Contextual Multi-armed Bandits for Web Server Defense

Tobias Jung Sylvain Martin Damien Ernst Guy Leduc
Montefiore Institute, University of Liège, 4000 Belgium
Email:{tjung,sylvain.martin,dernst,guy.leduc}@ulg.ac.be

***Abstract*—In this paper we argue that contextual multi-armed bandit algorithms could open avenues for designing self-learning security modules for computer networks and related tasks. The paper has two contributions: a conceptual and an algorithmical one. The conceptual contribution is to formulate the real-world problem of preventing HTTP-based attacks on web servers as a one-shot sequential learning problem, namely as a contextual multi-armed bandit. Our second contribution is to present CMABFAS, a new and computationally very cheap algorithm for general contextual multi-armed bandit learning that specifically targets domains with finite actions. We illustrate how CMABFAS could be used to design a fully self-learning meta filter for web servers that does not rely on feedback from the end-user (i.e., does not require labeled data) and report first convincing simulation results.**

I. Introduction

Tools such as blogs and wikis contribute nowadays to our freedom of speech. They collectively invite any visitor of a web page to take part in the author's effort by expressing relevant opinions or increasing the gathered knowledge. Although this allows individuals world-wide to communicate in unprecedented ways, the system can easily be abused for self-promotion or other more malicious purposes. For example, unprotected wikis are frequently defaced by spammers who inject links towards their own site, either to attract the regular visitors of that site, or to fool indexing engines and increase their own ranking. Since administration is mostly integrated into the web service itself (for example, by simply requiring a password to access plugin installation), these services are an easy target for an attacker who tries to get code remotely executed through a so-called *exploit* to gain control of the system, e.g., see [12]. Finally, as soon as on-line activity becomes a key source of revenue for its manager, unscrupulous competitors could try and make the service unavailable through a (distributed) denial of service attack on either bandwidth, state, or processing resources of the hosting infrastructure.

One of the chief difficulties inherent to the defense of web servers against these kind of attacks is that a security policy that works well against one sort of problem may be ineffective against other problems and sometimes even open avenues to new forms of abuse. CAPTCHAs [14], for instance, can be used to counter certain forms of voluntary denial of service attacks and are widely used as a first line of defense against spam. They are, however, highly annoying for regular users, should be avoided in case of visitors rush on a service (a.k.a. a flash crowd) and are uneffective against dedicated spammers. Similarly, user input tracking [1] could be a more effective and for human users far less annoying mechanism to counter the activity of bots. However, since user input tracking consumes significant resources on the server, it is best only selectively used. Finally, when a certain class of methods is particularly effective against a certain class of attacks, a specific instance of this defensive method can often be in-depth analyzed by dedicated hackers who can then provide an effective way to break this particular defense until the victim reacts with a patch.

Therefore, while we can imagine that a web server has different potential security policies at its disposal (which we will also call *actions* in the following), deciding which one to best apply in a given situation is not trivial. Current industry solutions are designed by experts and largely rely on hand-coded rules and database look-ups of signatures of known attacks or attack patterns. As is well known however, any such *static* approach comes with two main weaknesses: (1) it can not be easily automated and constantly requires a human in the loop to supervise the system and manually identify attacks and adapt rules which is a costly and error-prone procedure; (2) it can only respond to known attacks.

Based on the design of an abstract HTTP meta-filter which can choose, for each independent HTTP request arriving at the server, which security policy out of many possible ones to apply (where we include "apply no security policy" as just another action), our goal is to create a *self-learning* filter which does not rely on hand-coded rules but automatically determines from past experience what the "best" response should be for a given HTTP request. The main feature of this HTTP meta-filter is that it is self-learning and able to adapt to novel attack patterns. In particular, this self-learning does not rely on external feedback or labeled prior data; instead the system will monitor its own performance and generate internal rewards. These rewards will be stochastic and depend on the results of security challenges (e.g., whether or not a CAPTCHA was passed, how much time between the responses has elapsed, etc.) and other indicators derived from the subsequent HTTP requests within the same session.

To achieve these goals, we formulate this application scenario as an instance of a reinforcement learning problem; specifically, we formulate it as a *contextual multi-armed bandit problem*: contexts are the incoming HTTP requests, actions are choosing which particular security policy to apply, and reward is internally generated from subsequent traffic. While related techniques were to some extent already used in earlier work for intrusion and anomaly detection, e.g., finite state machines and hidden Markov models in [19], [18], [16], or temporal difference learning in [17], the novelty of this present work is to model the decision-making and reward process itself and to solve it with multi-armed bandits.

The remainder of the paper is structured as follows. Section 2 gives some background on multi-armed bandits and motivates why they are well-suited to model the problem of web server defense we have outlined above. The following Section 3 introduces our algorithm CMABFAS (contextual MAB in finite action spaces) which is based on the zooming algorithm in [13] and which has exactly the properties we need for our HTTP meta filter. Section 4 then describes in detail how we can map the real-world problem of web server defense to the conceptual framework of contextual MAB so that we can apply CMABFAS. Section 5 presents some simulation results and compares CMABFAS with a more naïve baseline implementation of MAB.

II. Background: Multi-armed bandits

From an abstract point of view, the role of our HTTP meta-filter is to map, or learn how to map over time, each incoming HTTP session to the most appropriate security policy. Here "most appropriate" is with respect to some numerical performance criterion which is based on the feedback the filter receives from the environment. While we leave the precise definition of this feedback or *reward* to Section 4, the very nature of this feedback poses a complex research challenge: (1) the feedback that we can get will be sparse and limited in that we only get feedback for the particular security policy that was *applied* (but no feedback for the alternative policies); (2) the feedback that we can get will be stochastic meaning that we have to try a single policy multiple times in order to get a reliable estimate of how good it is. Reasons for stochasticity are many. For example, the feedback could be the information if an associated CAPTCHA was passed or failed, how many attempts where necessary, how long did it take etc.

To achieve our goal, we formalize our HTTP meta-filter as a multi-armed bandit problem (MAB) with context. *Standard MABs* with k-arms are well-studied models for sequential decision-making when the outcome of actions is stochastic and its distribution a priori unknown [9], [3]. A *contextual MAB* is in principle like a standard MAB with the major difference that it is defined over some large set of elements or a continuous space. Now each element of the set corresponds to a separate standard MAB (in some formulations also the action space becomes large or continuous). Learning the reward distributions in contextual MABs is more challenging than it is for standard MAB since it is no longer possible to sample the same action multiple times. Instead, we have to impose "smoothness" as additional structural assumption; i.e., we assume that elements of the context space that are "similar" (with respect to some similarity measure) will also behave similarly. Using generalization we can then try to predict the outcome for new cases based on previously observed outcomes for similar cases. Contextual MAB are nowadays an active research topic with many relevant real-world applications, e.g., placement of web advertisements. See [15], [10], [11], [13], [8], [7] for some examples.

We believe that contextual MAB (but not standard MAB) are a good description of what the HTTP meta-filter motivated in Section 1 is trying to achieve: the contexts correspond to individual HTTP requests, the actions correspond to security policies the filter can choose from, and the rewards correspond to how the subsequent network traffic reacts to this chosen policy.

III. Description of CMABFAS

This section describes our algorithm CMABFAS. Note that we keep the presentation general, the actual application to the web server defense scenario will be described in the following Section IV. Our work is largely based on the contextual zooming algorithm described in [13], and inspired by the X-armed bandit learning algorithm described in [5]. Differences between CMABFAS and [13] are: a specialization to the finite action case and a modified scheme to estimate the expected rewards which works for more general metric spaces (we do not need the triangle inequality).

A. Notation

Let $\mathcal{X}$ denote a context space with elements $x \in \mathcal{X}$ and let $a \in \{1, \ldots, k\}$ denote the possible actions that can be chosen for each x (we assume that we have the same choice of actions available for each x). Each context x can be seen as an index to a conventional k-armed bandit: for each x we have a distribution of rewards $\mathcal{R}^a(x)$ under action a which models the stochastic response from the environment when performing action a in context x. Let $r^a(x)$ denote the random values drawn from the corresponding reward distribution, i.e., $r^a(x) \sim \mathcal{R}^a(x)$. We assume that $\mathcal{R}^a(x)$ has bounded support which, for notational convenience, is supposed to lie in the unit interval; thus we assume $\operatorname{supp} \mathcal{R}^a(x) \subseteq [0,1]$. In consequence, we also have $r^a(x) \in [0,1]$.

Let $\mu^a(x)$ denote the mean of the reward distribution $\mathcal{R}^a(x)$; i.e.,

$$\mu^a(x) := E_{r^a(x) \sim \mathcal{R}^a(x)}\big[r^a(x)\big]. \tag{1}$$

The essence of the problem we study is that $\mathcal{R}^a(x)$, and hence $\mu^a(x)$, is not known when making decisions; instead it is treated as a black-box from which only samples can be drawn. Our overall goal is, when presented with any context x, to be able to choose the action with the highest mean: $\operatorname{argmax}_a \mu^a(x)$. The algorithm we propose is based on taking and averaging samples in a smart way such that observations we made at one location x are reused to make an estimate about the reward distribution at another location x'.

Let context space $\mathcal{X}$ be equipped with a distance metric $d(x, x')$ which measures the distance between any two elements $x, x' \in \mathcal{X}$. We assume that d is a pseudo-metric and fulfills the conditions of (1) non-negativity: $d(x, x') \geq 0$; (2) symmetry: $d(x, x') = d(x', x)$; and (3) is equal to zero if and only if the arguments are identical: $d(x, x') = 0 \Leftrightarrow x = x'$, $\forall x, x' \in \mathcal{X}$. Furthermore we assume that, for notational convenience again, d is scaled such that the diameter of $\mathcal{X}$ with respect to d is equal to one; that is, $\sup_{x,x' \in \mathcal{X}} d(x, x') = 1$.

Finally, to make learning and generalization over the context space at all possible, we need to impose smoothness and restrict the variation of the mean functions μ^a. Specifically, we assume that each μ^a is Lipschitz with parameter $\lambda > 0$:

$$|\mu^a(x) - \mu^a(x')| < \lambda d(x, x'), \quad \forall x, x' \in \mathcal{X}, \forall a. \tag{2}$$

B. Objective

The game we get to play proceeds in rounds $t = 1, 2, \ldots$. At each round t we observe $x_t \in \mathcal{X}$; we suppose that we have no control over how x_t is generated from the set $\mathcal{X}$ and furthermore that the mechanics leading to the selection of an x_t are independent from whatever happened in all previous rounds of the game. Given x_t, we have to choose an action $a_t \in \{1, \ldots, k\}$. Executing action a_t lets us then observe the reward $r^{a_t}(x_t)$ which is sampled from $\mathcal{R}^{a_t}(x_t)$. Our goal is to use the results of the $t-1$ previous rounds of the game, i.e., the history

$$(x_1, a_1, r^{a_1}(x_1), \ldots, x_{t-1}, a_{t-1}, r^{a_{t-1}}(x_{t-1})),$$

to determine an action a_t such that the regret – a measure of performance – is minimized. In the bandit literature two types of regret are considered: here we take the cumulative regret which assumes that we have to play the game for a fixed number T of rounds and that we want to minimize over all T rounds the difference between the expected reward of the best possible action minus the expected reward of the action chosen at that round:

$$\text{regret}(T) = \sum_{t=1}^{T} |\mu^*(x_t) - \mu^{a_t}(x_t)|, \tag{3}$$

where $\mu^*(x_t) = \max_{a \in \{1,\ldots,k\}} \mu^a(x_t)$.

C. CMABFAS – Overview

Our algorithm CMABFAS works as follows. For each action a separately, we incrementally construct over time $t = 1, 2, \ldots$ a cover of the context space $\mathcal{X}$. The cover consists of ball-shaped regions where individual balls are centered on a certain subset of the contexts $\{x_1, \ldots, x_{t-1}\}$ seen so far. The cover is hierarchical with the radius of the balls exponentially fast decreasing with the level of hierarchy; e.g., $\mathcal{X}$ is covered at level 1 by a single ball of radius 1, at level 2 by balls of radius $1/2$, at level 3 by balls of radius $1/4$, and so on. Each ball aggregates the reward samples lying within: we only store their number and their sum. Each ball covering x_t can thus be used to estimate the expected reward $\mu^a(x_t)$ at query point x_t. Since we have to balance exploration and exploitation, we will augment this estimate by a UCB-like term (i.e., an upper confidence bound). Each ball covering x_t thus gives rise to a score which is composed of two parts: (1) the sample average within the ball, and (2) an uncertainty term which depends on the number of samples (the fewer samples we have in a ball, the less certain we can be about the correctness of their average) and the volume of the ball (the larger the region the ball covers, the more variation of μ^a is possible and thus the more samples of a different base quantity are lumped together). The "best ball" for each action is the one with the lowest score (the tightest upper bound), and among them the highest score indicates the best action. The cover is adaptively refined by adding new balls according to the following rules: (1) a new ball can only be created centered at (x_t, a_t); (2) a new ball can only be created if the number of samples in the parent ball exceeds a certain threshold (which grows inverse quadratically in the radius of the parent ball); (3) a new ball is only created if it will not overlap with already existing balls at the same level in the hierarchy. Informally speaking, CMABFAS works by ensuring that high resolutions are only attained in regions of $\mathcal{X}$ where the corresponding action is optimal and then exploiting that balls with increasingly smaller radius provide increasingly more accurate bounds for the expected reward.

From a practical point of view our algorithm possesses two properties which make it ideally suited for operation under heavy-load real-world conditions: (1) it is an *online* algorithm whose computational complexity and storage requirements are very low (this is so because we do not have to store and operate on an always growing number of individual data points, but only have to store and operate on balls of a fixed radius, which is a much smaller number[1] and where search operations can be efficiently implemented by appropriate space partitioning methods such as cover trees [4]; (2) it is an *anytime* algorithm which aims at producing the best possible solution in each step of its operation and does not need any kind of prior learning phase with data or tuning to start producing meaningful results.

D. CMABFAS – Additional notation

Before we come to the details of the algorithm, we need to introduce some more notation. Let $B_i^a \in \{1, \ldots, n_B^a\}$ be the index of the i-th ball and n_B^a the total number of balls in the cover for action a. Let $x(B_i^a) \in \mathcal{X}$ denote the location of ball B_i^a, that is, the location of its center in $\mathcal{X}$, and let $r(B_i^a) \in [0, 1]$ denote its radius. We say that an element $x \in \mathcal{X}$ lies in ball B_i^a, written as $x \in B_i^a$, if $d(x, x(B_i^a)) \leq r(B_i^a)$. Overloading the notation, we can identify with B_a^i also the region $B_i^a = \{x \in \mathcal{X} \,|\, d(x, x(B_i^a)) \leq r(B_i^a)\} \subset \mathcal{X}$. Let $n_t(B_i^a)$ denote the number of all the samples gathered up to time t lying in B_i^a, and let $\varrho_t(B_i^a)$ denote their corresponding sum of rewards. Let

$$\begin{aligned} \text{avg}_t(B_i^a) &:= \varrho_t(B_i^a)/n_t(B_i^a) \\ \text{conf}_t(B_i^a) &:= c \cdot \sqrt{\log T / n_t(B_i^a)} \\ \text{size}(B_i^a) &:= 2\lambda r(B_i^a) \end{aligned}$$

where c is a constant (which in practice will become a tunable parameter of the algorithm). We say that ball B_i^a is *full* (able

[1]The number of balls of radius r is trivially upper bound by the r-packing number of $\mathcal{X}$. A more tighter bound can be achieved by the near-optimality dimension or related concepts (e.g., see [5], [13]) which also take into account the specifics of the actual problem, i.e., μ^a.

to spawn a child), whenever $\text{conf}_t(B_i^a) < r(B_i^a)$.

E. CMABFAS – Algorithm

Initialization: At time $t = 0$ we initialize the individual cover for each action with a single ball: we create a ball centered on an arbitrary element $x \in \mathcal{X}$ and set its radius to 1 (such that it covers the whole space):

$$\begin{aligned} \forall a = 1, \dots, k: \ & \text{create } B_1^a \text{ with} \\ & x(B_1^a) := \text{any element of } \mathcal{X} \\ & r(B_1^a) := 1,\ n_0(B_1^a) := 0,\ \varrho_0(B_1^a) := 0. \end{aligned}$$

Every step: Now suppose that at time t x_t arrives. For each action a separately, we first compute the indices of those balls that contain x_t, which we will call *active balls* for x_t: $A^a(x_t) := \{B_i^a | x_t \in B_i^a\}$. From the set of active balls we then compute the set of *relevant balls* which consists of all balls $B_i^a \in A^a(x_t)$ which are either not full or allow the creation of a child (with radius $\frac{1}{2}r(B_i^a)$) centered on x_t such that it does not overlap (distance at least $\frac{1}{2}r(B_i^a)$) with an already existing ball at this level of the hierarchy: $R^a(x_t) := \{B_i^a \in A^a(x_t) \,|\, \text{conf}_t(B_i^a) > r(B_i^a) \vee \nexists B_j^a \in A^a(x_t) : r(B_j^a) = \frac{1}{2}r(B_i^a)\}$. For each ball in $R^a(x_t)$ we then compute its current score, which is a high-probability upper bound (see next section) for the error we make when we use the current in-ball sample average as a proxy for the true but unknown expected reward $\mu^a(x_t)$. We take the minimum over all the upper bounds as score $u(x_t, a)$ for the action in question (i.e., the tightest upper bound):

$$u(x_t, a) := \min_{B_i^a \in R^a(x_t)} \left[\text{avg}_t(B_i^a) + \text{conf}_t(B_i^a) + \text{size}_t(B_i^a)\right]. \tag{4}$$

Having such a u-score for each possible action a, we then choose the action which achieves the highest u-score:

$$a_t^* := \underset{a=1\dots k}{\text{argmax}}\ u(x_t, a). \tag{5}$$

The system executes action a_t^* and observes the stochastic outcome $r^{a_t^*}(x_t)$ which is drawn from the unknown distribution $\mathcal{R}^{a_t^*}(x_t)$. We then use this new observation to update all the balls that were active for the action chosen:

$$\begin{aligned} \forall B_i^a \in A^{a_t^*}(x_t): \quad & \varrho_{t+1}(B_i^a) = \varrho_t(B_i^a) + r^{a_t^*}(x_t) \\ & n_{t+1}(B_i^a) = n_t(B_i^a) + 1 \\ \text{all remaining balls:} \quad & \varrho_{t+1}(B_i^a) = \varrho_t(B_i^a) \\ & n_{t+1}(B_i^a) = n_t(B_i^a). \end{aligned}$$

Adaptive refinement: Having determined a_t^* and before updating, we test if the ball B^* achieving the minimum in (4) for action a_t^* is full (and thus allowed to spawn a child). If it is full, we add a new ball B^*_{child} with center x_t and radius $\frac{1}{2}r(B^*)$ and add its index to the list of active balls.

F. CMABFAS – Mathematical justification

The following section can be skipped by readers not interested in the mathematical details. We now motivate the core step in the CMABFAS algorithm and explain how, instead of storing and using all the individual samples, the average reward stored for a ball can be used to estimate the expected reward at a query point x_t (see the expression inside the min of (4)). In particular we will show the following:

Claim: Let B_i^a be an active ball for x_t, i.e., x_t lies inside B_i^a. Then the average reward stored for B_i^a, $\varrho_t(B_i^a)/n_t(B_i^a)$, is close to the expected reward at x_t, i.e., $\mu^a(x_t)$, with probability of at least $1 - T^{-2}$. More precisely, we have

$$P\left(\left|\frac{\varrho_t(B_i^a)}{n_t(B_i^a)} - \mu^a(x_t)\right| \le c \cdot \sqrt{\frac{\log 2^{\frac{1}{3}}T}{n_t(B_i^a)}} + 2\lambda r(B_i^a)\right) \ge 1 - T^{-2}. \tag{6}$$

To proof this claim, we need the following concentration measure:

Azuma-Hoeffding: Suppose Z_ℓ, $\ell = 1, 2, \dots$ is a martingale difference sequence (i.e., a sequence of random variables with the property that $E[|Z_\ell|] < \infty$ and $E[Z_{\ell+1}|Z_1, \dots, Z_\ell] = Z_\ell$) with bounded increments such that $|Z_\ell - Z_{\ell-1}| < b_\ell$ holds with probability 1 for a positive constant $b_\ell > 0$. Then for all indices k and all positive numbers L we have that

$$P(Z_k > L) \le \exp\left(\frac{-L^2}{2\sum_{j=1}^k b_j^2}\right). \tag{7}$$

Proof: In the following we will consider any fixed ball $B_i^a \in A^a(x_t)$ which is active for x_t. Let S_n be the ordered index set of samples lying in the ball, with $n := n_t(B_i^a)$ being their number (note that n itself is a random number). The sum of the rewards within the ball can then be written as

$$\varrho_t(B_i^a) = \sum_{s \in S_n(1:n)} r^a(x_s) \tag{8}$$

where $S_n(1:n)$ means looping over indices 1 to n of the set. Let

$$Z_\ell := \sum_{s \in S_n(1:\ell)} [r^a(x_s) - \mu^a(x_s)] \tag{9}$$

be the sum over the first ℓ elements in S_n. $\{Z_\ell\}_{\ell \in \mathbb{N}}$ then forms a sequence of martingales with bounded increments: $E[|Z_\ell|] < \infty$ is obvious and $E[Z_\ell|Z_1, \dots, Z_{\ell-1}] = Z_{\ell-1}$ holds because $E[r^a(x_s) - \mu^a(x_s)] = 0\ \forall s$. Moreover we have $|Z_\ell - Z_{\ell-1}| < 1, \forall \ell$. Thus we can apply A.H. to Z_ℓ; from (7) and the union bound we have that for any real number $\alpha > 0$ the two-sided inequality

$$P(|Z_\ell| \ge \alpha) \ \le \ 2\exp\left(\frac{-\alpha^2}{2\ell}\right), \quad \forall \ell \tag{10}$$

holds. Equation (10) says that the event of making an error greater than α (where "error" is the extent by which the random samples observed deviate from their true but unknown expected value) happens with probability that is at most $\delta := 2\exp(-\alpha^2/2\ell)$. Solving this for α, we obtain $\alpha = \sqrt{2\ell \log(2/\delta)}$. Plugging this α into (10) and dividing both sides of the expression inside P by ℓ, we obtain

$$P\left(\frac{1}{\ell}|Z_\ell| \ge c \cdot \sqrt{\frac{\log \frac{2}{\delta}}{\ell}}\right) \ < \ \delta \quad \forall \ell \tag{11}$$

where we use c to collect all the constants. We are free to choose the confidence δ; suppose we choose $\delta = T^{-3}$. Equation (11) is true for all $\ell \leq T$. Taking the union over all T events, we obtain with the union bound

$$P\left(\bigcup_{\ell \leq T}\left\{\frac{1}{\ell}|Z_\ell| \geq c \cdot \sqrt{\frac{\log 2^{\frac{1}{3}} T}{\ell}}\right\}\right) \leq \sum_{\ell \leq T} T^{-3},$$

which can also be written as

$$P\left(\frac{1}{n}|Z_n| < c \cdot \sqrt{\frac{\log 2^{\frac{1}{3}} T}{n}}\right) \geq 1 - T^{-2}. \qquad (12)$$

Now we want to turn (12) into a statement of how the in-ball average reward deviates from the expected reward at the query point, $\mu^a(x_t)$. First, we note that because of the Lipschitz assumption (2) we have $\forall s \in S_n$

$$\begin{aligned} |\mu^a(x_s) - \mu^a(x_t)| &\leq \lambda \cdot d(x_s, x_t) \\ &\leq \lambda \cdot \max_{x,x' \in B_i^a} d(x, x') \\ &= 2 \cdot \lambda \cdot r(B_i^a) \qquad (13) \end{aligned}$$

since both x_t and x_s lie in the same ball B_i^a with radius $r(B_i^a)$. Eliminating the absolute value and rearranging gives

$$\mu^a(x_t) - 2\lambda r(B_i^a) \leq \mu^a(x_s) \leq \mu^a(x_t) + 2\lambda r(B_i^a). \qquad (14)$$

Replacing $\mu^a(x_s)$ by the right side of (14) gives us a lower bound on Z_n:

$$\begin{aligned} |\frac{1}{n} Z_n| &= |\frac{1}{n}\sum_{s \in S_n(1:n)} [r^a(x_s) - \mu^a(x_s)]| \\ &\geq |\frac{1}{n}\sum_{s \in S_n(1:n)} [r^a(x_s) - (\mu^a(x_t) + 2\lambda r(B_i^a))]| \\ &\geq |[\frac{1}{n}\sum_{s \in S_n(1:n)} r^a(x_s)] - \mu^a(x_t)| - 2\lambda r(B_i^a). \qquad (15) \end{aligned}$$

Plugging (15) into (12) and using (8) gives

$$P\left(\left|\frac{\varrho_t(B_i^a)}{n} - \mu^a(x_t)\right| - 2\lambda r(B_i^a) < c \cdot \sqrt{\frac{\log 2^{\frac{1}{3}} T}{n}}\right) \geq 1 - T^{-2}, \qquad (16)$$

and we finally obtain the claim by adding $2\lambda r(B_i^a)$ inside P.

IV. CMABFAS for Web Server Protection

This section describes how we can map our original problem, detecting and preventing HTTP-based attacks on a web server (as described in Section I), to the general self-learning framework CMABFAS described in the previous section.

A. Defining the context space

The context space $\mathcal{X}$ is chosen to be the space of all possible HTTP sessions, which we represent by the information contained in the header of the initial HTTP request the filter observes. Specifically, we extract the fields of the Hyper-Text Transport Protocol [6] that identify the environment the user uses, i.e., web browser and operating system, and its previous

```
GET /latex2wiki/user_manual HTTP/1.1
Host: xorp.run.montefiore.ulg.ac.be
Connection: Keep-Alive
User-Agent: Mozilla/5.0 (compatible; Konqueror/4.4 ...
Accept: text/html, image/jpeg;q=0.9, image/png; ...
Accept-Encoding: x-gzip, x-deflate, gzip, deflate
Accept-Charset: iso-8859-1, utf-8;q=0.5, *; q=0.5
Accept-Language: en-US, en
Cookie: DokuWiki=svserjsogsf460sa64mtn97
```

K_x	V_x	s_A^k	$K_{x'}$	$V_{x'}$
mozilla	5.0	$\leftarrow +1 \rightarrow$	mozilla	5.0
compatible	$\emptyset$	$\leftarrow +.5 \rightarrow$	compatible	$\emptyset$
KHTML	4.4.5	$\leftarrow +.9 \rightarrow$	KHTML	4.4.7
Webkit	2.3	$\leftarrow +0 \rightarrow$		
		$\leftarrow +0 \rightarrow$	Gecko	1.7

Fig. 1. An HTTP request (top) and contribution s_A^k to $S_A(x, x')$ computation (bottom). Blue highlights points out prior knowledge of our website by the browser, yellow highlights the information the browser provides about itself.

knowledge about our website. This knowledge is observable through the URL of the initial request, the presence of a *referrer* URL (the location of the Web page the user comes from) and the presence of *cookies* associated with this request. Fig. 1 depicts such an HTTP request where it is likely that the user has already visited our site with this browser in the past. In this preliminary work, only the *URL path* of the request line (`/latex2wiki/user_manual` on Fig. 1) and the user-agent information are used to build the features of $\mathcal{X}$.

To measure distances in $\mathcal{X}$, we define two distinct similarity metrics over URL and user-agents that we later combine to obtain a single similarity measure. User-agent information is broken down into a list of *key,value* pairs, where the key is a sequence of alphanumeric characters and the value is a sequence of digits and dots. This maps common constructs such as "KHTML/4.4.5" into ($k =$KHTML$, v = 4.4.5$). Bare words in the user-agent string (e.g. `compatible`) are mapped to elements that have a special void value ($k =$compatible$, v = \emptyset$). The user-agent similarity of x and x' is then computed through

$$\begin{aligned} S_A(x, x') =& \#(K_x \cap K_{x'})/2max(\#K_x, \#K_{x'}) \\ &+ \sum_{k \in V_x} prefix(x_k, x'_k)/2max(\#V_x, \#V_{x'}) \end{aligned} \qquad (17)$$

where K_x is the set of keys of x, V_x the set of keys that have a non-void value for x and x_k the value associated with key k in x. For every such value, we normalize the length of the common prefix between v and v' over the longest chain.

To measure URL similarity S_U, we first need to extract the number of words[2] W_x, and the number of extra non-alphanumeric characters E_x in the URL[3]. $S_U(x, x')$ is then

[2]that is, uninterrupted sequence of `[A-Za-z0-9_-]` characters

[3]$E_x = 0$ when every word in x is separated by exactly one character

a weighted sum of W_x and E_x similarity, the identity comparison result between x and x' action and result codes, and the number of matching words between x and x'.

At the end, the distance $d(x, x')$ between x and x' is obtained by inverting and averaging the similarities, i.e.

$$d(x, x') := 1 - (0.5 * S_U(x, x') + 0.5 * S_A(x, x')). \quad (18)$$

Note that with this definition of d the requirement $\operatorname{diam}(\mathcal{X}) = 1$ is fulfilled.

B. Defining the actions

Unlike the modeling of the context space, where there is plenty of data available and the conceptual modeling can be tried out on the real world, the same cannot be done so easily for the actions and rewards as it would require a response from the environment which is very hard to *simulate*. For this first paper however, our experiments will have to be based solely on simulation. In the following we thus have to differentiate between how actions are modeled conceptually and how actions are modeled in our experiments.

The action the system has to decide about consists of choosing which particular security policy out of many possible ones to apply to a given session. Security policies are taken as high-level actions and affect on multiple levels how the server deals with a request. A security policy typically defines a *set of features* that will be disabled on the server (such as posting a public comment, editing a page, accessing administration interface, etc.) and a *security check* to perform before these actions could be re-enabled.

Having thus sketched how actions are to be modeled conceptually, we will now define abstracted actions for our first experiments. We define two abstract security policies called Type-1 and Type-2, where the first corresponds to a "sandbox" kind of response (where the visitor only has access to read-only, anonymous browsing, and all the "social" features of the site are silently disabled) and the second to a "throttling bandwidth" kind of response. The system can thus choose from the following three actions:

- **A0:** Apply no security policy and directly process this (and subsequent) HTTP requests unharmed.
- **A1:** Apply security policy Type-1 and process this (and subsequent) HTTP requests as dictated by the corresponding policy.
- **A2:** Apply security policy Type-2 and process this (and subsequent) HTTP requests as dictated by the corresponding policy.

C. Defining the rewards

Defining the rewards we face the same limitations as before with the actions: since we cannot *simulate* the response from the environment convincingly (without running a web server in the real world), we will have to resort to a simplified reward model for the experiments.

Defining the rewards conceptually is also the most critical modeling choice we face. The reward is a single scalar quantity that must capture the performance of the HTTP meta-filter. It has to be defined in such a way that by choosing actions which optimize it (which is what CMABFAS does), the meta-filter does what we, as its designer, want it to do. In our case here the reward has to account for two things: (1) did we make the right decision in enabling or disabling certain features; and (2) how "expensive" were the security tests necessary to arrive at this decision. While we imagine that the latter can be designed by a human expert without too much trouble, the first item poses a serious conceptual challenge: whether or not an HTTP session is controlled by a real human user is beyond the system to detect on its own accurately (otherwise we would not need the filter at all) and would require a human moderator to correctly label. However, we would rather like our meta-filter to be able to detect *by itself* if a request is initiated by a human or a bot (and thus generate the internal reward appropriately).

To achieve this goal, we propose to generate the reward from the session metrics such as the number of requests, the distribution of inter-request time and the number of "supporting" content that has been requested (such as style sheets, logos and other non-interactive pages). For example, for regular browsing content is requested in a logical sequence and a certain amount of time passes between the individual requests (the time it takes for a human to absorb the information and click on a new link). The browsing behavior of some automated software (not crawlers), however, is characterized by not following the logical sequence, requesting pages which do not exist, and very low inter-request time. Inferring from the session metrics what "type" of request we face is, however, unreliable and only an estimate: it is the strength of the learning framework we use to be able to deal with the inherent uncertainty (i.e., stochasticity) of the feedback.

Having thus sketched how rewards are to be modeled conceptually, we will now say how we will define concrete rewards for our experiment. Our experiments will be based on data extracted from the log file of a real web browser (see below). As we have described above, each such HTTP request constitutes a context $x \in \mathcal{X}$, and we can choose from the three actions A0,A1,A2. Before running the experiment, each HTTP request was internally mapped to one out of 10 classes (see below). Our reward generation is then only based on this class: for each class and each action we assign a parameter $\in [0, 1]$ which is the mean of a Bernoulli distribution. To generate the reward for context x_t under action a_t, we thus first find the class x_t belongs to, read the parameter from the table that corresponds to this class and action a_t, and then sample from this parameter a Bernoulli random number. *Note that CMABFAS does not "know" the precise mechanics of how the reward is generated; it can only observe the samples.*

D. Setting up the experiment

Our experiments are based on a dataset obtained from a real, locally hosted Web server that offers web mail, two wiki sites (one private and one public) and one blog where user frequentation is fairly modest, but still diverse in terms of software platforms. We collected 57,389 HTTP requests as logged by the Apache Web server between January, 10th and December, 31st 2011. Over this time period, we recorded – among a majority of regular traffic – several scans for vulnerable versions of the installed packages, remote code injection attempts [12], and instances of spam defacing the public wiki (reported by community users).

The 57,389 requests were grouped into 1126 sessions by an automated process that uses both source IP address and user-agent information to identify consecutive requests. To assist in the classification of those requests, a first step of automated processing extracted sessions that requested files that are typical of either indexing robots (i.e. `robots.txt`) or common web-browsers (i.e. `favicon.ico`). Traffic originating from within our internal network was easily identified and labeled as "internal". A series of simple pattern matching rules then extracted scanning and vulnerability exploit attempts on popular Web services. At the end, each class was carefully reviewed by a domain expert to avoid the absence of outsiders.

V. Simulation Results

A. Setup

To create the context space, we generated a corpus of 1126 HTTP requests and assigned each to a reward model as described in the previous section. Doing this gave us a corpus of requests out of which 99 are labeled as *robots*, 122 as *bad guys*, 3 as *break in*, 104 as *Likely real users*, 12 as *Internal tests*, 9 as *misconf*, 192 as *scanning*, 51 as *Students and team*, 461 as *Visitors* and 73 as *likely visitors* (labels starting with a capital letter identify classes for which action $A0$ would be the preferred one). As described above, the labels were assigned by a domain expert through inspection of features of the session as a whole, and are only used to internally generate the reward–they are not directly observable by CMABFAS.

Every time we simulate a new request x_t arriving at the web server, we draw a random index uniformly from this corpus, present the associated request to the CMABFAS learner, record the action CMABFAS recommends and generate stochastically a reward (which depends on both the request, i.e., the state, and the action) and return the reward back to CMABFAS.

Overall we examine three scenarios: one where the HTTP meta filter has to choose among three different actions A1,A2,A3, one where it has to choose among ten different actions A1,. . .,A10, and one where new data is injected in the middle of learning, simulating novel attack patterns. The three action scenario was described in the previous section, the ten action scenario is meant to illustrate the scaling capabilities of our algorithm and is obtained by just adding more (at this point, artificial) choices to the disposal of the filter. Note that increasing the number of choices makes the task of finding the best choice more difficult in that it requires more exploration. The novel attack scenario is generated by dividing the original data set into two separate parts, one used only for training (the size of this set was varied in the experiments and chosen from {50,100,250,500,750}) and one used for the novel attacks (this set was set aside first and was of fixed size 376). For the first half of the learning stage, which was 500,000 steps, only samples from the training set are drawn; in the second half only samples from the novel attacks set are drawn.

The remainder of the simulation parameters were as follows. The Lipschitz constant λ from Eq. (2) is set to 0.1. We perform a total of 10 independent runs and average the results; each single run consists of sequentially processing 1,000,000 independent HTTP requests.

B. Baseline

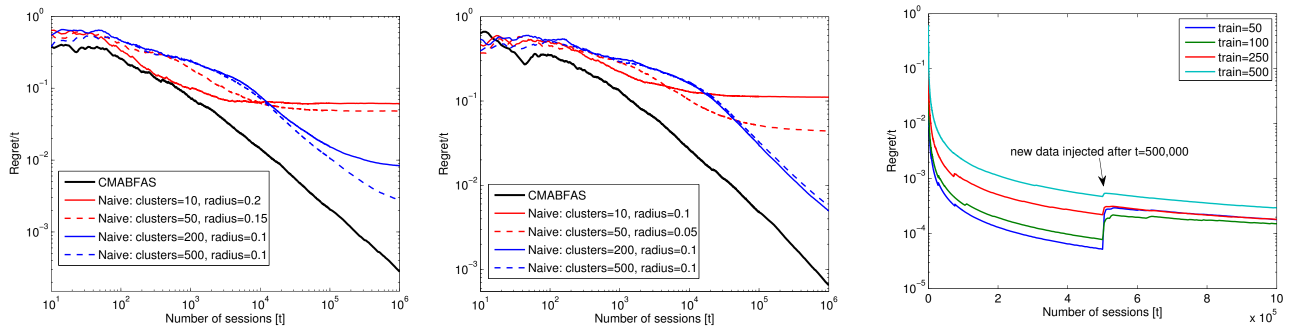
To properly evaluate the performance our algorithm CMAB-FAS, we define a naïve baseline method which works by incrementally (but non-adaptively) clustering the input space $\mathcal{X}$ and assigning a standard MAB with UCB$_1$(1.2) rule [2] to each cluster. Specifically, it works like this: let x_t be the current context. Find the nearest cluster according to the distance defined in (18). If the distance to the nearest cluster is greater than some parameter `max_radius` and the number of current clusters is below some other parameter `max_clusters`, we add a new cluster and initialize its counter to zero. Otherwise we assign x_t to the nearest cluster with index i^* and choose action a^* such that

$$a^* = \operatorname*{argmax}_a \; \frac{\varrho_t(i^*, a)}{n_t(i^*, a)} + \sqrt{\frac{1.2\log(n_t(i^*, a))}{n_t(i^*)}},$$

where $\varrho_t(i^*, a)$ is the sum of rewards for action a, $n_t(i^*, a)$ the number of samples for action a, and $n_t(i^*)$ the total number of samples, all with respect to cluster i^*. Choosing a^*, we observe, as before, a reward $r^{a^*}(x_t)$ after which we increment $\varrho_t(i^*, a^*)$, $n_t(i^*, a^*)$, and $n_t(i^*)$ accordingly. The hyperparameters of the algorithm, `max_radius` and `max_clusters`, were chosen by a coarse grid search: best performance was achieved for `max_radius`=0.1 and `max_clusters`=200 (our results also include some other settings of `max_clusters` and the respective best `max_radius` parameter.)

C. Results

The resulting performance of both CMABFAS and the baseline is shown in Figure 2 in terms of the cumulative regret (the regret can be evaluated because in this simulation experiment the reward distributions $\mathcal{R}^a(x)$ are known to us). The first panel shows that CMABFAS reaches a regret of about 0.0002 (which translates into about 450 out of 1,000,000 instances of not choosing the optimal action) whereas the best baseline setting achieves only 0.002 (which translates into about 3500 instances of not choosing the optimal action). The second panel shows that CMABFAS, along with the baseline, scales well when the number of available actions is increased. The third panel shows how CMABFAS is able to generalize over the context space and rapidly adapt to novel and previously unseen data; the vertical size of the step indicates how much performance is lost and how long it takes to absorb new knowledge into the system.



3 available actions   10 available actions   injecting new data

Fig. 2. Comparing CMABFAS with naïve clustering and standard UCB$_1$ MAB. Lower numbers indicate better performance.

VI. Conclusion

Defense of web servers running dynamic applications such as blogs and Wikis is traditionally performed by a mixture of a signature-based Intrusion Detection System (IDS) and a set of human interaction proof plug-ins such as the infamous CAPTCHAs. Both of these protection measures are highly vulnerable against innovation in black-hat hacking software, stressing the service maintainers to frequently upgrade their systems, but also frequently reverting their database to a former snapshot to undo the havoc caused by abusers of their service.

In this paper we have undertaken first steps towards making a complex decision-making HTTP meta filter become fully self-learning by formulating it as a contextual multi-armed bandit. The meta-filter can integrate several existing security measures, among which it has to choose without having access to prior labeled data. It does so based only on stochastic and sparse feedback.

The simulation results we present in this paper are encouraging: using solely similarity measures between URL and user-agent strings and internally generated rewards, we can correctly identify and filter out a large fraction of unwanted traffic and also quickly adapt to novel attack patterns. Building on this proof of concept, our future work will then have to answer the still open questions of (1) how to define the internal reward generator (which at this point is only *emulated* by an artificial Bernoulli distribution); and (2) how to define the actions and security policies. It then remains to examine with live interaction between end-users, meta filter, and application code running on the Web server itself the feasibility of our approach.

Acknowledgments

Sylvain Martin acknowledge the financial support of the Belgian National Fund of Scientific Research (FNRS). Tobias Jung acknowledges financial support from a ULg research fellowship. This work is also partially funded by EU project ResumeNet, FP7–224619.

References

[1] A. Ahmed and I. Traore. A new biometric technology based on mouse dynamics. *IEEE Trns. on Dependable and Secure Computing*, 4(3):165–179, 2007.

[2] J.-Y. Audibert, R. Munos, and C. Szepesvári. Tuning bandit algorithms in stochastic environments. In *Proc. of the 18th Int. Conf. on Algorithmic Learning Theory*, 2007.

[3] P. Auer, C. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002.

[4] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *Proc. of ICML*, 2006.

[5] S. Bubeck, R. Munos, G. Stoltz, and C. Szepesvari. Online optimization in X-armed bandits. In *Adv. in Neural Inf. Proc. Syst.(NIPS)*, pages 201–208, 2008.

[6] R. Fielding and U. I. et al. RFC2616: Hypertext Transfer Protocol – HTTP/1.1, 1999.

[7] A. György, L. Kocsis, I. Szabó, and C. Szepesvári. Continuous time associative bandit problems. In *Proc. of IJCAI'07*, pages 830–835, 2007.

[8] R. Kleinberg, A. Slivkins, and E. Upfal. Multi-armed bandits in metric spaces. In *40th ACM Symp. on Theory of Computing (STOC)*, pages 681–690, 2008.

[9] T. L. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Adv. in Appl. Math*, 6:4–22, 1985.

[10] T. Lu, D. Pál, and M. Pál. Showing relevant ads via lipschitz context multi-armed bandits. In *Proc. of AISTATS 14*, 2010.

[11] P. Rigollet and A. Zeevi. Nonparametric bandits with covariates. In *COLT*, pages 54–66, 2010.

[12] D. Secunia ApS. Secunia advisory sa39556, phpthumb() 'fltr[]' command injection vulnerability – cve-2010-1598, 2010.

[13] A. Slivkins. Contextual bandits with similarity information. *CoRR*, abs/0907.3986, 2009.

[14] L. von Ahn, M. Blum, and J. Langford. Telling humans and computers apart automatically. *Commun. ACM*, 47:56–60, February 2004.

[15] C.-C. Wang, S. R. Kulkarni, and H. Poor. Arbitrary side observations in bandit problems. *Advances in Applied Mathematics*, 34(4):903–938, 2005.

[16] Y. Xie and S.-Z. Yu. A large-scale hidden semi-markov model for anomaly detection on user browsing behaviors. *IEEE/ACM Transactions on Networking*, 17(1):54–65, 2009.

[17] X. Xu. Sequential anomaly detection based on temporal-difference learning: principles, models and case studies. *Applied Soft Computing*, 10:859–867, 2010.

[18] N. Ye, Y. Zhang, and C. M. Borror. Robustness of the markov-chain model for cyber-attack detection. *IEEE Transactions on Reliability*, 53(1):116–123, 2004.

[19] D. Y. Yeung and Y. X. Ding. Host-based intrusion detection using dynamic and static behavioral models. *Pattern Recognition*, 36:229–243, 2003.