboptimal defense actions after they are chosen enough times, and this term is also a constant. It is worthy to mention that the regret upper bound in Lemma 1 is an extension of the regret upper bound in [14] by taking into account utility delays and errors. When the utility delays and errors are zero, the upper bound reduces to the one in [14].

In [27], LAI AND ROBBINS proved that when the utility distributions are Bernoulli, for any strategy φ with same information set $\mathcal{F}$ denoted in Section II-B5, its *Regret* has a lower bound asymptotically: $\lim_{N\to\infty} R_\varphi(\mathbb{P}_\theta, N) \geq \sum_{d:\mu(\mathbb{P}_\theta,d)<\mu(\mathbb{P}_\theta,BR(\mathbb{P}_\theta))} \frac{\Delta(\mathbb{P}_\theta,d)\ln(N)}{KL(\mathbb{P}_{\mathbb{P}_\theta,d}||\mathbb{P}_{\mathbb{P}_\theta,BR(\mathbb{P}_\theta)})}$, where $KL(\mathbb{P}_{\mathbb{P}_\theta,d}||\mathbb{P}_{\mathbb{P}_\theta,BR(\mathbb{P}_\theta)}) \triangleq \sum_{\mathbb{P}_{\mathbb{P}_\theta,d}} \mathbb{P}_{\mathbb{P}_\theta,d} \ln \frac{\mathbb{P}_{\mathbb{P}_\theta,d}}{\mathbb{P}_{\mathbb{P}_\theta,BR(\mathbb{P}_\theta)}}$ is the Kullback-Leibler divergence [31] between the utility distribution $\mathbb{P}_{\mathbb{P}_\theta,d}$ of any suboptimal defense action and the utility distribution $\mathbb{P}_{\mathbb{P}_\theta,BR(\mathbb{P}_\theta)}$ of the optimal defense action. That is, if the utility distributions are Bernoulli, then the regret is always larger than the lower bound no matter what strategy φ the defender follows.

If the utility errors are very small; e.g., $\epsilon = 0$, the regret part in the upper bound (the second to fifth terms) in Theorem 1 is a logarithmic upper bound with respect to N and increases as slow as the lower bound when the utility distributions are Bernoulli. Therefore, the upper bound in Theorem 1 is the best possible upper bound which holds for any attack in Θ. From Lemma 1, we know that $R_{UCB-Z}(\mathbb{P}_\theta, N) = o(N)$ when $\epsilon = 0$. With equation (1), we know the damage of UCB-Z is optimal in the sense of $\lim_{N\to\infty}(1/N)D_{UCB-Z}(\mathbb{P}_\theta, N) = \lim_{N\to\infty}(1/N)D_{\varphi^*}(\mathbb{P}_\theta, N) = \mu(\mathbb{P}_\theta, BR(\mathbb{P}_\theta))$. That is, the asymptotic behavior of UCB-Z is as good as the optimal defense policy.