User Session Modeling for Effective Application Intrusion Detection

Kapil Kumar Gupta, Baikunth Nath (Sr. MIEEE) and Kotagiri Ramamohanarao

Abstract With the number of data breaches on a rise, effective and efficient detection of anomalous activities in applications which manages data is critical. In this paper, we introduce a novel approach to improve attack detection at application layer by modeling user sessions as a sequence of events instead of analyzing every single event in isolation. We also argue that combining application access logs and the corresponding data access logs to generate unified logs eliminates the need to analyze them separately thereby resulting in an efficient and accurate system. We evaluate various methods such as conditional random fields, support vector machines, decision trees and naive Bayes, and experimental results show that our approach based on conditional random fields is feasible and can detect attacks at an early stage even when they are disguised within normal events.

1 Introduction

Detecting intrusions is a challenge because it is important to detect malicious events at an early stage in order to minimize their impact. This becomes more important when attackers come up with previously unseen attacks even when the present systems are unable to detect all existing attacks with acceptable reliability [13]. Further, with more and more data becoming available in digital format and more applications being developed to access this data, the data and applications are a victim of malicious attackers who exploit the applications to gain access to sensitive data. Thus, there is need to develop robust and efficient intrusion detection systems which can detect such malicious activities at application layer.

Kapil Kumar Gupta, Baikunth Nath, Kotagiri Ramamohanarao
Department of Computer Science & Software Engineering, NICTA Victoria Research Laboratory, The University of Melbourne, Australia, 3010. e-mail: kgupta@csse.unimelb.edu.au, bnath@csse.unimelb.edu.au, rao@csse.unimelb.edu.au

Please use the following format when citing this chapter:

Gupta, K.K., Nath, B. and Ramamohanarao, K., 2008, in IFIP International Federation for Information Processing, Volume 278; *Proceedings of the IFIP TC 11 23rd International Information Security Conference*; Sushil Jajodia, Pierangela Samarati, Stelvio Cimato; (Boston: Springer), pp. 269–283.

Intrusion detection systems are classified as signature based, anomaly based or hybrid systems [5]. Hybrid systems generally employ machine learning methods while signature and anomaly based systems are often based on pattern matching and statistical methods. The advantage of hybrid systems is that they are trained using normal and anomalous data patterns together and hence can be used to label new unseen events reliably when compared with signature and anomaly based systems which are generally based on a threshold [21]. Intrusion detection systems can also be classified as network based, host based or application based [5].

In this paper, we propose an application intrusion detection system which models individual user sessions using a moving window of events. One of the main drawbacks of present application intrusion detection systems is that they are specific to a particular application and cannot be generalized [19], [20]. However, our proposed model is general and does not require application specific details to be encoded. It only needs to be trained with the logs associated with a particular application. As any application intrusion detection system, our system is meant to provide an additional line of defense and not to replace existing network based systems.

The rest of the paper is organized as follows; we explain our framework in Sect. 2 and discuss the data set in Sect. 3. We give our experimental results in Sect. 4. We then discuss related work in Sect. 5 and draw conclusions in Sect. 6.

2 Proposed Model

In general, there are two motives to launch an attack; either to force a network to stop some service that it is providing or to steal some information stored in a network. In this paper, we focus on the second motive, i.e., to detect malicious data access. However, what is normal and what is anomalous is not defined, i.e., an event may be normal when measured with respect to some criteria but the same may be called as anomalous when this criteria is changed. Thus, the objective is to find anomalous test patterns which are similar to the anomalous patterns which occurred during the training with the assumption that the underlying measuring criteria is unchanged and the system is trained such that it can reliably separate normal and anomalous events. The straight forward approach is to audit every data access request before it is processed and data is retrieved by the system. However, this is not the ideal solution due to the following reasons:

1. The number of data requests per unit time is very large and monitoring every request in real time applications severely affects system performance.
2. Assuming that we can somehow monitor every data request, the system must be regularly updated with new signatures to detect previously known attacks (it still cannot detect zero day attacks).
3. The system is application specific because the signatures are defined by encoding application specific knowledge.

Thus, monitoring every data request is often not feasible in real life environment. We also observe that real world applications generally follow the three tier architecture [1] which ensures application and data independence, i.e., data is managed separately and is not encoded into the application. Hence, to access data, an attacker has no option but to exploit this application. To detect such attacks, an intrusion detection system can either monitor the application requests or (and) monitor the data requests. As we discussed above, analyzing every data access is difficult and limits the detection capability of the intrusion detection system. Similarly, analyzing only the application requests does not provide useful information about the data accessed. Previous systems such as [6], [9] and [15] consider the application requests and the corresponding data requests separately and, hence, unable to correlate the events together resulting in a large number of false alarms. Before we explain our framework, we define some key terms which will be helpful in better understanding of the paper.

1. **Application:** An *application* is a software by which a user can accesses data. There exists no other way in which the data can be made available to a user.
2. **User:** A *user* is either an individual or any another application which access data.
3. **Event:** Data transfer between a user and an application is a result of multiple sequential events. Data transfer can be considered as a request-response system where a request for data access is followed by a response. An *event* is a single request-response pair. We represent a single event as an N feature vector. In this paper, we use the term *event* interchangeably with the term *request*.
4. **User Session:** A *user session* is an ordered set of events or actions performed, i.e., a session is a sequence of one or more request-response pairs. Every session can be uniquely identified by a session-id.

2.1 Framework

We represent a general framework for building application intrusion detection systems in Fig. 1. Our framework does not encode application specific knowledge making it useable for a variety of applications. To access data, a user accesses the application as in a simple three tier architecture. However, every request first passes through the session control. Session control is responsible for establishing new sessions and for checking the session-id for previously established sessions. For this, it maintains a list of all the valid sessions that are allowed to access the application and hence the data. Every request to access the application is checked for a valid session-id at the session control which can be blocked if it is found anomalous depending upon the installed security policy. The session control can be implemented as part of the application itself or as a separate entity.

Following checks from the session control, the request is sent to the application where it is processed. The web server logs every request. Similarly every data access is logged. The two logs are then combined to generate unified logs which are analyzed by the intrusion detection system as represented in the framework.

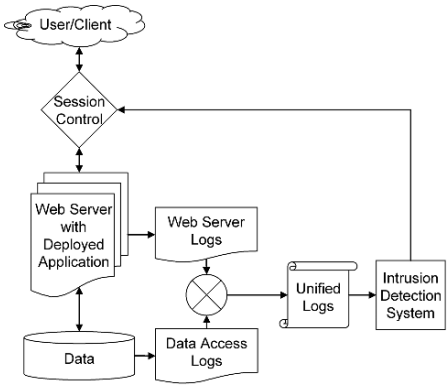

Fig. 1 Framework for building Application Intrusion Detection System

We represent the structure of a typical user session in Fig. 2. A user requests a resource which generates a web request. As we shall discuss later, we used a PHP application to generate data. We consider a web request to be a single request to render a PHP page by the web server and not a single HTTP GET request as it may contain multiple images, frames and dynamic content. The PHP page can be easily identified from the web server logs. This request further generates one or more data requests which depend on the logic encoded in the application. To capture user-application and application-data interactions, we utilize features of both the web server logs and the associated data access logs to generate unified logs. However, the number of data requests is extremely large as compared to the number of web requests. Hence, we first process the data access logs to generate simple statistics such as the number of queries invoked by a single web request and the time taken to process them rather than analyzing every data access individually. We then use the session-id which is present in both the web server logs and the associated data access logs to uniquely map the extracted statistics (obtained from the data access logs) to the corresponding web requests to generate unified logs.

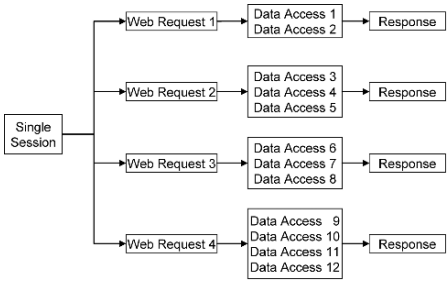

Fig. 2 Representation of a Single user Session

Thus, we generate a unified log format where every session is represented as a sequence of vectors and is represented by the following 6 features:

1. Number of data queries generated in a single web request.

2. Time taken to process the request.
3. Response generated for the request.
4. Amount of data transferred (in bytes).
5. Request made (or the function invoked) by the client.
6. Reference to the previous request in the same session.

Web access logs contain useful information such as the details of every request made by a client (user), response of the web server, amount of data transferred etc. Similarly, data access logs contain important details such as the exact data table and columns accessed, in case the data is stored in a database. Performing intrusion detection at the data access level, in isolation, requires substantially more resources when compared to our approach. Monitoring the two logs together eliminates the need to monitor every data query since we can use simple statistics. In order to gain data access an attacker follows a number of steps and hence, to reduce the number of false alarms and increase the attack detection accuracy, intrusion detection systems must be capable of analyzing entire sequence of events rather than considering every event in isolation [24]. To model such a sequence of event vectors, we need a method that does not assume independence among sequential events. Thus, we use conditional random field which we describe next.

2.2 Conditional Random Fields

Conditional random fields [18] offer us the required framework to build robust intrusion detection systems [11], [12]. The prime advantage of conditional random fields is that they are discriminative models which directly model the conditional distribution $p(y|x)$. Further, conditional random fields are undirected models and free from label bias and observation bias which are present in other conditional models [16]. Generative models such as the Markov chains, hidden Markov models, naive Bayes and joint distribution have two disadvantages. First, the joint distribution is not required since the observations are completely visible and the interest is in finding the correct class which is the conditional distribution $p(y|x)$. Second, inferring conditional probability $p(y|x)$ from the joint distribution, using the Bayes rule, requires marginal distribution $p(x)$ which is difficult to estimate as the amount of training data is limited and the observation x contains highly dependent features. As a result strong independence assumptions are made to reduce complexity. This results in reduced accuracy [22] and hence these methods are not considered in this paper. Instead, conditional random fields predict the label sequence y given the observation sequence x, allowing them to model arbitrary relationships among different features in the observations without making independence assumptions. The graphical structure of a conditional random field is represented in Fig. 3.

The following mathematical description of a conditional random field is motivated from [18]. Given X and Y, the random variables over data sequence to be labeled and the corresponding label sequences, let $G = (V, E)$ be a graph with vertices

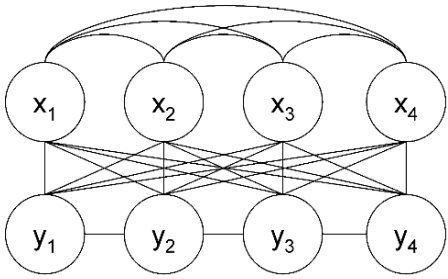

Fig. 3 Graphical Representation of a Conditional Random Field. x_1, x_2, x_3, x_4 represents an observed sequence of length four and every event in the sequence is correspondingly labeled as y_1, y_2, y_3, y_4. Further, every x_i is a feature vector of length '6'.

V and edges E such that $Y = (Y_v)$ where $v \in V$ and Y is represented by the vertices of the graph G, then, (X,Y) is a conditional random field, when conditioned on X, the random variables Y_v obey the Markov property with respect to the graph: $p(Y_v|X,Y_w,w \neq v) = p(Y_v|X,Y_w,w \sim v)$, where $w \sim v$ means that w and v are neighbors in G, i.e., a conditional random field is a random field globally conditioned on X. For a simple sequence (or chain) modeling, as in our case, the joint distribution over the label sequence Y given X has the form:

$$p_\theta(y|x) \propto \exp\Big(\sum_{e\in E,k} \lambda_k f_k(e,y|_e,x) + \sum_{v\in V,k} \mu_k g_k(v,y|_v,x)\Big) \tag{1}$$

where x is the data sequence, y is a label sequence, and $y|_s$ is the set of components of y associated with the vertices or edges in subgraph S. Also, the features f_k and g_k are assumed to be given and fixed. The parameter estimation problem is to find the parameters $\theta = (\lambda_1, \lambda_2, ...; \mu_1, \mu_2, ...)$ from the training data $D = (x^i, y^i)_{i=1}^N$ with the empirical distribution $\tilde{p}(x,y)$. Recently the conditional random fields have been shown to work very well for intrusion detection [11]. The reason for this is that they make no unwarranted assumptions about the data, and once trained they are very efficient and robust. During testing, the Viterbi algorithm is employed which has a complexity of $O(TL^2)$, where T is the length of the sequence and L is the number of labels. The quadratic complexity is problematic when the number of labels is large, such as in the language tasks, but for intrusion detection we have a limited number of labels (normal and anomalous) and thus the system is efficient.

3 Data Description

To perform our experiments we collected data locally by setting up an environment that mimics a real world application environment. We used an open source, online shopping application [2] and deployed it on a web server running Apache version 2.0.55 and connected to a database server running MySQL version 4.1.22. Every access to the web server and the data server was logged. We collected both the normal and the attack data. The data set is made freely available and can be downloaded from [10].

To collect the normal data we asked the students in the department to access the application. The system for data collection was online for five consecutive days. From the data we observed that about 35 different users accessed the application which resulted in 117 unique sessions composed of 2,615 web requests and 232,655 database requests. We then combined the web server logs with the data server logs to generate the unified logs in the format discussed in Sect. 2.1. Hence we have 117 sessions with only 2,615 events vectors which include features of both the web requests and the associated data requests. We also observed that a large number of user sessions were terminated without actual purchase resulting in abandoning the shopping cart. This is a realistic scenario and in reality a large number of the shopping carts are abandoned without purchase. A typical normal session in the data set is represented in Fig. 4.

```
0,0,301,369,GET /catalog HTTP/1.1,-,normal
0,0,200,28885,GET /catalog/ HTTP/1.1,-,normal
131,1,200,28480,GET /catalog/index.php,http://dummydata.xyz/catalog/,normal
108,1,200,27121,GET /catalog/product_info.php,http://dummydata.xyz/catalog/index.php,normal
```

Fig. 4 Representation of a Normal Session

To collect attack data we disabled access to the system by any other user and generated attack traffic manually based upon two criteria; first, the attacks which do not require any control over the web server or the database such as SQL injection and, second, the attacks which require some control over the web server such as website defacement and others. The events were logged and the same process to combine the two logs was repeated. We generated 45 different attack sessions with 272 web requests that resulted in 44,390 data requests. Combining them together we got 45 unique attack sessions with 272 event vectors. A typical anomalous session in the data set is represented in Fig. 5 which depicts a scenario where the deployed application has been modified by taking control of the web server.

```
98,1,301,369,GET /catalog HTTP/1.1,-,attack
113,0,200,28623,GET /catalog/index.php HTTP/1.1,-,attack
208,1,200,35467,GET /catalog/checkout_shipping.php HTTP/1.1,http://dummydata.xyz/catalog/index.php,attack
158,0,200,47801,GET /catalog/checkout_shipping_address.php HTTP/1.1,http://dummydata.xyz/catalog/checkout_shipping.php,attack
218,0,200,40401,GET /catalog/checkout_payment.php HTTP/1.1,http://dummydata.xyz/catalog/checkout_shipping_address.php,attack
```

Fig. 5 Representation of an Anomalous Session

4 Experiments and Results

We used the CRF++ toolkit [17] and the weka tool [23] for the experiments. Further, we developed python and shell scripts for data formatting and implementation. We

perform all experiments ten times by randomly selecting training and testing data and report the average. We use exactly the same samples for all the four methods. It must be noted that methods such as *decision trees, naive Bayes and support vector machines* are not designed for sequence labeling. However, for our purpose these methods can be applied by treating the data as *relational rather than considering them as sequences*. To experiment with these methods, we convert every session to a single record by appending sequential events at the end of the previous event and then labeling the entire session as either normal or as attack. For the support vector machines we experimented with three kernels; *poly-kernel, rbf-kernel and normalized-poly-kernel*, and varied the value of c between 1 and 100 for all of the kernels [23]. In the experiments we *vary the window size 'S' from 1 to 20* and analyze its effect on the attack detection accuracy. Window of size $S = 1$ indicates that we consider only the current request and do not consider the history while a window of size $S = 20$ shows that a sequence of 20 events is analyzed to perform the labeling. We report the results for measuring the effectiveness of attack detection using *precision, recall* and *F-measure*. However, due to space limitations, we do not present the results for efficiency. Nonetheless, the efficiency for our system was comparable to that of other methods.

Very often, attackers hide the attacks within normal events, making attack detection very difficult. We define the *disguised attack parameter*, 'p' as follows:

$$p = \frac{number\ of\ Attack\ events}{number\ of\ Normal\ events + number\ of\ Attack\ events}$$

where number of Attack events > 0 *and number of Normal events* $>= 0$

The value of 'p' lies in the range (0,1]. The attacks are not disguised when $p = 1$, since in this case the number of normal events is 0. As the value of 'p' decreases when the number of normal events is large, the attacks are disguised in a large number of normal events. In order to create disguised attack data, we add a random number of attack events at random locations in individual normal sessions and label the events as attack. This results in hiding the attacks within normal events such that the attack detection becomes difficult. We perform experiments to reflect these scenarios by varying the number of normal events in an attack session such that 'p' between 0 to 1.

4.1 Experiments with Clean Data (p = 1)

Figure 6 shows how the *F-measure* vary as we increase the window size 'S' from 1 to 20 for $p = 1$. We observe that conditional random fields and support vector machines perform similarly and their attack detection capability (*F-measure*) increases, slowly but steadily, as the number of sequential events analyzed together in a session increases. This shows that modeling a user session results in better attack detection accuracy compared to analyzing the events individually. However, decision trees and naive Bayes perform poorly and have low *F-measure* regardless of the window size 'S'.

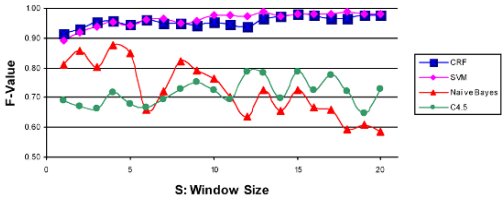

Fig. 6 Comparison of F-measure ($p = 1$)

4.2 Experiments with Disguised Attack Data (p = 0.60)

In order to test the robustness of the methods, we performed experiments with disguised attack data. We compare the results for all the four methods (conditional random fields, decision trees, naive Bayes and support vector machines) in Fig. 7 where we set $p = 0.60$. We observe that the conditional random fields performs best, outperforming all other methods and are robust in detecting disguised attacks. Their attack detection capability increases as the number of sequential events analyzed together in a session increases with the window size 'S'. Support vector machines, decision trees and the naive Bayes did not perform well when the attack data is disguised in normal events.

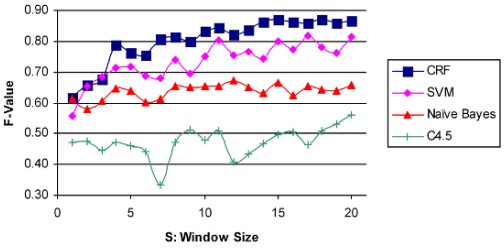

Fig. 7 Comparison of F-measure ($p = 0.60$)

Figures 8, 9, 10 and 11 represents the *precision, recall* and *F-measure* for conditional random fields, decision trees, naive Bayes and support vector machines.

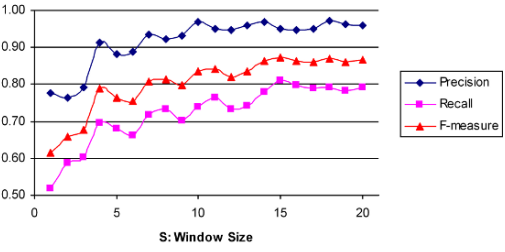

Fig. 8 Results with Conditional Random Fields

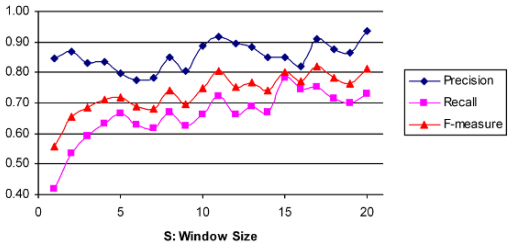

Fig. 9 Results with Support Vector Machines

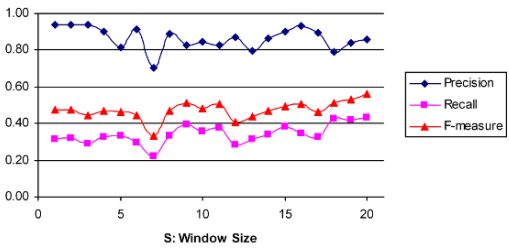

Fig. 10 Results with Decision Trees

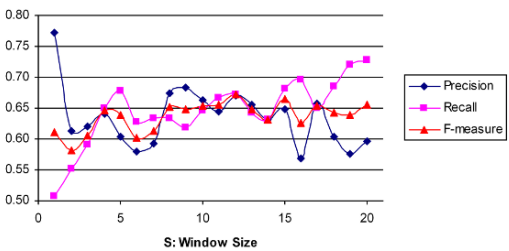

Fig. 11 Results with Naive Bayes

Figure 8 suggests that conditional random fields have high *F-measure* which increases steadily as the window size 'S' increases. The maximum value for *F-measure* is 0.87 at $S = 15$. This suggests that conditional random field generates less false alarms and the system performs reliably even when attacks are disguised.

For support vector machines, best results were obtained with *poly-kernel* and $c = 1$ and are reported in Fig. 9. We observe that support vector machines have moderate precision but low recall and hence low *F-measure*. The highest value for *F-measure* is 0.82 when $S = 17$.

Figure 10 represents that decision trees have very low *F-measure* suggesting that they cannot be effectively used for detecting anomalous data access when the attacks are disguised. The detection accuracy for decision trees remains fairly constant as 'S' increases. This is because the size of the decision tree remains constant even when the number of features increases since the goal of building a decision tree is to build a smallest tree with a large number of leaf nodes resulting in better classification. Hence, even when we increase the number of features, the size of the tree does not vary and their attack detection capability does not improve.

Results from Fig. 11 suggest that naive Bayes have low *F-measure* which fluctuates as the window size 'S' increases. There is little improvement in *F-measure* which remains low. The maximum value for *F-measure* is 0.67 at $S = 12$ suggesting that a system based on naive Bayes classifier is not able to detect attacks reliably.

4.3 Effect of 'S' on Attack Detection

In most situations, we want 'S' to be small since the complexity and the amount of history that needs to be maintained increases with 'S' and the system cannot respond in real time. Window size of 20 and beyond is often large resulting in delayed attack detection and high computation costs. Hence, we restrict 'S' to 20.

Table 1 Effect of 'S' on Attack Detection when $p = 0.60$

Size of Window 'S'	Decision Trees	Naive Bayes	Support Vector Machines	Conditional Random Fields
1	0.47	0.61	0.56	0.62
2	0.47	0.58	0.66	0.66
3	0.44	0.61	0.69	0.68
4	0.47	0.65	0.71	0.79
5	0.46	0.64	0.72	0.76
6	0.44	0.60	0.69	0.76
7	0.33	0.61	0.68	0.81
8	0.47	0.65	0.74	0.81
9	0.51	0.65	0.70	0.80
10	0.48	0.65	0.75	0.83
11	0.51	0.66	0.80	0.84
12	0.41	**0.67**	0.75	0.82
13	0.44	0.65	0.77	0.84
14	0.47	0.63	0.74	0.86
15	0.50	0.66	0.80	**0.87**
16	0.50	0.63	0.77	0.86
17	0.47	0.65	**0.82**	0.86
18	0.51	0.64	0.78	0.87
19	0.53	0.64	0.76	0.86
20	**0.56**	0.66	0.81	0.86

We observe that conditional random fields perform best and their attack detection capability increases as the window size increases. Additionally, when we increase 'S' beyond 20 (not shown in the graphs), the attack detection accuracy for conditional random fields increases steadily and the system achieves very high *F-measure* when we analyze the entire session together. From Table 1, we observe that decision trees analyzes 20 events together to reach their best performance while con-

ditional random fields achieve same performance by analyzing only a single event (i.e., $S = 1$). Similarly, naive Bayes peaked their performance at $S = 12$ while conditional random fields achieved the same performance at $S = 3$. Finally, support vector machines reach their best performance at window size $S = 17$ while the conditional random fields achieve the same performance at $S = 10$. *Hence, using conditional random fields attacks can be detected with higher accuracy at lower values of 'S' resulting in early attack detection and an efficient system.*

4.4 Effect of 'p' on Attack Detection ($0 < p < 1$)

We varied 'p', between 0 and 1 to analyze the robustness of conditional random fields with disguised attack data. In Fig. 12, we represent the effect of 'p' on conditional random fields for different values of 'S'. We make two observations; first, as 'p' decreases, it becomes difficult to detect attacks and second, irrespective of the value of 'p', the attack detection accuracy increases as 'S' increases.

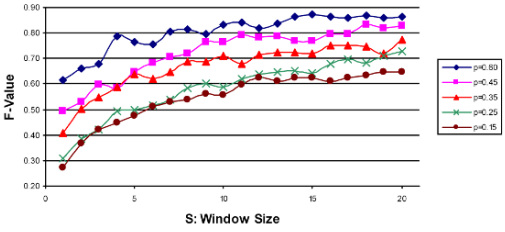

Fig. 12 Results for Conditional Random Fields when $0 < p < 1$

4.5 Discussion of Results

From the experiments, we observe that both conditional random fields and support vector machines performed very well when attacks were not disguised. However, support vector machines did not perform well while the conditional random fields were robust and detected disguised attacks reliably. This is because support vector machines perform classification in spatial domain thereby separating the classes by defining hyper-planes and distance measures. This results in higher attack detection accuracy when classes are distinct, but when the attacks are disguised, the system performs poorly. Decision trees and naive Bayes performed poorly in both cases. This is because they consider features of an event independently to label a particular event and then combine the results of all the features but do not consider the correlation between them. When the number of features is less, the error due to loss of correlation is less which increases with the number of features. Also, when 'S'

increases, decision trees select subsets of features and does not use all of the input features and hence, their attack detection accuracy does not improve. However, conditional random fields can model long range dependencies among a sequence of events since they do not assume independence among the event vectors and perform effectively even when the attacks are disguised. This is critical as intrusion is not a one step process and an attacker performs sequence of steps to gain control of the data. Conditional random fields can capture long range dependencies in the sequence of events, and hence, perform better.

Also note that an experienced attacker may try to disguise attacks within more than 20 normal events. Even then, our system can detect attacks as the system does not consider events independently. Nonetheless, there is a tradeoff between disguise attack parameter 'p' and window size 'S'. In general, for better attack detection, 'S' must be increased when 'p' decreases. *The advantage of conditional random fields is that higher attack detection occurs at lower values of 'S' which is desirable.*

5 Related Work

The field of Intrusion Detection started in around 1980's and many techniques have been proposed for building intrusion detection systems. A number of surveys [4], [21] compare various well known intrusion detection methods such as data mining approaches which include association rules and frequent episodes, clustering, naive Bayes classifier, hidden Markov models, decision trees, support vector machines, Bayesian network approaches, neural networks, conditional random fields and others. Detecting data breaches has mainly focused on finding anomalous data queries based on these methods [6], [15]. These methods, however, consider data access patterns in isolation of the events which generates the data request. Similarly, systems which model application access such as web-application firewall [8] do not consider underlying data access and simply perform protocol analysis [9].

Methods for detecting malicious database modifications include mining data dependencies among data items to create dependency rules [15], clustering of data queries [26], modeling time difference between multiple transactions [14], building role profiles using naive Bayes classifiers to model normal behaviour [6], user profiling based on user query frequent item-sets [25], defining distance measures to determine the closeness of a set of attributes that are referenced together [7] and fingerprinting of data queries [19], [20]. These approaches are rule based, expensive to build and their signatures must be updated regularly. Additionally, they cannot detect previously unseen attacks and are specific to a particular application.

This is different from our work as we first combine the application access logs and the associated data requests to generate unified logs and then use session modeling to analyze a sequence of events together rather than analyzing them individually to detect malicious data access. Our system is application independent and therefore can be widely used. Finally, we model application-data interaction which does not depend upon a user and therefore does not change overtime when compared with

user profiling based approaches. We also compare our work with [3] and [9]. In [3], the authors describe a tool for performing intrusion detection at application level. Their system uses Apache web server to implement an audit data source and the collected information is used to monitor the behaviour of the web server. This is different from our work as we are interested in analyzing the behaviour of a web application in conjunction to the underlying data. The system in [9] analyzes application layer protocols at the network level. This is also different from our work as we are interested in preventing malicious data access and our system operates at the application layer rather than at the network layer.

6 Conclusions

In this paper we proposed and evaluated a novel approach to analyze user sessions using sliding window for effective intrusion detection at application level. We also argued that combining application access logs and the corresponding data access logs to generate unified logs eliminates the need to analyze them separately thereby resulting in an efficient and accurate system. From our experiments we conclude that as the window size 'S' increases, the attack detection accuracy increases which justifies our motive of modeling user sessions rather than analyzing the events individually. Our results show that conditional random fields performed better than other methods and were robust to disguised attack data. Further, our framework is scalable for future applications and is not specific to any particular application except for data gathering. Finally, following good software engineering practices and taking care of logging mechanism during application development would not only help in application testing and related areas but would also provide necessary framework for building better and efficient application intrusion detection systems.

Acknowledgements We thank the students at the Department of Computer Science and Software Engineering, The University of Melbourne for helping us to collect data for our experiments.

References

1. ANSI/X3/SPARC Study Group on Data Base Management Systems: Interim Report, FDT (bulletin of ACM SIGMOD) 7, No.2, 1975.
2. osCommerce, Open Source Online Shop E-Commerce Solutions. Last accessed: January 08, 2008. `http://www.oscommerce.com/`.
3. M. Almgren and U. Lindqvist. Application-Integrated Data Collection for Security Monitoring. In *4th International Symposium on Recent Advances in Intrusion Detection*, pages 22–36. LNCS, Springer-Verlag, Vol (2212), 2001.
4. S. Axelsson. Research in Intrusion-Detection Systems: A Survey. Technical Report 98-17, Department of Computer Engineering, Chalmers University of Technology, 1998.
5. R. Bace and P. Mell. *Intrusion Detection Systems*. Gaithersburg, MD : Computer Security Division, Information Technology Laboratory, NIST, 2001.

6. E. Bertino, A. Kamra, E. Terzi, and A. Vakali. Intrusion Detection in RBAC-Administered Databases. In *21st Annual Computer Security Applications Conference*. IEEE, 2005.
7. C. Y. Chung, M. Gertz, and K. Levitt. DEMIDS: A Misuse Detection System for Database Systems. In *3rd International IFIP TC-11 WG11.5 Working Conference on Integrity and Internal Control in Information Systems*, pages 159–178. Kluwer Academic Pub., 1999.
8. L. Desmet, F. Piessens, W. Joosen, and P. Verbaeten. Bridging the Gap Between Web Application Firewalls and Web Applications. In *4th ACM workshop on Formal methods in security, FMSE*, pages 67–77. ACM, 2006.
9. H. Dreger, A. Feldmann, M. Mai, V. Paxson, and R. Sommer. Dynamic Application-Layer Protocol Analysis for Network Intrusion Detection. In *15th Usenix Security Symposium*, pages 257–272, 2006.
10. K. K. Gupta, B. Nath, and K. Ramamohanarao. Application Intrusion Detection Dataset. `http://www.csse.unimelb.edu.au/~kgupta`.
11. K. K. Gupta, B. Nath, and K. Ramamohanarao. Layered Approach using Conditional Random Fields for Intrusion Detection. *IEEE Transactions on Dependable and Secure Computing*. In Press.
12. K. K. Gupta, B. Nath, and K. Ramamohanarao. Conditional Random Fields for Intrusion Detection. In *21st International Conference on Advanced Information Networking and Applications Workshops*, pages 203–208. IEEE, 2007.
13. K. K. Gupta, B. Nath, K. Ramamohanarao, and A. Kazi. Attacking Confidentiality: An Agent Based Approach. In *IEEE International Conference on Intelligence and Security Informatics*, pages 285–296. LNCS, Springer Verlag, Vol (3975), 2006.
14. Y. Hu and B. Panda. Identification of Malicious Transactions in Database Systems. In *7th International Database Engineering and Applications Symposium*, pages 329–335. IEEE, 2003.
15. Y. Hu and B. Panda. A Data Mining Approach for Database Intrusion Detection. In *ACM symposium on Applied Computing*, pages 711–716. ACM, 2004.
16. D. Klein and C. D. Manning. Conditional Structure versus Conditional Estimation in NLP Models. In *ACL-02 Conference on Empirical methods in Natural Language Processing Vol (10)*, pages 9–16. Association for Computational Linguistics, Morristown, NJ, USA, 2002.
17. T. Kudu. CRF++: Yet another CRF toolkit. Last accessed: February 9, 2008. `http://crfpp.sourceforge.net/`.
18. J. Lafferty, A. McCallum, and F. Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *18th International Conference on Machine Learning*, pages 282–289, 2001.
19. S. Y. Lee, W. L. Low, and P. Y. Wong. Learning Fingerprints for a Database Intrusion Detection System. In *7th European Symposium on Research in Computer Security, Vol (2502)*, pages 264–279. LNCS, Springer-Verlag, 2002.
20. W. L. Low, J. Lee, and P. Teoh. DIDAFIT: Detecting Intrusions in Databases Through Fingerprinting Transactions. In *4th International Conference on Enterprise Information Systems*, pages 264–269, 2002.
21. A. Patcha and J.-M. Park. An Overview of Anomaly Detection Techniques: Existing Solutions and Latest Technological Trends. *Computer Networks*, 51(12):3448–3470, 2007.
22. C. Sutton and A. McCallum. An Introduction to Conditional Random Fields for Relational Learning. In *Introduction to Statistical Relational Learning*. MIT, 2006.
23. I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.
24. N. Ye, X. Li, Q. Chen, S. M. Emran, and M. Xu. Probabilistic Techniques for Intrusion Detection Based on Computer Audit Data. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 31(4):266–274, 2001.
25. Y. Zhong and Xiao-Lin-Qin. Research on Algorithm of User Query Frequent Itemsets Mining. In *3rd International Conference on Machine Learning and Cybernetics, Vol (3)*, pages 1671–1676. IEEE, 2004.
26. Y. Zhong, Z. Zhu, and X. Qin. A Clustering Method Based on Data Queries and Its Application in Database Intrusion Detection. In *4th International Conference on Machine Learning and Cybernetics, Vol (4)*, pages 2096–2101. IEEE, 2005.