Speculative Authorization

Pranab Kini[*], Konstantin Beznosov[†]

Laboratory for Education and Research in Secure Systems Engineering
lersse.ece.ubc.ca
University of British Columbia
Vancouver, Canada

Technical report LERSSE-TR-2010-002[‡]

Abstract

As enterprises aim towards achieving zero latency for their systems, latency introduced by authorization process can act as an obstacle towards achieving their goal. We present *<u>Sp</u>eculative <u>A</u>uthorizatio<u>n</u>* (SPAN), a prediction technique to address the problem of latency in enterprise authorization systems. SPAN predicts the possible future requests that could be made by a client, based on the present and past behavior of the client. Authorization decisions to the predicted requests are fetched even before the requests are made by the client, thus reducing the authorization latency virtually to zero. Our implementation indicates that systems deploying SPAN can obtain zero authorization latency for almost 60% of the requests made by the client. We discuss the additional load incurred by the systems to compute responses to the predicted requests, and provide measures to reduce the unnecessary load. We also compare the benefits of deploying caching and SPAN in the same system, and find that SPAN can effectively improve the performance of systems with smaller sizes of cache.

[*]pranabk@ece.ubc.ca

[†]beznosov@ece.ubc.ca

[‡]This and other LERSSE publications can be found at lersse-dl.ece.ubc.ca

Contents

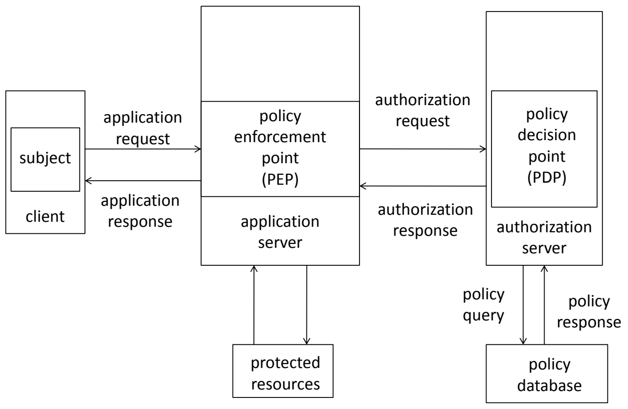


Figure 1: Architecture of enterprise access control systems

1 Introduction

With the emergence of tighter corporate policies and government regulations, access control has become an integral part of the business requirements. Modern access control architectures [Kar03, Ora08] follow the request-response architecture as shown in Figure 1. In this architecture, a subject makes an application request to the policy enforcement point (PEP) for accessing a protected resource. The PEP converts this request to an authorization request and forwards it to the policy decision point (PDP). The PDP queries the policy database to verify the ability of the subject to access the requested resource. It computes an authorization response and sends it back to the PEP, which enforces it. The policy database is assumed to contain the security policy information for all the protected resources in the system.

The authorization process adds to the already existing latency for accessing resources. The process of computing the authorization response can take a few milliseconds to several seconds depending on the policy associated with the request and subsequent process involved in computing the response [BSF02, BEJ09]. Neilsen suggests that a response time of greater than 0.1 second makes end users feel that the system is not responding instantaneously [Nie93]. A study by Amazon reported roughly 1% sales loss as the cost of a 100 ms extra delay. Another study by Google found a 500 ms extra delay in displaying the search results may reduce revenues by up to 20% [KHS07]. Thus, on one hand, authorization is imperative for securing the protected resources in an enterprise, but on the other hand, it increases the latency in the system, thus hampering responsiveness and revenue. In the rest of the paper, the term latency refers to the delay introduced in the system by the authorization process. The total latency of computing an authorization response is the sum of the delays introduced by (1) communication channels between the PEP and the PDP (communication delays), (2) authorization calls made to the policy database (communication delays), (3) subsequent time for computing the response by the PDP (processing delays), and (4) the queuing of requests at the PDP awaiting responses when the system is heavily loaded (queuing delays).

To address the need for reducing the overall latency, we propose *Speculative Authorization* (SPAN), a technique that predicts future requests 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. We apply the principles of machine learning [Hec95] for analyzing the series of requests made by the subject in every session. As shown in Figure 2, the PEP sends the authorization requests to the PDP. The PDP not only computes the responses to the requests but also forwards a copy of these requests to SPAN. Based on the series of received requests, SPAN predicts the requests that could possibly be made by the subjects in their sessions, and sends the predicted requests

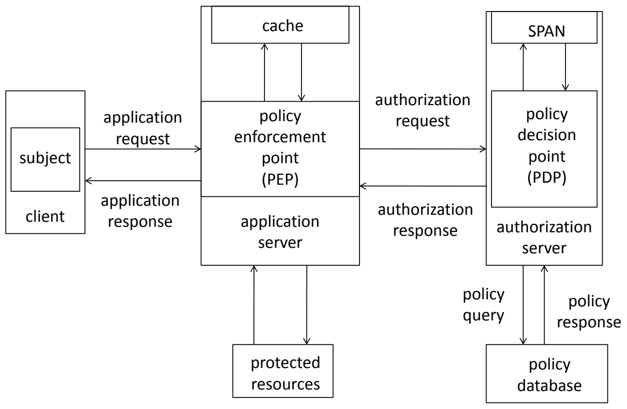


Figure 2: Architecture of SPAN

to the PDP. The PDP computes the responses to these predicted requests and sends it to the cache that is collocated with the PEP. If the subjects make the same requests as predicted by SPAN, responses are obtained from the cache. As responses are available in the cache even before a request is made, overall latency is reduced. This is a "push" model where SPAN is collocated with the PDP. SPAN could also be located at the PEP side as a "pull" model.

SPAN is designed to be comparable to several algorithms developed for web page prefetching [AKT08, DK04, EVK05, CHM$^+$03, SH05, SYLZ00, BBB09, MDLN01, YHN03, YZ01, YLW04, PP99]. In web page prefetching algorithms, the likelihood of web pages being requested in the same session is found using techniques such as Markov models [AKT08, DK04, EVK05, CHM$^+$03, SH05], association rule mining [YHN03, YZ01, YLW04], n-grams [SYLZ00, PP99], SVM [AKT08], or clustering [SH05, CHM$^+$03]. However, none of these algorithms consider the identity of the subjects making those requests. In enterprises, subjects' identity and authorization policies restrict the access of the subjects to a subset of resources. This restriction influences the requests made by the subjects. The likelihood of two resources being requested in the same session would not only depend on the resources themselves but also on the authorization policies governing the resources. Thus, authorization policies introduce an additional constraint for prefetching. In section 3.1.3, we find the possible shortcomings of using web page prefetching algorithms in access control systems. We designed SPAN to take the authorization policies into account without having access to the authorization policies of the system. It finds the likelihood of two resources being requested by subjects by observing the past sessions of the subjects.

To make predictions, we form Markov chains of all possible sequences of requests. To accommodate authorization policies, a naive technique would find if every subject has ever requested the formed sequences. This technique has two drawbacks (1) large memory consumption: The memory required is directly proportional to the number of subjects and the number of sequences formed. (2) to make predictions with certain degree of confidence, the number of times (frequency count) subjects request for certain sequences has to be comparatively higher than the requests for other sequences. Counting the number of times each sequence was requested by every subject might not result in comparable frequency counts as required. To overcome these problems, web clustering techniques [SH05, CHM$^+$03] are adopted, where sequences are clustered based on their likelihood to be requested in the same session. SPAN also uses a clustering approach for grouping the sequences, and probabilistically associates subjects to the clusters. In clustering techniques used for web page prefetching [SH05, CHM$^+$03], users are randomly assigned to the clusters. This is contrary to the approach in SPAN where the clustering also finds the probability of associating subjects to clusters. The probabilistic association indirectly takes the authorization policies into account. To achieve this goal, SPAN extends Latent Dirichlet Allocation (LDA) model proposed by Blei et al. [BNJ03]. LDA was developed for unigram models whereas SPAN extends it to accommodate the sequences of requests.

Research by Sen and Hansen [SH05], and Deshpande and Karypis [DK04] suggests that higher order Markov models improve the predictive capabilities of the prefetching algorithms. However, increasing the order of Markov models increases the number of parameters in the system, which results in higher memory usage and state-space complexity [DK04]. The number of parameters of Markov models are m^n, where m is the number of resources and n is the order of the Markov model. The parameters increase exponentially as the order of the Markov model increases. To overcome this exponential increase, we build 'Multiple first order Markov models' within SPAN. Our design reduces the number of parameters to $n * m^2$. The parameters of our model increase linearly as the order of the Markov model increases.

We evaluated SPAN using two datasets that represented log traces from access control systems. Our first dataset contained accesses made by students, teaching assistants and course instructors in WebCT [Web] for a course at our university. Different subjects had different rights on the resources of the course. We obtained the second dataset from requests made by users in the 'Fighters Club' (FC) application of Facebook [NRC08]. In this application, users could start a virtual fight with their friends and request for help from other friends. This represents an access control system where subjects (users) can only request for resources (friends) that they are authorized.

In addition to implementing SPAN, we also implemented algorithms proposed by Cadez et al. [CHM+03], and Deshpande and Karypis [DK04], and the first and second order Markov models. For the second dataset, SPAN achieved an improvement of 11%, 31%, 21% and 23% in hit rate as compared to other algorithms. Corresponding improvement for the first dataset was 2%, 21%, 2%, and 11% . We believe that the increase in the hit rate is due to the adopted clustering technique that takes authorization policies into account.

Caching is a simple and inexpensive solution of improving the performance of systems. It has been widely adopted in commercial products like Tivoli Access Manager [IBM08]. On the other hand, prefetching requires a learning process that is time consuming and memory intensive. We simulated caching and prefetching in the same system. For the WebCT dataset, we found that prefetching helps in improving the performance by atleast 26% in addition to the performance improvement achieved by caching. For the FC application, the improvement is over 40%. In fact, we observe that increasing the size of the cache doesn't improve performance for FC. Prefetching algorithms are beneficial to improve the performance of such systems.

Predictions increase the number of computations that have to performed by the PDP. Not all predicted requests are eventually requested by the subjects. Computing responses to such requests can be expensive where the cost of computing responses is high. To minimize the unnecessary load, we propose a 'confidence level' metric for computing responses to predicted requests. We propose to compute responses to only those requests whose confidence level is above the threshold. For all the algorithms that we implemented, we found that the load on the PDP reduces considerably using this approach.

To summarize, our contributions are:

1. We proposed SPAN for prefetching resources in access control systems that takes authorization policies into account without actually having direct access to the policies.
2. We built a 'Multiple first order Markov models' within SPAN to reduce the exponential growth in parameters to a linear growth.
3. We implemented algorithms that have been proposed in the literature and found that SPAN performs better than those algorithms.
4. We proposed a 'confidence level' metric for prefetching the responses that can reduce the unnecessary load on the PDP.
5. We compared prefetching algorithms against caching and evaluated the effectiveness of prefetching over caching in access control systems.

The rest of the paper is organized as follows: The related work is presented in Section 2. In Section 3, we present the shortcomings of the algorithms in web page prefetching and present our technique to overcome these. We present the evaluation of our algorithm in Section 4. Finally, in Section 6, we present the conclusion drawn from the results and discuss future work.

2 Related Work

One of the approaches to combat the problem of latency is to replicate the authorization service components such as the PEP, PDP and policy database. This increases availability [IBM08], but such systems scale poorly, and become technically and economically infeasible when the number of entities in the system reaches thousands [KLW05, Vog04].

Another state-of-the-practice approach to improving the overall latency is to cache authorization decisions at the PEP-side [BZP05, SSL+99]. If a subject makes the same request as before, the response is fetched from the PEP-cache serving the subject instantaneously. Caching helps reduce the overall latency in authorization systems, but it can be effective only in those cases where subjects repeatedly make the same requests. To overcome this problem, Crampton et al. [CLB06] propose Secondary and Approximate Authorization Model (SAAM), where the cached responses are stored at the secondary decision point (SDP) that is 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 [WRB07]. SAAM algorithms for role based access control are proposed by Wei et al. [WCBR08]. The models proposed for SAAM compute the responses after receiving the request from the subjects. If the time required for computing response is relatively high, as in the case of distributed authorization systems [BSF02], the responsiveness of the system is affected. Also, SAAM is proposed for the computing the secondary authorizations for policies based on the Bell-Lapadula model [BL73, BL75, CLB06] that restricts its usage.

Kohler and Schaad [KS08] proposed 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 made of certain predefined sub-processes. When a user starts a business process, their architecture extracts the permissions required to complete all the sub-processes of the process. This reduces the latency involved in computing the response for every sub-process. However their approach depends on predefining sub-processes for every business process. The success of this implementation depends on defining all the possible sub-processes for every business processes in an enterprise. Our approach finds probabilistically the dependencies between requests without any prior knowledge of business processes.

Several predictive models have been proposed for prefetching web pages [AKT08, DK04, EVK05, CHM+03, SH05, SYLZ00, BBB09, MDLN01, YHN03, YZ01, YLW04]. In these models, the algorithms learn the surfing patterns of the users on a website during the training phase. In the testing phase, when users surf web pages, the models compare the surfing patterns with the learnt patterns to predict the most likely web page(s) that would be visited by the user. For the training and testing phases of the algorithms, sequences of requests made by the users are captured by the algorithms.

Techniques using Markov models [DK04] and its variants [AKT08, DK04, EVK05, CHM+03] are used to find the popularity of web surfing patterns. Deshpande and Karypis [DK04] developed regular first, second, and third order Markov models for predicting web pages. They found that higher order Markov models give better predictions but face the problems of higher memory usage (state-space complexity) and reduced coverage. Sen and Hansen [SH05] also justify these claims through their results. To overcome these problems, Deshpande and Karypis [DK04] propose techniques to intelligently combine the Markov models that have low state-space complexity while maintaining the coverage and accuracy of the model. They have presented three schemes for pruning, called (1)frequency pruning, (2)confidence pruning, and (3) error pruning. Results obtained in their paper does not show significant improvement in implementing their techniques over regular Markov models. The maximum gain of predictive accuracy is found to be less than 1%. However, the pruning techniques presented in this paper provide directions in reducing the search space in the testing phase. In effect, such techniques reduce the number of unnecessary requests that would be prefetched by the prediction algorithms. Deshpande and Karypis [DK04] have proposed their pruning techniques that remove the states that have little or no chance of being requested in the testing sets, eventually reducing the search space during the testing phase. If their proposed algorithms are implemented in access control systems, additional load on the PDP will be reduced as responses to requests having little or no chances of being requested by the subjects will not be computed.

Awad et al. [AKT08] combine two techniques, namely Markov models and Support Vector Machines [BC00], to predict the surfing patterns of the users. Although the techniques are quite powerful in prediction, longer training times can hamper the performance of the systems. The paper reports a training time of 26 hours training $23,028$ requests. We implemented our algorithms on training sets of $50,000$ requests and above. The training time of our algorithm is proportional to the number of requests and the subjects in the system. Our algorithm is iterative in nature with each iteration taking 12-47 minutes.

Cadez et al. [CHM$^+$03] and Sen and Hansen [SH05], propose clustering the web pages using statistics obtained from first order Markov models. Our approach extends their clustering technique. They cluster the sequences of requests from the statistics obtained from first order Markov models in the training phase. Users are randomly assigned to one of these clusters in the testing phase. However, we probabilistically find the association of every subject to the clusters formed in the training phase. In addition, Cadez et al. [CHM$^+$03] form 17 categories of web pages and assign 10 to $5,000$ pages to every category. They have implemented their approach to predict the categories of web pages, while the implementation at the level of web pages is proposed as future work. Implementing an algorithm to predict $5,000$ pages as compared to 17 categories, increases the number of parameters that is directly proportional to the difference between the number of pages and categories. We started by implementing their algorithm to accommodate web pages (in our case, permissions), and not restrict to the category of pages. Our implementation accommodated the increase in the parameters, which is a contribution within itself. Next, we extended their approach to suite the requirements of access control systems.

Pitkow and Pirolli [PP99] and Su et al. [SYLZ00] propose n-gram techniques to find the popular surfing patterns on a website. Su et al. [SYLZ00] propose *WhatNext* prediction algorithm that uses n-grams modeling techniques. They build a model with n-grams where the gram size is greater than or equal to 4. The surfing patterns of users are predicted based on the n-grams that are found in the trainng phase. Pitkow and Pirolli [PP99] find the sequences of different sizes in the training sets. In the testing set, when users start surfing the website, their surfing patterns are matched to the sequences in the training sets and predictions are made for the users. Deshpande and Karypis [DK04] have proposed that the datasets have to be large enough for attaining a better predictive capability when the order of Markov models increase. A 2-gram reduces to first-order Markov model when transitional probabilities are considered. It should be noted that n-gram techniques are Markov models of order that is one less than the gram size and the analysis remains the same as for Markov models. Thus, approaches proposed by Pitkow and Pirolli [PP99] and Su et al. [SYLZ00] face the problems of state-space complexity and low coverage, similar to those proposed in Deshpande and Karypis [DK04].

Bonnin et al. [BBB09] propose a technique to skip several places in longer n-grams to generate lower-order Markov models to reduce the state-space complexity. However, they fail to address the state-space complexity that the skipping process results in. Association rule mining [AS94] has been used to find popular surfing patterns in web pages [MDLN01, YZ01, YHN03, YLW04]. To gather confidence in the popular surfing patterns, these however need large amount of training sets.

The main difference between surfing patterns in web pages as compared to the access patterns in access control systems, is that the latter is dominated by access control policies of the organization. Access control policies are based on subject attributes, e.g. group memberships, roles, etc. Thus resources that are accessible to a certain group of subjects might not be accessible to others. Also access rights available for subjects vary over resources. Models used for training web pages consider only resources and don't consider the access control policies that govern the resources. In practice, the access control policies dominate the access patterns of the subjects in the enterprise. The predictive model in access control systems should consider the access rights possessed by subjects while predicting the requests for subjects. We take this fact into consideration while developing our model, which differentiates our approach from web page prediction approaches.

So far, we discussed batch approach that is the traditional approach to machine learning [Dav04]. In this approach, the datasets are divided in distinct training and testing sets. In the training phase, algorithms attempt to find a model based on the requests present in the training set. This model is evaluated against the testing set. In actual systems, there is a possibility that the online behavior might change over a period of time. Traditional batch approach cannot capture this changing behavior for making predictions. If training is performed on the old behavior of the subjects while predictions are made on the new behavior, it might result in several incorrect predictions. To capture the changing behavior of the subjects, online approach should be adopted in which system sends feedback to the algorithms for correcting the trained model according to changing behavior of the subjects. We have not come across any online algorithms for web page predictions. We adopted a combination of batch and online algorithms, where we first train the model using requests from the training set. Later, we use requests from testing set to run the training phase again and adjust the model to the changing behavior of the subjects.

3 Problem Formalization and Approach

In this section, we discuss the training phase 3.1 and testing phase 3.2 of SPAN.

Similar to other algorithms designed to make predictions, SPAN's design is divided in two phases: training and testing. In the training phase, SPAN needs to capture the sequences of requests made by the subjects from the history of accesses, and cluster them. These clusters are used to make predictions in the testing phase. In an enterprise system, a subject authenticates itself to the system. The time period between a subject logging in and out of the system is referred to as a *session*(Se). During a session, a subject requests for a series of actions on the protected resources termed as *permissions*. Based on the access control policies that govern these resources, the subject obtains an *allow* or a *deny* response. In every session, subjects request for permissions that can be represented as $Se = \{P_1,P_2,P_3,...,P_l\}$. In this sequence, P_1 represents the first requested permission, followed by P_2, and P_l is the last permission requested by a subject before logging out of the system[1]. The popularity of the permissions for the subjects, can be captured by counting the number of times, subjects request these permissions. Higher frequency counts indicate that the permissions are likely to be requested in the future. In the training phase, frequency counts decide what sequences lie in which cluster. While training phase is an offline process, testing phase is an online process.

The concept of predicting permissions in access control systems is similar to predicting web pages surfed by users in web pages. However, algorithms proposed for web page prefetching have certain shortcomings, and cannot be directly applied to access control systems. To explain the need for specific algorithms to make predictions in access control systems and differentiate them from the algorithms proposed for web page prefetching, we consider a hypothetical website with 5 web pages that is controlled by the authorization policies. The link structure for the website is as shown in Figure 3. An arrow between p_i and p_j (e.g., from p_1 to p_2) indicates that there exists a hyperlink on p_i that points towards p_j. Table 1 represents the access control matrix that characterizes the rights of the subjects to view different web pages on the website. Websites deployed by enterprises like banks, are classic examples of our hypothetical website, where different employees, and customers have different views, based on their identity and role

[1]Note that the capitalized tokens (e.g. P_1, P_2) represent the random variable denoting permissions. The actual assignment to the random variables are represented by lower case tokens (e.g. p_1, p_2).

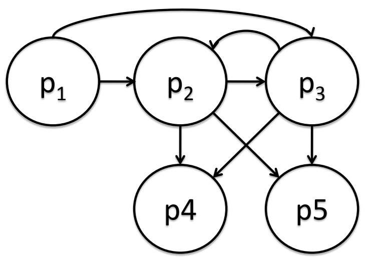


Figure 3: Link structure for hypothetical website

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

Table 1: Access control matrix in the enterprise

in the bank. We assume that our website is designed in such a way that the view presented to a subject contains only those hyperlinks that the subject is authorized to access. This avoids unauthorized requests made by the subjects. For example, the view of page p_3 presented to Bob would contain hyperlinks for pages p_2 and p_4, but a hyperlink for p_5 would not exist. To summarize, the view presented to every subject would not only depend on the structure of the website, but also on the corresponding access rights. We now describe the training phase of SPAN.

3.1 Training phase

Since sequences of requests are to be predicted, we choose Markov chains for building SPAN. Based on the past behavior exhibited by the subjects, Markov chains of all the possible sequences are formed, and the number of times these sequences are repeated across the trace are recorded. Sequences having higher counts indicate that they are likely to be repeated more often, as compared to the sequences with lower frequency counts. In a real world setting, the sequences of requests could be very long because subjects have a large number of resources that they can request. Also, subjects might not repeat their requests for resources in the same sequence every time. This would result in lower frequency counts for the sequences. To overcome the problem of lower frequency counts, we propose to split the sequences into smaller subsequences. This technique provides higher frequency counts for individual subsequences.

Session	**Alice**	**Bob**	**Mike**
Se_1	p_1, p_2, p_3	p_1, p_2, p_3	p_1, p_3, p_2
Se_2	p_2, p_3, p_4	p_2, p_3, p_4	p_2, p_3, p_5
Se_3	p_2, p_3, p_5	p_2, p_3, p_4	p_2, p_3, p_5
Se_4	p_1, p_2, p_3	p_1, p_3, p_2	p_1, p_2, p_3
Se_5	p_1, p_2, p_3	p_1, p_3, p_2	p_1, p_2, p_3

Table 2: 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

Table 3: Sample of transitions made by the subjects using first order Markov models

3.1.1 Smaller subsequences

Table 2 represents the different sequences of web pages that were visited by the 3 subjects across 5 sessions. Referring to Table 2, we find that Alice has requested for the sequence p_2, p_3, p_4 only once, but the subsequence p_2, p_3 has been requested in all her sessions. Such shorter subsequences provide higher frequency counts, compared to the entire sequence of requests made by subjects. We split the longer sequences into smaller subsequences of fixed length. For this example, we form first order Markov chains [SH05, DK04] as shown in Table 3. Every cell in the table, represents the frequency counts of transitions made by the subjects. The last column represents the total number of times the subsequences were requested by all the subjects.

While the last column gives an overall picture about the sequences of requests, it is different when compared to requests made by individual subjects. Algorithms designed for web page predictions using Markov models [AKT08, DK04, EVK05, CHM+03, SH05], association rule mining [YHN03, YZ01, YLW04], n-grams [SYLZ00, PP99], SVM [AKT08], or clustering [SH05, CHM+03], use the frequency counts from the last column to develop their algorithms. Next, we discuss a possible shortcoming of using this approach for access control systems.

3.1.2 Shortcoming of web prefetching algorithms

In this section, we find a possible shortcoming of using web page prediction algorithm for access control systems. Referring to Table 3, we find that the total frequency count of viewing p_5 after p_3 is 3. Similarly, the frequency counts of viewing p_2 and p_4 after p_3 are also 3, each. Let us suppose that Bob has logged into the system, and predictions have to be made for Bob. When Bob accesses p_3, the algorithms designed using frequency counts from the last column of Table 3, would predict p_5 as one of the probable pages, as Bob's future request. However, Bob is not authorized to view p_5. In fact, the view of p_3 presented to Bob wouldn't contain any hyperlink for p_5. The algorithms designed for access control systems should assign a zero probability to such transitions. The only pages that Bob can request after visiting p_3 are p_4 and p_2. From this example, we observe that the web page prediction algorithms can't accommodate authorization policies for making predictions, but these policies influence the requests made by the subjects.

To avoid making predictions for requests that a subject is not authorized, prediction algorithms developed for access control systems should analyze the sequence of requests in a way that captures the underlying access control policies of the system. To achieve this goal, one of the possible solutions is to look at the frequency counts of transitions made by individual subjects, but this results in lower frequency counts of transitions. As algorithms depend on higher frequency counts to make decisions, using this approach might result into lower predictive accuracy. To overcome this problem, we combine the frequency counts of transitions made by individual subjects and the total frequency counts, in the clustering approach presented in Section 3.1.4.

Research by Sen and Hansen [SH05], and Deshpande and Karypis [DK04], suggests that higher order Markov models improve the predictive capabilities. However, increasing the order of Markov models increases the number of parameters in the system, resulting in higher memory usage and state-space complexity [DK04]. We build Multiple first order Markov models

(MfoMm) within SPAN to address these problems. Before presenting our clustering technique, we describe MfoMm to understand the way in which we form our sequences of requests.

3.1.3 Multiple first order Markov models (MfoMm)

From Table 3, we find that the overall frequency counts of transition from p_3 to pages p_2, p_4 and p_5 are 3, each. Such equal frequency counts create ambiguity in prediction as all three pages are equally likely to be visited. This ambiguity could arise even when frequency counts of individual subjects are considered. A solution for overcoming this problem is to build higher order Markov models. They provide better prediction capabilities [DK04, SH05]. However, as the order of the Markov model increases, the memory requirements and the state-space complexity increase, and larger training sets are required for obtaining higher frequency counts [DK04, SH05]. The number of parameters of Markov models are m^n, where m is the number of resources and n is the order of the Markov model. The parameters grow exponentially as the order of Markov models increase. MfoMm captures the features of higher order Markov models while maintaining the low memory requirements and state-space complexity of the first order Markov models. The parameters of this model are $n * m^2$. They increase linearly to the number of steps in the model.

To explain the combination process, we consider a sequence with 3 requests. We assume that the first two requests have already been made by the subject and our model attempts to predict the third request. Following the example from Section 3.1.1, we assume that the first two requests were made for p_2 and p_3. The first step of MfoMm is similar to the one described earlier, where the model finds the probability of p_2, p_4, and p_5 being requested after p_3, i.e., it finds $p(p_2|p_3)$, $p(p_4|p_3)$, $p(p_5|p_3)$. We don't consider any access control policies at this stage. They are incorporated in our clustering algorithms. The next step considers a first order Markov model that checks the request made by the subject just before p_3 was requested. In our example, the subject requested for p_2. As the first step indicates that p_2, p_4, and p_5 are the likely requests after p_3, we determine the likelihood of p_2, p_4 and p_5 being requested in the same session, where p_2 is the first request, and p_2, p_4, or p_5 are the third requests in a sequence. In this step, we skip the second request. The tuples of requests formed would appear in the form $(p_2, ?, p_2)$, $(p_2, ?, p_4)$ and $(p_2, ?, p_5)$, and we find $p(p_2|p_2, ?)$, $p(p_4|p_2, ?)$, $p(p_5|p_2, ?)$. The '?' sign represents any request made by the subject that was preceded by p_2 and followed by either p_2, p_4, or p_5, and not necessarily p_3. Now, if p_4, is the third request made by the subject, it could be attributed to 3 possibilities: (1) because p_2 was the first request, or (2) because p_3 was the second request, or (3) because p_2 was the first request and p_3 was the second request. Thus the probability of p_4, given that p_2, and p_3 have been the first and second requests is given by,

$$p(p_4|p_2, p_3) = 1 - [(1 - p_4|p_2) * (1 - p_4|p_3)]$$

Our model considers the dependencies between only two requests at any point of time. Thus, this model can be thought of as an extended version of first order Markov model. Since only two requests are considered at a time, the parameters of the model grow linearly, and it is proportional to the number of steps of the MfoMm. For a sequence, where l requests have been made, the $(l+1)^{th}$ request can be predicted using the following formula,

$$p(P_{l+1} = p_i|P_1, P_2, ..., P_l) = 1 - \prod_{j=1}^{l} [1 - p(P_{l+1} = p_i|P_j = p_j)] \tag{1}$$

Equation 1 gives the probability of every request being made in a session. If we suppose that the number of steps considered to calculate the probability of each likely request is $G - 1$, the probability of the entire sequence (Se) having N permissions is given by,

$$p(Se) = p(P_1) * \prod_{i=2}^{N} \left[1 - \prod_{j=1}^{G-1} (1 - p(P_i = p_i|P_{i-j} = p_{i-j})) \right] \tag{2}$$

Since this equation represents a single session of a subject, it does not provide sufficient statistics for interpreting the behavior of the subjects in the enterprise. In our clustering technique, we clusters the sequences of requests by combining the individual subject's frequency counts and total frequency counts to gain the statistics.

3.1.4 Clustering

In this section, we propose the clustering technique. To achieve our goal, we cluster the available sequences formed using frequency counts from Table 3.

1. A sequence is the series of all the requests made by a subject in a session denoted by x = (P_1,P_2, P_3,..., P_{Ns}). Our model assumes that L unique sequences are formed by all the subjects. Every sequence is represented as x_j where $1 \leq j \leq L$
2. The system is assumed to have M subjects denoted by u_i ($1 \leq i \leq M$) and every subject is assumed to log in and out of the system several times creating a number of sessions for itself. We represent subjects and their corresponding sessions as u_i = (x_{i1},x_{i2}, x_{i3},..., x_{iQ}), where Q denotes the number of sessions made by a subject.
3. We assume that there are N permissions in the system that could be requested and we denote them as p_1,p_2, p_3,..., p_N.
4. We split the sequence of requests made across various sessions into subsequences each denoted as y and assume that there are T unique subsequences formed. Every subsequence is represented as y_t ($1 \leq i \leq T$).

Our goal is to find the probability of a subject requesting for permissions in a sequence. We represent this probability as $p(x_j|u_i)$. As our model depends on obtaining comparable frequency counts of the requests made, we split the sequence x_j into several smaller subsequences y_t, each having S permissions. We believe that smaller subsequences would be repeated more often than longer ones. This helps in improving the frequency counts from the available data. We group the subsequences into clusters that would help us build a complete sequence for predictions. Using Equation 2, the probability of a subject requesting for a subsequence is given by,

$$p(y_t|u_i) = p(p_{1_t}|u_i) * \prod_{i=2}^{S} \left[1 - \prod_{j=1}^{G-1} (1 - p(p_i|p_{i-j}, u_i)) \right] \tag{3}$$

We make the same assumption as LDA and cluster the subsequences. We assume that there are K clusters formed. We adopt a soft-clustering approach, where every subject is assumed to be associated with multiple clusters with different degrees of affiliation. The association of every subject belonging to the clusters can be represented by a vector π_i, where $\pi_{i,c}$ denotes the probability of subject i belonging to cluster c. The sum of the probabilities in vector π_i is equal to one. For all the M subjects in the system, matrix θ would represent the probabilities of all the subjects belonging to the clusters. This matrix has a dimension of $M * K$. The sum of the elements in any row equals one. The concept of soft-clustering is contrary to hard clustering where every subject is assumed to request permissions from only one cluster. The graphical model representation of the concept is shown in Figure 4. In this figure, 'S' and 'C' denote all the subjects and clusters in the system, and θ denotes the $M * K$ dimensional matrix. Subsequences too belong to one or more clusters with different degrees of affiliation. Considering the same analogy used for subjects, matrix ϕ represents the probability of subsequences being affiliated to the clusters. The size of the matrix is $T * K$. Note that the clusters are actually latent (unobserved) in the model.

With the latent clusters in the model the probability of a subsequence being requested by a subject would be

$$p(y_t|u_i) = \sum_{c=1}^{K} p(y_t|z_c)p(z_c|u_i) \tag{4}$$

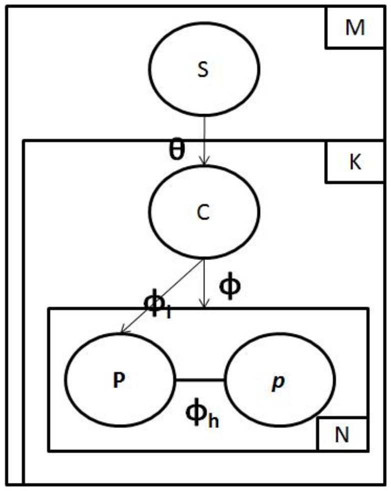


Figure 4: Graphical model representation of SPAN. The boxes are plates representing replicates. The outer plate represents subjects, while the inner plate represents the clusters and sequences.

By substituting the value for y_t from equation 3, we obtain

$$p(y_t|u_i) = \sum_{c=1}^{K} p(p_{1_t}|z_c) * \prod_{i=2}^{S} \left[1 - \prod_{j=1}^{G-1} (1 - p(p_i|p_{i-j}, z_c))\right] * p(z_c|u_i) \tag{5}$$

The first term $p(p_1|z_c)$ represents a matrix of size $N * K$ that we represent by ϕ_I and the second term would result in a matrix size of $N * N * K * G$ represented by ϕ_T. Many entries in the second term would be zero, creating a sparse matrix[2]. We designed our algorithms by taking the sparse nature of this matrix into account for reducing the memory required to store this matrix and efficiently retrieving the information out of these matrix. For the sake of simplicity, we present our analysis by considering the second term as a first order Markov model, instead of considering MfoMm. The analysis for MfoMm remains the same as regular first order Markov model. With this assumption, Equation 5 can be written as,

$$p(y_t|u_i) = \sum_{c=1}^{K} p(p_{1_t}|z_c) * \prod_{i=2}^{S} p(p_i|p_{i-1}, z_c) * p(z_c|u_i) \tag{6}$$

The likelihood of all the subsequences and the clusters is given by,

$$p(D, Z|\theta, \Phi) = \prod_{c=1}^{K} \prod_{t=1}^{T} \prod_{i=1}^{M} [p(p_{1_t}|z_c)^{N_{1,c}} * \prod_{i=2}^{S} p(p_i|p_{i-1}, z_c)^{N_{i,i-1,c}} * p(z_c|u_i)] \tag{7}$$

Here D and Z represent all the subsequences and clusters, respectively. In this equation Φ encompasses the 3 parameters, ϕ, ϕ_I, ϕ_T. Given a part of subsequence accessed by a subject, our aim is to find the probability of the subject requesting other permissions in the subsequence. We adopt a Bayesian approach and attach priors to all the parameters required for making predictions. A subject can be associated with one of K clusters in k possible ways, representing a multinomial distribution with a parameter θ. From the theory of Bayesian analysis [Hec95], we choose a dirichlet prior with hyperparameter α for the parameter θ of the multinomial distribution. Similarly, subsequences can be associated with the clusters in k different ways, representing a multinomial distribution with a parameter ϕ. We choose a dirichlet prior with

[2]A sparse matrix is a matrix populated primarily with zeros

hyper parameter β for the parameter ϕ. Corresponding priors for ϕ_I and ϕ_T are denoted by β_I and β_T respectively. The prior distribution for the parameter θ given the hyperparameter α is,

$$p(\theta|\alpha) = \frac{\Gamma(\alpha_0)}{\Gamma(\alpha_1)\cdots\Gamma(\alpha_K)} \prod_{c=1}^{K} \theta_c^{\alpha_c - 1} \tag{8}$$

where $\Gamma(\cdot)$ denotes the Gamma function and $\alpha_0 = \sum_c \alpha_c$. Similar distributions can be obtained for parameters ϕ, ϕ_I and ϕ_T with dirichlet prior having the hyperparameters as β, β_I and β_T respectively. The posterior distribution of our model is obtained by multiplying the priors with the likelihood (equation 7). The posterior distribution is given by,

$$p(D, Z|N, U, \alpha, \beta, \beta_I, \beta_T) \propto \prod_{c=1}^{K}\prod_{t=1}^{T}\prod_{i=1}^{M}[p(p_{1t}|z_c)^{N_{(p_{1}y_t, c)}} * \prod_{j=2}^{S} p(p_j|p_{j-1}, z_c)^{N_{(p_j, p_{j-1}, z_c)}} * p(z_c|u_i)^{N_{(z_c, u_i)}} \\ \times \prod_{c=1}^{K}\theta_c^{\alpha_c - 1}\prod_{c=1}^{K}\phi_c^{\beta_{I_c}-1}\prod_{c=1}^{K}\prod_{i=2}^{S}\phi_c^{\beta_{T_{(i|i-1,c)}}-1} \tag{9}$$

Therefore,

$$p(D, Z|N, U, \alpha, \beta, \beta_I, \beta_T) \propto \prod_{i=1}^{M}\prod_{c=1}^{K}\theta_c^{N_{(z_c,u_i)}+\alpha_c-1}\prod_{j=1}^{N}\prod_{c=1}^{K}\phi_{I_c}^{N_{(p_j,z_c)}+\beta_{I_c}-1}\prod_{c=1}^{K}\prod_{i=2}^{S}\prod_{j=1}^{N}\phi_{T_c}^{N_{(i,j,z_c)}+\beta_{T_{(i,j,c)}}-1} \tag{10}$$

In the above equations, ϕ and β are not directly taken into account, but they would be required for computing the initial and transition probabilities. The parameters of this model are obtained by using the Expectation Maximization (EM) algorithm [PMB77]. This algorithm optimizes the non-concave function through gradient accent using two steps: the expectation step (E-step) calculates the joint likelihood of observing the data, given the current estimates of the model parameters, and the maximization step (M-step) optimizes the model parameters from the likelihood of data that is calculated in the E-step. The EM algorithm is an iterative algorithm that iterates between the E and the M steps until convergence. The criteria for convergence is the step that maximizes the likelihood of the data. Since logarithms are monotonic transformations, maximizing the log-likelihood gives the same result as maximizing the likelihood.

- In the E-step, we find the probability of the clusters given the subjects and their requested sequences. The probability of the clusters is given by,

$$p(z_c|y_t, u_i) = \frac{p(y_t|z_c)p(z_c|u_i)}{\sum_c[p(y_t|z_c)p(z_c|u_i)]} \tag{11}$$

  Substituting Equation 4 in Equation 11, we get,

$$p(z_c|y_t, u_i) = \frac{p(p_{1_t}|z_c) * \prod_{i=2}^{S} p(p_i|p_{i-1}, z_c) * p(z_c|u_i)}{\sum_{c=1}^{K} p(p_{1_t}|z_c) * \prod_{i=2}^{S} p(p_i|p_{i-1}, z_c) * p(z_c|u_i)} \tag{12}$$

  This equation calculates the probability of a cluster for a subject given one sequence. Overall, M subjects would access T sequences through K clusters. Thus, this equation would have to calculate $K * M * T$ values.

- In the M step, the parameters that are required to estimate the E step are optimized. We optimize the parameters θ and ϕ in this step.

$$\theta_{i,c} = \frac{\alpha_c + \sum_{t=1}^{S} p(y_{t_{i,c}})}{\sum_{c=1}^{K}[\alpha_c + \sum_{t=1}^{S} p(y_{t_{i,c}})]} \tag{13}$$

$$\phi_{t,c} = \frac{\beta_c + \sum_{i=1}^{M} p(y_{t_{i,c}})}{\sum_{c=1}^{K} [\beta_c + \sum_{i=1}^{M} p(y_{t_{i,c}})]} \tag{14}$$

$$\phi_{I_{j,t,c}} = \frac{\beta_{I_c} + \sum_{t=1}^{S} \mathbb{I}_{j,t,c}\phi_{t,c}}{\sum_{c=1}^{K} [\beta_{I_c} + \sum_{t=1}^{S} \mathbb{I}_{j,t,c}\phi_{t,c}]} \tag{15}$$

$$\phi_{T_{i,j,t,c}} = \frac{\beta_{T_c} + \sum_{t=1}^{S} N_{(i,j,c)}\phi_{t,c}}{\sum_{c=1}^{K} [\beta_{T_c} + \sum_{t=1}^{S} \sum_{t=1}^{S} N_{(i,j,c)}\phi_{t,c}]} \tag{16}$$

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

1. Given a dataset containing sequences of requests, form unique sequences of fixed sizes.
2. Count the number of times each sequence was requested by every subject. Also calculate the total number of times these sequences were requested by all the subjects.
3. Fix the step size for MfoMm and count the frequency count of transitions between any two requests of the datasets.
4. Once all the frequency counts are obtained, start the EM algorithm that can be summarized as follows:
   (a) Randomly initialize the values in the Equation 12 for the E-step. Choose the values of the hyperparameters and set iteration $i = 0$.
   (b) Calculate the parameters of the M-step from the values obtained in E-step.
   (c) Using the new parameters of the M-step, calculate the values for every element of the cluster using Equation 12 of the E-step.
   (d) Using the new E-step calculate the log-likelihood.
   (e) If difference between the log-likelihood of current iteration and previous iteration does not change considerably, terminate the algorithm, else $i \longleftarrow i+1$, and go to step b.

The EM algorithm is run iteratively until it converges. The memory requirements of implementing SPAN are $O(MTK + MK + KT + NK + KN^2)$. Thus, the memory required would be directly proportional to the number of subjects, unique sequences, and transitions formed in the systems. It also depends on the number of clusters that are formed. The time complexity of the algorithm can be given by $O(IKMTN(N+1))$, where I is the number of iterations required for the EM algorithm to converge. As the number of subjects and permissions grow in the system, the time required for the algorithm to converge also grows linearly. However, not all subjects request for all the sequences in the system. This gives rise to matrices that are sparse in nature. While implementing the algorithm, we have taken the sparse nature of the matrices to find the association of subjects and sequences to the clusters. Next, we present details about the testing phase of the algorithm.

3.2 Testing phase

From the clusters obtained during the training phase, if a subject u_i requests for sequence y_t of length t, the membership of y_t requested by u_i can be found by,

$$p(z_c|y_t, u_i) \propto p(y_t|z_c) * p(z_c|u_i)$$

$$p(z_c|y_t, u_i) \propto \sum_{c=1}^{K} p(p_1|z_c) * \prod_{i=2}^{S} p(p_i|p_{i-1}, z_c) * p(z_c|u_i) \tag{17}$$

The proportionality is used because the equation needs to be normalized over all clusters. Once the membership is obtained, the $(t+1)^{th}$ request can be predicted using the following equation,

$$p(y_{t+1}|y_t, u_i) = \sum_{c=1}^{K} p(y_{t+1}|z_c) * p(z_c|y_t, u_i) \tag{18}$$

In this section, we described the design of SPAN. We discussed the shortcomings of using web prediction techniques for access control system. SPAN overcomes the shortcomings by considering the identity of subjects to form clusters. Forming clusters is an offline process that involves analyzing the past behavior of subjects. The testing phase of the algorithm is an online phase, where SPAN predicts the requests for subjects logged into a system. In the next section, we report the experiments conducted and results obtained to evaluate the design of SPAN.

4 Evaluation

While the previous section described the model for building SPAN, we now present the experimental evaluation. We used a simulation based approach for evaluating SPAN. In Section 4.1, we first describe the datasets obtained for evaluation and our methodology of processing them. In Section 4.2, we describe our experimental setup. Next, we describe the measurement criteria in Section 4.3, and finally, in Section 4.5, we present our results.

4.1 Evaluation datasets and processing

The first step towards conducting the experiments is to record the requests made by the subjects in the system. There are two options to collect this information. The first option is an online process where SPAN records the sequences of requests. It then runs the modeling algorithms on the recorded requests. In this online process, the number of requests recorded before running the algorithms would depend on the settings specified by the administrator. Future requests are then predicted by SPAN. The second option is to obtain log traces, where the sequences of requests have already been recorded. We adopted the second option to conduct our experiments. We divided the log traces into two parts: the training set and the testing set. The data present in the training set was used to build the model. This step was termed the *training phase*. The testing set was used to evaluate the effectiveness of the prediction model. This step was termed the *testing phase*.

First, we explain the structure of tuples to be extracted from the sequences of requests available in the log files. We propose to extract tuples of the following structure *(su,se,r,a,p,i)*, where su is the subject making the request, se is the session in which the request is made, r is the requested resource, a is the action requested on the resource, p is the PEP making the request, and i is a unique identifier for every tuple. In some situations, the log files don't contain explicit information about the attributes required for SPAN. If a subject cannot be directly extracted from the trace, we suggest considering the IP address of the requestor as the subject. In the case where information about the session is not explicit in the log trace, time difference between requests can be used to extract session information. Following the approach adopted in [SH05], a session can be defined as all the requests made by the subject within a span of 2 hours from the first request. In cases where actions are not specified in the log traces, we recommend omitting the actions, and just focussing on predicting the resources that would be requested. This is common for web pages, where all requests are generally *read only*. For accommodating an architecture with multiple PEP's, the tuple maintains the identifier of the PEP making the requests. This has two benefits: (1) The PDP gains knowledge about the PEP making the requests and sends back the predicted responses to the correct PEP. (2) The information can be used for auditing purposes. Finally, i is the unique identifier of the request made. The response from the PDP has the following structure *(re,p,i,c)*, where re is the response to the request with identifier i. In this tuple, c uniquely identifies the response tuple; and p is a variable to accommodate an architecture with multiple PDP's.

Our first dataset was obtained from WebCT, provided by the University of British Columbia (UBC). WebCT [Web] (Course Tools) or Blackboard Learning System, now owned by Blackboard, is an online proprietary virtual learning environment system that is sold to colleges and other institutions and used by over 10 million students in 80 countries for e-learning [Web]. Instructors can add lecture notes for the courses they offer and add tools such as discussion boards, mail systems, assignment systems, and live chat. Instructors can also provide grading system through WebCT. Students registered for the course can read lecture notes and participate in the discussions, view and submit their assignments, and chat to other registered members of the course. They can view their grades and maintain an email account. There are other roles like teaching assistants, administrators, etc., that can be added to the course. The latest version of this software is now called "Webcourses". We believe that WebCT is a good example of a typical web application, where people with different identities have different levels of access to the resources in the system. Thus, we decided on evaluating SPAN algorithms on the traces obtained from WebCT.

We obtained an anonymized trace of $210,000$ requests from UBC. The trace contained information about requests made in an online course offered at the university. It contained information about the subjects, their actions in various sessions, the time when they made those requests, and the role in which they logged in the system. The course had 11 instructors, 3 teaching assistants, and 42 students, for a total of 56 subjects. Every resource had different levels of access. We considered every resource with different actions as a separate resource. Thus, a resource, say A, having actions *read* and *write* would represent two resources. The first one would be A-*read*, and the second one would be A-*write*. In this way, there were 4696 resources.

From the WebCT trace, we obtained 3 different sets of sub-traces: (1) the entire trace (2) requests from 1 to $100,000$ (3) requests between $75,000$ to $175,000$. The first sub-trace followed the standard procedure of running the experiments on all the available requests in the datasets, and is described by most of the approaches mentioned in our related work. After manually observing the complete trace, we found that several requests at the start of the trace were made by the instructors of the course. Most of their actions pertained to adding resources in the system, and these actions were often not repeated in the later part of the trace. Also, students couldn't perform most of these actions. Thus, in our second sub-trace we captured the first $100,000$ requests as the entire trace. In this sub-trace, the training phase mostly got trained on the behavior exhibited by the instructors. The testing phase had requests made by all the subjects in the system. Our second observation was made for requests in the range $75,000$ - $175,000$. We observed that it mostly contained requests for reading course contents, and discussing topics on the discussion boards. The reason for obtaining these sub-traces was to understand the effect of different access patterns on the performance achieved.

We obtained our second dataset from requests made by the users in the 'Fighters Club' (FC) application of Facebook [NRC08]. It is one of the first games to be launched on Facebook, and it evolved over a period of 9 months to have been played by over 3.44 million users. FC allows users to pick virtual fights with their Facebook friends that lasts from 15 to 48 hours. For the duration of the fight, each player may request support from their Facebook friends, who then help the individuals team defeat the opposing users team through a series of virtual hits. Users on FC can have one of the three possible roles. (1) Offender: The user instigating the fight is the offender. (2) Defender: The Facebook friend picked on by the offender is the defender. (3) Supporter: The offender and the defender may receive support from their Facebook friends who are called the supporters. Every fight has a unique fight id.

The trace obtained for the FC application can be associated to requests made by subjects in access control systems. When a facebook user starts a fight, he or she gets a unique fight id. Users can be considered as subjects, and fight ids as their sessions. In every fight, users receive help from their friends. The order in which they receive help, can be considered as a sequence of requests made by subjects in their sessions. To summarize, the offender or defender act as the subjects, fight ids as their sessions, and supporters are the permissions to be predicted. A second aspect of this trace is that users can receive help only from their friends. This is similar to those access control systems where subjects can request for only those permissions that they are authorized to access. From the vast number of facebook users, offenders and defenders can receive help from a small subset of users, who are their friends. This is similar to subjects

Trace	**Number of subjects**	**Number of unique requests**	**Number of unique sequences**	**Training time per iteration of clustering (minutes)**	**Time for predicting each request (ms)**
WebCT1	56	$4,696$	$22,418$	28	2.34
WebCT2	56	$2,642$	$12,984$	16	2.52
WebCT3	56	$1,482$	$9,318$	15	2.04
FC1	50	$1,780$	$5,671$	12	9.6
FC2	100	$2,424$	$10,092$	24	9.6
FC3	200	$3,211$	$19,116$	47	8.28

Table 4: Summary of the datasets used for experiments.

requesting for a subset of permissions from the entire subspace of permissions.

The FC dataset contained over 23 million requests made by $43,669$ users. The memory and time required to form clusters in our algorithm directly depends on the number of subjects and the unique sequences of requests formed. The present implementation of our algorithm does not support the memory required for such huge datasets. To meet the memory requirements of our algorithm, we formed 3 sub-traces of requests made by 50, 100, and 200 users. We randomly picked all the users for our sub-traces. We started with 50 users to compare the time required to form clusters using this dataset, as compared to WebCT that contained 56 users. We then doubled the number of subjects in every subtrace, which increased the number of requests and unique sequences of requests formed. Time required to form clusters in each subtrace, gives an idea about the scalability of the algorithm.

We simulated our third dataset using Zipf distribution to model the uneven popularity of permissions to the subjects. Zipf distributions have been widely used to model heterogeneous popularity distributions (e.g., web page popularity [BCF+99], web site popularity [AH02], and query term popularity [KLVW04].) A set of data obeys Zipf's law if the frequency of an item is inversely proportional to (some non-negative) power of its rank (determined by frequency of occurrence). More formally, suppose we have a frequency distribution $(x_1, f_1), \ldots, (x_n, f_n)$, where data item x_i occurs f_i times and $f_1 \geqslant f_2 \geqslant \cdots \geqslant f_n$. Then the distribution obeys Zipf's law if

$$f_i \propto \frac{1}{i^\alpha}$$

for some $\alpha \geqslant 0$. The simulated dataset had a total of $100,000$ requests made by 100 subjects over 3000 permissions. For every session, we simulated 100 requests being made by the subjects. We varied the value of α from 0 and 1.5 in steps of 0.5. As the value of α becomes smaller, the popularity distribution becomes less skewed, collapsing to a uniform distribution when $\alpha = 0$. Using this distribution, we were interested in predicting the behavior of subjects in web page prediction environment.

4.2 Experimental setup

In this section, we present the setting of our experiments. We conducted all our experiments on a machine with Intel Pentium 1.73GHz dual-core processor having cache memory of 1 MB and RAM memory of 2 GB. Our training and testing phases were implemented in Matlab version 2009a that contains a statistical tool box.

We divided every sub-trace into training and the testing set for the training and testing phases of SPAN. Our simulation testbed is as shown in Figures 5. In the training phase, we fed the requests from the training set into SPAN. SPAN found the number of unique requests in the trace. It formed all the possible short sequences of requests from the long sequences. It recorded the number of times the short sequences were repeated in the entire trace. It also recorded the

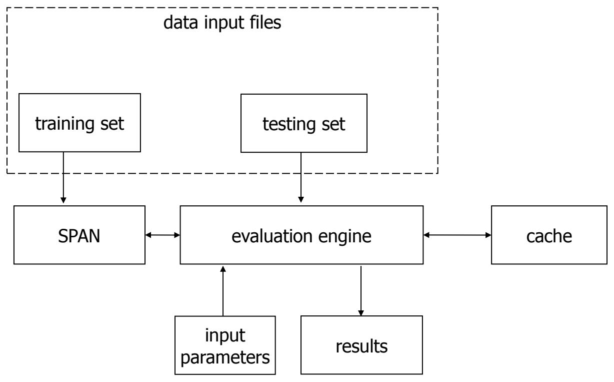


Figure 5: Experimental setup for evaluating SPAN

number of times every subject requested for these short sequences. It calculated the transition between any two requests in the trace. After recording all these numbers, SPAN clustered the short sequences of requests.

We had to choose priors and the number of clusters for implementing the training phase. We conducted our experiments for prior values of 0.1, 0.3, 0.6, and 1. For the log traces we considered, the log-likelihood of clustering was not greatly affected by the different values of the priors. Thus, we decided to run all the experiments for a prior of 0.3. To select the number of clusters, we varied the number of clusters between $1 - 10$ in steps of 1 and recorded the log-likelihood for every run. The number of clusters in the training phase that provided the maximum log-likelihood during convergence was chosen as the number of clusters for the testing phase. We determined that 4 and 7 clusters gave the optimum value of log-likelihood for the WebCT and FC datasets, respectively. Thus, we decided to use 4 and 7 clusters for the two datasets. We followed the same convergence criteria as proposed by Cadez et al. [CHM+03]. If the log-likelihood between two steps of EM algorithm differed by less than 0.1%, we assumed that the algorithm converged. As the algorithm initially starts with random assignments of probability values, we restarted the algorithm if it failed to converge in 25 steps.

In the testing phase, the evaluation engine read the requests from the testing set one after the other. The evaluation engine forwarded a copy of the requests to SPAN. SPAN predicted the probable requests for sequences of requests sent by the evaluation engine. Based on the setting specified by the input parameters and requests available in the cache, the evaluation engine computed the achieved performance. We discuss the detailed implementation of the input parameters and cache in Sections 4.2.1 and 4.2.2, respectively.

For each of our sub-traces, Table 4 provides details about the number of subjects, unique permissions, and unique sequences formed. Every sequence in the table is of size 3. The table also provides the average time required for every iteration of training phase, and the average time required to make a prediction. For every sub-trace, we carried the training and testing phase of the algorithm two times. As we started the clustering algorithm with random assignments, the two runs of experiments in the training phase were used to confirm that unique clusters were formed each time. For both the runs, we ensured that the difference between the log-likelihood at the time of convergence was less than 0.1%. Table 4 indicates that the training time increases as the number of subjects and sequences of requests increase in the system. However, the time required to predict every request is relatively small. The two runs in the testing phase confirmed the results obtained in that phase.

We observed that the approaches for web page predictions cited in Section **??**, fix a number of requests from the trace as training set, and the remaining requests become the testing set. The training and the testing phase of the algorithms are conducted on these fixed sets of requests. We wanted to analyze the effect of different sizes of training and testing sets on performance. We wanted to observe if larger training sets gave better performance on smaller testing sets. To evaluate this, we decided to vary the number of requests in training and testing sets. We divided each of our traces into different sized training and testing sets. The training sets contained 50% to 90% requests from the total number of requests in a trace. Requests in the testing sets varied from 50% to 10%, respectively. We measured the performance for all sub-traces varying the size of training and testing sets.

We implemented our next set of experiments to evaluate MfoMm, built within SPAN. We varied the number of steps of the algorithm from 1 to 4. As the number of steps increased, the time required to form clusters increased incrementally. A different training phase was required for each step of MfoMm. We maintained the number of steps common between the training and the testing phase. This means that if the training phase was trained for a step size of 3, we tested the algorithm using the same step size.

Next, we describe the settings of input parameters and cache that were collocated with the evaluation engine.

4.2.1 Input parameters

For every sequence, SPAN predicts the likely requests that would be made by the subjects with certain probability. We set two input parameters for the evaluation engine as described below:

1. SPAN predicts requests with certain probability. In the first setting, the evaluation engine considers the most probable request for every sequence and verifies it against the requests read from the testing set. In the second set of experiments, three most probable requests for every sequence are considered and checked against the requests read from the testing set. Considering more requests per sequence improves the performance, but increases the load on the PDP because in a real world setting, the PDP has to compute responses to the predicted requests. The input parameters set in this experiment gives an idea about the tradeoff between performance gain v/s additional load encountered by the PDP.
2. Requests predicted with higher probability values are more likely to be requested than those predicted with lower values. The probability with which SPAN predicts the requests can be converted to confidence levels. Confidence levels are nothing but probability values ranging between $0-1$ scaled to percentage values ranging between $0-100$. Thus, a confidence level of 0.01% corresponds to probability value of 0.0001 and 10% corresponds to 0.1. We varied the confidence level from 0.01% to 10%. The evaluation engine considered predictions from SPAN as valid, only if they were predicted with a confidence level greater than the preset confidence level. In a real world setting, to reduce the number of unnecessary computations made by the PDP, we propose that responses should be precomputed only if requests are predicted with a certain confidence level. Note that our confidence levels are quite low. This is due to the fact that SPAN has to predict permissions in a sequence from thousands of available permissions in the system, which results in very low probability for each predicted permission.

We now describe our design for combining SPAN and cache in a system.

4.2.2 Cache implementation

To evaluate the performance obtained by SPAN as compared to caching techniques, we implemented two types of cache: FIFO and LRU. For each implementation, we recorded the performance obtained by stand alone cache. We combined each implementation of cache with SPAN and recorded the performance obtained by the combination. In our experiments, the evaluation engine cached the requests that it read from the testing set. The size of the cache was a percentage of total size of permissions in the system. We varied the size from 0% to 100%. Initially the caches were filled from the requests available in the training set. At the start of the

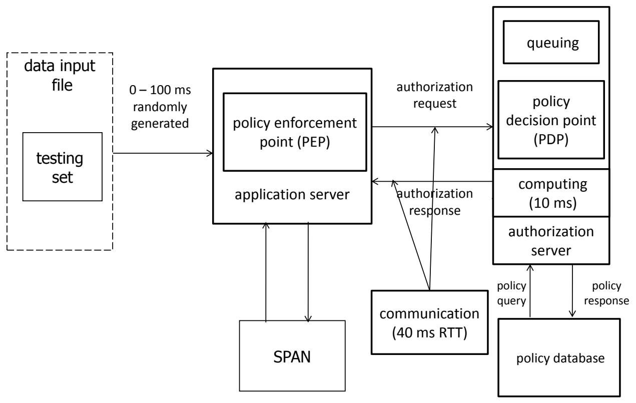


Figure 6: Simulation setup for latency calculation

experiment, the FIFO and LRU caches read requests from the training set equal to its size using the FIFO and LRU techniques, respectively. Reading the requests from the training phase was termed as the warming phase [WCBR08]. In the testing phase of stand alone caching system, if requests read from the testing set were found in the cache, it was considered as an improvement in performance. In the implementation where SPAN was combined with one of the caches, if requests read from the testing set were either found in the cache or predicted correctly by SPAN, it was considered as an improvement in performance. Using this experiment, we were interested to find the performance obtained by combining SPAN and cache, over stand alone caching technique.

4.2.3 Batch and Online Algorithms

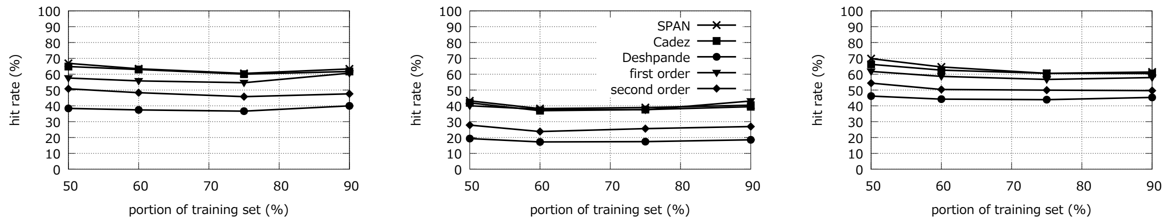
Table 4 summarizes training time required for every dataset. To make predictions with certain degree of confidence, the number of times (frequency count) subjects request for certain sequences has to be comparatively higher than the requests for other sequences. To adopt to the changing behavior of the subjects in the system and gather sufficient frequency counts, we adopted a combination of batch and online algorithms. Initially clusters were formed using the methodology described in Section 4.2. We performed online training process for WebCT2 and WebCT3 datasets. WebCT2 had a changing access pattern in the training and testing set, whereas WebCT3 had a fairly similar pattern in both sets. Training time required for the online phase would suggest SPAN's ability to accommodate changing behavior of the subjects in the system. To implement the online training, we added first 10,000 requests from the testing sets of the datasets to the training sets. At the same time we removed the first 10,000 requests from the training sets. This maintained the number of requests in the training sets. We also repeated online training for 20,000 requests. Adding 10,000 and 20,000 requests for online training also sufficed the condition on frequency counts for judging the changing behavior of subjects.

Next, we present our evaluation metrics before we present our results.

4.3 Measurement criteria

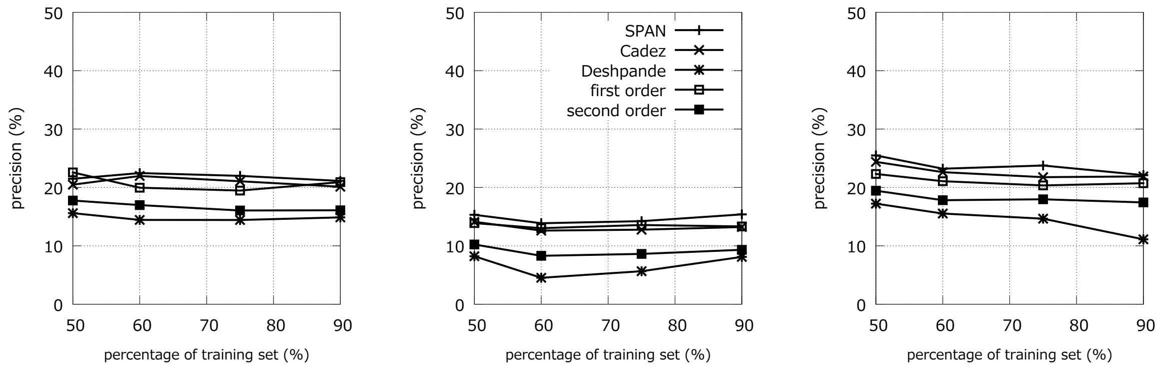
In this section, we define the metrics used to evaluate SPAN for different settings in which we conducted the experiments.

First, we study the hit rate, which we define as the ratio between the number of precomputed responses used to serve the requests made by subjects, to the total number of requests made by the subjects. A high hit rate indicates that the model can predict future requests efficiently



(a) WebCT1 (b) WebCT2 (c) WebCT3

Figure 7: Hit rate obtained for WebCT dataset when 3 most probable responses are fetched for every sequence.



(a) WebCT1 (b) WebCT2 (c) WebCT3

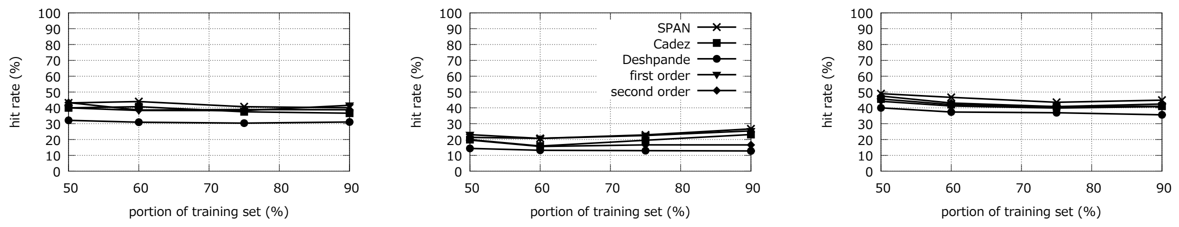
Figure 8: Precision obtained for WebCT dataset when 3 most probable responses are fetched for every sequence.

and compute the responses even before the request is made by the subjects. As the PDP precomputes the responses to these predicted requests and pushes them to the PEP cache, a higher hit rate indirectly indicates that latency is virtually zero. In our experimental setup, hit rate was the ratio of the number of requests considered by the evaluation engine after taking the input parameters and cache into consideration to the total number of requests read from the testing set.

As computing authorization responses can be expensive in certain applications, responses to predicted requests that are not requested by the subjects can be considered an unnecessary overload on the system. To gain knowledge about the unused predictions, we study the precision of the algorithm. We define precision as the ratio of the total number of precomputed responses used to serve the requests made by subjects, to the total number of responses precomputed by the PDP. Precision is an indirect measure to calculate the additional load on the PDP. A higher precision indicates that the PDP precomputes minimum number of responses that are unused, thus reducing the unnecessary overhead on the system. For our experiments, precision was the ratio of the number of requests in the evaluation engine that matched the requests read from the testing set to the total number of requests in the evaluation engine after taking the input parameters and cache into consideration.

Confidence level affects the number of computations to be performed by the PDP. Thus, we study the drop in the number of computations for increasing confidence levels. In our experiments, we calculate the drop in the valid requests present in the evaluation engine for increasing confidence levels.

Now, we consider a simulation setup to understand the mapping of our evaluation metrics



(a) WebCT1 (b) WebCT2 (c) WebCT3

Figure 9: Hit rate obtained for WebCT dataset when only one most probable response is fetched for every sequence.

into latency reduction

4.4 Simulation setup for latency reduction

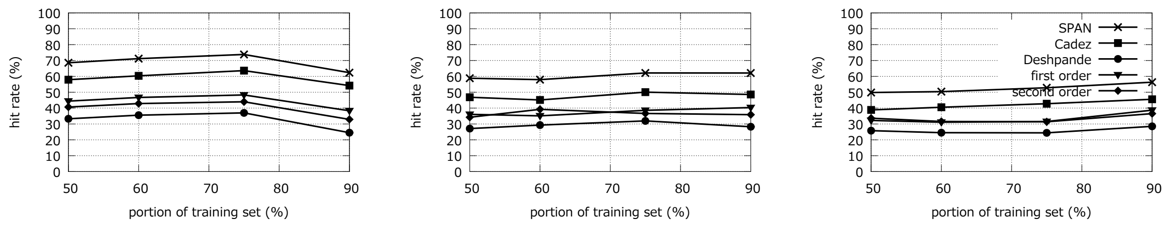
To gain knowledge about the reduction in latency using SPAN, we built an experimental testbed as shown in Figure 6. We split the evaluation engine in two modules: the PEP and the PDP. The PEP read every request from the testing set between an interval of $0 - 100$ ms from the previous request. We used uniform distribution to generate this interval. The analogy of generating requests in this interval was to simulate a real world scenario, where requests arrived at the PEP within an interval of 100 ms from the previous requests. The PEP forwarded these requests to the PDP and SPAN. SPAN predicted the possible requests that could be read from the testing set by the PEP. The PEP sent the predicted requests to the PDP. We introduced fixed communication delays of 40 ms between the PEP and the PDP, and a computation delay of 10 ms for every request as proposed in [Wei09] and [BGR07], respectively. We assumed that the communication delay between the PDP and policy database is negligible as compared to the delay between the PEP and the PDP. Queuing delays get 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. The PDP prioritized requests read from the testing set over requests predicted by SPAN. In this setup, we calculated the latency experienced by the system for all the requests read from the testing set. Latency was the time difference between a request being read by the PEP from the testing set to the time it received a response from either its cache or the PDP.

We now present the results obtained from all our experiments.

4.5 Results

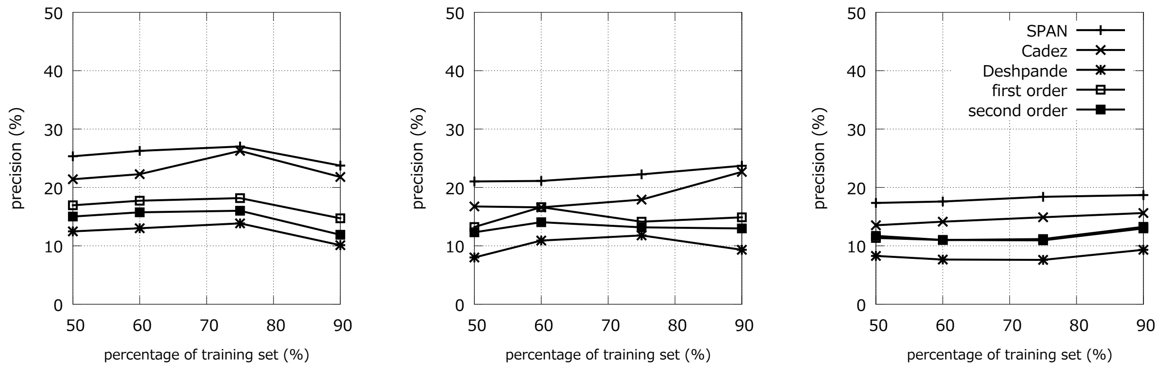
4.5.1 Hit rate and precision for different sizes of training and testing sets

Figures 7 and 8 show the hit rate and precision for different sub-traces of WebCT dataset, when 3 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. SPAN achieves an average hit rate of 63%, 41%, and 64% for the three sub-traces of WebCT. Corresponding precision is 21%, 13%, and 23%, respectively. We observe that the hit rate and precision for the WebCT3 is much higher than WebCT2. We selected WebCT3 trace to contain common access patterns between training and testing sets as compared to WebCT2. This implies that the hit rate and precision are higher when the patterns found in the training and testing sets are similar to each other. The average improvement in hit rate achieved by SPAN as compared to the first order, second order, and the algorithm proposed by Deshpande, is 6%, 15%, and 25%, respectively. Corresponding improvement in precision is 2%, 5%, and 7%. Overall, SPAN achieves better hit rate and precision as compared to the other algorithms we implemented.



(a) FC1 (b) FC2 (c) FC3

Figure 10: Hit rate obtained for FC dataset when 3 most probable responses are fetched for every sequence.



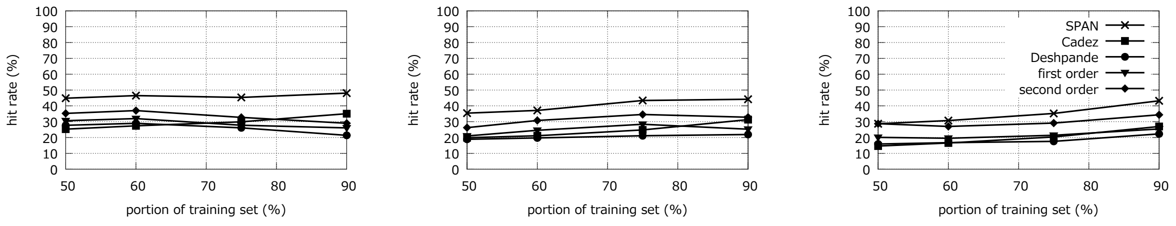
(a) FC1 (b) FC2 (c) FC3

Figure 11: Precision obtained for FC dataset when 3 most probable responses are fetched for every sequence.

However, we observed that our results closely matched the results obtained for Cadez. The improvement in hit rate and precision for SPAN ranged between 2% and 4%, when compared against Cadez. The possible reason for low improvement can be explained as follows: the WebCT dataset contained requests mostly by students and instructors of a course, and most of the requests were made for accessing the course material. The access control policies were the same for all subjects on these resources. This dataset closely matched a dataset that would be obtained for web pages without access control policies. Since SPAN and Cadez are implemented using clustering approach, SPAN could not achieve a significant improvement over Cadez for this dataset.

Figures 10 and 11, show the hit rate and precision obtained for the FC dataset. The average hit rate obtained by SPAN is 70%, 60%, and 50% for the 3 sub-traces of the FC dataset. Corresponding precision is 25%, 22%, and 18%, respectively. SPAN outperforms all the other algorithms in terms of hit rate and precision. The average improvement in hit rate achieved by SPAN is 10%, 21%, 20%, and 31% as compared to Cadez, first order, second order and Deshpande algorithms. The average improvement in precision obtained by SPAN is 3%, 6%, 8%, and 11%, respectively. In the FC application, Facebook users could receive requests only from their friends. From a total of 43,669 users, who played the game, each user could receive help from a small set of users. This is 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 clustering technique that considers the identity of subjects boosts the hit rate in such systems, as compared to the technique proposed in Cadez.

Figure 13 indicates the hit rate obtained for the simulated dataset. The hit rate increased



(a) FC1 (b) FC2 (c) FC3

Figure 12: Hit rate obtained for FC dataset when only the most probable response is fetched for every sequence.

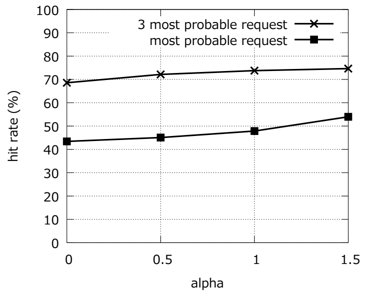


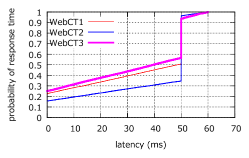
Figure 13: Hit rates obtained by varying coefficient α in Zipf distribution.

for increasing values of α. The hit rate for 3 most probable requests being predicted improved from 70% to 75% when the value of α increased from 0 to 1.5. Similarly, the hit rate improved from 43% to 53%, when the value of α changed from 0 to 1.5 in the most probable case.

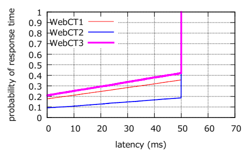
In all our datasets, hit rate drops when only the most probable request is considered by evaluation engine, but the precision increases considerably. For systems where computing authorization responses are expensive, higher precision would be preferred to decrease the additional load on PDP. For the WebCT dataset, the hit rate drops to 45%, 23%, and 50% from 63%, 41%, and 64% when the most probable requests considered by the evaluation engine changes from 3 to 1. By definitions of hit rate and precision, we note that precision is equal to hit rate when only 1 predicted request per sequence is considered by the evaluation engine. Thus the precision increases to 45%, 23%, and 50% as compared to 25%, 22%, and 18%. The precision increases by approximately $50\% - 100\%$. For the FC dataset, the hit rate drops to 48%, 40%, and 36% from 70%, 60%, 50%. In other words, the precision increases to 48%, 40%, and 36% from 25%, 22% and 18%. The improvement in precision is around $60 - 100\%$ for this dataset. From Figures 10, and 12, we observe that the second order Markov models perform better than the first order Markov models when only the most probable request per sequence is considered. From Figures 10 and 12, we also observe that the performance of Cadez drops considerably when only one request per sequence is predicted.

4.5.2 Latency calculation

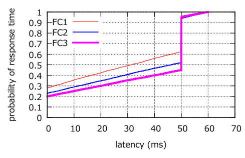
Figure 14 shows the cumulative distribution functions (CDFs) of the simulations conducted to demonstrate that SPAN reduces the latency introduced by the authorization process in access control systems. In Figure 14(c), we note that the probability of zero latency for the FC1 trace is 0.28. This implies that in 28% cases, a response would be present in the PEP cache when a request arrives at the PEP, thus reducing the authorization latency to zero. Next, we observe that in 62% cases, the probability of receiving a response is less than 50 ms, which is the delay in obtaining a response without SPAN in the system. Note that the hit rate obtained by this



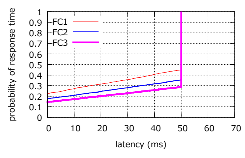
(a) 3 most probable requests for WebCT



(b) most probable request for WebCT

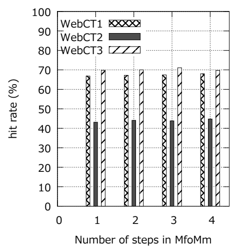


(c) 3 most probable requests for FC

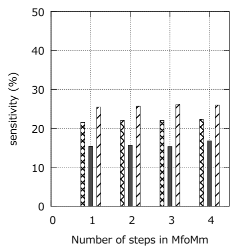


(d) most probable request for FC

Figure 14: CDF's showing the response times for different traces of WebCT and FC



(a) Hit rate obtained for different step sizes



(b) Precision obtained for different step sizes

Figure 15: Hit rate and precision obtained for different step sizes of Multiple first order Markov models (MfoMm) in WebCT

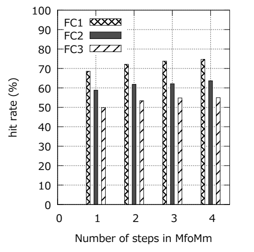
trace is 68%. Thus, a hit rate of 68% indicates that in approximately 68% cases, a subject would receive a response less than the time required for actual authorization process.

Comparing Figures 14(c) and 14(d), we find that the performance is improved when 3 most probable predicted requests are considered as compared to only one most probable request. However, the increase in the performance can negatively effect some of the requests, which would experience a queuing delay in the system. The queuing delay is depicted in the tail of the CDF in Figure 14(c). This tail cannot be seen in Figure 14(d) indicating lesser queuing delays. Finally, the hit rate obtained in FC1 is better than FC2. This is indicated by the curves in Figures 14(c) and 14(d) showing the reduction in latency obtained for FC1 and FC2.

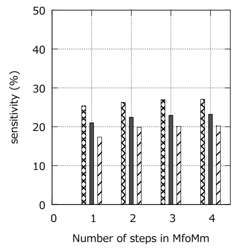
4.5.3 Hit rate, precision, and PDP computations for different step sizes of Multiple first order Markov models (MfoMm)

Figure 15 shows the hit rate and precision obtained for WebCT, using different step sizes in MfoMm. We observe that varying MfoMm does not provide much benefit in terms of hit rate and precision for this dataset. We obtained an improvement in hit rate of less than 1% for WebCT.

Figure 16 shows the hit rate and precision obtained for FC dataset. Changing the step size of MfoMm from 1 to 2, improves the hit rate by $4\% - 5\%$ for all the sub-traces of FC. Corresponding increase in the precision is between $1\% - 3\%$. In FC application, Facebook users receive help from their friends during the duration of any fight. Generally, the order in which the help is received cannot be restricted to particular sequences. Requests in access control systems, like accessing files in repositories, or requesting permissions on different directories in a domain, are examples where subjects might not make the requests in same sequence again and again. Looking into the past states using MfoMm would help in improving the hit rate and precision of such systems.

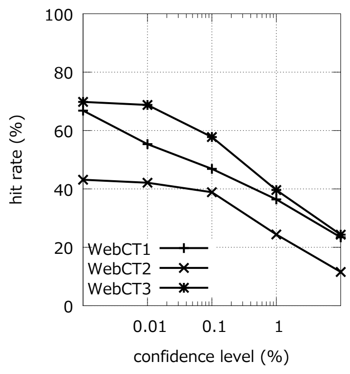


(a) Hit rate obtained for different step sizes

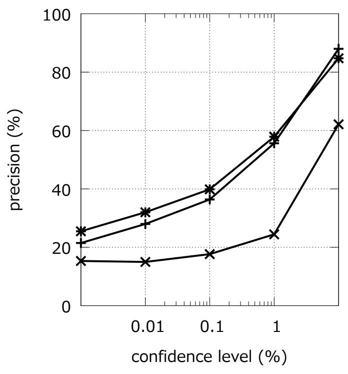


(b) Precision obtained for different step sizes

Figure 16: Hit rate and precision obtained for different step sizes of Multiple first order Markov models (MfoMm) in FC



(a) Change in hit rate as confidence level varies



(b) Change in precision as confidence level varies

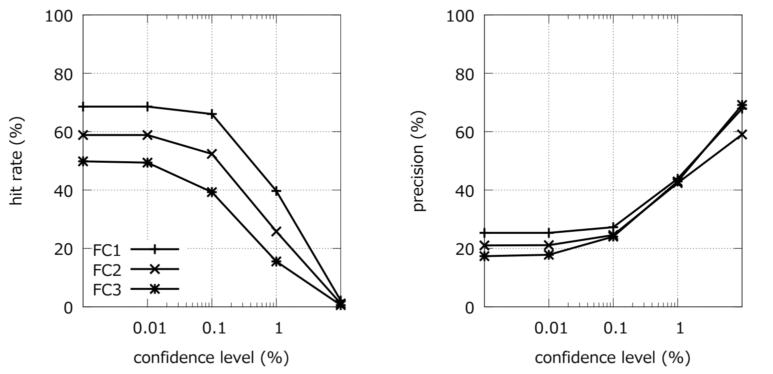
Figure 17: Change in hit rate and precision as the confidence level is varied for WebCT dataset

4.5.4 Hit rate, precision, and PDP computations for different confidence levels

As confidence level increases, the hit rate reduces, whereas precision increases. This expected behavior is observed for all the sub-traces of both datasets.

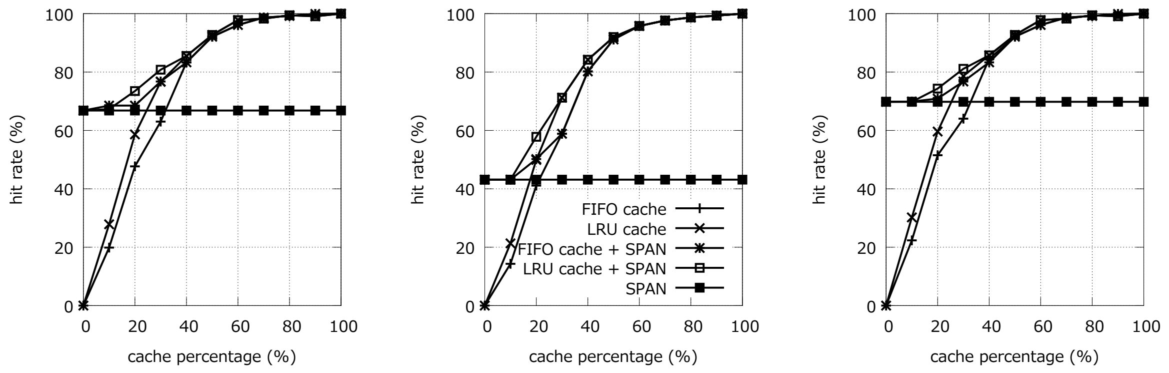
Figures 17(a) and 17(b) give details about the hit rate and precision achieved by the WebCT dataset as the confidence level is varied in the system. Hit rate and precision are not much affected for confidence level of 0.01%. However, as the level of confidence increases further, the hit rate drops and precision increases considerably. The hit rate achieved by SPAN for a confidence level of 10% is 23%, 11%, and 24% for the 3 sub-traces. The corresponding precision is 87%, 62% and 84%. To understand the effect of change in the confidence level on the number of requests to be computed by PDP, we studied the relative drop in the PDP computations for varying confidence levels. The relative drop in the number of computations is 91%, 93%, and 89%, respectively, when the confidence level reaches 10%. For the FC dataset, hit rate drops and precision increases linearly for SPAN beyond a confidence level of 1%, as seen in Figure 18.

Our experiments show that setting a higher confidence level decreases the hit rate, but the number of computations that the PDP has to perform drop considerably, reducing the additional load on the PDP. Our results demonstrate that confidence level could be a good metric for precomputing responses to the predicted requests. The confidence level could be set to higher values for systems where computing authorization responses are expensive.



(a) Change in hit rate as confidence level varies (b) Change in hit rate as confidence level varies

Figure 18: Change in hit rate and precision as the confidence level is varied for FC dataset



(a) Change in hit rate for WebCT1 dataset (b) Change in hit rate for WebCT2 dataset (c) Change in hit rate for WebCT3 dataset

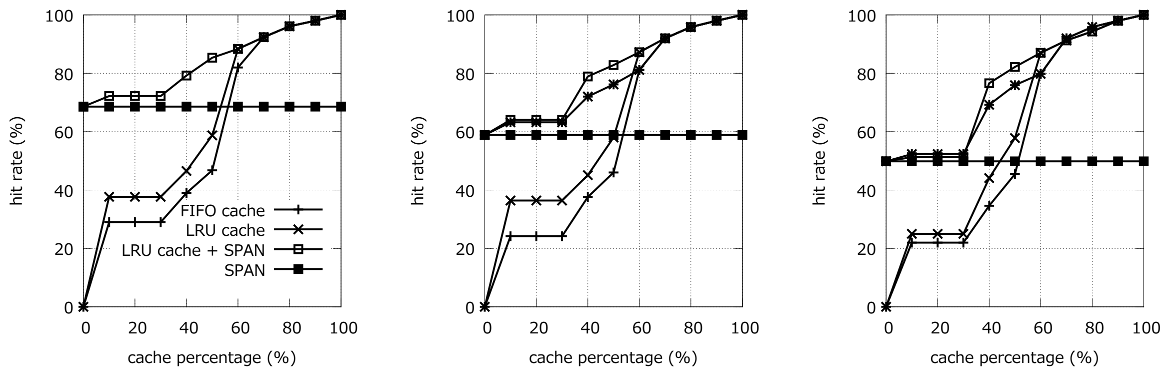
Figure 19: Change in hit rate for implementations of FIFO cache, LRU cache, and their combinations with SPAN for WebCT dataset

4.5.5 Hit rate for cache implementation

Figures 19 and 20 show the results obtained by combining caching techniques with SPAN as compared to stand alone caching for WebCT and FC datasets. From the results obtained, we find that the combination improves the overall hit rate of the system. For smaller sizes of cache, predictions made by SPAN can improve the hit rate of the system, whereas for larger sizes of cache, the permissions obtained from the cache improve the hit rate. Overall, we found that combining SPAN and LRU cache performs better than combining SPAN and FIFO cache.

Figure 19 shows the results obtained for WebCT. In this dataset, subjects repeatedly request for the same resources in their sessions. Thus, as the size of the cache grows, the evaluation engine finds more requests in the cache, resulting in improved cache performance.

For the FC dataset (Figure 20), we observe that FIFO and LRU caches obtain a maximum hit rate of less than 30% and 40% for cache sizes of up to 30%. In this dataset, subjects request for permissions mostly once per session, but repeat their behavior across different sessions. In such cases, SPAN provides better hit rate as it depends on the behavior of the subjects in the past sessions. However, as the size of the cache grows, the storage capacity of cache increases and information from the past sessions can be stored in the cache, improving the performance obtained by the caching techniques.



(a) Change in hit rate for FC1 dataset (b) Change in hit rate for FC2 dataset (c) Change in hit rate for FC3 dataset

Figure 20: Change in hit rate for implementations of FIFO cache, LRU cache, and their combinations with SPAN for FC dataset

5 Discussion

The results of our experiments indicate that SPAN leads to a high hit rate and precision in predicting requests for access control systems. As our approach is comparable to the prediction techniques proposed for web predictions, we followed the same procedure of building algorithms and measuring the hit rate. In addition, we also calculated the precision and additional load that could be encountered by PDP. In this section, we discuss our understanding on the implementation of SPAN in access control systems, based on the results obtained.

While predicting requests improves the performance of systems, it also causes the PDP to compute additional responses. If subjects do not make the same requests as predicted, responses computed for the predicted requests are a waste of PDP's computational power. A good prediction algorithm should achieve high hit rate and impose minimum additional load on the PDP. While a high hit rate indicates that the system is capable of reducing the latency of access control systems, a high precision indicates that fewer responses computed by the PDP are unused in the system. SPAN achieved high hit rate and precision, as compared to the other algorithms we implemented. We found that the algorithms discussed in our related work do not consider precision for evaluating their approaches, but precision is an important metric for determining the additional load on the PDP.

For all the algorithms we implemented, SPAN achieved better improvement in hit rate and precision for the FC dataset as compared to WebCT. WebCT mostly contained accesses made by instructors, teaching assistants, and students in a course, and the popular sequences of requests were made for accessing course materials, to which everyone had access. The popular behavior of individual subjects was not significantly different from the overall behavior exhibited by all the subjects. Our results for this dataset shows that SPAN does not achieve significant improvement in performance over Cadez. SPAN and Cadez, both follow clustering approach in the training phase. The only difference is that SPAN considers the behavior of individual subjects while forming clusters, whereas Cadez does not. On the other hand, in our FC dataset, Facebook users could receive help only from their friends. This dataset represented a system where subjects could request for only those permissions that they were authorized. In this case, the behavior of individual subjects is different from overall behavior of subjects. Clusters formed by considering individual subjects access patterns, result in better predictions. To summarize, SPAN performs better for enterprises, where the access patterns of individual subjects are different from the overall access patterns of all the subjects in the system.

The access patterns found in the training and testing sets influence the hit rate and precision. For the WebCT dataset, WebCT3 gave better hit rate and precision, as compared to WebCT2.

WebCT2 was trained on the sequences of requests made mostly by instructors, and tested on requests made by the instructors, teaching assistants, and students. The actions of instructors in the training set added course materials and students could not perform these actions. The testing set contained requests made by everyone for accessing course materials. The access pattern in the training set was different from the pattern found in the testing set. On the contrary, the training and testing sets of WebCT3 contained requests by students for accessing course materials. This resulted in similar access patterns in the training and testing set. We conclude that having similar access patterns in the training and testing set results in higher hit rate and precision.

For every sequence, SPAN and other prediction algorithms predict possible future requests with certain probability. Computing responses to all the predicted requests can increase the load on the PDP. To reduce these computations, we implemented an evaluation engine and set input parameters to the engine. We found that the hit rate dropped by 50% when only the most probable request was considered, as compared to 3 most probable requests. However, the precision increased by $50\% - 100\%$. From our results, we also observe an interesting fact. The number of most probable requests affected the performance of the algorithms. We found that first order Markov models performed better than second order Markov models, when 3 most probable requests were considered. However, the performance reversed when only the most probable request was considered. For the FC dataset, we observe that the hit rate and precision obtained by Cadez dropped considerably, when only 1 response per sequence was considered, as compared to 3 initially. To summarize, the number of probable requests considered for satisfying the subjects had a direct impact on the hit rate and precision achieved by all the algorithms.

SPAN predicts requests with certain probability. A higher probability for a request indicates that it is likely to be requested in the future, as compared to the request predicted with lower probability. Responses can be fetched only to those requests that are predicted with certain probability. This would be a good criteria for reducing the additional load encountered by the PDP. As observed from our experiments, neglecting requests predicted with lower probability (confidence levels) reduced the additional load on the PDP considerably. Enterprises, where computing authorization responses are not expensive, more responses could be fetched. This would result in high hit rate, but the precision would be low. On the other hand, if computations are expensive, fewer responses could be fetched based on our proposed confidence level approach, resulting in reduced additional load on the PDP.

Caching has been a popular technique for improving the performance of systems. Policies are cached even in commercial access control products like Tivoli Access Manager [IBM08]. We have not seen speculative authorizations being used to improve the performance of systems. Results obtained from implementing SPAN and caching in the same system demonstrated that this configuration can boost the hit rate. For WebCT, increasing the size of the cache reduced the difference in hit rate between stand alone caching, and combined SPAN and caching implementation. WebCT represented a dataset, where subjects requested access on same resources repeatedly in a session. In systems like these, speculation authorizations obtains higher hit rate when the size of the cache is small. As the size of the cache grows, more responses are stored in the cache. If subjects make the same requests repeatedly, responses to these requests are found in the cache. Hence, the difference in hit rate is reduced. On the contrary, in our FC dataset, Facebook users generally received help from their friends, only once per session. In these systems, caching cannot improve the performance to a large extent. SPAN can improve performance of such systems, where caching is of little use and predictions are made on the behavior exhibited by the subjects in their past sessions.

Results obtained for Multiple first order Markov models demonstrated an improvement in hit rate, when applied to FC dataset as compared to WebCT. In the FC dataset, Facebook users received help from their friends in certain order, but this order was not fixed each time. MfoMm finds the probability of two requests being requested in a session, not necessarily one after the other. Thus, it improves the performance in systems like FC or file systems, where subjects are not restricted to ordering of their requests on permissions.

Our experiments indicate that the training time required for forming clusters is proportional

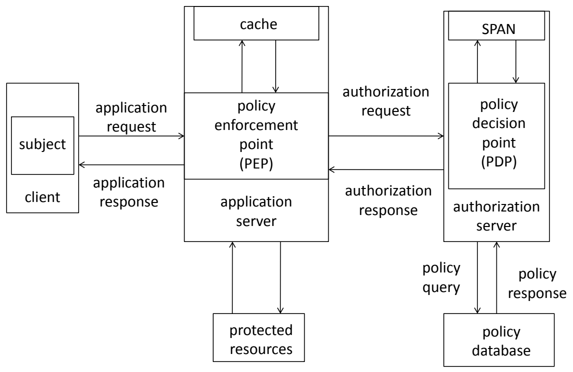


Figure 21: Architecture of SPAN's implementation on PDP side

to the number of subjects and unique sequences of requests formed in the system. However, the time required to make a prediction is quite low for all the datasets. In fact, it is independent of the number of subjects and unique sequences formed. It depends on the number of clusters formed in the system. The time required to make a prediction for WebCT is lesser than FC, where the number of clusters are 4 and 7 respectively.

So far, we interpreted the results of our experiments that demonstrates the effectiveness of SPAN in access control systems. Another aspect to prediction is to accommodate the change in behavior of subjects or access control policies that affect the access patterns exhibited by the subjects of the system. We discuss this aspect in the next section.

5.1 Batch and online algorithms

As noted in Table 4, we found that the time required to form clusters is directly proportional to the number of subjects, sequences of permissions, and the iterations required to form the clusters. To accommodate changes in behavior of subjects in systems, training phase has to be repeated. However, time required to form clusters is an expensive step for SPAN. Limiting the number of iterations would limit time required to run the training phase repeatedly for accommodating the changes. For this purpose, whenever a training phase was repeated, we started the clusters assignments from the one obtained in the previous training phase of the dataset. Using this process, WebCT2 dataset converged in 3 iterations, whereas the WebCT3 dataset converged in just a single iteration for both sets of requests. This experiment demonstrates that SPAN can accommodate changes in the behavior of the subjects by simply running the training phase of the algorithms again. This step is less time consuming compared to the batch training phase performed at the start of every experiment that required between 15 and 25 iterations to converge.

In the next section, we provide the possible configurations of implementing SPAN in access control systems.

5.2 Implementing SPAN in access control systems

SPAN could be completely collocated with the PDP side as shown in Figure 21, or split between the PEP and PDP, as shown in Figure 22.

5.2.1 SPAN collocated with the PDP

SPAN can be collocated with the PDP as shown in Figure 21. In this configuration, both the training and prediction phases of SPAN are implemented at the PDP-side. In the training

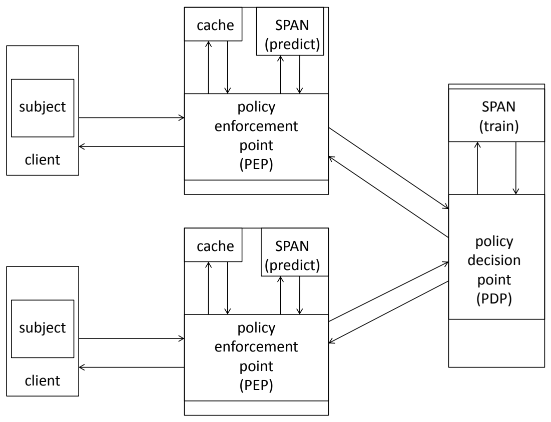

cache
SPAN (predict)
subject
client
policy enforcement point (PEP)
SPAN (train)
policy decision point (PDP)
cache
SPAN (predict)
subject
client
policy enforcement point (PEP)

Figure 22: Architecture of SPAN's implementation split between PEP and PDP

phase, the PDP not only computes the responses to the incoming requests from the PEP, but also sends a copy of the requests to SPAN. After a certain number of requests are recorded, SPAN runs the training phase of the algorithm.

In the testing phase, when the PEP sends requests to the PDP, a copy of requests are sent to SPAN. Based on the sequences of requests received, SPAN predicts the requests that could be made by the subjects. The PDP computes the responses to these predicted requests and sends it to the cache, collocated with the PEP. This mechanism reduces the latency to virtually zero. However, there is a disadvantage. When PEP finds responses in its cache, it would not make authorization requests to the PDP. This would break the sequence of authorization requests flowing from PEP to the PDP. As SPAN depends on the sequences of requests to make predictions, its predictive capability would be affected. To avoid this, the PEP has to inform the PDP about the requests made by the subjects even if responses are found in the cache. In a single PEP-PDP configuration, SPAN could be collocated with the PEP to avoid this drawback.

However, the PEP-PDP configuration is generally designed for a single PDP to support multiple PEP's. We provide the implementation of SPAN in this scenario next.

5.2.2 SPAN split between PEP and PDP

In this configuration (Figure 22), the training and testing phases of SPAN are split between the PDP and the PEP, respectively. The training phase is implemented at the PDP, whereas the testing phase is implemented at the PEP. Implementing the training phase at the PDP has two advantages:

1. The training phase of SPAN depends on higher frequency counts to form clusters. Aggregating the authorization requests from all the PEP's would boost the frequency counts.
2. The training phase is intensive in terms of time. Clusters could be formed once at the PDP and transmitted to the PEP's.

For the testing phase, SPAN could be collocated with the PEP's. When SPAN predicts future requests, the PEP could check if the cache contains the response before sending the authorization request to the PDP. This configuration also avoids the problem of breaking the sequence of requests for SPAN.

5.3 Shortcomings in SPAN

Although SPAN achieves good hit rate and precision, there are certain shortcomings that we address in this section.

First, SPAN requires time to adjust to any policy changes made in enterprise authorization systems. SPAN is built on the frequency counts obtained from transitions made by subjects for accessing resources. If a subject gets access to a new resource, or is denied access to a resource, previously allowed, it would take some time for SPAN to detect the changes. Policy changes would change the behavior exhibited by subjects in an enterprise. This change will not be captured immediately by SPAN, as it relies on comparable frequency counts of transitions to make predictions. SPAN would predict incorrect requests for the subjects. However, this does not cause any security threat to the enterprise systems. SPAN only predicts the requests, but decisions are still made by the PDP. Policies changes will affect the hit rate and precision obtained by SPAN. This is the fundamental problem for all approaches [AKT08, DK04, EVK05, CHM$^+$03, SH05, SYLZ00, BBB09, MDLN01, YHN03, YZ01, YLW04] built on frequency counts of transitions. Accommodating policy changes to make predictions is an open problem for speculative authorizations.

The second shortcoming of SPAN is the time required to train SPAN is directly proportional to the number of subjects and unique sequences of requests formed in the system. For our second dataset, we found that increasing the number of subjects increased the number of permissions and requests made by the subjects. It had a direct impact on the training time that increased from 12 minutes for 50 subjects, to 47 minutes for 200 subjects. In the current implementation, this shortcoming affects the scalability of SPAN.

5.4 Summary

We now summarize the section to list our assumptions in SPAN, its applicability and the tuning of input parameters to obtain better performance.

1. SPAN is designed for access control systems where information about subject's identity and sequence of requests made by the subjects can be obtained.
2. From the results obtained for our two datasets, we conclude that SPAN achieves better performance where access pattern of individual subject is different from overall access patterns of all the subjects in the system.
3. The number of most probable responses to be computed depends on the time required for the PDP to compute every response. If the time required to compute a response is comparatively higher than the average time of incoming requests from the PEP, only the most probable response should be computed. If not, the requests made by the PEP would experience a queuing delay at the PDP. The PDP should be configured in such a way that computing responses to requests made by the subjects are prioritized as compared to requests predicted by SPAN.
4. From the results obtained for our WebCT2 and WebCT3 traces, we found that the access patterns found in the training phase and actual prediction phase determines when the training phase should be rerun. If the number of permissions in the system does not undergo frequent changes and the behavior of subjects on those permissions is fairly constant over a period of time, the training phase of the algorithm need not be run over and over again.
5. Finally, we conclude that SPAN can be used to improve the performance of enterprise authorization systems consisting of a few hundred subjects. Scaling SPAN to accommodate larger enterprises is a direction of future work.

6 Conclusion

In this paper, we presented *Speculative Authorization* (SPAN) that predicts authorization requests, likely to be made by subjects in access control systems. Unlike web page prediction algorithms, our model considers the authorization policies for making predictions. We implemented two web prediction techniques, and evaluated the hit rate and precision obtained by those against SPAN. For the two datasets used for our experiments, SPAN achieved a hit rate between $30-70\%$, a gain of $2-55\%$ as compared to the other algorithms. Precision varied from $15-45\%$. We proposed confidence level metric for computing responses to the predicted requests. For systems where computing policies decisions are expensive, our strategy would help in reducing the additional load on the PDP, while improving performance. We also implemented caching and SPAN in the same systems. Our results demonstrate that combining SPAN and caching can further improve the performance of access control systems as compared to stand alone caching technique.

6.1 Future work

Although our approach improves the performance of enterprise authorization systems, in terms of reducing in latency, there is an area for future research.

Recall that the time required for the training phase is proportional to the number of subjects in the system. Time required for training can affect the scalability of the systems. A *role-based access control* policy (RBAC) controls access based on the roles a subject is assigned and the permissions that are allowed for those roles. Having been introduced more than a decade ago, RBAC [FK92, SCFY96] has been deployed in many organizations for access control enforcement, and eventually matured into the ANSI RBAC standard [ANS04]. In RBAC, instead of directly assigning permissions to subjects, the subjects are assigned to roles and the roles are mapped to permissions. Subjects are assigned appropriate roles according to their job functions in an enterprise. Generally, the number of roles in an enterprise are much lesser than the number of subjects in an enterprise. Considering the roles instead of identity of subjects, can improve the time required in the training phase. However, a subject can possess multiple roles. The clustering algorithms built for prediction using role based access control should consider this criteria.

7 Acknowledgements

Members of the Laboratory for Education and Research in Secure Systems Engineering (LERSSE) gave valuable feedback on this paper. A special thanks to Nando de Freitas for his inputs during the initial phase of this project.

References

[AH02] L.A. Adamic and B.A. Huberman. Zipf's law and the Internet. *Glottometrics*, 3(1):143–50, 2002.

[AKT08] Mamoun Awad, Latifur Khan, and Bhavani Thuraisingham. Predicting WWW surfing using multiple evidence combination. *The VLDB Journal The International Journal on Very Large Data Bases*, 13:401–417, 2008.

[ANS04] ANSI. ANSI INCITS 359-2004 for role based access control, 2004.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules. In *Proceedings of the 20th International Conference on Very Large Data Bases, (VLDB'94)*, pages 487–499, Santiago, Chile, September 12-15 1994.

[BBB09] Geoffray Bonnin, Armelle Brun, and Anne Boyer. A low-order markov model integrating long-distance histories for collaborative recommender systems. In *IUI '09: Proceedings of the 13th international conference on Intelligent user interfaces*, pages 57–66, New York, NY, USA, January 13-16 2009. ACM.

[BC00] Kristin Bennett and Colin Campbell. Support vector machines: hype or hallelujah? *ACM SIGKDD Explorations Newsletter*, 2:1–13, 2000.

[BCF$^+$99] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of the Conference on Computer Communications (INFOCOM'99)*, pages 126–134, New York, NY, USA, 1999. IEEE Computer Society.

[BEJ09] Josh Bregman, Brian Eidelman, and Chris Johnson. Oracle fusion middleware security. http://fusionsecurity.blogspot.com/2009/10/impact-of-oracle-entitlement-server-oes.html, 2009.

[BGR07] Lujo Bauer, Scott Garriss, and Michael Reiter. Efficient proving for practical distributed access-control systems. In *Computer Security—ESORICS 2007: 12th European Symposium on Research in Computer Security*, volume 4734 of *Lecture Notes in Computer Science*, pages 19–37, 2007.

[BL73] D. E. Bell and L. J. LaPadula. Secure computer systems: Mathematical foundations. Technical Report ESD-TR-74-244, MITRE, March 1973.

[BL75] D. E. Bell and L. J. LaPadula. Secure computer systems: Unified exposition and multics interpretation. Technical Report ESD-TR-75-306, MITRE, March 1975.

[BNJ03] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Topic modeling. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[BSF02] Lujo Bauer, Michael A. Schneider, and Edward W. Felten. A general and flexible access-control system for the web. In *Proceedings of the 11th USENIX Security Symposium*, pages 93–108, Berkeley, CA, USA, August 5-9 2002. USENIX Association.

[BZP05] Kevin Borders, Xin Zhao, and Atul Prakash. CPOL: high-performance policy evaluation. In *Proceedings of the 12th ACM conference on Computer and Communications Security (CCS'05)*, pages 147–157, New York, NY, USA, November 7-11 2005. ACM Press.

[CHM$^+$03] Igor Cadez, David Heckerman, Christopher Meek, Padhraic Smyth, and Steven White. Model-based clustering and visualization of navigation patterns on a web site. *Data Mining Knowledge Discovery*, 7(4):399–424, 2003.

[CLB06] Jason Crampton, Wing Leung, and Konstantin 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)*, pages 111–120, Lake Tahoe, CA, USA, June 7–9 2006. ACM Press.

[Dav04] Brian D. Davison. *Learning web request patterns*, pages 435–460. Springer, 2004.

[DK04] M. Deshpande and G. Karypis. Selective Markov models for predicting web page accesses. *ACM Transactions on Internet Technology*, 4(2):163–184, 2004.

[EVK05] Magdalini Eirinaki, Michalis Vazirgiannis, and Dimitris Kapogiannis. Web path recommendations based on page ranking and markov models. In *WIDM '05: Proceedings of the 7th annual ACM international workshop on Web information and data management*, pages 2–9, New York, NY, USA, November 04 2005. ACM.

[FK92] D. Ferraiolo and R. Kuhn. Role-based access controls. In *Proceedings of the 15th NIST-NCSC National Computer Security Conference*, pages 554–563, Baltimore, MD, USA, 1992. National Institute of Standards and Technology/National Computer Security Center.

[Hec95] David Heckerman. A tutorial on learning with Bayesian Networks. Technical report, Microsoft Research, 1995.

[IBM08] IBM. Tivoli access manager, version 6.0. http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp topic=/com.ibm.itame.doc_6.0/am60_admin16.htm, 2008.

[Kar03] G. Karjoth. Access control with IBM Tivoli Access Manager. *ACM Transactions on Information and Systems Security*, 6(2):232–57, 2003.

[KHS07] Ron Kohavi, Randal M. Henne, and Dan 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*, pages 959–967, New York, NY, USA, 2007. ACM.

[KLVW04] Alexander Klemm, Christoph Lindemann, Mary K. Vernon, and Oliver P. Waldhorst. Characterizing the query behavior in peer-to-peer file sharing systems. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 55–67, New York, NY, USA, 2004. ACM.

[KLW05] Zbigniew Kalbarczyk, Ravishankar K. Lyer, and Long Wang. Application fault tolerance with Armor middleware. *IEEE Internet Computing*, 9(2):28–38, 2005.

[KS08] Mathias Kohler and Andreas Schaad. Proactive access control for business process-driven environments. In *ACSAC '08: Proceedings of the 2008 Annual Computer Security Applications Conference*, pages 153–162, Washington, DC, USA, December 812 2008. IEEE Computer Society.

[MDLN01] Bamshad Mobasher, Honghua Dai, Tao Luo, and Miki Nakagawa. Effective personalization based on association rule discovery from web usage data. In *WIDM '01: Proceedings of the 3rd international workshop on Web information and data management*, pages 9–15, New York, NY, USA, November 9 2001. ACM.

[Nie93] Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[NRC08] Atif Nazir, Saqib Raza, and Chen-Nee Chuah. Unveiling facebook: a measurement study of social network based applications. In *IMC '08: Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, pages 43–56, New York, NY, USA, 2008. ACM.

[Ora08] Oracle. Oracle entitlements server: Programming security for web services. Technical report, Oracle, September 2008.

[PMB77] Dempster A. P., Laird N. M., and Rubin D. B. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[PP99] James Pitkow and Peter 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*, pages 13–13, Berkeley, CA, USA, October 11-14 1999. USENIX Association.

[SCFY96] Ravi Sandhu, Edward Coyne, Hal Feinstein, and Charles Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[SH05] R. Sen and M. Hansen. Predicting web users' next access based on log data. *Journal of Computational and Graphical Statistics*, 12(1):1–13, 2005.

[SSL$^+$99] R. Spencer, S. Smalley, P. Loscocco, M. Hibler, D. Andersen, and J. Lepreau. The Flask security architecture: System support for diverse security policies. In *Proceedings of the 8th USENIX Security Symposium*, pages 123–140, Washington, D.C., August 23-26 1999. USENIX Association.

[SYLZ00] Zhong Su, Qiang Yang, Ye Lu, and Hongjiang Zhang. Whatnext: a prediction system for web requests using n-gram sequence models. In *Proc. of the First International Conference on Web Information Systems Engineering*, pages 214–221, June 19 - 20 2000.

[Vog04] Werner Vogels. How wrong can you be? Getting lost on the road to massive scalability. In *the 5th International Middleware Conference*, Toronto, Canada, October 20 2004. ACM Press. Keynote address.

[WCBR08] Qiang Wei, Jason Crampton, Konstantin Beznosov, and Matei Ripeanu. Authorization recycling in RBAC systems. In *Proceedings of the thirteenth ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 63–72, Estes Park, Colorado, USA, June 11–13 2008. ACM.

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

[Wei09] Qiang Wei. *Towards Improving the Availability and Performance of Enterprise Authorization Systems.* PhD thesis, Electrical and Computer Engineering,The University of British Columbia, Vancouver, 2009.

[WRB07] Qiang Wei, Matei Ripeanu, and Konstantin Beznosov. Cooperative secondary authorization recycling. In *Proceedings of the 16th ACM/IEEE International Symposium on High-Performance Distributed Computing (HPDC)*, pages 65–74, Monterey Bay, CA, June 27-29 2007. ACM Press.

[YHN03] Qiang Yang, Joshua Zhexue Huang, and Michael Ng. A data cube model for prediction-based web prefetching. *Journal of Intelligent Information Systems*, 20(1):11–30, 2003.

[YLW04] Qiang Yang, Tianyi Li, and Ke Wang. Building association-rule based sequential classifiers for web-document prediction. *Data Mining Knowledge Discovery*, 8(3):253–273, 2004.

[YZ01] Qiang Yang and Henry Hanning Zhang. Integrating web prefetching and caching using prediction models. *Journal: World Wide Web*, 4(4):299–321, 2001.