Speculative Authorization

Pranab Kini and Konstantin Beznosov

***Abstract*—We present *S̲peculative A̲uthorization̲* (SPAN), a prediction technique that reduces authorization latency in enterprise systems. SPAN predicts requests that a system client might make in the near future, based on its past behavior. SPAN allows authorization decisions for the predicted requests to be made before the requests are issued, thus virtually reducing the authorization latency to zero. We developed SPAN algorithms, implemented a prototype, and evaluated it using two real-world data traces and one synthetic data trace. The results of our evaluation suggest that systems employing SPAN are able to achieve a reduced authorization latency for almost 60% of the requests. We analyze the tradeoffs between the hit rate and the precision of SPAN predictions, which directly affect the corresponding computational overhead. We also compare the benefits of deploying both caching and SPAN together, and find that SPAN can effectively improve the performance of those systems which have caches of a smaller size.**

***Index Terms*—Access control, machine learning, prediction.**

I. Introduction

With the emergence of tighter corporate policies and government regulations, access control has become an integral part of business requirements. Modern access control architectures [1] follow the request-response model (dotted lines in Figure 1). In this model, a request from a client (subject) is intercepted by the policy enforcement point (PEP), which converts this request to an authorization request, and forwards it to the policy decision point (PDP). The PDP, with the help of the policy database, computes an authorization response and sends it back to the PEP for enforcement.

Resource access latency directly affects end user experience, and as a result, company revenue. Neilsen suggests that a response time of greater than 100 milliseconds (ms) makes end users feel that the system is not responding instantaneously [2]. A study by Amazon reported an approximate 1% loss in sales as the cost of a single extra 100 ms delay. Another study by Google found that an extra 500 ms delay in displaying the search results may reduce its revenues by up to 20% [3]. Computing an authorization response can take from a few milliseconds to several seconds [4], [5]. Thus, although on the one hand, authorization is imperative for securing protected resources, on the other hand, it increases latency, hampering systems' responsiveness and ultimately the revenue of the organization.

One straightforward approach would be to compute the list of all accessible resources, as soon as the user logs in, and cache them; a technique used in classical capability list based systems. This approach is most appropriate for systems with fairly small and static collection of resources, and permission mechanisms that allow inexpensive queries of the accessible resources. However, it breaks down in most systems based on access control lists and other similar permission mechanisms where the list of authorized users can only be queried through the reference to the corresponding resource. It is also suboptimal for systems with very large or highly dynamic collection of resources.

The authors are with the Department of Electrical and Computer Engineering, University of British Columbia, 2332 Main Mall, Vancouver, BC V6T 1Z4. E-mail: pranabk, beznosov@ece.ubc.ca.

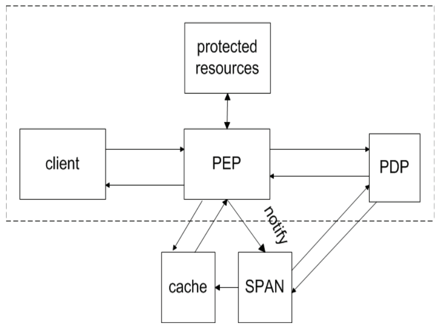


Fig. 1. Architectures of enterprise access control systems and SPAN

To address the need for reducing authorization latency (referred in this paper as just *latency*), we propose *S̲peculative A̲uthorization̲* (SPAN), a technique that predicts which future requests are likely to be made by a subject in a session. A *'session'* is defined as the time period between a subject logging in and out of the system. As shown in Figure 1, before the PEP sends authorization requests to the PDP, it (1) notifies SPAN of the new request, and (2) checks the cache to see if it already contains the corresponding response. Based on the series of received requests, SPAN is able to predict which future requests would likely be made by the subjects, and sends the predicted requests to the PDP. SPAN places the responses received from the PDP into the cache. If the subjects make the requests predicted by SPAN, the PEP finds the corresponding responses in the cache, reducing the latency to cache lookup. Otherwise, it sends the request to the PDP as usual.

In designing SPAN, we used concepts similar to those employed by web page prefetching techniques [6], [7], [8], [9], [10], [11], [12], [13], [14]. However, none of these algorithms consider the identity of the subjects making those requests. In enterprise authorization systems, authorization policies restrict the access of the subjects to a subset of resources, which influence the requests made. The likelihood of requests made on resources would not only depend on the resources themselves, but also on the authorization policies governing them, which introduces an additional constraint for prefetching. This constraint is taken into account in SPAN's design by associating the identity of subjects to their requests, differentiating SPAN from the approaches for web page prefetching.

To make predictions, SPAN forms Markov chains of all the possible sequences of requests and probabilistically (1) clusters the sequences and (2) associates subjects to the clusters. To achieve the association and clustering, SPAN extends the Latent Dirichlet Allocation (LDA) model [15], developed for unigrams, by adopting it to Markov models. Our approach of finding associations of subjects to the clusters differentiates our technique from clustering in web page prefetching [8], [9] where users are randomly assigned to the clusters.

We evaluated SPAN using two datasets of request traces from real-world systems and one synthetic dataset. Our first dataset (WebCT) contained requests made by students, teaching assistants, and course instructors in WebCT [16] for a course at our university. Different subjects had different rights for the course materials. We obtained the second dataset (FC) from requests made by users in the 'Fighters Club' application of Facebook [17]. In this application, users could start a virtual fight with their friends, and request help from other friends. This represents an application system where subjects (users) can only request resources (friends) for which they have rights. We generated our third dataset (Zipf) using Zipf distribution, which was shown to closely match real world distributions of web page requests [18].

We compared SPAN with the first and second order Markov models, as well as with the algorithms proposed by Cadez et al. [8] and Deshpande and Karypis [7]. For FC, SPAN achieved an improvement of 11%, 31%, 21%, and 23% in hit rate as compared to other algorithms. The corresponding improvement for WebCT was 2%, 21%, 2%, and 11%. We believe that the increase in the hit rate is due to the combination of both association and clustering in our technique. Our results also demonstrate that higher hit rates are directly proportional to reduced latency. The percentage of requests experiencing reduced latency is less than or equal to the hit rate obtained.

We also evaluated SPAN when it is combined with caching. Simulation results suggest that SPAN improves the performance of systems considerably for smaller sizes of cache. We also found that the performance is significantly higher for FC as compared to WebCT.

To summarize, the contributions of this paper are as follows:

1) We proposed SPAN, an approach for predicting future authorization requests that takes authorization policies into account without having direct access to the policies.
2) We compared SPAN with existing web prefetching algorithms, and found that SPAN performs better than those algorithms.
3) We evaluated SPAN when it is combined with PEP-side caching and found SPAN to be more effective than caching for smaller cache sizes.

The rest of the paper is organized as follows: Related work is presented in Section II. In Section III, we present the shortcomings of the algorithms in web page prefetching, and present SPAN. We discuss the evaluation of SPAN in Section IV. Finally, in Section VI, we conclude the paper, and discuss future work.

II. Related Work

One approach to combating latency is to replicate the authorization service components. This also increases availability, but such systems scale poorly, and become technically and economically unfeasible when the number of entities in the system reaches thousands [19].

Another state-of-the-practice approach that is used to improve latency is the caching of authorization decisions at the PEP-side. If a subject makes the same request as before, the response is fetched from the PEP cache, reducing the latency to cache look-up. However, it can be effective only in those cases where the same subjects repeatedly make the same requests. To overcome this problem, Crampton et al. [20] propose the Secondary and Approximate Authorization Model (SAAM), where cached responses are stored at a secondary decision point (SDP), collocated with the PEP. The SDP infers responses to requests that do not have their responses stored in the cache. The idea is further explored by SDP's co-operating with each other to make decisions [21]. SAAM algorithms for computing secondary authorizations for policies based on the Bell-Lapadula model [22] and RBAC [23] have been also proposed [20], [24]. The main limitation of the SAAM approach is that it requires different inference algorithms to be developed for each policy model, whereas the SPAN algorithm is agnostic to the authorization logic used by the PDP.

Kohler and Schaad [25] propose an architecture for predicting the actions required to complete the business processes in enterprises. Their approach is based on the assumption that the execution of every business process is comprised of certain predefined sub-processes. As soon as a business process is started, permissions for all sub-processes within that process are computed, reducing latency. While their architecture depends on defining all the possible sub-processes for every business process, our approach probabilistically finds the dependencies between requests without any prior knowledge of the business processes.

Several predictive models have been proposed for web page prefetching [6], [7], [8], [9], [10], [11], [12], [13]. All the approaches follow batch learning [26], where datasets are divided into distinct training and testing sets. In the training phase, a prediction model is developed using the training set. In the testing phase, the surfing pattern of users in the testing set is compared to the prediction model, and the web page(s) that are most likely to be accessed are predicted. However, there is a possibility that the behavior of users might change over time, and this is something which cannot be captured by batch learning. If training is performed using old behavior, the prediction of requests that follow the new behavior might suffer from a low predictive capability. To capture the changing behavior of the subjects, iterative learning could be adopted, in which feedback is provided for correcting the parameters of the prediction model according to changing behavior of the subjects. We adopted an iterative learning technique, where we first train the model using requests from the training set. Then, we use requests from the testing set to not only make predictions, but also to run the training phase again in order to adjust the model incrementally to the changing behavior of the subjects.

We now summarize the most relevant approaches to web page prediction. Techniques that use Markov models [7] and its variants [6], [7], [8] are used to find the popularity of web surfing patterns. Deshpande and Karypis [7] and Sen and Hansen [9] developed first, second, and third order Markov models for predicting web pages. They found that increasing the order of models results in better predictions, but that they encounter higher memory usage and increased state-space complexity. Deshpande and Karypis [7] proposed three pruning techniques—(1) frequency pruning, (2) confidence pruning, and (3) error pruning—that are combined with Markov models to overcome the problems. Their reported evaluation results [7] do not demonstrate a significant improvement in predictive capabilities, but their ideas offer some guidance on how to reduce search space and improve confidence in making predictions. A technique that combines Markov models and Support Vector Machines (SVM) is proposed by Awad et al. [6]. However, longer training times— up to 26 hours for 23,028 requests [6]—can be a bottleneck for their algorithms. In comparison, our SPAN algorithm completes the training for 50,000 requests in under two hours.

We adopted the clustering technique proposed by Cadez et al. [8], which clusters sequences of requests formed using first order Markov models. While their approach randomly assigns users to clusters, SPAN, in contrast, probabilistically finds the association of subjects to clusters, implicitly taking access control policies into account.

A number of prediction techniques based on n-grams are proposed for finding popular surfing patterns of web pages [10], [14]. Su et al. [10] propose an algorithm for gram size greater than or equal to 4. The request frequency counts need to be large in order to attain better predictive capability using n-grams [7], and they also face the problems of state-space complexity and low coverage. Bonnin et al. [11] propose to skip several places in longer n-grams resulting in lower order Markov models, but they don't address state-space complexity. Association rule mining has been used to find popular surfing patterns in web pages [12], [13]. Yet, to have confidence that the predicted surfing patterns will be sufficiently accurate, these techniques need large frequency counts of request sequences.

To summarize, access to protected resources, unlike public web pages, is controlled by authorization policies. As a result, different subjects might have different access rights over the same resource. Resources accessible to certain subjects might not be even accessible to others. Models used for training web pages consider only the resources, but not the identity of subjects. On the contrary, the SPAN prediction algorithm not only considers the resources, but also takes into account the identity of subjects who request access to those resources.

III. Problem Formalization and Approach

SPAN algorithms are divided into two parts: training and testing. In the training phase, the sequences of requests made by subjects in the past are clustered. These sequences are used to make predictions in the testing phase.

A. Training phase

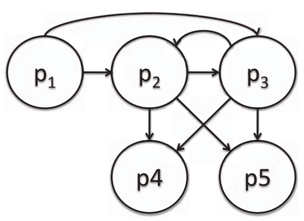
In the training phase, SPAN first obtains all the possible sequences of requests made by the subjects in the past. Next, it splits the sequences into subsequences of fixed length, and counts the number of times these subsequences were repeated. It also counts the number of times every subject requested these subsequences. Finally, it clusters the subsequences based on these two counts. In the next subsection, we explain the need to split into subsequences, and to maintain two different sets of counts for clustering in SPAN. Afterwards, we describe our clustering approach.

1) Building Blocks: To explain the rest of SPAN's design, let us consider a hypothetical website with five web pages, access to which is controlled by authorization policies. The link structure is shown in Figure 2(a), in which an arrow between pages p_i and p_j indicates that there exists a hyperlink on p_i that points to p_j. Figure 2(b) represents the authorization policy for our example.

Figure 2(c) represents different sequences of web pages visited by the three subjects over five sessions. It can be seen that Alice has requested pages in sequence p_2, p_3, p_4 only once, but subsequence p_2, p_3 repeats in all of her sessions. Such shorter subsequences have higher frequency counts. As higher frequency counts improve the predictive capabilities of Markov chains [7], [9], SPAN splits longer sequences into smaller subsequences of fixed length. SPAN forms a first order Markov model, an instance of which in our example is shown in Figure 2(d). Every cell in the table, represents the frequency count of transitions made by subjects. The last column represents the total number of times subsequences were requested by all subjects.

Unlike SPAN, algorithms for web page prefetching develop their predictive models using frequency counts only from the last column. While this column shows the popularity of subsequences in general, it does not provide information about the behavior of individual subjects. For example, let us suppose that Bob requests p_3. Algorithms designed using total frequency counts would predict p_5 as one of the probable pages for Bob, because the total frequency count of viewing p_5 after p_3 is the same as for p_2 and p_4. However, Bob has never requested to view p_5 (probably because he is not authorized, see Figure 2(b)), and the chances of him requesting this page in the future are minimal. As web page prefetching algorithms do not consider the identity of the subjects, they cannot take individual access patterns in their prediction models into account. SPAN algorithms, on the other hand, do take into account the transitions made by individual subjects. To maintain high-frequency counts while taking the behavior of individual subjects into account, SPAN combines frequency counts of transitions made by individual subjects and the total frequency counts in clustering request subsequences.

2) Clustering: By definition of conditional probability for first order Markov models, the probability of a subject 'u_m' requesting a subsequence 'y_t' of length J is shown in Equation 1.



(a) Link structure

Pages	Alice	Bob	Mike
p_1	allow	allow	allow
p_2	allow	allow	allow
p_3	allow	allow	allow
p_4	allow	allow	*deny*
p_5	allow	*deny*	allow

(b) Access matrix

Session	Alice	Bob	Mike
s_1	p_1, p_2, p_3	p_1, p_2, p_3	p_1, p_3, p_2
s_2	p_2, p_3, p_4	p_2, p_3, p_4	p_2, p_3, p_5
s_3	p_2, p_3, p_5	p_2, p_3, p_4	p_2, p_3, p_5
s_4	p_1, p_2, p_3	p_1, p_3, p_2	p_1, p_2, p_3
s_5	p_1, p_2, p_3	p_1, p_3, p_2	p_1, p_2, p_3

(c) Sequence of requests made by the subjects

Transition	Alice	Bob	Mike	Total
p_1, p_2	3	1	2	6
p_1, p_3	0	2	1	3
p_2, p_3	5	3	4	12
p_3, p_2	0	2	1	3
p_3, p_4	1	2	0	3
p_3, p_5	1	0	2	3

(d) Transitions made by the subjects using first order Markov models

Fig. 2. Hypothetical web site, authorization policy, request sequences, and frequency counts for first order Markov model

$$p(y_t|u_m) = p(p_{1_t}|u_m) * \prod_{j=2}^{J} [p(p_j|p_{j-1}, u_m)] \tag{1}$$

As observed in Equation 1, first order Markov models predict the next permission based on the current permission being requested by the subject and fails to capture the history of past requests in a subsequence. Clustering the request subsequences retains the benefits of the first order Markov model, while gathering information on the history of accesses for making better predictions [8]. We cluster subsequences of requests using frequency counts for individual subjects, as well as total counts (see Figure 2(d) for an example).

We assume that there are a total of M subjects, N permissions, T subsequences, and K latent clusters formed. With latent clusters in the model, equation 1 would be

$$p(y_t|u_m) = \sum_{k=1}^{K} p(p_{1_t}|z_k) * \prod_{j=2}^{J} p(p_j|p_{j-1}, z_k) * p(z_k|u_m) \tag{2}$$

The likelihood of all the subsequences and the clusters, represented by Y and Z respectively, is given by,

$$p(Y, Z|\theta, \Phi) = \prod_{k=1}^{K} \prod_{t=1}^{T} \prod_{m=1}^{M} [p(p_{1_t}|z_k)^{N_{1,k}} * \prod_{j=2}^{J} p(p_j|p_{j-1}, z_k)^{N_{j,j-1,k}} * p(z_k|u_m)^{N_{m,k}}] \tag{3}$$

We adopt a Bayesian approach [27] and attach priors α and β to the two parameters θ and Φ, which denote the association of subjects and subsequences to the clusters. Clustering the subsequences into K factions is achieved using the Expectation Maximization (EM) algorithm [28]. The parameters are optimized in two steps: the expectation step (E-step) calculates the probability of subjects requesting subsequences from particular clusters, given the current estimates. The maximization step (M-step) optimizes these parameters using the probabilities calculated in the E-step. To make the calculations more tractable, we use the log of probabilities (log-likelihood), which operates with probability sums. The memory requirements for clustering are $O(M \times T + N^2)$ and the time complexity is $O(I \times M \times T \times N^2)$. A detailed explanation of our notation and clustering technique using the Bayesian approach can be found in Section Sup:II-B.[1]

We summarize the design of the training phase using the following steps:

1) Given all sequences across various sessions, find unique subsequences of a fixed size.
2) Count the number of times each subsequence was requested by every and all subjects.
3) Execute the EM algorithm, which can be summarized as follows:
    a) Randomly initialize the values of z_k for the E-step and choose the values of the hyperparameters (α-s and β-s).
    b) Calculate the parameters θ and Φ in the M-step from the values obtained in E-step.
    c) Using the new parameters of the M-step, execute the E-step, i.e., the probabilities of subjects requesting for subsequences from different clusters.
    d) Calculate the sum of log-likelihoods. If the difference between the log-likelihood of current iteration and previous iteration does not exceed the predefined threshold, terminate the algorithm, or else go to step b.

B. Testing phase

Once SPAN has observed that u_m requested the sequence $y_t = \{p_1, \ldots, p_J\}$ of length J, it computes the probability of y_t and u_m being associated with each cluster z_k: $p(z_k|y_t, u_m) \propto p(y_t|z_k) * p(z_k|u_m)$. SPAN computes the probability of each permission in the system to be requested next by subject u_m, given the previously requested sequence y_t and a set of latent clusters:

$$p(p_{J+1}|y_t, u_m) = \sum_{k=1}^{K} p(p_{J+1}|z_k) \cdot p(z_k|y_t, u_m) \tag{4}$$

SPAN selects a pre-configured number of permissions with the highest probabilities, which are used for precomputing authorization responses to be cached by the PEP.

So far, we have discussed the batch learning approach where clusters are formed only once. Its predictive performance could

[1]References to all figures, tables, and sections from the supplementary document are prefixed with "Sup:". The reader can obtain this supplementary document from the publisher's web site.

significantly degrade if access patterns significantly change subsequent to the training phase. To overcome this problem, we adopted an iterative technique of retraining the model after a batch of requests have been read from the testing set. We adopted this strategy since a single request is unlikely to result in dramatic change to model parameters, and model rebuilding is relatively expensive.

IV. Evaluation

We used a simulation based approach for evaluating SPAN. In this section, we first describe the datasets obtained for evaluation, followed by an explanation of our experimental setup. Then, we describe the measurement criteria, and present results.

A. Datasets

To evaluate SPAN, we needed datasets that contained information about subjects requesting actions on protected resources in different sessions. We chose not to use traces of access to publicly available web pages, such as [29]. In such datasets, all users have access to all pages, whereas in systems with protected resources, subjects differ in their rights to access those resources. These differences affect the request patterns. As a result, we used traces of requests made on protected resources by different subjects.

Our first dataset was a WebCT request trace from a distance education course conducted at the University of British Columbia (UBC). WebCT [16] is a commercial online learning environment utilized in 80 countries by colleges and other institutions with a user base of over 10 million. Instructors can add course lecture notes and add tools, such as discussion boards, mail, an assignment system, and live chat. Instructors can also collect and grade assignments using WebCT. Students who are registered for the course can read lecture notes, participate in the discussions, view and submit their assignments, view their grades, exchange mail, and chat with other registered students, as well as the course teaching assistant(s) and the instructor(s). There are also other roles such as course designer, administrator, etc., that can be added to the course. WebCT is an example of a typical web application, where users have different levels of access to the system resources.

We obtained an anonymized trace of 210,000 requests for one course offered at UBC over four months, from January to April, 2007. Each record in the trace contained the subject id, its action, the session id, time of the request, the resource name, and the subject's role. The course had 11 instructors (but only 5 made more than 500 requests, and only 2 of the 5 were active throughout the entire course), 3 teaching assistants, and 42 students, for a total of 56 subjects. Different subjects could perform different actions on WebCT resources. For example, instructors could read and modify course materials, whereas students could only read them. In the WebCT dataset, we counted 4,696 unique permissions, each permission comprised of resource and action. The distribution of requests by different roles was not spread evenly throughout the dataset. The first 15,000 requests were mostly made by instructors who were setting up the course structure and content. The rest were requests by students, TAs, and instructors. To understand the effect of different access patterns on SPAN performance, we evaluated SPAN using two sub-traces (the first 100K requests, and requests between 75K and 175K), in addition to the whole WebCT dataset.

We obtained our second dataset from requests made by the users in the Fighters Club (FC) application of Facebook [17].[2] This game has been played by over 3.44 million users. FC allows users to pick virtual fights with their Facebook friends and select fight durations lasting from 15 to 48 hours. During the fight, each player may request support from their Facebook friends, who then help the individual's team defeat the opponents through a series of virtual hits. In each fight, a user can have one of the three possible roles: (1) an *offender*, who instigates the fight with a (2) *defender*, and (3) a *supporter* who helps either offender or defender (but not both) in the fight. It was important for our evaluation that every fight has a unique id. Our FC dataset contained over 23 million requests made by 43,669 users. The FC trace was appropriate for SPAN evaluation because it had protected resources, subjects, and sessions. Users can be considered as subjects, and fight ids as their sessions. In every fight, users receive help from some of their Facebook friends. The order in which offender or defender receives help can be considered a sequence of requests made by subjects in their sessions on protected resources.

We synthesized our third dataset using Zipf distributions, which have been widely used to model heterogeneous popularity distributions (e.g., web site popularity [18], and query term popularity). The simulated dataset had a total of 100,000 requests made by 100 subjects over 3,000 permissions. Every session had 100 requests. We varied the value of α from 0 (uniform distribution) to 1.5 in steps of 0.5.

B. Experimental setup

In this section, we present experimental settings. We conducted all our experiments on a machine with Intel Pentium 1.73 GHz dual-core processor, which had a cache memory of 1 MB and a RAM memory of 2 GB. SPAN training and testing phases were implemented in Matlab version $2009a$.

In the training phase, SPAN counted the total number of unique requests in the training set, and recorded all subsequences of length three. For each subsequence, SPAN recorded the number of times each subject requested the subsequence and the total number of times the subsequence occurred. SPAN also recorded all transitions between the requests (i.e., subsequences of length two).

To select the priors α and β, we conducted preliminary experiments with both priors having values of 0.1, 0.3, 0.6, and 1, and found that the relative difference between log-likelihoods of clustering was less than 5%. Thus, we decided to run all our experiments with both priors equal to 0.3. We varied the number of clusters between 1 and 10, in steps of 1, and measured the log-likelihood for every run. For the testing phase, the number of clusters that we chose was the number that provided the maximum log-likelihood during convergence

[2]This dataset is available online at [30].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

in the training phase.[3] We followed the convergence criteria of Cadez et al. [8], which assumes that the algorithm has converged if the difference of log-likelihoods between two iterations of EM algorithm is less than 0.1%. If it failed to converge in 25 steps, we restarted the training phase with a new random assignment of probability values in E-step. We found that four clusters for WebCT and Zipf distribution, and seven for FC were optimum. As we started the clustering algorithm with random assignments, two runs in the training phase were used to confirm that unique clusters were formed each time.

In the testing phase, the evaluation engine read the requests from the testing set sequentially. It then forwarded a copy of each request to the SPAN module, which predicted future requests based on the sequences of past requests. Based on its configuration parameters and requests available in the cache, the evaluation engine computed the predictive performance of SPAN. We discuss the configuration parameters and cache in Sections IV-B1 and IV-B2. We conducted two runs of the testing phase to measure SPAN's predictive performance.

For each of the WebCT and FC datasets, Table I provides details about the number of subjects, unique permissions, and unique sequences formed. The table also lists the average number of iterations required during the training phase, and also to make a prediction in the testing phase. Table I indicates that while the training time increases with the number of subjects and sub-sequences, the prediction times remain short.

We divided each trace into training and testing sets, with the former always comprising the first part of the trace. To analyze the effect of different proportions between training and testing sets on SPAN performance, we varied the size of the training set from 50% to 90% of the trace.

1) Configuration parameters: We set two configuration parameters in the evaluation engine to determine for which of the predicted requests the PDP should compute responses. In any run, the evaluation engine used only one of these two parameters:

- When the engine used the *top-n* parameter, it selected only n requests with the highest probabilities (ignoring the rest), and passed them to the PDP for computing corresponding responses.
- When the evaluation engine used the *confidence level* parameter, it selected only the requests whose probabilities were above this preset threshold value.

The use of top-n requests, as well as the setting of different confidence levels allowed us to study tradeoffs between the gain in predictive performance and the additional load on the PDP.

2) Cache implementation: We combined conventional caching and SPAN to compare the performance of each technique, as well as the two in combination. In conventional authorization subsystems, responses are cached by the PEP. Since the SPAN testbed did not include a PEP, the evaluation engine cached the incoming requests, i.e., the requests it read from the testing set. To simulate the limited size of the PEP cache, our cache also had a limited capacity, which varied from 0 to 100% of all the requests. At the beginning of each experiment, the cache was fully populated with requests from the training set, using either the FIFO or LRU cache replacement policy. During each run, the cache was updated with requests read from the testing set.

[3]Notice that the number of clusters is specific to a dataset, and does not have to be changed each time the training phase is executed.

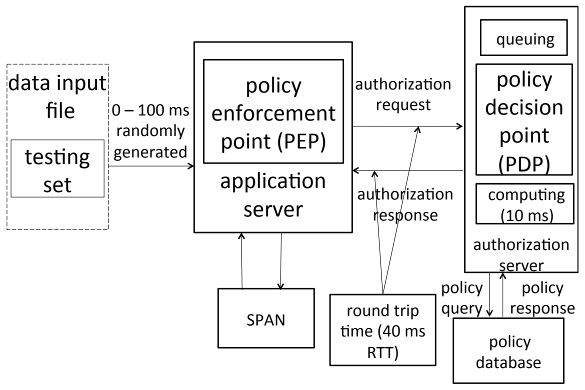


Fig. 3. Simulation setup for latency calculation

3) Iterative learning technique: For evaluating the iterative learning technique, we first trained SPAN by setting the size of the training set to 50% and used the remainder as the testing set. For every iteration of SPAN training, we moved an additional 10K requests from the testing set to the training set, while keeping the number of requests in the training set constant by removing requests from the initial part of training set. For instance, in WebCT2, we trained SPAN on requests 10K to 60K, and measured its performance. The initial assignments of subjects and subsequences to clusters were chosen from the previous iteration. We performed two sets of iterative learning for every set. We also repeated these experiments with the increment of 20K.

C. Measurement criteria

First, we studied the hit rate, defined as the ratio of correctly predicted requests to the total number of requests made by the subjects. As the PDP precomputes the responses to the predicted requests and pushes them to the PEP cache, a higher hit rate indirectly indicates lower latency for accessing protected resources.

Predicted requests that are not utilized can lead to an unnecessary load on the PDP. To study the number of unused predictions, we calculated the precision, which we defined as the ratio of requests correctly predicted to the total number of requests predicted. A higher precision rate indicates that the PDP precomputes fewer responses that are unused.

D. Simulation setup for evaluating latency reduction

We built an experimental testbed shown in Figure 3 to study latency reduction. We split the evaluation engine into two modules: the PEP and the PDP. The PEP read every request from the testing set in a period of 0 to 100 ms from the end of the previous request, using uniform distribution. The PEP forwarded these requests to the PDP and SPAN. SPAN

Trace	**Request range**	**Number of subjects**	**Number of unique requests**	**Number of unique sub-sequences of length 3**	**Training time per iteration of clustering (minutes)**	**Time for making each prediction (ms)**
WebCTALL	$1-210K$	56	$4,696$	$22,418$	28	2.34
WebCT2	$1-100K$	56	$2,642$	$12,984$	16	2.52
WebCT3	$75-175K$	56	$1,482$	$9,318$	15	2.04
FC1	N/A	50	$1,780$	$5,671$	12	9.6
FC2	N/A	100	$2,424$	$10,092$	24	9.6
FC3	N/A	200	$3,211$	$19,116$	47	8.28

TABLE I
SUMMARY OF THE DATASETS USED FOR EXPERIMENTS.

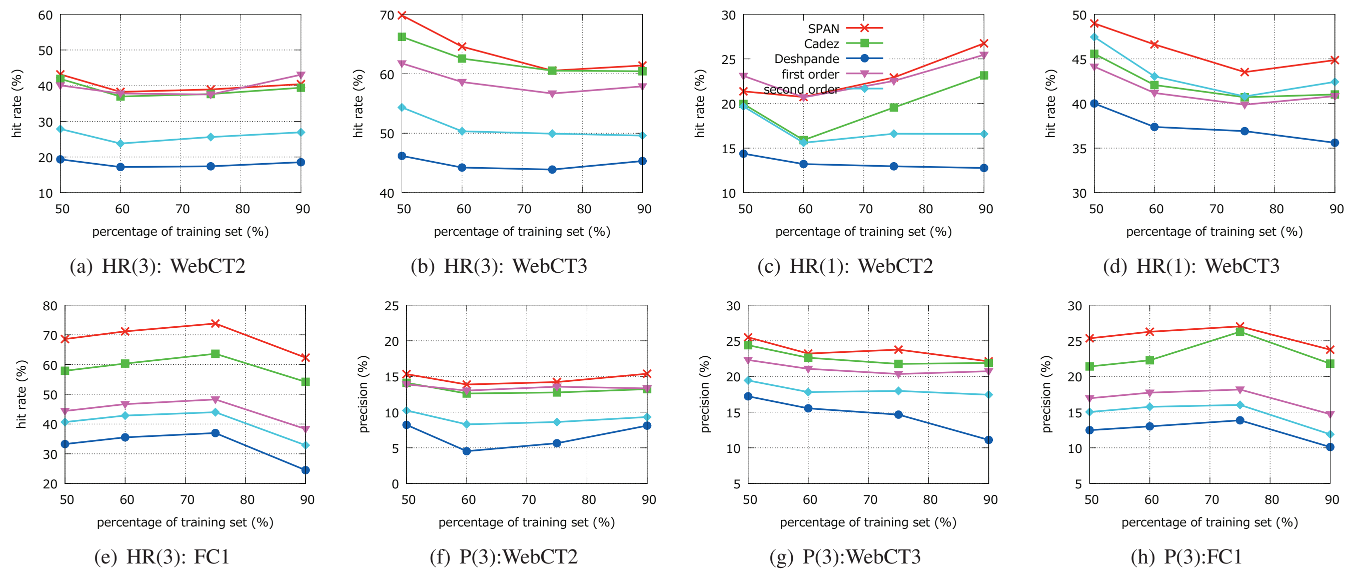


Fig. 4. Hit rate (HR) and precision (P) for WebCT2, WebCT3, and FC1 when either 3 or 1 most probable responses were fetched for every sequence.

predicted the future request(s) based on the received requests and sent them to the PEP, which forwarded these predicted requests to the PDP. We simulated fixed round trip delays of 40 ms between the PEP and the PDP, as done by Wei et al. [31], and a computation delay of 10 ms for every request. Queuing delays are added at the PDP depending on two factors: first, the rate at which requests arrive from the PEP to the PDP; and second, the delay introduced by the computation process at the PDP. Queuing delays are zero if the PDP is idle when requests arrive from the PEP. In our simulation, the PDP prioritized requests read from the testing set over requests predicted by SPAN. Latency was the difference between the time a request was read by the PEP from the testing set and the time it received a response from either its cache or the PDP. Latency calculations were performed for a training and testing set ratio of 50:50.

E. Results

1) Hit rate, precision, and latency reduction: Figure 4, Sup:2, Sup:3, and Sup:4 show the hit rate and precision for different traces of the WebCT and FC dataset, when the one most probable and the three most probable requests are considered by the evaluation engine. The figures show that hit rate and precision are not much affected by different sizes of training and testing sets.

Observe that the hit rate and precision for the WebCT3 4(b), 4(d), and 4(g) are much higher than for WebCT2 4(a), 4(c), and 4(f). The WebCT3 trace corresponds to the time of the course when requests were made mostly for reading and discussing course contents. This implies, unsurprisingly, that the hit rate and precision are higher when the patterns found in the training and testing sets are similar to each other.

SPAN achieves an average hit rate of 63%, 41%, and 64% when the three most-probable requests are considered for WebCTAll, WebCT2, and WebCT3. Corresponding precision is 21%, 13%, and 23%, respectively. The average improvement in hit rate achieved by SPAN, as compared to the first order, second order, and Deshpande algorithm, was 6%, 15%, and 25%, respectively. The corresponding improvement in precision was 2%, 5%, and 7%. Overall, SPAN achieved a better hit rate and precision. We also observed that SPAN closely matched the Cadez algorithm in performance, offering improvement only in the range of 2 to 4%. A possible reason for this is provided in Section V.

Figures 4(e), 4(h), Sup:2, Sup:3, and Sup:4 show the hit rate and precision obtained for the FC dataset. The average hit rate obtained by SPAN was 70% (FC1), 60% (FC2), and 50% (FC3). The corresponding precision was 25% (FC1), 22% (FC2), and 18% (FC3). SPAN outperformed all the other algorithms in terms of hit rate by 10 to 31% and precision 3 to 11%.

For all datasets, when only the most probable request was considered by the evaluation engine, the hit rate dropped by 10 to 20%, but the precision almost doubled, as can be seen in Figures 4, Sup:2, Sup:3, and Sup:4. Interestingly, for the FC dataset, the performance of Cadez dropped considerably when only one request per sequence was predicted. Precision equals hit rate when the one most probable request is considered.

Figure 5 shows the cumulative distribution functions (CDFs) for the experiments conducted to study the reduction in latency achieved by the SPAN and Cadez algorithms. From Figures 4, 5, and Sup:2 we observe that higher hit rates correspond to lower latencies for both datasets. We can also conclude that the proportion of requests experiencing reduced latency for obtaining responses was less than, or approximately equal to, the hit rate obtained for that dataset. Consider hit rate and latency for FC1, as an example. In Figure 4(e), we saw that the hit rate obtained for FC1 was 68% for training and testing set sizes of 50% each, when 3 most probable requests were considered. Correspondingly, Figure 5(c) indicates that responses were received by the PEP in less than 50 ms for 62% of requests. Furthermore, Figure 5(c) shows that 28% of these requests experienced no delay at the PEP in obtaining a response. In other words, responses were precomputed and placed in the PEP cache before the corresponding requests were made by the subjects. A similar correlation between latency and the hit rate can be observed for the Cadez algorithm. In the case of Cadez, when the three most probable requests were considered for FC1, responses were received by PEP in less than 50 ms for 51% of requests (Figure 5(c)), and a corresponding hit rate of 57% was obtained (Figure 4(e)). However, a queuing delay was observed when the three most probable requests were considered, as the tail in Figures 5(a) and 5(c) suggest. However, this tail cannot be seen for the most probable requests case (Figures 5(b) and 5(d)).

2) Combining SPAN and Caching: Figures 6 and Sup:5 show the hit rates and the latency calculation obtained by caching, SPAN, Cadez, and their combinations. Combining caching and prefetching algorithms improved the overall hit rate and reduced latency for all datasets and cache sizes. Furthermore, prefetching appears to have played a dominant role in improving the hit rate for smaller cache sizes. As our cache with the least recently used policy performed just slightly better than with first-in-first-out (FIFO), we show the results for FIFO only.

In the case of SPAN, a better hit rate for cache sizes of up to 20% for WebCT and up to 50% for FC was achieved. As the cache size increased, the hit rate continued to improve for cache and cache+SPAN configurations. However, after the cache reached a certain size (30% for WebCT and 60% for FC), SPAN's contribution to the hit rate improvement became negligible. In the FC dataset (Figure 6(b), Sup:5(c), Sup:5(d)), increasing the cache size from 10% to 30% did not result in any improvement in the hit rate. This anomaly is probably because subjects in the FC dataset mostly only requested permissions once per session, which resulted in most cached responses never being used again in the same session.

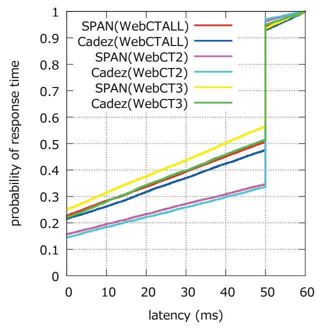
We performed latency calculations for cache sizes of 20%, 40%, and 60%. The correlation between the hit rate and latency holds true even in the case of caching and thus, we report results only for a cache size of 40%. From Figure 6 and Sup:5, we observed the following: (1) For systems with only caching, the probability of latency reduction was equal to the hit rate obtained for the dataset. (2) In the case of the FC dataset, when caching was combined with SPAN, the probability of reduction in latency improved considerably (to 68% in FC1 for example Figure 6(d)) as compared to the standalone use of caching (40%). This combination achieved a slight improvement when compared to the standalone use of SPAN (62%). (3) Also, the combination of caching and SPAN improved the probability of zero latency from 40% (caching) or 28% (SPAN) to 45% in the case of FC1. (4) Similar observations hold true for algorithm proposed by Cadez et al. (5) However, in the case of WebCT there was no improvement achieved by combining prefetching and caching over standalone caching. Since caching dominated prefetching, there was no improvement achieved by the combination of the two techniques over standalone caching.

3) Iterative learning technique: For implementing the iterative learning technique, we started the training phase from the previous stable assignment of clusters . This process reduces the number of iterations from 15 to 25 initially, to less than 5 iterations. For example, for WebCT3 and WebCT2 traces, our SPAN prototype found stable clusters in 1 and 3 iterations, respectfully, when it was relearning with an additional 10,000 requests from the corresponding testing sets. Table II shows the hit rates obtained for different traces of WebCT and FC datasets for iterative learning. The hit rates did not vary much for any set within a trace.

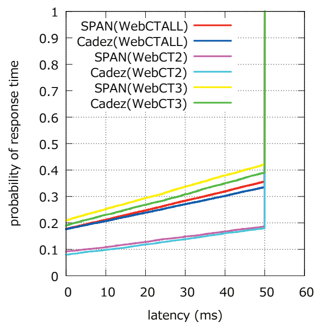
V. Discussion

In our evaluation, SPAN achieved a higher hit rate and precision, as compared to the other algorithms we evaluated. A higher hit rate leads not only to a reduced latency, but also to an increased computational load on the PDP, and higher precision results in fewer precomputed responses being unused.

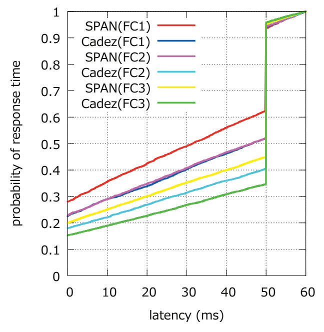
The results of our evaluation show that SPAN performs better when the request patterns of individual subjects are different from overall request patterns. SPAN performs as well as the prediction algorithm by Cadez et al. [8] for the WebCT dataset, and it outperforms the Cadez et al. algorithm for FC. The popular sequences of requests made in WebCT were for accessing course materials, to which everyone had access; resulting in similar request patterns for most subjects. The WebCT dataset closely matched a dataset that would be obtained for web pages without access control policies. Since SPAN and Cadez are implemented using a clustering approach, SPAN could not achieve a significant improvement over Cadez for this dataset. On the other hand, Fight Club players could receive help only from their Facebook friends, which resulted in different requests made by different subjects. From a total of 43,669 users, who played the game, each user could receive help from a small set of other users. This is



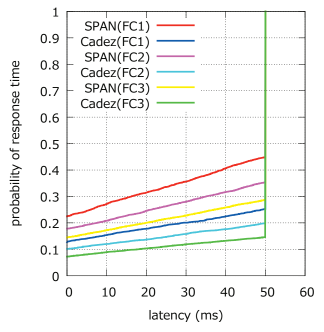
(a) 3: WebCT



(b) 1: WebCT

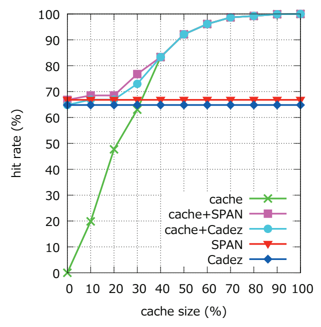


(c) 3: FC

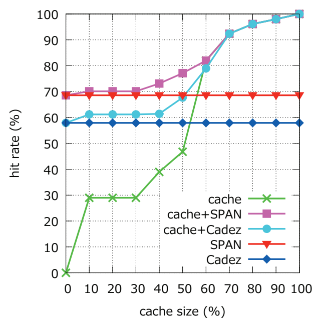


(d) 1: FC

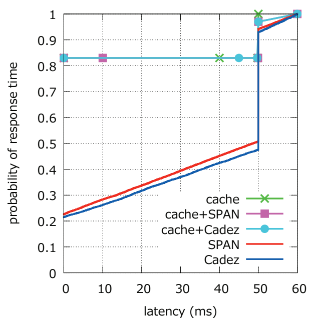
Fig. 5. Latency calculation for WebCT and FC datasets when either 3 (a, c) or 1 (b, d) most probable responses were fetched for every sequence.



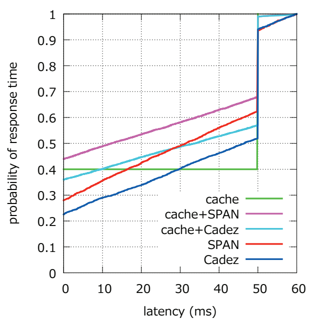
(a) hit rate: WebCTALL



(b) hit rate: FC1



(c) latency: WebCTALL



(d) latency: FC1

Fig. 6. Hit rates (a,b) and latency reduction (c,d) for implementations of FIFO cache, SPAN(3), Cadez(3), and their combination for WebCT and FC datasets.

similar to the accesses found in access control systems, where subjects can request action on a small set of permissions from the available pool of permissions. Our design of SPAN, which considered individual subjects' request patterns might have resulted in better predictions for the FC dataset. However, SPAN outperforms the other algorithms used in the evaluation for both datasets.

Limiting the number of precomputed responses increased precision—and therefore reduced the load on the PDP—at the expense of the hit rate. Results (Supplementary file Sup:III-A) show that precision increased by 50 to 100% but the hit rate decreased by almost 50%, when the number of precomputed requests was reduced from three most probable to the most probable one. Increasing the probability threshold for predicted requests also gave similar results. Limiting the number of precomputed responses can be a good strategy where the PDP's computing resources are either expensive or limited.

SPAN achieved a better hit rate for smaller cache sizes, but caching dominated the performance for larger sizes of cache. Overall, combining caching and SPAN demonstrated a boost in hit rate for all sizes of cache, as compared to standalone caching techniques or SPAN.

The increased hit rate resulted in the reduction of latency. In the case of FC1, for example, for the 62% of the requests, responses were received in less than 50 ms. The impact of such reduction on the user experience depends, of course, on the overall waiting time, which includes not only the authorization and communication but also the application delays. While it's hard to assess the impact of these savings on all applications,

	WebCTAll	WebCT2	WebCT3	FC1	FC2	FC3
1^{st} set (10K)	66.8	43.2	69.9	68.6	58.8	49.8
2^{nd} set (10K)	64.4	38.3	67.6	67.1	57.9	50.4
3^{rd} set (10K)	63.5	40.9	66.5	69.8	61.1	52.8
1^{st} set (20K)	66.8	43.1	69.8	68.6	58.8	49.8
2^{nd} set (20K)	63.4	42.4	69.4	65.3	65.3	56.3
3^{rd} set (20K)	64.8	43.7	67.9	69.5	61.0	57.4

TABLE II
HIT RATES (%) OBTAINED FOR ITERATIVE LEARNING. 10K AND 20K REPRESENT THE REQUESTS CHANGED FOR EVERY NEW ITERATION. THE 1^{st} SET IS THE ORIGINAL TRAINING SEQUENCE FOR THE TRACE.

online commerce companies report an approximate 1% loss in sales as the cost of a single extra 100 ms delay, and search engines find that an extra 500 ms delay in displaying the search results may reduce their revenues by up to 20% [3].

From Table I, we found that training time is large, which is directly proportional to the number of subjects, unique sequences of requests, and iterations required to form clusters. However, the time required to make predictions is low. In fact, it only depends on the number of clusters formed, which is much smaller than the number of subjects and sequences.

Although SPAN achieves a better hit rate and precision, compared to other state-of-the-art systems, it is not perfect. If, for example, due to a policy change, a subject's access pattern changes, it would take some time for SPAN to learn the new pattern. Until then, SPAN's hit rate and precision might be degraded. This is a core limitation of all predicting approaches [6], [7], [8], [9], [10], [11], [12], [13], as discussed in Section II. We have partially addressed this limitation by our iterative learning technique. Note that when using SPAN, incorrect predictions do not result in incorrect authorization decisions being enforced by the PEP, as SPAN only predicts requests but responses are computed by the PDP, which has access to the current policy. Also, in the current design, SPAN does not address the issue of multiple users using the same account to access protected resources. This would be a possible topic for future research.

VI. Conclusion

In this paper, we presented an approach for *Speculative Authorization* (SPAN), which predicts the authorization requests that are to be made by subjects in enterprise authorization systems. After evaluating the results obtained with our approach using one synthetic dataset and two real-world datasets, SPAN achieved a hit rates between 30% and 70%, and reduced latency for almost 60% of the requests. This is an improvement of 2 to 55%, as compared other prominent web prefetching algorithms. We also simulated caching and prefetching in the same system, and found that SPAN improves the performance of authorization systems considerably for smaller sizes of cache.

In future studies, role-based access control (RBAC) [23] could be explored as one of the solutions for reducing the length of the training phase. This time is proportional to the number of subjects, which can affect the scalability of SPAN. With RBAC, subjects would be assigned to roles, and these roles could be used during cluster assignments. A subject can, however, possess multiple roles, which would have to be considered while forming clusters.

References

[1] G. Karjoth, "Access control with IBM Tivoli Access Manager," *ACM Transactions on Information and Systems Security*, vol. 6, no. 2, pp. 232–57, 2003.

[2] J. Nielsen, *Usability Engineering*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[3] R. Kohavi, R. M. Henne, and D. Sommerfield, "Practical guide to controlled experiments on the web: Listen to your customers not to the hippo," in *KDD '07: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2007, pp. 959–967.

[4] L. Bauer, M. A. Schneider, and E. W. Felten, "A general and flexible access-control system for the web," in *Proceedings of the 11th USENIX Security Symposium*. Berkeley, CA, USA: USENIX Association, August 5-9 2002, pp. 93–108. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.1.2230

[5] J. Bregman, B. Eidelman, and C. Johnson, "Oracle fusion middleware security," http://fusionsecurity.blogspot.com/2009/10/impact-of-oracle-entitlement-server-oes.html, 2009.

[6] M. Awad, L. Khan, and B. Thuraisingham, "Predicting WWW surfing using multiple evidence combination," *The International Journal on Very Large Data Bases*, vol. 13, pp. 401–417, 2008.

[7] M. Deshpande and G. Karypis, "Selective Markov models for predicting web page accesses," *ACM Transactions on Internet Technology*, vol. 4, no. 2, pp. 163–184, 2004.

[8] I. Cadez, D. Heckerman, C. Meek, P. Smyth, and S. White, "Model-based clustering and visualization of navigation patterns on a web site," *Data Mining Knowledge Discovery*, vol. 7, no. 4, pp. 399–424, 2003.

[9] R. Sen and M. Hansen, "Predicting web users' next access based on log data," *Journal of Computational and Graphical Statistics*, vol. 12, no. 1, pp. 1–13, 2005.

[10] Z. Su, Q. Yang, Y. Lu, and H. Zhang, "Whatnext: A prediction system for web requests using n-gram sequence models," in *Proceedings of the First International Conference on Web Information Systems Engineering*, June 19 - 20 2000, pp. 214–221.

[11] G. Bonnin, A. Brun, and A. Boyer, "A low-order Markov model integrating long-distance histories for collaborative recommender systems," in *IUI '09: Proc. of the 13th Int. Conf. on Intelligent User Interfaces*. New York, NY, USA: ACM, January 13-16 2009, pp. 57–66.

[12] B. Mobasher, H. Dai, T. Luo, and M. Nakagawa, "Effective personalization based on association rule discovery from web usage data," in *WIDM '01: Proc. of the 3rd Int. Workshop on Web Information and Data Management*. New York, NY, USA: ACM, November 9 2001, pp. 9–15.

[13] Q. Yang, T. Li, and K. Wang, "Building association-rule based sequential classifiers for web-document prediction," *Data Mining Knowledge Discovery*, vol. 8, no. 3, pp. 253–273, 2004.

[14] J. Pitkow and P. Pirolli, "Mining longest repeating subsequences to predict world wide web surfing," in *USITS'99: Proceedings of the 2nd Conference on USENIX Symposium on Internet Technologies and Systems*. Berkeley, CA, USA: USENIX Association, October 11-14 1999, pp. 13–13.

[15] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Topic modeling," *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.

[16] "Blackboard Vista, a course management system," http://www.blackboard.com/.

[17] A. Nazir, S. Raza, and C.-N. Chuah, "Unveiling Facebook: A measurement study of social network based applications," in *IMC '08: Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement*. New York, NY, USA: ACM, 2008, pp. 43–56.

[18] L. Adamic and B. Huberman, "Zipf's law and the Internet," *Glottometrics*, vol. 3, no. 1, pp. 143–50, 2002.

[19] Z. Kalbarczyk, R. K. Lyer, and L. Wang, "Application fault tolerance with Armor middleware," *IEEE Internet Computing*, vol. 9, no. 2, pp. 28–38, 2005.

[20] J. Crampton, W. Leung, and K. Beznosov, "Secondary and approximate authorizations model and its application to Bell-LaPadula policies," in *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT'06)*. Lake Tahoe, CA, USA: ACM Press, June 7–9 2006, pp. 111–120. [Online]. Available: http://portal.acm.org/citation.cfm?id=1133075

[21] Q. Wei, M. Ripeanu, and K. Beznosov, "Cooperative secondary authorization recycling," in *Proceedings of the 16th ACM/IEEE International Symposium on High-Performance Distributed Computing (HPDC)*. Monterey Bay, CA: ACM Press, June 27-29 2007, pp. 65–74.

[22] D. E. Bell and L. J. LaPadula, "Secure computer systems: Mathematical foundations," MITRE, Technical Report ESD-TR-74-244, March 1973.

[23] ANSI, "ANSI INCITS 359-2004 for role based access control," American National Standards Institute, 2004.

[24] Q. Wei, J. Crampton, K. Beznosov, and M. Ripeanu, "Authorization recycling in RBAC systems," in *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies (SACMAT)*. Estes Park, Colorado, USA: ACM, June 11–13 2008, pp. 63–72. [Online]. Available: http://portal.acm.org/citation.cfm?id=1377836.1377848

[25] M. Kohler and A. Schaad, "Proactive access control for business process-driven environments," in *ACSAC '08: Proceedings of the 2008 Annual Computer Security Applications Conference*. Washington, DC, USA: IEEE Computer Society, December 8-12 2008, pp. 153–162.

[26] B. D. Davison, *Learning web request patterns*. Springer, 2004, pp. 435–460.

[27] D. Heckerman, "A tutorial on learning with Bayesian Networks," Microsoft Research, Tech. Rep., 1995. [Online]. Available: ftp://ftp.research.microsoft.com/pub/tr/tr-95-06.pdf

[28] D. A. P., L. N. M., and R. D. B., "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society, Series B*, vol. 39, no. 1, pp. 1–38, 1977.

[29] M. Arlitt and T. Jin, "1998 world cup web site access logs," http://www.acm.org/sigcomm/ITA/, 1998.

[30] A. Nazir, S. Raza, and C.-N. Chuah, "Online Social Networks: Anonymized Data from Third-Party Facebook Applications," http://www.ece.ucdavis.edu/rubinet/data.html, 2008.

[31] Q. Wei, "Towards improving the availability and performance of enterprise authorization systems," Ph.D. dissertation, Electrical and Computer Engineering, The University of British Columbia, Vancouver, 2009.

Biographies

Pranab Kini received his masters degree from the department of Electrical and Computer Engineering at the University of British Columbia (UBC), Canada, in 2010. His masters thesis focused on improving the performance of enterprise access control systems. Currently, he works as a Research Engineer at UBC, where he is involved in developing a disaster management platform, using distributed architecture, for better correspondence between emergency responders and experts. Prior to joining UBC, he worked at Infosys Technologies Limited, India, as a software engineer in the field of identity and access management.

Konstantin (Kosta) Beznosov is an Associate Professor at the Department of Electrical and Computer Engineering, University of British Columbia, where he directs the Laboratory for Education and Research in Secure Systems Engineering. His research interests are usable security, distributed systems security, secure software engineering, and access control. Prior UBC, he was a Security Architect at Hitachi Computer Products (America) and Concept Five. Besides many academic papers on security engineering in distributed systems, he is also a co-author of Enterprise Security with EJB and CORBA and Mastering Web Services Security books, as well as XACML and several CORBA security specifications. He has served on program committees and/or helped to organize SOUPS, CCS, NSPW, NDSS, ACSAC, SACMAT, CHIMIT. Prof. Beznosov is an associate editor of ACM Transactions on Information and System Security (TISSEC) and International Journal of Secure Software Engineering (IJSSE).