CERIAS Tech Report 2002-36

ON-THE-FLY INTRUSION DETECTION FOR WEB PORTALS

by Radu Sion, Mikhail Atallah, Sunil Prabhakar

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907

On-the-fly Intrusion Detection for Web Portals (draft) [*]

Radu Sion
Computer Sciences & CERIAS
Purdue University
(sion@cs.purdue.edu)

Mikhail Atallah
Computer Sciences & CERIAS
Purdue University
(mja@cs.purdue.edu)

Sunil Prabhakar
Computer Sciences & CERIAS
Purdue University
(sunil@cs.purdue.edu)

Abstract

Remote access to distributed hyper-linked information proves to be one of the killer *applications for computer networks. More and more content in current inter and intra nets is available as hyper-data, a form easing its distribution and semantic organization.*

In the framework of the Internet's Web-Portals and Pay-Sites, mechanisms for login based on username and password enable the dynamic customization as well as partial protection of the content. In other applications (e.g. commercial intra-nets) various similar schemes of authentication are deployed.

Nevertheless, stolen passwords are an easy avenue to identity theft, in both public and commercial data networks. Once a perpetrator enters a system, assuming an authorized user's identity, the task of actually detecting this intrusion becomes non-trivial and is often ignored completely.

Thus, in addition to the initial authentication step we propose a runtime intrusion detection mechanism, required to maintain a virtually continuous user authentication process and detect identity theft and password misuses.

The current paper focuses on designing a pervasive intrusion detection method for hyper-data systems, based on training on and analyzing of access patterns to hyper-linked data, aiming at detecting intruders and raising a red flag at the content provider's side. Our solution is based on a new technique, on-the-fly adaptive training *for normality on streams of data access patterns. This enables runtime intrusion detection through analysis of correlations between current patterns and the adaptive past-knowledge. Such a method is to be used in conjunction with current username-password protection schemes. We introduce the motivation behind our solution , discuss the novel detection and training metrics and propose a real-life deployment design. We implement the main algorithm and perform experiments for assessing its intrusion detection ability, with very encouraging results. We also discuss the deployment of our method for detecting automatic spam-bot accesses.*

[*]Portions of this work were supported by Grants EIA-9903545, ISS-0219560, IIS-9985019, IIS-9972883 from the National Science Foundation, Contract N00014-02-1-0364 from the Office of Naval Research, and by sponsors of the Center for Education and Research in Information Assurance and Security.

1 Introduction

In the framework of the Internet's Web-Portals and Pay-Sites, mechanisms for login based on username and password enable the dynamic customization as well as partial protection of the content. In other applications (e.g. commercial intra-nets) various similar schemes of authentication are deployed. Nevertheless, stolen passwords are an easy avenue to identity theft, in both public and commercial data networks. Once a perpetrator entered a system, assuming an authorized user's identity, the task of actually detecting this intrusion becomes non-trivial and is in most cases ignored completely. Thus, in addition to the initial authentication step, runtime intrusion detection mechanisms are required to maintain a virtually continuous user authentication process and detect identity theft and password misuses.

In this paper we introduce a new solution for intrusion detection in systems with an underlying hyper-linked content structure (e.g. Web Sites, Grid Portals). Our solution is based on a new technique, *on-the-fly adaptive training* for normality on streams of data access patterns. This enables runtime intrusion detection through analysis of correlations between current patterns and the adaptive past-knowledge.

We envision a broad applicability of our research. A suggested implementation as a web-server module, as outlined in Section 4, could be used in additionally enforcing current weak pay-site authentication mechanisms. Deployment inside a company or government agency intra-net could be used to detect identity theft and immediately revoke credentials for access to sensitive internal information. An additional level of privacy control can be guaranteed by deployment in the framework of a hospital medical records access software. The main contributions of this work are:

- Identification of the problem of training for normality patterns in data access systems with underlying hyper-linked content
- Formulation of a theoretical algorithmic solution for on-the-fly intrusion detection using normality training of hyper-linked content access patterns and an experimental evaluation thereof
- A design proposal for an implementation in a concrete real-world environment, e.g. secure web-server

The paper is structured as follows. Section 2 defines a succinct model for the web-portal framework and introduces the problem, its associated motivation and main challenges. Section 3 outlines our solution and discusses novel concepts such as on-the-fly training and transition queuing. Section 4 discusses several aspects of our solution (including the potential for detecting automatic spam-bot accesses in both *anomaly* and *misuse* detection scenarios) and proposes an implementation design for a solution deployment. Section 6 introduces avenues for future research.

2 Problem

We define a *web-object* to be a digital data object composed of useful content (i.e. meant for the content consumer, e.g. web-user) and *web-links* (aka. hyperlinks). A web-link is a construct that allows accessing of an associated web-object, for example through user-driven GUI actions such as mouse-clicks. A web-user "clicks" on a web-link with the intention to "load" (e.g. access for consumption) the associated web-object.

Note: For brevity reasons, here we are assuming naturally that a user can only "click" on web-links embedded in the *currently* accessed (i.e. loaded) web-object, that is, no direct URL input (e.g. by typing into an "URL bar") is possible. This assumption simplifies access pattern processing and can be relaxed immediately through the aid of technical counter-measures, which are not the subject of this research.

A *web-portal* can be viewed as a set of web-objects (e.g. web-pages) that are structured in a certain "browsing-graph" through the associated web-links contained in each web-object. There is a finite set of authorized web-users that are to be allowed access to the web-portal's content. Each of the individual web-users can be associated directly with a digital "profile", a highly user-specific data set, containing at least some authorization tokens such as a *username* and an associated *password*. To access the web-portal, the web-user is to provide the username and password as input to a certain authorization mechanism. Once authorized the user is offered full access to the web-portal content.

From a content-centric view, a web-portal is composed of it's data objects $\mathbb{O} = \{O_1, O_2, ...O_n\}$ and the associated web-link transitions $T(O_i) \forall O_i \in \mathbb{O}$. Thus a web-portal is a graph construct $\mathbb{W} = (\mathbb{O}, \mathbb{T} = \cup_{\forall O_i \in \mathbb{O}} T(O_i))$.

2.1 Challenges

The main issues of a password protection access model derive from the fact that password theft and reuse is an extremely easy avenue for web-portal intrusion (or unauthorized access). As a solution we propose an additional layer of on-the-fly authentication through a dynamic modeling mechanism for normality in web-user data access patterns. This mechanism is to be used as a runtime aid for web-portal intrusion detection and ultimately prevention.

Existing research addresses the issue of detecting anomalies in embedded systems [4], software [1], user typing behavior [5] [6] etc. In these efforts (and others [2]) pre-training on a state of normality is deployed in the construction of a model which is then to be used in the detection of "anomalies" (potential intrusions). There are several characteristics of this approach that become drawbacks when applied in the web-portal framework. These are the main challenges in constructing a mechanism for training on hyper-linked data access patterns.

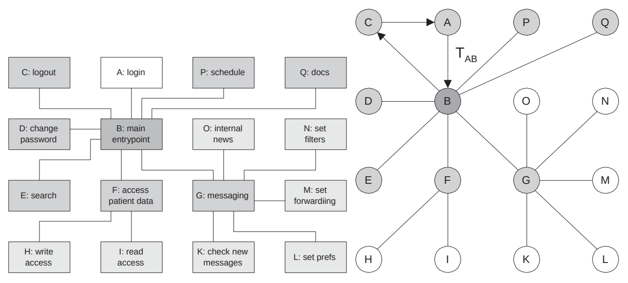

Figure 1. (a) Sample hospital intra-net web-portal. A likely authorized access pattern could present a trace of A,B,G,O,P,F,I. A suspicious pattern could show up as A,B,D,F,H,G,M,L,N. (b) Training occurs not at state level (e.g. web-page) but on transitions between states (e.g. "web-click").

- Whereas embedded systems (for example), are usually characterized by relatively stable and static state transition graphs, current web-portals feature a highly dynamic content behavior. An intrusion detection scheme has to be able to adapt to this dynamism. Pre-training presents the inherent flaw of producing a model often outdated by the time it is to be applied for intrusion detection.

- Given the nature of web content, *new* content has to be dealt with naturally. For example the creation of a new web-object will likely result immediately in data accesses to it. This has to be taken into account when analyzing data access patterns and an associated "normal" behavior, and, more important, the training algorithm has to naturally accommodate it.

- Last but not least, in a web-portal, it is not very intuitive *how* to enforce a lengthy (usually a *must*, in the approaches discussed above) pre-training phase. It is unreasonable to assume it can be imposed on the users of the web-portal without incurring negative customer feedback and associated customer-base losses.

Given the above, the training mechanism is to naturally be highly dynamic and adapt to changes in data access patterns. A trade-off is to be observed between this adaptability and the ability to detect intrusions. Normality needs to be defined accordingly and an associated adjustable normality-threshold has to be made available.

3 A Solution

We initiate our construction by establishing the nature of the input to the access pattern normality training algorithm.

In other words, what is a training set entry composed of (i.e. what is the training alphabet [4]) ?

The immediate, naive, solution would consider the data objects $O_i \in \mathbb{O}$ and associated access times as symbols in our training set. Given the specifics of the web-portal framework this would present an important drawback, namely the impossibility of capturing the main user-web-portal interaction behavior, the "click".

"Clicking" on a certain web-link semantically "links" two separate contexts, the "current" web-object *and* the "target" web-object, corresponding to the web-link. A certain "target" web-object can have many different "current" web-objects containing web-links pointing to it. The behavior of the user is not only characterized by the interest in the "target" but also by the "path" taken to get there. This information is not captured easily by a training alphabet corresponding to the data objects $O_i \in \mathbb{O}$.

We propose to use as training input actual *transitions* (e.g. $T_{ij} \in \mathbb{T}$) associated with web-links. The transition information naturally captures both the "target" context and the "path" taken to get there. A transition $T_{ij} \in \mathbb{T}$ is characterized by the two web-objects it links through the web-user "click" and a a time-stamp [1] ($T_{ij}.ts$).

Thus our training mechanism receives as input a stream of transition events (see Figure 2). It accordingly constructs an on-the-fly model for data access normality and then uses the (dynamically changing) model to identify abnormal behavior in the input stream [2].

3.1 Transition Queuing

From the discussion above we can derive two required characteristics of a desired solution. It has to dynamically adapt to changing trends in data access patterns (with an adjustable degree of sensitivity), and, at the same time, it should enable the timely detection of unauthorized accesses (i.e. intrusions).

With respect to the change adaptability, at one extreme is a system that does not adapt at all, but is rather based on normality pre-training. This scheme is suitable for cases of long-term, stable patterns of normality in the state transitions (eventually hard-coded and pre-determined, e.g. in hardware devices over a narrow set of inputs).

At the other extreme, a system adapts its normality model continuously from each and every access pattern input. While this can be a very effective solution for problems of the branch prediction type, it suffers from the impossibility to actually isolate a potential intrusion (e.g. occurring "now") from the normal (e.g. occurred "past"), as every occurring pattern is immediately "absorbed" into the normality model.

Our solution provides a trade-off between both extremes. A FIFO structure (see Figure 2, "transition queue") is used to delay the absorption of training data items (i.e. observed transitions) into the runtime normality model. As transitions are observed at certain moments in time, they are introduced in the transition queue and will only exit (and enter the next phase, the actual training process) after a certain *queuing timeout* elapses. The transitions to be found in the queue at a given moment in time are called "queuing window". Through delaying of the transition absorption in the training model, the solution presented provides the necessary separation between a potential intrusion (i.e. whose patterns are to be found in the transition queue) and a model of normality. At the same time, as it considers *all* inputs in the training process it naturally adjusts to changes in the data access patterns.

[1] Here in absolute UTC time; "system" time is explored in Section 3.2.3.

[2] The collection of this "web click" type of data has already been discussed in numerous commercial frameworks, including [3].

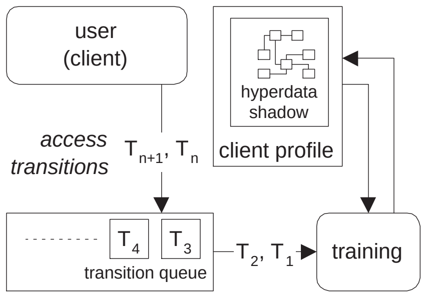

Figure 2. Transition-queued on-the-fly training.

The main assumption behind the use of transition queuing is based on the understanding that a sound real-time intrusion detection mechanism detects intrusions necessarily within few time-spans of their happening. If an intrusion goes undetected for too long a time-period its effects (e.g. the attack) are already consumed with a high probability [3]. In the worst case, if the detection process fails to detect an intrusion, this solution absorbs the associated (abnormal) transitions in the normality model. The dynamicity of the training process (see below) aims to gradually overcome the effect of such a case. Nevertheless we would argue that not much more can be done in this scenario. The inability to detect the intrusion necessarily limits the potential of the system and inherently, from this perspective, the intrusion transition patterns become "normal" (after all who can detect them as being abnormal ?).

The queuing delay is an adjustable parameter in transition queuing. Different queuing window sizes need to be considered for different applications, in order to be able to effectively "catch" abnormal patterns and enable the detection process to a maximum.

3.2 On-the-fly Training and Intrusion Detection

After the stream of observed transitions is delayed by queuing, it enters the training process. In this process, a model of normality is updated continuously from this data. A simultaneous intrusion detection process uses this model to detect intrusion patterns in the transition queue.

3.2.1 Hyper-data Shadow

Working in a structured (i.e. hyper-linked) data environment (e.g. web-portal) enables the construction of a normality model data structure that relates in a natural bijection with the underlying web-portal data. In other words, because

[3] As discussed further, in the case an intrusion is detected, the transition queue is "flushed" and the normality model training re-started with new data.

```
while (true) do
  // training:
  1. tr ← tr_queue.nextFIFOBlocking() // blocks until timeout
  2. tr_shadow.get(tr.name).profile.add(tr.time())
  // detection:
  3. tr_shadow_copy ← tr_shadow.copy()
  4. norm ← 0
  5. while ((tr ← tr_queue.nextFIFO()) ≠ null) do
       //: weight at time of occurence
       a) norm ← norm + weight(tr.name,tr.time())
       //: absorb effect into shadow copy
       b) tr_shadow_copy.get(tr.name).profile.add(tr.time())
  6. if (norm < norm_min) then
       //: call intrusion handler
       a) intrusion_handler()
       b) tr_queue.flush()
```

Figure 3. Training and Detection Algorithm.

transitions are structure-conditioned (i.e. a transition can only occur if there exists an associated web-link), a natural data structure for a transition model is exactly a weighted directional graph (see Figure 1 (a), (b)) of version of the web-portal hyper-linked content. This data structure is called *hyper-data shadow*. Thus, the hyper-data shadow is a directed graph data structure in which each node is associated with a web-object (i.e. "browsing state") and the edges with transitions corresponding to web-links.

In the detection process, a traversal of the graph is performed according to the analyzed input patterns (e.g. the queuing window data) and a "normality metric" is computed. Each edge in the graph has a certain associated "transition profile" ($u()$) FIFO data structure which contains time-stamp information about the occurrences of the transition it corresponds to

$$u(T_{ij}) = [t_1, t_2, ..., t_p] \forall T_{ij} \in \mathbb{T} \tag{1}$$

where t_i are timestamps at which the transition was observed and p upper-bounds the size of the transition profile according to available storage space and number of transitions. The hyper-data shadow is updated each time a transition becomes input for the training process (i.e. by exiting the transition queue). If full (e.g. already p elements), the corresponding transition profile is "shifted" by discarding the oldest element, then the incoming transition is inserted.

3.2.2 Normality Metric

An automated abnormality detection mechanism requires a computable "normality metric", a function of the system state that closely models the desired definition of "system normality". In this solution we are modeling normality from a content-centric perspective with respect to web-portal users actions, more specific data access (e.g. web-clicks).

An analysis of a web-click trace has several types of information available, including *content semantics*, *structural paths* and *access timing information*. In the web-portal framework absolute timing information is not practically relevant as discussed in Section 3.2.3. Using semantics associated with a certain content in modeling user behavior is a promising avenue situated at an area boundary (e.g. natural language and intrusion detection) suited for future research. In the present paper we address the definition of normal user behavior from a structural path perspective, that is, we are concerned most with the web-portal browsing patterns. This is why the metric we propose quantifies normality with respect to observed browsing patterns.

The *transition weight* of a certain transition T_{ij} at a certain moment t is defined (see Section 5 for an experimental analysis of other weights) by

$$weight(T_{ij}, t) = ||u(T_{ij})|| \times \frac{avg(u(T_{ij}))}{time} = \frac{\sum_{t_i \in u(T_{ij})} t_i}{time} \tag{2}$$

The transition weight concept captures the time-locality and weighted frequency of the observed transition. Given a sorted set (FIFO) of transitions (e.g. the transition queuing window) $\mathbb{Q} = [Q_1, Q_2, ..., Q_w] \subset \mathbb{T}$ the normality metric for $\mathbb{Q}$ at time t is then defined by

$$normality(\mathbb{Q}, t) = \frac{1}{||\mathbb{Q}||} \times \sum_{\forall Q_i \in \mathbb{Q}}^{*} weight(Q_i, t) \tag{3}$$

In the formula above, the computation of the transition weights is done on a special copy of the hyper-data shadow, gradually trained with the incoming transitions in $\mathbb{Q}$ (in the order they appear in $\mathbb{Q}$). Each transition weight is computed only after the previous transitions from $\mathbb{Q}$ were already considered and used in training the hyper-data shadow copy. The symbol $\sum^*$ was used to denote the fact that after each new element (transition weight) in the sum is added to the sum result, the corresponding transition is used in training the hyper-data shadow copy. This happens *before* adding the next weight. The purpose of the incremental training process of the hyper-data shadow copy, in the detection mechanism, is twofold. While it preserves the original hyper-data shadow, it also ensures that normality is measured consistently, by naturally considering also the incoming transitions in $\mathbb{Q}$. This addresses a scenario in which for example part of the elements in $\mathbb{Q}$ induce (through training) a drastic change to the hyper-data shadow. If not trained for, the detection metric would result in a different, inconsistent value (because it does not consider the natural flow of transitions).

The normality metric matches the given series of transitions against the dynamic model of the observed past. By using transition weights as base components it captures both the "freshness" and the frequency characteristics of this model. A minimum allowable normality value $normality_{min}$ defines the threshold below which an intrusion is suspected and the corresponding "intrusion handler" is signaled. Upon detecting an intrusion, the transition queue output is disabled and training is stopped until the intrusion handler deals with the intrusion. Then the queue is flushed and training commences.

3.2.3 System Time

Absolute time values are not suited for an analysis of user browsing patterns from an content-centric perspective. Most web-portal users are not in a "glued to the screen" working

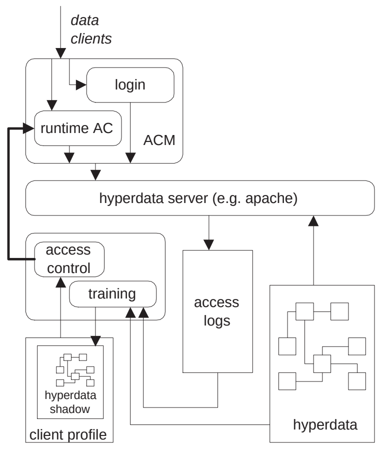

Figure 4. System Architecture Overview

mode and time locality cannot be assessed properly by using absolute time. Lunch-hour breaks and night-shifts do significantly shift time locality in equations such as 2 and 3 without a real correspondence in actual user behavior.

For the purpose of computing weights and normality metrics, we propose the deployment of a Lamport-style system time mechanism. As each normality computation is done per web-user, each web-user's "time" will be defined as the *total number of transitions already performed* by this user. This involves keeping a transition counter for each user [4] An algorithm summary is illustrated in Figure 3

4 Discussion

In comparison to the "Markov detector" deployed in [4], to a certain extent, our solution can be viewed as a "Markov detector with memory". The Markovian nature of the model derives from the fact that it practically samples normalized occurrence frequencies for input symbols (i.e. transitions). The model also features memory as it considers time-locality of observed inputs and weights "older" occurrences less than "new" ones, see Equation 2.

We believe this dual nature to provide much of the power of our solution. The normality model is built by sampling occurrences for transitions, while the time-weighted memory effect provides for adaptability to access pattern changes.

We propose an implementation architecture as a web-server component (see Figure 4). Server access is to be partly regulated by the runtime access patterns analysis module. This module implements the detection and training algorithm (see Figure 3) which identifies abnormal patterns based on stored training user profiles and notifies the access control module (ACM) which acts as a *intrusion handler*. Runtime access data can be collected either via web logs (in the case of an external module implementation) or directly by using server module callback functionality inside the web-server. This enables a notification for each and every web-object access to be received and processed directly by the training module. Due to space constraints, more details are omitted.

[4]*Transition queuing* is necessarily linked to absolute time locality as discussed in Section 3.1 and the queuing timeout is expressed in absolute time. System time values are only used in computing normality metrics and transition weights.

Web-Bots

Automated web-bots are more and more used for automated web-data collection and indexing. Spam-bots are a particular case of web-bots, used for the specific purpose to (often illegally) collect personal contact information for the purpose of targeted product advertisements and general marketing. We believe our solution can be deployed as an effective tool in detecting automated web-bots by using a slightly different deployment scenario. Whereas in the password-protected web-portal a claimed association between real browsing patterns and a certain username was to be verified, when detecting automated web-bots there is no such claim [5] Rather two avenues present themselves. In one scenario, a model of the expected web-bots is built by training for the known associated access patterns (i.e. *misuse detection*). While this case should offer accurate detection of known patterns, it suffers from the impossibility of detecting new (unknown) automated web-bots.

Another scenario deploys training in the construction of a general model of normal web-user access on trusted non-web-bot accesses (i.e. *anomaly detection*). This model is then matched with incoming (un-trusted) accesses and an estimation of the access patterns normality is done. The question asked is whether the given accesses are of a normal (i.e. real, non-automated) web-user or come from an automated browsing tool. In this scenario we estimate a higher rate of detection failure and false positives. Nevertheless we believe that the potential to detect new, previously unseen, web-bots is a benefit often out-weighting these disadvantages.

A combination of the two deployment scenarios could also be envisioned, under the form of a "chain" of detectors. The first stage, detection using a model of a known set of web-bots, could be followed by a stage of "abnormality" assessment. Thus, if no known patterns are discovered initially, this second assessment attempts to detect unknown automatic web-bots, albeit with higher false-positive and false-negative rates.

Given current space limitations, a more in depth analysis is out of scope here. We performed several detection experiments as outlined in Section 5. Further work with well-known web-bots and associated browsing patterns are to reveal the detection effectiveness of our method.

5 Experiments

We implemented our detection algorithm in Java and performed experiments to test its detection ability. The results are extremely encouraging. A web-user is simulated through a dynamically defined finite state machine (FSM). The states in the FSM correspond to web-objects and an actual "run" corresponds to a web-user browsing session.

Initially a given FSM is "run" a certain amount of time and the FSM transitions are fed into the on-the-fly training algorithm. After a certain time (TRAINING_LENGTH) the FSM and its transition probabilities are modified for a limited (INTRUSION_LENGTH) time period (modeling an intruder

[5]A web-bot usually browses *public*, non-protected web content, thus no username-password-identity association is available.

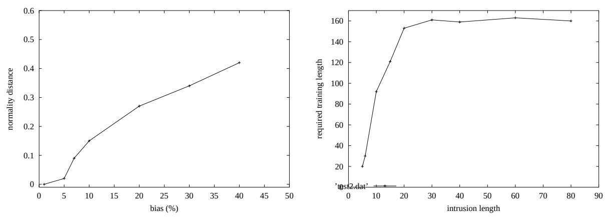

Figure 5. (a) Observed intrusion detection effectiveness as a function of data access pattern deviation from normal (bias). (b) Detecting intrusions of increasing complexity, requires increasingly more "normality" knowledge.

access). The modifications are gradually introduced biases in the FSM transition probabilities and can be quantified in terms of the original (i.e. normal) FSM transition probabilities (i.e. "bias" factor in Figure 5 (a)). The (now biased) FSM is then "run" for a certain time and the on-the-fly detection algorithm's ability to detect an intrusion (i.e. the FSM changes) is assessed by comparing the normality metrics output for the original (unbiased) FSM and the ones for the biased FSM.

Figure 5 (a) shows that as the bias factor increases for the modified FSM, the intrusion detection process becomes more and more effective (intuitively expected), i.e. the distance between the normality metrics for the normal and biased case increase. Figure 5 (b) displays the (15 runs averaged) dependency between TRAINING_LENGTH and INTRUSION_LENGTH for a certain guaranteed minimum detection threshold. In other words, the question answered here is: how "much" (TRAINING_LENGTH) normality do we need to see in order to be able to detect intrusions of "this" (INTRUSION_LENGTH) length. There exists an upper-bound effect in this graph, probably determined by the finite nature of the FSMs. After a certain point the normality model likely becomes an exact copy of the FSM, and thus, seeing additional access patterns (past that point) won't help. More experiments should explore the dependency between the FSM state graph diameter and this upper bound effect. Also naturally, intrusions of smaller access pattern sizes prove to be harder to detect, especially if the introduced FSM bias is small.

In an attempt to increase the effectiveness of our method, and better capture distinguishing access patterns, we experimented with different metrics of normality and associated transition weights. Several variants were assessed, including the following transition weights:

$$weight(T_{ij}, t) = \frac{1}{||u(T_{ij})||} \times \sum_{t_i \in u(T_{ij})} \frac{1}{time - t_i} \qquad (4)$$

$$weight(T_{ij}, t) = \frac{max(u(T_{ij}))}{t \times ||u(T_{ij})||} \times \sum_{t_i \in u(T_{ij})} |min(u(T_{ij})) - t_i| \qquad (5)$$

While the weight defined in equation 4 aims to capture time locality, it suffers from the inability to also capture and weight the transition frequency. Thus, for example, a low-frequency transition seen very recently might weight much more than a very high-frequency transition seen in the near past but not recently. Equation 5 aims to capture both time locality and transition frequency. Unfortunately it only considers relative (to the transition itself) time locality and misses the semantic link to system time. One very old high-frequency transition can thus be weighted much more than a more recent lower-frequency one. This impacts both the dynamicity and detection ability of normality metric. The transition weight (equation 2) and metric proposed in Section 3.2.2 proved to be most accurate in detecting patterns generated by the abnormal (biased) generator.

Given the current space constraints a more detailed experimental analysis is out of scope here. Nevertheless it is to be noted that the results are extremely encouraging. For example, a deviation (bias) as low as 3% from the "normal" access pattern can be accurately detected.

6 Conclusions

In the present paper we defined the problem of intrusion detection through normality training on content access patterns in systems with an underlying hyper-linked structure (e.g. web-portals). We introduced an algorithmic solution appropriate for the particularities of the framework. Our solution dynamically adapts to changes in content access patterns while at the same time detecting intrusions. It is based on novel concepts such on-the-fly training and transition queuing.

Several issues warrant research continuation, including the ability to handle web-object semantic shifts (e.g. same web-object but different content) through content summarization, the ability to detect *structural classes* ("partitions" of the hyper-linked content that are usually accessed simultaneously. Different types of hyper-linked data and associated theoretical and deployment issues should be analyzed.

Further research should focus on an actual proof-of-concept implementation in the framework of an existing content providing server (as proposed above) and on assessing its effectiveness through training and real-life testing in a large-scale experiment.

References

[1] A. Ghosh, J. Wanken, and F. Charron. Detecting anomalous and unknown intrusions against programs. In *Proceedings of the 1998 Annual Computer Security Applications Conference (ACSAC)*, December 1998.

[2] Wenke Lee and Salvatore Stolfo. Data mining approaches for intrusion detection. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX, 1998.

[3] Martin Lurie. Web click stream analysis using linux clusters. In *Linux Journal (www.linuxjournal.com)*, November 2001.

[4] Roy A. Maxion and Kymie M.C. Tan. Anomaly detection in embedded systems. *IEEE Transactions on Computers*, 51(2):108–120, February 2002.

[5] Fabian Monrose and Aviel D. Rubin. Keystroke dynamics as a biometric for authentication. *Future Generation Computer Systems*, 16(4):351–359, 2000.

[6] Dawn Xiaodong Song, David Wagner, and Xuqing Tian. Timing analysis of keystrokes and timing attacks on SSH. In *Tenth USENIX Security Symposium*, 2001.