The Use of Frequent Episodes in Intrusion Detection

Liubov Kokorina

Masteroppgave
Master i Teknologi - Medieteknikk
30 ECTS
Avdeling for informatikk og medieteknikk
Høgskolen i Gjøvik, 2009

Avdeling for
informatikk og medieteknikk
Høgskolen i Gjøvik
Postboks 191
2802 Gjøvik

Department of Computer Science
and Media Technology
Gjøvik University College
Box 191
N-2802 Gjøvik
Norway

The Use of Frequent Episodes in Intrusion Detection

Liubov Kokorina

2009/07/01

Abstract

In today's intrusion detection systems (IDSs), a trade-off between efficiency and accuracy must be achieved. Because of that, the decision on structures for representing patterns of normal and intrusive behavior are of crucial importance as well as pattern discovery techniques relevant for the detection of as many current attacks as possible. In this thesis we evaluate compatibility of so-called frequent episodes to intrusion detection by studying various attacks and episodes constructed of the attacks' events. We also describe several possibilities of the episode structure discovered under our experiment.

In the thesis, we discuss architecture issues of episode-based hybrid IDSs, combining misuse and anomaly approaches to take advantages of both of them. In addition, we propose a model for a new IDS on episodes. The model is built on episode discovery with sliding window and new episode analysis techniques, which we designed for intrusion detection. The first one is a modification of a frequent episode discovery technique, which is widely used in anomaly detection. The new techniques deal with rare episode discovery and event distribution analysis, which are supposed to characterize event series. The experiment demontstrates some results of the techniques in findnig similarities and regularities of automated (generated by computer programs) attacks.

Liubov Kokorina, 2009/07/01

Contents

List of Figures

List of Tables

1 Introduction

1.1 Topic Covered by the Thesis

The main security goals of any organization are defending the assets from outsiders and providing stable access to all legitimate users in the organization. The assets and necessary access must be described in detail in the organization's security policy in order to specify security goals. Violation of security policy is commonly called intrusion. There are a lot of intrusions of different kinds, which affect organizations in various ways. Intrusion detection is an analysis of information with the purpose of finding signs of intrusions.

Intrusion detection systems (IDSs) collect and process system and network events in order to find intrusions. They can detect intrusions afterwards (offline IDSs) or in real time (real-time IDSs). A real-time IDS detects intrusions on an early stage and allows to stop the attack or eliminate its consequences. Offline IDSs are useful for forensics and vulnerability assessment: they point on weaknessies of information systems and, in that way, help to achieve effective security. An IDS usually consists of three modules: audit, detection, and decision. The audit module collects information on events on the system's location (host, network, application). The detection module processes the audit data and selects intrusive events. And then, the decision module estimates the risks of the intrusions.

There are two common selection principles in the detection module: misuse and anomaly detection. The first one means comparing audit data to attack signatures – patterns of known attacks. The anomaly detection principle involves normality checks: all behavior that deviates from the predefined normal profile is supposed to be intrusive. There are several methods for detection of intrusive events: statistics, data mining, artificial intelligence, etc. This work discusses the possibility of building a real-time hybrid IDS, combining both principles and based on a data mining approach called frequent episodes.

Episode, as a data structure, was first presented in [3] as a partially ordered set of events that occur relatively close to each other (i.e. within one time window). Having specified the time window size and frequency threshold, one can decide if an episode is frequent. There have been proposed several algorithms for finding all frequent episodes (FEs) that belong to a given episode class from a given sequence of events. Frequent episode discovery (FED) algorithms can be applied widely to intrusion detection, for example, to extract useful event features for preprocessing of raw data, generate attack patterns for misuse detection, build normal profiles for anomaly detection and discover patterns from audit data, which can be matched against attack signatures or normal profiles.

1.2 Keywords

Site security monitoring, frequent episodes, data mining

1.3 Problem Description

The audit module of an IDS, usually, collects large amounts of unformatted data. These raw data must include at least a set of features required for behavior distinguishing. On the other side, data, containing many useless features, make the detection process inefficient. Hence, the input data should be built exactly on a carefully chosen set of necessary features.

The detection module requires initialization before use. The initialization involves uploading or generating intrusion patterns in case of misuse detection or a normal profile in case of anomaly detection. Then, intrusion detection is a process of matching data from the audit module's output against these templates – intrusion patterns or the normal profile. Both initialization and detection operate with data structures, which represent real data, preserving characteristics that are important for intrusion detection. So, the decision on the data structures is critical and must be made prior to the IDS building process. The algorithm for discovery of these structures is significant too, because it influences the system's efficiency. It can be used to extract useful information from audit data and produce templates for matching. The algorithm should have the following properties: fast enough (especially, for real-time intrusion detection) and precise (all structures must be discovered).

The process of matching the data structures also requires a fast algorithm for comparison. It may encode the structures and contain comparison rules. Absolute matching can produce matches poorly if the data structures are complex. So, despite that the approximate matching schemes are usually complex and need to be appropriately set up, only they can provide effective matching in detection modules based on data mining. The sets of data structures (the templates and the audit module's output), even obtained by using a good discovery algorithm, may be redundant (contain structures that are combinations of or similar to other structures) and include many useless structures, which influence matching efficiency. To optimize the matching procedure, the sets can be "pruned" by applying an effective pruning algorithm. Building of such an algorithm is a much more difficult task in comparison to discovery algorithms, since it must significantly reduce the sets (increase efficiency) without major loss of precision.

Misuse-based IDSs alert when a data structure from the audit data matches against an intrusion pattern. Obviously, they are not able to detect unknown attacks. Anomaly-based IDSs classify system behavior reflected in the audit data, usually, into two classes: normal and intrusive, although there are no sharp limits between them. Thus, the IDSs of this kind produce a large amount of false positives (alarm but no intrusion) and false negatives (intrusion but no alarm), which are commonly used as measures of the IDSs' accuracy. False negatives mean missed intrusion, and they must be eliminated. A lot of false positives flood with alarms and are not good either. The misuse and anomaly approaches can be combined to allow a misuse-based IDS detect unknown attacks or to increase precision of an anomaly-based IDS. The combination of the approaches is a big challenge because of the difference in the core ideas.

1.4 Justification, Motivation and Benefits

The IDSs work with audit data, which are, actually, a sequence of events ordered in time, and these order relationships between events, in addition to various event characteristics, are supposed to contain evidence of intrusions. Ideally, we would represent an event sequence as a tree,

which connects the events, according to their partial order in the sequence, to analyze these event relationships. But such tree structures are large for long sequences of thousands of events such as audit trails, and, therefore, the algorithms for finding such large structures are quite slow.

Episodes are a more efficient way of representing partial order relationships between events. The main difference is that we do not need to reflect all relationships, but only those, which we consider frequent and, hence, important and not accidental. There exist several FED algorithms, which are claimed to be fast and precise. Besides, we can vary the speed and precision by changing correspondent thresholds. Hence, we believe that a FE-based IDS may excel today's IDSs.

We noticed that the FE definition, based on the common thresholds, is not always suitable for intrusion detection. For example, different connections often have different TTL ("time-to-live") value, and some attacks can be slow and other require several events in a short time interval to succeed. By studying patterns, left by attacks in audit trails, we discuss the applicability of frequent episodes to intrusion detection. Based on discovered possibilities, we construct a new hybrid IDS on episodes.

1.5 Research Questions

The main research question is: Is it possible to build an efficient and accurate IDS, based only on FEs? To give an answer, we should analyze episode properties, including their applicability to intrusion detection, and construct an IDS – module by module. We assume that the IDS's input is a sequence of non-simultaneous events with an appropriate set of features. Of course, two or even more events can occur simultaneously, but they are somehow ordered for simplification, and it, obviously, may decrease the system's precision, which mainly depends on the chosen thresholds.

So, the main question can be divided into the following sub-questions:

- What information can we get from FE counting results?
- Are episode-based attack patterns good enough?
- How can an IDS be constructed only on FEs?

1.6 Methodology

Since the idea of applying FEs to intrusion detection is not new, we are interested in deepen compatibility assessment, which can be made by literature study, episode and intrusion analysis, and modeling and building an experimental IDS. Literature study is the most important method here. We capture and analyze available information on frequent episodes and their applicability to intrusion detection.

Building an IDS, usually, begins with modeling, continued by implementation and, finally, testing. First, the system should be split up into several modules. Such modularity is necessary for implementation and testing, since an IDS is a complex system. Then, a data flow model, where every module is a "black box" with input and output of fixed sizes and data types, must be drawn. Each module's functionality is specified and clearly described. A final model of our experimental IDS is analyzed to assure the modules' compatibility and integrity. If the analysis

gives unsatisfactory results, the model must be revised. The main algorithms required for our IDS must be implemented. We have chosen the C# object-oriented programming language for our IDS implementation because it is simple and combines advantages of C++, Delphi and Java such as bounds checking, portability of source code and automatic garbage collection.

For the experiment we require some traffic data. It can be either real or simulated data. Simulated data depends on quality of data generating, while real data should be chosen carefully to be a "representative sample" of reality. Real data also require pre-processing. A pre-processing method can be chosen under the experiment when the first raw data will be captured. Since we study attacks and only real data can provide us with real intrusions, we conduct the experiment on real data. The data is captured by the Wireshark program [4] at Gjøvik University College on one or two computers, which are configured as honeypots and placed outside the firewall.

The experiment is to assure correctness of the theoretical results. To choose appropriate thresholds, we need to implement the algorithms, execute them many times with different thresholds and compare results. We repeat the experiment multiple times with various parameters (thresholds) and, then, analyze effectiveness of the IDS's modules.

1.7 Intrusion Detection Systems

Before we discuss the FEs' applicability to intrusion detection, we introduce modern intrusion detection techniques. We start with definitions and then explain common IDS building principles.

1.7.1 Intrusions

One of the first articles on security monitoring, [5], defines an attack as an action that takes advantage of a threat, meaning by threat a possibility to violate confidenciality, integrity and availability. Further, [6] mentions that an intrusion, generally, is a successful attack, but in the term "intrusion detection system" the intrusion and attack definitions coinside, independently on the result. Probably, the original purpose of the IDSs was successful attack detection, but Denning, who presented the first IDS model in [7], used the intrusion term meaning a violation of security policy.

We prefer to follow the intrusion definition given by [8] because it is precise and close to the frequent episode definition. They define attack as "a malicious or suspicious event detected by the IDSs" and, based on this, define intrusion incident as "a sequence of related attacks within a time frame against a single computer system by an attacker to achieve some goals". By malicious event the authors of [8] mean "an event generated by a single attempt to violate certain security policies, regardless of whether the attempt achieves its goal", while suspicious event is "a non-malicious event [that do not violate the security policy] generated by an attempt that has strong logical connections with the malicious events".

The authors of a hybrid IDS, described in [1], divide abnormal traffic events into intrusions (events that are known to be intrusive) and anomalies (likely/possibly intrusive events) as two different event kinds without any intersection. They define intrusion as an attempt of unauthorized access to a host and anomaly as an unexpected network connection or a single packet, which cannot be defined as normal.

1.7.2 Attack Taxonomy

We look only at network layer attacks, since our work (especially, the experiment) is limited by attacks/intrusions that can be detected by network traffic log analysis. Such attacks are directed to a host and usually exploit TCP/IP vulnerabilities, i.e. its design and implementation flaws. The common attack classification is built on the attack goals: Denial of Service (DoS) (for example, SYN Flood), unauthorized remote access, or Remote-to-User (Dictionary); unauthorized root access (Buffer Overflow) and information gathering, or Probe (Portscan). The Spam problem can also be addressed on the network layer. In Appendix A, we provide some attack descriptions and signature examples, which are related to our work and the experiment.

The attacks can also be classified based on the number of connections involved (single or multiple), attack duration (quick, normal, or slow), and attackers' resource distribution (usual or distributed). An intruder can coordinate several intrusions to achieve a specific goal. An "innocent" Probe attack, for example, is often followed by a harmful attack, which uses the information gathered during the probing. Such multistage intrusions are difficult to fully detect, even afterwards.

1.7.3 IDS Definition and Concepts

[9] defines IDS as a system to monitor audit trails and alert on suspicious events and incidents. More generally, IDSs are software- or hardware-based systems for monitoring and analysis of events in the networks, systems or applications, which are aimed to automatically detect security policy violations – intrusions – in addition to auditability they provide [10]. The earliest automatization attempts were made in the 1950s and the first (anomaly-based) intrusion detection model was presented in 1986 by [7]. Now intrusion detection systems and techniques attract attention of many researchers. Our overview is based on a good IDS systematization from [10]. The article describes the main intrusion detection concepts: monitoring approach, IDS architecture, and analysis strategy,– by referring to the structure of the intrusion detection process, which includes location-dependent information gathering, information analysis, and response.

Information Gathering

From the IDS definition, we can conclude that the process of information gathering is a simple capturing (and logging) of raw event data on a specific location: network, host, or application. Depending on the location, IDSs can be divided into three categories: network-based, host-based, and application-based. Other information sources are outputs from various system management tools like integrity checkers, vulnerability assessors, or other IDSs. [10] claims that these sources can significantly increase quality of IDS results with attention to sensitivity and reliability.

Network monitoring is the most popular information source for IDSs. Network devices with switch ports spanning (uniting all ports on a switch) and Ethernet packet capturing programs, such as Wireshark, provide IDSs with logs of all network packet traffic on the devices. By filtering the traces, the data volume may be reduced, for example by choosing only the packets, addressed to the device (IP-address). Host-based (or system) monitoring logs operating system events, and, analogously, events from applications are collected during application monitoring, which stores the applications' parameters as well. These monitoring layers supplement each other and, when used together, give the most complete picture of activity inside and around the system.

Information Analysis

The information from the monitoring module must be prepared and processed by the analysis module, where the decisions on presence of intrusions will be made. There are two common decision schemes: misuse detection and anomaly detection. They divide all IDSs into two main categories: misuse-based IDSs and anomaly-based IDSs. The first category is characterized by attack signatures – patterns of known intrusions. An attack is detected, if a bit of real activity in form of captured information has matched against an attack signature. Anomaly-based IDSs are built on the presumption that legal activity produces patterns native only to normal behavior. Such systems generate patterns from the input data and compare them to normal activity patterns captured in advance from attack-free sequences of events. Any deviation from normal behavior is supposed to be result or a part of an intrusion. We look closer at these approaches later in the thesis.

To take advantages of both approaches, IDS architects suggested combining misuse and anomaly detection techniques in so-called hybrid IDSs. In a hybrid IDS, the misuse module detects known intrusions and protects the anomaly module against masked attacks, while the anomaly module handles new attacks. [11] claimed that signature-based misuse concept is rarely combined with anomaly-based IDSs in existing products. But we should add here that the architecture of the commercial IDSs is kept secret, and it is likely that such combination is already widely used.

By the monitoring frequency, we can classify IDSs in the following way: real-time and interval-based (also called offline). In a real-time system, information is analyzed almost simultaneously to capturing. It gives the user possibility to react in time, stop attacks, or eliminate negative consequences. Interval-based monitoring is used in case of limited system resources. Audit data are sent to the analysis module in blocks, corresponding to a given time interval. We can notice that, if there are no block overlap and intrusion status, most of the complex and slow attacks can be missed.

Response Module

The response module defines the system's reaction on the detected intrusions. There are two kinds of IDS responses: passive (reports, logs, alerts) and active (attack interruption). A passive response module informs the IDS user or another system on the detection results, but does not react on intrusions. [10] mentions that, in some IDSs, response may be given in form of alarms – immediate messages – or reports – complex documents on intrusive/unusual events for a chosen period of time.

Active response can be divided into man-in-the-loop and automated responses. The first response type presupposes human involvement, while the second handles intrusions independently. Automated response modules work best with attacks, generated automatically. They may have various goals: gaining information on intrusions, attacker recognition or environment correction. IDSs run often in their own environment, separated from the target of monitoring. A stand-alone automated response changes the IDS behavior according to specific rules, built into the IDS, for example, changes a threshold's value to achieve more sensitive detection. Another kind of automated response is integrated response, which defends the target by changing sys-

tem/network configuration with integrated measures.

Feature Extraction

Lee [12] defines an additional IDS process, which describes intrusion detection at an abstract level,– feature extraction. IDSs collect intrusion evidence from the audit data. Expert knowledge on the evidence is the kernel of feature extraction. According to [12], the experts cannot be fully substituted by IDSs, but their work can become more efficient. To automate the feature extraction process, we can apply a data mining technique. Data mining is a way of gathering of useful information from a large data amount. Cluster analysis, association rules and frequent episode rules are data mining techniques that were applied to feature extraction.

Lee pointed out that the main challenge of data mining in intrusion detection is feature extraction and construction, not classification algorithms. Examples of features are service and flags. A popular way of feature discovery and testing is KDD CUP (Network Intrusion) – a tcpdump of the DARPA dataset [13]. This simulated dataset contains a wide spectrum of attacks. An outstanding feature research based on frequent episodes was presented in [14].

Simple features are often complemented by constructed features. The constructed features are usually built on window-based statistics of other features, like count or percent of connections to the same destination port with a specific flag. To detect intrusions hidden in rare packets (which do not produce frequent patterns), Lee noticed the necessity of free text mining, because even constructed features are useless in some cases.

Misuse Approach

The main idea of the misuse approach is collecting of intrusion signatures and checking if the audit data contain such signatures. Intrusion signature is a data pattern, which is “native” only to the corresponding attack and can be used to detect the attack from various event sequences. So, the attack signatures must be minimal and complete at the same time. They can be atomic (inside one event) or composite (characteristics of an event sequence). Good attack signatures are created by security experts with attention to the current security policy. Known signatures of intrusive behavior are usually stored in a database.

Under intrusion detection, a pattern matching tool searches attack patterns (from the database) in the real data. Every match means intrusion and triggers an alert in pre-defined cases. The main challenge of matching algorithms is resource efficiency on large amounts of data. Matching must be accurate, but it should detect attacks even with some variations. Obviously, no misuse-based IDS is able to detect new intrusions. Besides, the signature database needs regular updates to provide effective intrusion detection. SNORT [15] is an example of a misuse-based IDS.

Anomaly Approach

The attackers improve known attacks and invent new, and thus they may easily overtake any misuse-based IDS. The anomaly approach is focused on normal behavior patterns, which the IDS’ owner has control over. When a new kind of activity becomes acceptable (does not contradict to security policy), the normal behavior pattern database must be updated; otherwise the activity will be classified as an intrusion and will result in false positives. Attacks and deviations from normal activity are anomaly by definition and deserve the IDS user’s attention. So, the main problem of the anomaly-based IDSs (ADSs) is a high rate of false positives – intrusion alarms

generated by legitimate activity – and false negatives – no alarm on intrusion. Simultaneous minimizing of both rates is difficult, and the task is a compromise-based balancing between the rates using tuning mechanisms of the ADSs.

Patterns of normal behavior are more complex than attack signatures, and advanced recognition and matching techniques are often applied in order to build an effective and accurate ADS. The most common techniques are classified in [10] in the following way: statistical analysis, quantitative analysis, learning techniques, advanced techniques. Statistical analysis is the first anomaly detection technique. It was built on statistics of users, processes and resources for a period of time or a session. Quantitative analysis uses numerical characteristics of events and sequences in predefined time intervals. Decisions on anomaly are usually taken, based on triggers and thresholds, which can be simple (constant) or heuristic (adaptive). Learning techniques apply artificial intelligence models, like neural networks, to train ADSs on normal activity trails and distinguish between abnormal and normal behavior. Several other advanced anomaly detection techniques, for example, data mining and autonomous agents, have also been proposed recently [10].

[16] points out that the basic anomaly detection premise on intrusiveness of anomalies made by Denning in 1987 [7] is not actual for the modern network environment; therefore, the ADSs generate many false positives and are difficult to tune. The authors discuss common assumptions of the anomaly approach with the aim of finding the wrong ones, which cause the ADSs' drawbacks. The problem domain assumptions, according to [16], are the following: attacks are rare and anomalous, and anomalies are malicious. The training data assumptions are availability of attack-free data, representativeness of simulated data, and static nature of the network traffic. They recommended combining the anomaly approach with the signature-based misuse approach to reduce the influence of the imprecise assumptions on the IDSs' performance.

1.7.4 Challenges of Intrusion Detection

Building of an IDS is a complex process, and there are many limitations and barriers on the way to effective intrusion detection. The main challenge of every IDS is threshold selection, or tuning. Even in the misuse approach, alert decisions are based on matching results and probability of intrusion that the applied signature corresponds to. The quality of configuration and tuning of an IDS to a specific security policy has a huge influence on the IDS's performance.

Anderson [9] mentions that the choice of the thresholds, regulating an IDS's rates, is critical, but a patient attacker can go round the thresholds, for example, by slowing the attack. The author explains the "poor performance" of today's IDSs by the following facts:

- The networks are noisy: they contain random and eventual packets;
- The attack rate is usually low (and the noise level is significant in comparison to it);
- Many attack signatures depend on software and often are specific for each software version;
- Encrypted traffic cannot be properly checked;
- Not all attacks can be detected on the packet layer and eliminated by packet filtering;
- Parallel and distributed attacks on global level are difficult to detect.

The IDS researchers address the problems and propose solutions. The task is to combine the proposals to get an IDS, which can solve all the problems at the same time. For example, [1] states that signature matching is usually aimed at detection of only single-connection attacks, while anomaly-based systems work well on multiple-connection attacks; and, hence, the authors suggest an approach combination in form of a hybrid IDS.

The more sophisticated the attacker is, the more challenging the balancing between efficiency and accuracy becomes. An example of advanced intrusions is a complex multistage intrusion. [8] proposes to discover logical attack relations, using attack results, to improve intrusion detection efficiency, obtain a complete picture of complex intrusion incidents, and reduce the number of alerts. Hence, the order relationships are important for multistage intrusions. The problem is closely related to our work and we refer to it in the next chapters.

1.8 Frequent Episodes

The theory on frequent episodes is relatively new. Here we provide an overview of it. Only deep understanding of the episode structure and episode discovery techniques can allow us to analyze the episode compatibility to intrusion detection.

1.8.1 FE Definition

A frequent episode (FE) is a data structure, which was introduced by Mannila et al. in 1995 as a "collection of events occurring frequently close to each other" [3]. An episode was also defined generally as a partially ordered set of events. The authors improved their theory in [17] and [18] by introducing a more general and formal episode definition based on predicates. We limit us by the first episode definition. It is built on smaller terms, and we start with the definition of an event. An event is defined in form of $(A \mathtt{fromE}, \mathtt{integer} t)$, where A is an event type from a set of event types E and t is the time of occurrence of the event. Generally, an event type may have several attributes, but we use this simplified form without loss of generality.

So, we can introduce an event sequence s on E with notion $(S, \mathtt{integer} T_s, \mathtt{integer} T_e)$, where $S = \{(A_1, t_1), (A_2, t_2), ..., (A_n, t_n)\}$ is an ordered sequence of events: A_i from E for all $i = 1, ..., n$, and $t_i \leq t_{i+1}$ for all $i = 1, ..., n-1$; T_s – starting time, T_e – ending time, $T_s \leq t_i < T_e$ for all $i = 1, ..., n$. The FE discovery (FED) algorithms, usually, take such sequences as an input. A pre-processed audit trail simply fits into this event sequence definition, and therefore, we can apply a FED algorithm to it (we come back to the algorithms later).

Then, an episode is a set of events, occurring close enough in time. Thus, we need to define the time window width for our episode. According to [18], a window is "a slice of an event sequence" defined above and can be noted as $(w, \mathtt{integer} t_s, \mathtt{integer} t_e)$ where $t_s < T_e$, $t_e > T_s$, and $w = \{(A_p, t_p), ..., (A_q, t_q)\}$ where (A_j, t_j) from S and $t_s \leq t_j < t_e$ for $j = p, ..., q$, where $p < q$.

Now we can denote the width of the window w as $\mathtt{width}(w) = t_e - t_s$ and introduce the notion $W(S, \mathtt{integer} win)$ for a set of all windows w on S, such as $\mathtt{width}(w) = win$. From here, we can go back to the event sequence and redefine it as "a sequence of partially overlapping windows" [18] to make the definitions clearer. The total number of such windows is, obviously, $(T_e - T_s + win - 1)$. The authors noticed that the first and the last windows contain maximum

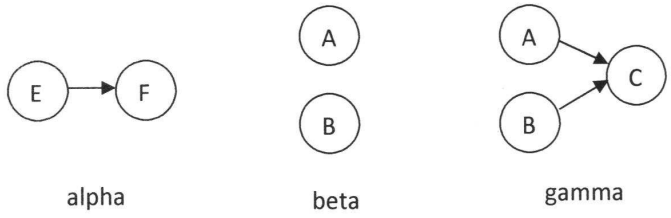

Figure 1: Episode Types

one event each. We can add that the first window contains (A_1, t_1) if $t_1 = T_s$ or empty otherwise. Analogously, the last window contains only (A_n, t_n) if $t_n = T_e - 1$, or empty otherwise.

Hence, the authors conclude that an event at the beginning or the end of the sequence is "observed in equally many windows" in comparison to an event in the middle of the sequence. So, we avoid deformations and get equitable results when using this efficient window counting method. That is why, to decide on frequency of an episode, the authors count time windows, containing the episode, instead of episode instances. For a specified N - number of time windows, we say that an episode is frequent, if it occurs in at least N windows.

Formally, the frequency of an episode α in s is $fr(\alpha, s, win) = |w$ from $W(s, win)$: w contains $\alpha| \ / \ |W(s, win)|$. Alpha is frequent in s with respect to win if $fr(\alpha, s, win) \geq min\_fr$, where min_fr is the frequency threshold. The authors introduced a notion $F(s, win, min\_fr)$ for a collection of all frequent episodes in s with respect to win and min_fr.

1.8.2 Episode Properties

Now we return from the abstract definitions to concrete episode examples and practical properties. The authors gave three examples of episodes in their first article. They are shown in Figure 1. They called episode $alpha$ "serial" and defined it as events, occurring in the sequence in a predefined order, not necessarily following each other. It means that the order is important and there can be other events in between. Another episode, $beta$, was called "parallel", meaning that "no constraints on the relative order ... are given". The order does not matter here. Episode $gamma$ was not classified, but the authors noticed that it is a combination of the two episode types. For example, it can be described as a serial episode (AB,C), where AB is a parallel episode (A,B). Thus, a complex episode, such as $gamma$, can be presented as a recursive combination of parallel and serial episodes.

The authors mentioned two episode relationships: "sub" ($\preceq$) and "super"($\succeq$) [18], e.g. episode $beta$ is a subepisode of $gamma$, while $gamma$ is a superepisode of $beta$. Obviously, an episode is at least as frequent as its superepisode. Formally the property was described in the following form:

Property1. If an episode $alpha$ is frequent in an event sequence s; then all subepisodes $beta \preceq alpha$ are frequent.

Let us look at the example in Figure 2. There are three event types in the example: A, B, and C. Only three windows are shown in the figure. Each window contains a set of events occurred

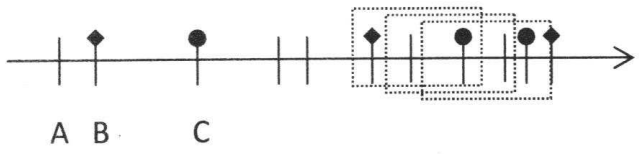

Figure 2: Counting Example

in a specific order. The first set is BAC, the second: ACA, the third: CAC. The last window does not contain event B, by the window definition. If we set the frequency threshold at 2 and want to list all frequent episodes from the following set of serial episodes of order 2: AA, CC, BA, AC, CA, and BC, the number of windows, containing episodes AA, CC, BA, and BC, is 1, episode CA: 2, and episode AC: 3. Hence, we have two frequent episodes: CA and AC. Episode B is a subepisode of BA. Notice that, if B is not frequent, BA cannot be frequent either. This property is useful in episode discovery.

Other episode properties, which were not mentioned by the authors, but which are useful for intrusion detection, are discussed later.

1.8.3 Episode Constrains

[18] mentions that it is possible to generate rules of episodes and notices that "episode rules show the connections between events more clearly than frequent episodes alone". Later, [1] points out that the difference between association rules and FERs is that "an association rule is aimed at finding interesting interrelationships inside a single connection record" while "a FER describes the interrelationship among multiple connection records".

By noting a FER in form of $beta \Rightarrow gamma$ (follows), where $beta \preceq gamma$ (subepisode), [18] calculates the episode rule confidence (conditional probability of $gamma$ occurrence given beta occurred) as a fraction $fr(gamma, s, win)/fr(beta, s, win)$, where w – the window size, in which the rule is detected; s – the sequence. [1] operates with active FERs: a FER is active if $s \succeq s_0$ and $c \succeq c_0$, where s_0 is the minimum support threshold and c_0 is the minimum confidence level chosen by experts. For a FER of 4 events: L1, L2, R1, and R2 – and 2 episodes: (L1,L2) and (L1,L2,R1,R2) – in form of (L1, L2 $\rightarrow$ R1, R2), the authors define the support and confidence in the following way: support base s – probability of the episodes from the left-hand side of the FER to occur (s=P(L1, L2)); confidence level c – probability of the whole sequence to occur if the episodes from the left-hand side have occurred (c=P(L1, L2, R1, R2)/s).

The episode, as a data structure, was analyzed, changed, and improved by several researchers. Even a language for specifying the episodes was developed in [19]. The most of works related to frequent episodes are based on restricted episode definitions or extending the approach to episode sets. For example, [20] defined a frequent closed set of episodes and built "representative episodal association rules" on the definition.

Some critical remarks on frequent episodes has been made. Wojciechowski [21] noticed that many real events could not be described by only one attribute and mining of episodes of event templates (predefined sets of event attributes) did not allow discovering a little bit different episodes. Casas-Garriga [22] noticed that the user-defined window limited interesting episodes discovery. They proposed "unbounded episodes". [23] introduced a "generalized" episode notion by "incorporating event duration constraints explicitly into the pattern's definition".

Bettini et al. [24] focused on event structures in discovery of temporal patterns, which are initial settings, such as event characteristics with temporal constraints defined by user. By "temporal patterns" the authors meant "instantiations of the variables in the structure that appear frequently in the time sequence". So, if an event is represented by a set of variables, the temporal pattern definition is similar to the definition of frequent episode. A method of "finding event sequences that match event structures" was built, based on a "timed automata with granularities" and introduced in the paper. The authors proposed a procedure that "exploits the properties of granularities and event structures" in temporal constraints.

A new definition of frequency based on non-overlapped occurrences appears in [25]. Two new episode types were also introduced: sectorial episode, where a set of events is followed by an event [26], and diamond episode, where a set of events follows an event and is followed by another one [27].

1.8.4 FE Discovery

Mannila et al. consider the following problem [18]: "Given a class of episodes, an input sequence of events, a window width, and a frequency threshold, find all episodes of the class that occur frequently enough". Later, the problem with some modifications was addressed by other researchers. Here we present a short overview of episode discovery algorithms.

The first algorithm, considering the problem, WINEPI, was presented in [3]. It is based on a sliding window approach: incremental checks of episode occurrences in a window and counting windows, containing the episode. This approach, together with an alternative minimal occurrences approach MINEPI, was described in detail in [17] and [18]. The main difference is in exact counting of an episode's occurrences instead of windows, containing the episode. According to the authors, MINEPI is more efficient for episode rules discovery on time intervals twice as long as the window width. The main restriction of all Mannila's algorithms is excluding simultaneous events. This fact is an important limitation with attention to possible application areas.

As we mentioned above, the episode can be defined as a set of partially ordered events. Mannila and Meek [28] presented a method for finding partial orders (limited only by series-parallel orders) from large collections of event sequences, which, by definition, can be used for frequent episodes discovery. The approach is based on representing of event sequences as acyclic directed graphs reflecting the appropriate partial orders.

Several modifications were made on the frequent discovery algorithms. [19] proposed a new data structure – sequential pattern tree – to improve efficiency of the frequent episode discovery algorithms. [21] presented extensions to some algorithms for discovery of frequent episodes, consisting of simple events, to handle episodes of complex events. Levelwise episode mining was

proposed in [29].

Mielikainen [30] considered only frequent serial episodes discovery problem. Since a frequent serial episode is only a subset of events with frequency above a given threshold, the problem is similar to "iceberg" [31] and "hot list" [32] queries. Thus, there are some algorithms for frequent serial episode discovery, such as the algorithms proposed in [33] and [34], which have had no reference to frequent episodes by now.

The simultaneous events problem was addressed by Huang and Chang([35] and [36]). They defined a complex sequence as a sequence containing simultaneous events and proposed a modification for the MINEPI algorithm and a new algorithm for discovery of frequent episodes from complex event sequences. Laxman et al. [25] present fast algorithms for counting frequent episodes under non-overlapping occurrences and suggest including the Hidden Markov Model learning into discovery models. [23] proposed a discovery algorithm for generalized episodes.

[19] introduced a new data structure-sequential pattern tree – to improve efficiency of the frequent episodes discovery algorithms. [21] presented extensions to some algorithms for discovery of frequent episodes, consisting of simple events, to handle episodes of complex events. [22] proposed an algorithm for unbounded episodes mining. [26] and [27] proposed algorithms for discovery of sectorial and diamond episodes. The algorithms are useful in bacterial culture data, as it was noticed in the articles, and, thus, they may be useful for intrusion detection, since these areas have a lot in common.

Many algorithms for frequent episode discovery have been proposed, and Cheng et al. [37] pointed at the lack of critical analysis of existing algorithms for frequent itemset discovery and provided some preliminary results of comparing of the algorithms.

Our experiment requires an implementation of a FED algorithm. The total order relations between events are very important in our case, and, hence, we decided to discover only serial episodes and avoid simultaneous events. We are not interested in the algorithm efficiency at this stage. So, we have chosen an appropriate set of Mannila's WINEPI algorithms for serial episode discovery from [18]: algorithms number 2, 3, and 5. A clear description of the algorithms and their structure from [18] is available in Appendix B. In the algorithms, the episodes are represented by arrays of event types (in their original order). Collections of episodes are also arrays sorted lexicographically (alphabetically).

2 Previous Work

2.1 FE Applicability to Intrusion Detection

Now we focus on intrusion detection and present a wide overview on the different contexts in that the frequent episodes theory have been applied, as well as the FE improvements suggested by various researchers.

2.1.1 Feature Extraction and Pattern Mining

The idea of using frequent episodes in intrusion detection belongs to Lee and Stolfoc [38]. Their work was focused on data mining – an automated process of extracting specific structures from a large amount of data. The authors defined intrusion detection as a "data analysis process" and tried to maximally automatize IDS building by using association rules and frequent episodes. The anomaly IDS approach is based on finding patterns of normal activity, while the misuse approach supposes constructing intrusion patterns; and both end up with matching some audit data against the patterns. The paper covers both approaches since it is focused on pattern discovery.

The authors mentions that, to collect accurate patterns, it is necessary to understand the structure of audit data, collect enough training data, and make right choice of features. In their approach, frequent episodes and FERs were used to discover relationships between events. They proposed to extract normal patterns in form of episodes and rules by applying the original FED algorithm from [3] (together with a rule mining technique) to different audit data pieces. The process flow of this offline framework called MADAM ID (for Mining Audit Data for Automated Models for Intrusion Detection) is shown in Figure 3.

Some audit data are captured from multiple runs of sendmail and tcpdump programs with different settings in [38]. After that, the set is "pruned": all rare rules are deleted. The "profile rule set" is used for feature selection and generation of intrusion patterns and normal profiles. To make the normal profile matching flexible, the authors proposed "scoring functions" built on support and confidence values for estimating the deviation degree. The authors claimed that such automatization makes intrusion detection more accurate and efficient.

The authors made a number of improvements in some later works. For example, the feature extraction technique was applied in [14] to choose the right time window size and, thus, increase the IDS accuracy. In [39] they discuss the use of FER discovery for mining attack signatures from intrusion data. Lee [29] notices that only DoS and Probing attacks produce "intrusion only" frequent episodes. To get such frequent episodes for a slow Probing attack, Lee suggests sorting connections by destination host and applying the connection number window of 100 connections instead of the 2 seconds long time window. Lee calls the packets produced by Remote-to-User and unauthorized root access attacks "unstructured" and notices that an attack packet is, usually, related to a single connection. The attacks can be discovered by finding "frequent" anomalies in packet content (e.g. connection flag), using domain knowledge.

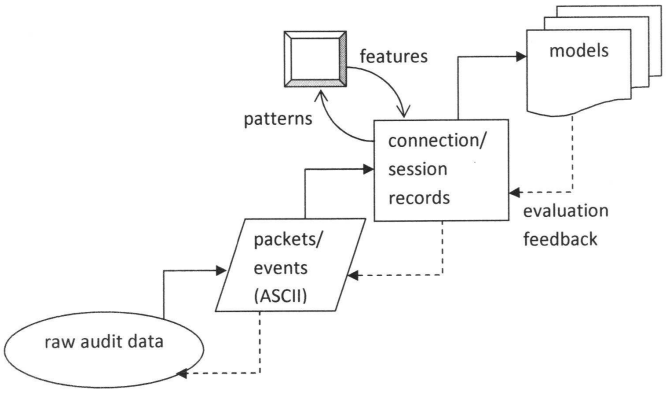

Figure 3: MADAM Intrusion Detection Framework by Lee et al.

2.1.2 Episode Modifications

Domain Knowledge

Frequent episodes were originally proposed to be applied to alarm sequences in telecom networks. The application's approach was also described in detail by Hätönen et al. [40] and [41]. Being focused on telecom event monitoring, some of the articles' authors extended the approach by adding a new dimension. The basic frequent episode discovery algorithms use one-dimensional distribution of events – in time. But domains, where the events occur, play an important role for intrusion detection, according to [42]. For example, presence of a specific frequent episode in an audit event log may occasionally mean intrusion, but the events' topology will indicate it as a "false alarm" since the events are actually independent. The authors give the following examples of commonly available knowledge: domain hierarchy, taxonomies, and "definitions of control hierarchy and material flows" between domain objects.

They propose using the topological information "as a basis for domain specific distance measures" to prune out irrelevant rules and accidental events. The authors claim that their "distance constraints" approach is general and can be applied both to finding related events and pattern mining. The paper describes various aspects of using domain knowledge in frequent episode discovery in detail. The authors conclude that this kind of knowledge is useful for pruning "unnatural and impossible combinations of events away from the result set" or for efficient "natural" event partitioning (which can, in its turn, reduce the algorithm's execution time).

Prior to the work described above, [39] suggested axis and reference attributes to involve domain information into the episode discovery process and, thus, reduce the rules set. The episode topology problem was also addressed by Devitt et al. [43]. The paper pointed out that non-topological data mining can extract "not plausible" sequences "from the point of view of network topology constraints". The authors described their "Topographical Proximity (TP) approach", which was based on the idea of using "topographical information embedded in alarm data".

Time Marks

Mooney and Roddick [44] notice that "while many sequences are associated with absolute time values, most sequence mining routines treat time in a relative sense". Of course, the real timestamp is an important characteristic of the connection events, especially for decisions on the "normality" in anomaly detection. For example, http-packets in the evening are normal activity while ftp-packets may not be. The authors describe the opportunities of the new approach: it "enables sequences to be examined not only in respect of the order and occurrence of tokens but also in terms of pace". They also proposed an algorithm, called INTEM with timing mark support.

Frequent episodes, according to Zaki([45] and [46]), can be considered as a particular case of frequent sequences, because sequences are frequent "across many input-sequences" and episodes – within one sequence. The paper points at the lack of attention on mining sequences with "syntactic constraints" such as time interval or time frame.

Fuzzy Frequent Episodes

To make the FE-based intrusion detection methods more flexible, Luo and Bridges [47] propose integrating fuzzy logic into the episode frequency definition. They describe the fuzzy frequency concept and propose a new fuzzy FED algorithm and modifications of some known FED algorithms for offline intrusion detection. Later, in [48], the authors present some modifications to the algorithms for real-time intrusion detection.

A similar approach is described by Guan et al. [49], and the main difference is that some fuzzy data mining methods (not frequent episodes discovery algorithms) are modified to be applicable to discovering of fuzzy frequent episodes for offline anomaly detection. Their experimental results on fuzzy frequent episodes indicate that this approach "can provide effective approximate anomaly detection".

2.1.3 Alarm Investigations

As we mentioned before, the idea on frequent episodes came to Mannila et al. when they were working on the alarm investigation problem in telecommunication networks. The problem is similar to the multistage intrusion challenge: discover alarm relationships to analyze alarm "roots", filter out redundant alarms, and even predict faults in the networks.

Interesting results on the FE applicability to IDS alarm investigations were presented in [50]. Frequent episodes and FER discovery techniques for serial and parallel episodes were used in mining of alarms produced by some experimental IDSs. Several "interesting" patterns were discovered, including episodes, representing attack tools, attack warning episode rules in form of attack indicator–>massive attack, and episodes, containing only legitimate activity. The authors pointed on some drawbacks of frequent episodes in alarm investigations: insignificant automatization improvement and large amounts of "irrelevant or redundant" patterns. Thus, they decided to exclude frequent episodes from their framework. But they also added that FED techniques are "very homogeneous and repetitive" and may simplify intrusion detection.

2.2 A Hybrid IDS on FERs

In 2004 Qin and Hwang [11] proposed a new anomaly-based intrusion detection system on frequent episode rules (FER), which can be used for all TCP, UDP, and ICMP connections. The

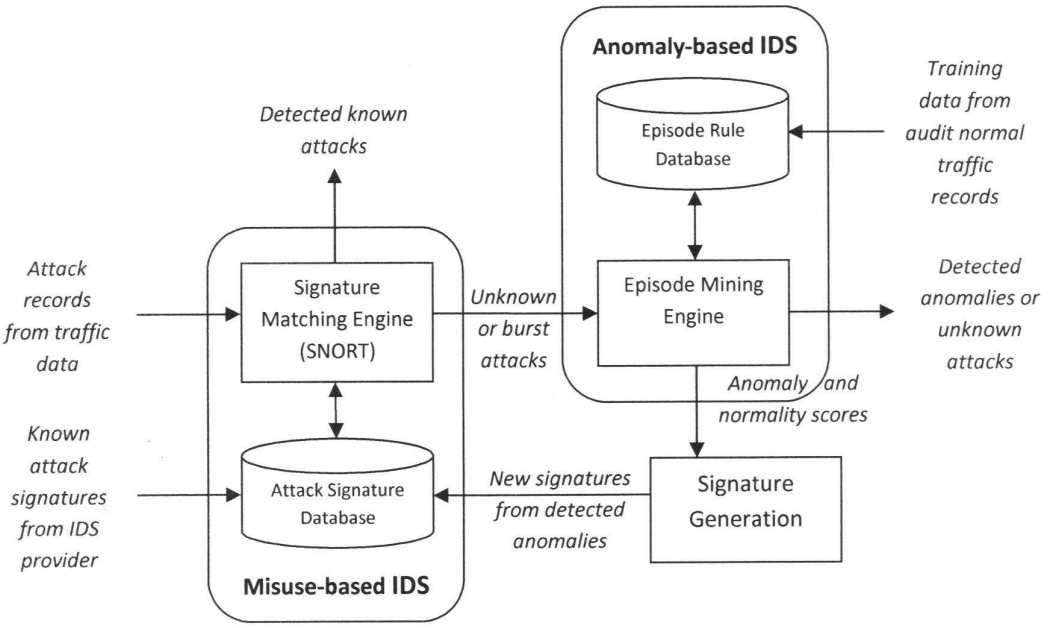

Figure 4: The Hybrid IDS on FERs from [1]

point was to generate effective FERs – rules that are used frequently for intrusion detection – to simplify and accelerate the IDS's work. The authors used FERs instead of frequent episodes to capture relationships between episodes as well. Later, they improved the approach and suggested a hybrid IDS based on FERs (Figure 4) in [1], which we describe in detail.

The authors pointed at two main drawbacks of the FER approach and proposed solutions:

- "Many attacks are triggered by a single connection and may not generate anomalous FERs". They solved this problem by paying attention to rare attributes of single connections.
- "A single attack may last for a long period of time". To be able to detect such attacks, they used connection sequence number in a trail of packets directed to the same destination instead of timestamps.

Building of the hybrid IDS was based on two assumptions: "Frequent episodes are mostly resulted from normal users" and "A rare episode is likely caused by intruders". The authors proposed a "weighted signature generation scheme" to capture both attack signatures and normal behavior patterns.

2.2.1 Feature Selection

To pre-process the raw audit data, a feature set must be chosen. In order to detect new kinds of attacks, the system should collect various packet information for each connection, including rare felts (because the attacker can exploit them). An anomaly-based IDS needs commonly a lot of

Feature	Description
Connection Features	
timestamp	Time when the connection begins
duration	Duration of the connection in seconds
ip_proto	IP protocol type
src_ip	Source IP address
dst_ip	Destination IP address
service	Network service on the destination, e.g. http, ftp
icmp_type	ICMP message type
src_types	Bytes sent by the source
dst_types	Bytes sent by the destination
flags	SF, S0, REJ
Temporal statictical features	
src_count	Number of connections from the same source
dst_count	Number of connections to the same destination
service_count	Number of connections for the same service
syn_dst%	% of connections with the same feature and SYN errors
service_dst%	% of connections per service to the same destination
syn_service%	% of connections with SYN error to the same port
ave_duration	Average duration of connections for the same service
ave_src_bytes	Average bytes sent by the source
ave_dst_bytes	Average bytes sent by the destination

Table 1: Connection Features and Temporal Statistics in HIDS Construction [2]

resources (time and memory). The authors pointed out that the choice of event features is important since it influence the ADS' performance. The following features were used: connection features (timestamp, dst_ip, src_ip, flags,...) and temporal statistics (for connections with the same reference features, e.g. number of connections from the same source, procent of connections per service to the same destination, average duration of connections for the same service). Table 1 provide a comlete list of the features.

The authors stated that the use of source or destination host as a reference attribute would limit the influence of background traffic on the ADS's accuracy [2]. According to [1], since FERs do not capture domain knowledge, many FERs are "ineffective or useless". According to the paper, the features are "platform and language independent". The authors also mentioned that the influence of suspicious packets in the training data on the detection rate is low if the packets do not contain attacks.

2.2.2 Detection Process

The Hybrid IDS consists of the following elements (Figure 5): feature extraction module, episode rule mining engine, anomaly detection engine, alarm generation module and normal profile database. Pre-processed audit data (connection records from audited internet traffic) are first investigated by the signature matching engine, which works with the SNORT database. On the output we get records that contain normal activities and unknown or burst attacks. Packets indicated by SNORT as intrusive are excluded from the traffic before input to the anomaly-based

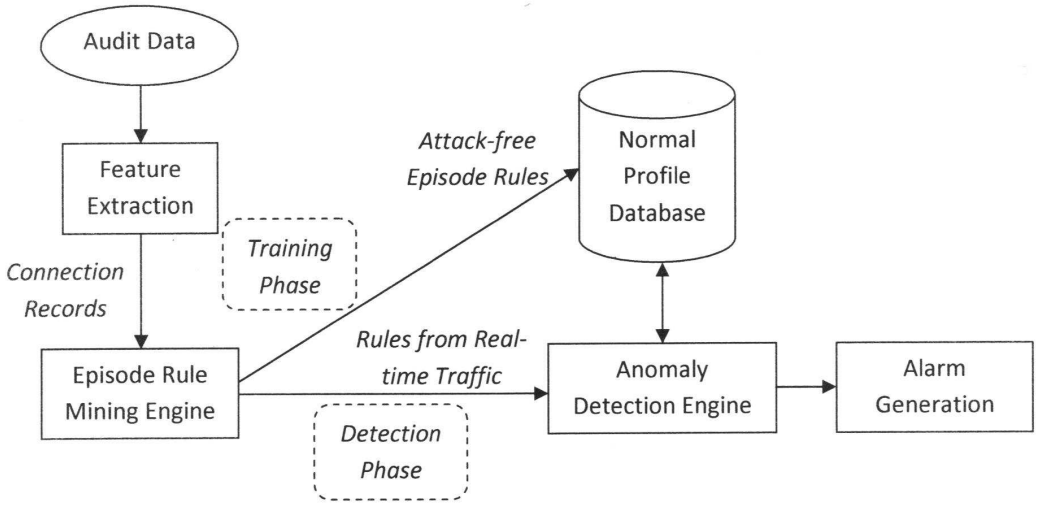

Figure 5: Data Mining Scheme for the Hybrid IDS [1]

IDS. The packet-level alerts are generated at the same time.

The anomaly detection engine is connected to the normal profile database, which is built and continuously updated on frequent episodes, and works on an episode rule mining algorithm. At the training phase, the database without attacks is generated. At the detection phase, each frequent episode rule from real traffic is matched against rules from the normal profile database. To minimize redundancy of the episode rules, the authors suggested a rule pruning technique based on the following principles: keep the FER left-hand side as short as possible (apply transposition from left to right when applicable); eliminate every FER that can be implied by another FER (including transitive FERs); one FER per frequent episode; and avoid complex FERs.

As we mentioned before, [1] discussed the problem on threshold choosing for FER support values. A high support threshold leads to missing rare anomaly-indicating rules because infrequent patterns are ignored. For a low support threshold, a lot of redundant rules are discovered and, because of that, an effective rule pruning (redundant rule elimination) technique is needed. So, based on Lee's levelwise algorithm [38], the authors suggested a base-support mining method that allows capturing both frequent and rare episodes. The authors also proposed threshold differentiation for common and uncommon services: an uncommon but normal service may produce a FER with low support value, while a higher support threshold is appropriate for common services.

The base-support algorithm calculates minimal occurrences and compares axis attributes:

$$\mathrm{Basesupportfraction} = \mathrm{support}(X\mathrm{withallattributes})/\mathrm{support}(X\mathrm{withonlyaxisattributes})$$

Abnormal connection is alerted in two cases: no match with any normal FER (unknown anomaly) and match, but the occurrence number exceeds threshold (massive anomaly). The massive anomaly threshold was chosen by comparison of maximal and minimal occurrences of

episode rules in the training normal-only data. The authors also mentioned an anomaly confirmation procedure based on temporal statistics and error flag checks to reduce false alarm rate. After matching against the normal profile, every traffic connection is assigned anomaly and normality scores ($AS + NS = 1$).

Then, cluster analysis is applied to group similar abnormal connections and form abnormal behavior patterns. When the anomaly/normality score – a sum of all AS and NS of matching connections – is calculated for each pattern, exclusively good anomaly patterns ("the most specific and discriminative") can be found: those with highest anomaly scores (more anomaly match) and lowest normality scores (few false alarms). The abnormal behavior episodes are used by the signature generation unit to construct additional signatures for the SNORT database. The unit is a link that connects the anomaly module with the signature-based module.

The experimental implementation of the hybrid IDS described in the article excels the SNORT performance by 33% and has only 3% false positive rate, according to the authors. Their experiment was conducted on KDD CUP datasets with background traffic of real data in form of Internet traces without packet payload.

3 The New Hybrid IDS

3.1 The IDS Model

According to the pruning methods used in the HIDS, most of the final FERs are $A \rightarrow B$ or $A \rightarrow (B, C)$, where A, B, C are events (not necessarily different). The main difference between FER $A \rightarrow (B, C)$ and frequent episode (A, B, C) is their additional characteristics. The FER has a confidence value for a given window size, i.e. probability of (A, B, C) if A occurred. The frequent episode has only a frequency value (an integer number) for the same window size. If we divide the whole episode's frequency by the frequency of event A, we go over to the FER idea. If the episode (A, B, B) represent an intrusion, while the episode A is a normal behavior pattern, then the FER $A \rightarrow (B, B)$ with $confidence > 0$ indicates intrusion. Similarly, the episode (A, B, B) with $frequency > 0$ indicates intrusion. If the intrusiveness depends on the confidence level, the frequency should be compared to the frequency of the episode A.

So, we noticed that FERs are more informative than frequent episodes, but the information is useful if the decision on the attack presence is taken based on relative frequency: the frequency of an abnormal episode divided by the frequency of a normal subepisode. We can see no advantages of such related frequency; moreover, the absolute frequency is more useful since it does not deform the reality. For example, if the security policy states that 5 login attempts from one IP to the FTP Server is a Dictionary attack, only absolute frequency allows us comparison with the threshold. We can capture order relationships between episodes by complex episodes and apply event (episode) distribution analysis described later in those cases, where FERs could be useful. So, we try to use only episodes in intrusion detection.

We build a model of a new hybrid IDS. The IDS architecture is presented in Figure 6 and its modules are described below. The architecture is a modification of the HIDS from [1]. The real-time episode discovery, unlike the offline discovery, requires a new algorithm for episode counting, working with a well-organized episode database. We focus on offline intrusion detection, and real-time FED is out of the scope of this thesis.

3.1.1 Data Pre-processing

The main task of the data pre-processing module is to reduce the input event sequence by capturing only necessary information from the events. The capturing criteria are event features, useful for intrusion detection. Feature selection is out of the scope of this thesis, thus we assume that data pre-processing is based on an appropriate feature set. In addition, we propose to improve the module and, by this, increase performance of the system. The improvement is a new link between the data pre-processing module and the decision module. In our IDS, the decision module can modify the feature set and switch between packet number and timestamp to capture all the information required for accurate intrusion detection and complete characteristics of various attacks.

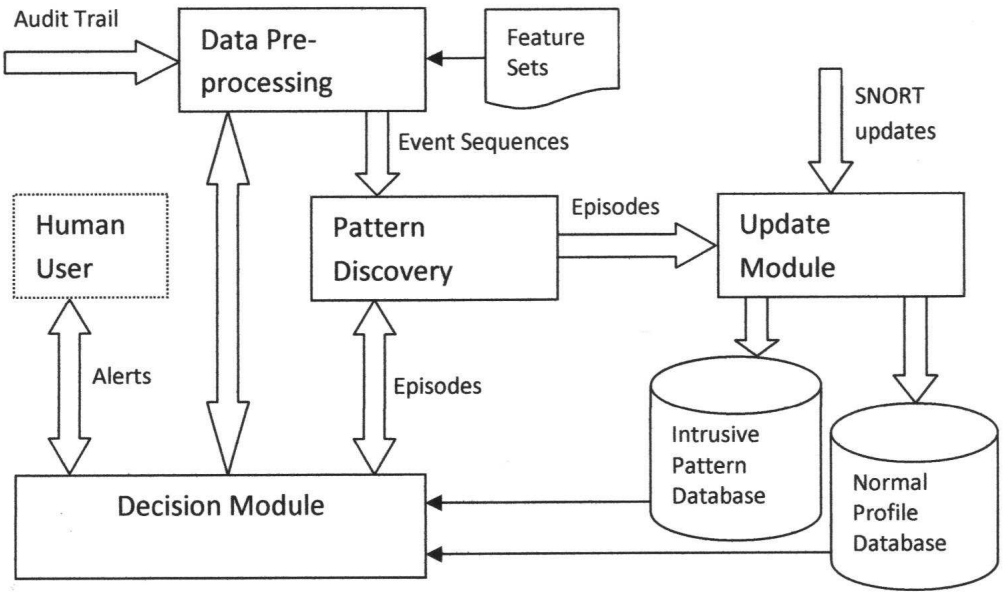

Figure 6: The New Hybrid IDS Model

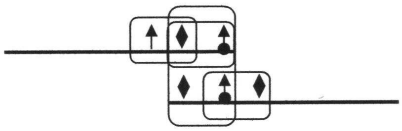

Figure 7: Overlap

Continuous Discovery

Usually, an audit trail consists of many files, and the number of files depends on the period chosen for audit data saving. Under the episode discovery, we, obviously, miss episodes that start at the end of one file and end at the beginning of the next file if we use separated files. We must simply "melt" the files together to get only one packet trail. The final event sequence often is very long. To distribute calculations between several computers, we suggest a modification to the algorithm for right counting the episodes in an audit trail stored in several files.

At first, we should correct the file's trail distribution. We need to include some redundant data to the files to count the episodes on the file shifts. Figure 7 shows examples of windows containing episodes, which may be missed by dividing the sequence without overlap: $T_e[i] = T_s[i+1]$, where i is a file number, T_s – start timestamp, T_e – end timestamp.

An overlap of the window size gives us $T_e[i] = T_s[i+1] + win - 1$, thus the last window

calculated in the file i is $(T_s[i+1]-1; T_s[i+1]+win-1)$; and the first window calculated in the file $i+1$ is $(T_s[i+1]; T_s[i+1]+win)$. So, we miss no episodes despite the sequence separation. To count with several window sizes, the overlap should be of the maximal window size, while the window under episode discovery in file i must start no later than $T_s[i+1]-1 = T_e[i]-max(win)$. Here we should notice that the overlap technique works only on timestamps. If packet number is used instead of timestamp, the episode discovery algorithm must be changed to handle "reset" of packets numeration.

In one of the recent publications on frequent episodes, [51], such sequence separation –"a set of events at each time slot in terms of various intervals (hours, days, weeks, etc.) "– is called a complex sequence. The authors point at wide applicability of such sequences. They modify the MINEPI algorithm to make it able to handle complex sequences and propose a new, more efficient, algorithm EMMA for episode discovery in complex sequences.

3.1.2 Pattern Discovery

The normal behavior patterns and attack signatures are represented by episodes in our IDS. The HIDS is built on the assumption that frequent episodes commonly describe normal behavior, and rare episodes most likely represent intrusions. We do not make such assumption in our IDS since the variety of normal and intrusive behavior is large and the assumption limits the natural variety. The variety can be reflected by threshold differentiation. The HIDS' threshold differentiation for common and rare services may become a weakness of the system. The service defined as rare for a given network and its environment can be common for another network. Moreover, it may become common for the given network too. The following conclusion is obvious: the differentiated thresholds must be updated continuously.

In the definition of a frequent episode, the phrase "occur frequently enough" makes the definition "subjective"; we have to define which frequency is "enough" by ourselves. It includes choosing the time window width and frequency threshold. The decision of the window size plays a crucial role. If the window is too big, we may get too many episodes, and a lot of them will not reflect the reality. If it is too small, we miss the episodes with long intervals between events, and thus we miss long-time intrusions, consisting of intrusive events well-spread in time. Such events can be spread in space as well (a network may be attacked on several IP- addresses or from many IP-addresses). It makes intrusion detection even more difficult. Unfortunately, the episodes do not carry domain information. [29] and [18] suggested adding domain knowledge to the episodes. Analysis of domain-specific episodes, however, would be out of the scope of this thesis.

For effective intrusion detection, we introduce 3 new episode-based pattern discovery techniques, supported by the Winepi counting method. We noticed that the log analysis in intrusion detection is to discover the following:

- attack signatures in form of fixed structures or specific events,
- anomalies and normal behavior patterns represented by similar structures, and
- other useful patterns in form of regularity of activity.

The attack signatures refer to misuse-based intrusion detection. The misuse module of our

IDS is responsible for discovery of the signatures. The similarity patterns can be of two kinds: equal (all features from the feature set coincide) or with variations (one or more features are different). For patterns of the first kind we suggest our own FED modification, which is also useful for attack signature discovery in the misuse module. For other similarity patterns we propose a new rare episode discovery (RED) technique. The regularity patterns can be found by using the event distribution analysis that we constructed based on Winepi's properties.

3.1.3 Frequent and Rare Episode Analysis

An episode, as a data structure, reflects order relationships between events. But, by the definition, an episode may consist of only one event (we call it atomic episode). In this case the terms "episode" and "event" coincide. This fact is very important to us because an intrusive behavior, as well as a normal behavior, can be represented (classified, detected) by only one event. So, an episode, containing only one event, can capture the necessary event features and, therefore, can be used in single attack detection. But most of the behavior can be reflected, using simple episodes (containing only 2 events) or more complex episodes (i.e. episodes of simple episodes). We think that this episode classification fits best to intrusion detection.

Since thousands of events happen every hour in a usual network, the amount of events to use in intrusion detection is enormous; and it may be too difficult and resource demanding to extract all episodes (especially, complex episodes) for such event sets. And we can make the task easier by limiting us with only frequent and rare episodes. [52] mentions that frequent episodes, most likely, represent normal behavior, while rare ones mean intrusion. But attacks like SYN Flood, Dictionary, and Portscan can be characterized by frequently occurred episodes. The researchers point out that an event with an erroneous flag (that means rejected or failed connection), which, usually, is rare, is a first sign of intrusion. This is true in most of the cases, but one or two occasionally rejected connections have nothing to do with intrusion, though it happens rarely. An intrusion, however, may be detected from a connection sequence, containing many failed connections.

The importance of rare episodes is obvious. [1] proposed to update the SNORT rule database by extracting rare episodes and converting them into appropriate SNORT rules. From another point of view, SNORT rules can be converted to connected sets of episodes and specific actions in such a way that every episode triggers an action. The main disadvantage of SNORT is in its rule limitation: a rule represents only one event and can be converted to an atomic episode. So, we can use SNORT rules as a fundament for an episode database for intrusion detection and supply it by simple and complex episodes to deepen traffic analysis. Continuous database updates allow detection of new intrusions. The false positive rate depends on the choice of thresholds.

We propose a new, more flexible and precise threshold scheme. As we can see in Appendix A and deminstrate by our experiment, the attacks produce different event patterns. The same we can say about normal activity. For any fixed window size, the frequency of an episode depends on the episode (pattern) structure. We do not fix the window size, but fix the frequency threshold for each episode structure. Now we take decisions based not on the episode's frequency, but on the window size, for which the episode is frequent (frequency is not lower than the frequency threshold) or rare (frequency is low).

The idea of rare episode discovery is exploiting the information of sudden appearances of episodes. If a given episode (A, B) appears for the first time for $win = w$, i.e. the frequency is $fr(win) = 0$ for all $win < w$, it means that the distance between the events A and B is not shorter than $(w - 3)$ because, by the window definition, we have $(w - 1)$ packets, including A and B. We notice that the episode's frequency $fr(w) > 0$, not necessary equal to 1. But $fr(w) = 1$ if there is only 1 episode (A, B) in the sequence. Such information can be useful to intrusion detection. We check this statement and describe the technique on examples in the next section.

3.1.4 Event Distribution Analysis

Any periodical events are, obviously, generated by a computer and are, most probably, intrusive. Often the task of distinguishing between normal and intrusive behavior can be transformed to the task of distinguishing between computer-generated and human-generated events/packets. We noticed that such distinguishing can be made by episodes, and propose some methods for event distribution analysis and, thus, regularity discovery.

The Winepi episode counting technique can be used to analyze the distribution of events (or episodes) over a given sequence. The idea is to vary the thresholds and compare the results. For example, if an event A has frequency near 1 (the number of windows, containing the event, is about the number of all windows, covering the sequence), but the frequency of the episode (A, A) is close to 0, it means that the events are distributed over the sequence equally with the interval about the window size. This property can be useful especially in detection of slow attacks.

If $Frequency(EpisodeE) = Frequency(Episode(E, E))$, all windows, containing event E, contain at least 2 such events (i.e. episode (E, E)). So, events E are so close to each other that time intervals between them are shorter than the window size. If the number of episodes (E, E, E) is N times lower, it means that only $1/N$ of all windows, containing (E, E), contain (E, E, E). If all events E present one attack, we can conclude that the attack is periodical, i.e. packets are generated with equal time intervals. Based on this observation, we constructed an algorithm for regularity detection:

1. Count all events R;

2. $Window = (T_e - T_s)/R.count$; Count RR (result: RR.count);

3. If RR.count is near 0 then a slow attack is detected; if close to R.count, do next step; else the events R are randomly distributed.

4. If RR.count = R.count, then $window/ = 2$; count RR (result: RR.2count);

5. If RR.2count is near 0, then events are equally distributed but over a half of the sequence; else the events are very close together or are grouped into $(R.count - RR.2count)$ groups.

The following examples show the algorithm's effectiveness for discovery of regularities. For all sequences, the number of events N = 15 and the events are coded as R or 0. R is the event we test on regularity, 0 is any other event. The total number of events R in the sequence $R.count = 5$, and so, $window = 15/5 = 3$.

1. R00R000R000R00R, RR.count (3) = 0; $\Rightarrow$ equally distributed.

2. R00R000R0R0000R, RR.count(3) = 1; $\Rightarrow$ randomly distributed.

3. R0R0000R0R0000R, RR.count(3) = 2; $\Rightarrow$ randomly distributed.

4. RR00000RR00..0R, RR.count(3) = 4, RR.2count(2) = 2; $\Rightarrow$ 2 groups.

5. R0R0R0R0R000..0, RR.count(3) = 4, RR.2count(2) = 0; $\Rightarrow$ equally distributed over a half of the sequence.

6. RRRRR000..0, RR.count(3) = 5, RR.2count(2) = 4; $\Rightarrow$ grouped into 1 group.

7. RRR00RR000..0, RR.count(3) = 5, RR.2count(2) = 3; $\Rightarrow$ grouped into 2 groups.

Another method for regularity discovery, which works on the same episode order, exploits effects of window size variations on frequency of periodical structures. We may expect that the number of windows, containing a given episode, increase equally for each period (if the periods are equal) with increasing of the window size. But the total number of windows with at least 1 such episode is not necessary a sum of the periods' results, but only if the periods do not intersect and the distance between the periods is bigger than the window size. Hence, the observation of increase of the episode number counted by the Winepi method, while incrementing the window size by 1, provides information on the regularity and allows to define the period of the regular episodes. In the Experiment section, we describe the method in detail on an example.

Events may have several features. Under the experiment we melt all features to one for simplification. But it does not hinder to observe the result variations that we can get by changing the feature set. Removing and insertion of some specific features, like source and destination ports, is an effective technique for event distribution analysis. The idea of comparison of numbers of episodes calculated on a complete and reduced feature sets is similar to the axis attributes approach from [14].

Let us note an event as $E(f_1, .., f_k)$, where $f_1, .., f_k$ are the event features. Now, we make an important observation: If an event $E(f_1, f_2)$ has count $N * M$, but the same event with one extra feature $E(f_1, f_2, f_3)$ has count $max(E(f_1, f_2, f_3)) = N$ for all possible f_3, it means that there exist at least M different f_3 that caused the variation between these counts. The observation can also be generalized and used in event distribution analysis.

3.1.5 Databases and Their Updates

The HIDS uses the SNORT database of attack signatures. The system updates the database based on results from anomaly detection engine: new signatures are generated of abnormal behavior patterns. The HIDS exploits the advantages of usual SNORT updates. But the local SNORT database may contain redundant signatures because of updates from different sources. The problem must be addressed. What about reporting the new signatures to SNORT and unifying the update process? The update delay can be large, and the temporary patterns may be stored in the local SNORT database, but cleaned out under the next SNORT update.

Another obvious disadvantage of the signature detection module of the HIDS is blind storing patterns suggested to be intrusive to the signature database. An unknown pattern can represent a new normal behavior and, if so, it should be stored in the normal patterns database, which, as we could notice, has not been updated since the training phase. Both databases need updates. The system requires a mechanism to temporally store alerted patterns and handle alert responses on the correctness of the system's decision from the administrator.

We support the idea on the use of the SNORT attack database for detection of single connection attacks. But we suggest its converting to an episode database. An algorithm for converting

a SNORT signature to an episode, once implemented, can be reused for updating the episode database from new SNORT signature updates. Similar reuse will be achieved for the matching algorithm. In our IDS, the attack signature database has the same structure as the normal pattern database: both are episode-based. Hence, the matching of the patterns discovered from a real traffic data can be made by the same algorithm in the misuse and anomaly modules. So, we achieve unifying of the decision module's input: only episode structures from the misuse and anomaly modules come into the decision module.

An update of the attack signature database after an episode generation for a new attacks (proposed for the HIDS) we supply with a normal pattern database update on an episode of a new normal behavior. The decision on normality or intrusiveness of new episodes can be taken based on responses from the human administrator, who receive the alerts from the decision modules. The alerts, connected to new episodes, should require a response on the intrusiveness of the new patterns. Depending on the response, one of the databases will be updated by the update module, containing the SNORT convertor and a unique full access to the databases. Such isolation of the modification rights, while all other modules have just read access to the databases, eases future securing of the system and serves its integrity.

The update module is also used for the normal profiles database's updates via the link, connecting it to the pattern discovery module. On the training phase, the pattern discovery module generates episodes of normal behavior if an attack-free event sequence is on the input to the data pre-processing module.

3.1.6 The Decision Module

[1] used anomaly and misuse approaches in the HIDS, because, as it mentions, misuse-based IDSs are good at detection of single connection attacks, but not at multiple connection attacks. At the same time, the authors limited theirselves by an invisible assumption that there is no overlap between the sets of single and multiple connection attacks. It resulted in a wrong decision, as we think: the anomaly-based module (ADS) of the HIDS is separated from the SNORT module and the input to the ADS is already filtered traffic. Such separation leads to incomplete information in the ADS. The attacks become more complex, and such separation is a vulnerability that can be exploited. For example, the attacker can make the ADS generate an attack signature that will filter out normal traffic and result in denial of service caused by the HIDS.

The decision module in our system assimilates the misuse, anomaly and decision modules. It makes the attack intrusion more flexible and complete. As we mentioned before, the decision module can modify the feature set, switch between timestamp and packet number. For example, the idea of comparison of episodes with all attributes and only axis attributes episodes (proposed for HIDS) is useful also in our system, but we modify it and use in event distribution analysis. The decision module also has control over the thresholds in the pattern discovery module.

Real-time intrusion detection, using frequent episodes, requires efficient algorithms for discovery, pruning, and matching frequent episodes. Normal behavior, usually, is not unique for different networks. To collect the normal episodes that describe normal behavior in a network, we need packet traces captured in several months to train the system. In addition, user behavior is dynamical and changes slow or fast because of the variety of the user's tasks. Hence,

anomaly detection requires continuous learning in the detection phase. The decision module informs the pattern discovery module on which patterns should be inserted to which database. Positive human responses on new attack patterns will result in new attack signatures, while negative responses – in new normal behavior patterns.

The key elements of the decision module is matching and pruning algorithms for episode patterns. The pruning algorithm can be reused on the system's output to remove redundant alerts. The matching and pruning techniques is a complex topic, and it is out of the scope of this thesis. To propose decision methods, both normal behavior and attack patterns need to be studied. But we focus us on the attacks alone and analyze several episode patterns in the next chapter.

3.2 Model Summary

In this chapter, we discussed the desicions on the existing HIDS architecture, found several weaknesses, and proposed solutions. Our model is an improvement of the HIDS. IDS building is a ballancing on trade-offs, which is complex and needs attention of many researchers. Our episode-based hybrid architecture is the first step on the way to an accurate and efficient IDS based on frequent episodes.

4 Experimental Work

4.1 Experiment Description

To register properties of episodes and analyze their ability to capture information, which can be useful to intrusion detection, the following experiment was completed. At the Gjøvik University College, we prepared an unprotected network connection (not protected by the firewall) and two personal computers: a usual one and one with three network cards. Of software we had Microsoft Visual Studio .NET 2003 to implement the Winepi algorithms, Wireshark for network packet capturing, filtering and converting packet information to a text file with an event sequence, and a honeynet to decoy attackers to penetrate our system.

4.1.1 Honeynet

A honeynet is a network of honeypots. A honeypot is a system configured in a way to attract attackers while capturing all incoming and outgoing traffic. There are no legitimate activity on honeypots and, thus, all its traffic is malicious [53]. There are two types of honeypots: low-interaction with emulated services and high-interaction with real services [54]. So, [54] defined a honeynet as "a high-interaction honeypot designed to capture extensive information on threats". But, first of all, it is an architecture of a controlled network. The main honeynet component is the gateway, which isolates the honeypots (and, thus, may be compared to a wall). No traffic into and from the honeynet can avoid the gateway called HoneyWall [55].

For our purposes, we deploy a honeynet to get the most real picture of intrusions. At first, the HoneyWall was installed on the computer with three network interfaces: the external interface eth0 connected to the Internet, the internal interface eth1 connected to the honeynet, and the administration interface eth2 for remote administration of the gateway. The first two interfaces are bridged to hide the honeynet, that is why they share one IP address. This internal placement of the honeynet can be useful in internal threats detection, according to [55]. For a quick and simple installation, we used HoneyWall CDROM image [56] and its documentation [57]. The installation process was fully automated and did not need any manual interactions.

The next step was to configure the Honeywall. We logged in with default login and password, changed to root and got into the Dialog Menu. We were asked on our IP-address and some other addresses. For our HoneyWall we chose secured Linux installation Minimized Fedora Core 6. To capture only incoming and outgoing honeynet traffic, we tuned Wireshark on the internal interface eth1. All packets were captured with full payload. The intrusive activity was lured by our honeypot – the other computer connected to the gateway via eth1. To make it attractive, we installed Windows XP OS with ServU-FTP, MS-SQL and Apache pluss two Linux OSs (Ubuntu 8.4 with VNC Server and Debian), running on VMware, there. No Honey tool was installed there, because every PC connected to the Honeywall became a honeypot automatically.

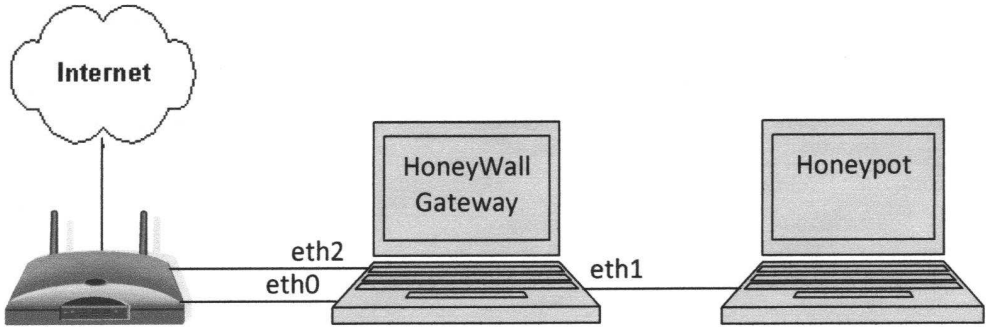

Figure 8: Our Honeynet Architecture

4.1.2 Pre-experiment

We conducted a pre-experiment to check the installation. We connected and powered the honeynet, and started capturing with Wireshark 06.03.2008 at 18:25. There was no activity from our side until 08:20 next day when normal activity (Internet searching, online article reading) began. We applied the following filter to the traffic (only on displaying, not capturing): ip.dst == 128.39.44.100. A lot of FTP requests on User root came from Argentina (200.61.190.145). Some attempts to establish ssh connection were done from Sweden (84.244.4.119). SPIM packets were sent from several different IP addresses from the same town in China. A lot of TCP packets followed by some HTTP and Socks came also from China (202.120.43.208). From a Taiwan IP-address (122.116.112.161, 60.250.66.176) we got similar packets. One more interesting packet came from India (203.94.243.191) – a UDP packet with Source port chmd and Destination port ms-sql-m. Even from United Stated (131.118.64.7, 118.168.177.51) we got some malicious packets. All this traffic assured us that the HoneyWall and Wireshark were working properly.

4.2 Data Pre-processing

Under the experiment, the data capturing was divided in 8 series of about 24 hours each. The series have different levels of activity and contain packet traces of various attacks. The audit trail from Wireshark must be converted to an event sequence, which can be used as an input to our FED program. The process involves filtering, feature selection, and event coding. The filtering reduces data amount. The main challenge is not to filter out packets useful for intrusion detection. To solve the noise problem, we focus on packets directly related to our host (IP=128.39.44.100): ip.host == “128.39.44.100”. It means that the source/destination IP is a reference attribute.

Further, we have two alternatives: ingress (only incoming packets) and ergress (both incoming and sent packets). As we noticed above, most of the attack signatures are ingress-based, and ergress signatures can often be substituted by ingress ones. Besides, focusing on only ingress packets significantly reduces the events amount on the input (and, thus, the number of episodes

on the output). Hence, we chose only ingress traffic and refined the Wireshark filter to ip.dst_host == “128.39.44.100”. Now, most of the packets have the same destination port (our IP), except ICMP packets. So, we do not need to include “destination IP” to the feature set. The only information we ignore here is the destination IP address of the ICMP packets, which is not important for our experiment.

Our data captured on a honeypot with no legal activity contain only intrusions or casual packets. Hence, the use of timestamps instead of packet numbers is preferable. But we can use the packet number instead of time because several intrusions are active at the same time and intrusive packet sequences are, usually, similar to legal activity. We will catch periodical attacks if we use timestamps, but if there are no other attacks in between, packet numbers are good enough. Besides, a reduction of time precision for very close events results in simultaneous packets without order. So, we miss important information, since we assumed that the total order relationship is important for intrusion detection. Thus, the main feature is chosen: packet number.

Our results are dependent on the feature set. Since we were limited in resources, we decided to keep the feature set as simple as possible (without any info-data like flags) and complete enough to indicate common intrusions: packet number, source IP, source port, destination port, protocol (service). The experiment consists of two similar subexperiments: one on the complete future set and another on a reduced set – without the source port. It means that all features, except the source port, are axis attributes.

Then, the network events in form of packets captured by Wireshark need to be converted to a text file and coded to be used on the input of the FER program. We made a simple program to automatize the coding procedure. It reads the chosen features for each event from a txt-file and connects them, excluding the packet number, to a “word” using “_” in between. On the output we have a sorted list of events in form of an inp-file where each line contains packet number and word of the event’s features.

Finally, we must decide which traffic logs to use. As we already mentioned, we captured intrusive packets for 8 days (8 packet traces), and some days contains different sets of attacks, the other sets are alike. We pay attention to the following attacks: FTP Dictionary, TCP/UDP/ICMP Portscans, HTTP/TDS Exploits and SPIM. To reduce the data amount, we constructed an event sequence of two packet traces with clear attack signatures by appending the day 7 packet trail (with FTP Dictionary and HTTP Exploits) to the day 2 trail (with the rest of interesting attacks). At the same time, we exclude a lot of FTP Dictionary packets, because the experiment goal is to detect intrusions effectively, and if the Dictionary attack cannot be detected after several (not many) series of 3 password guesses, the system is not effective enough.

4.3 Threshold Selection

As we mentioned before, an episode of order 1 represents an event. We use such episodes to unify our experiment data and store them in the same form. Episodes of order 2 and 3 give us information on relations between 2 and 3 events, and it may be extremely useful in intrusion detection. Episodes of a higher order are complex and applicable to a few intrusions. Thus, we try to detect attacks by episodes of order not higher than 3.

Focus on only frequent episodes reduces possibilities of intrusion detection by means of

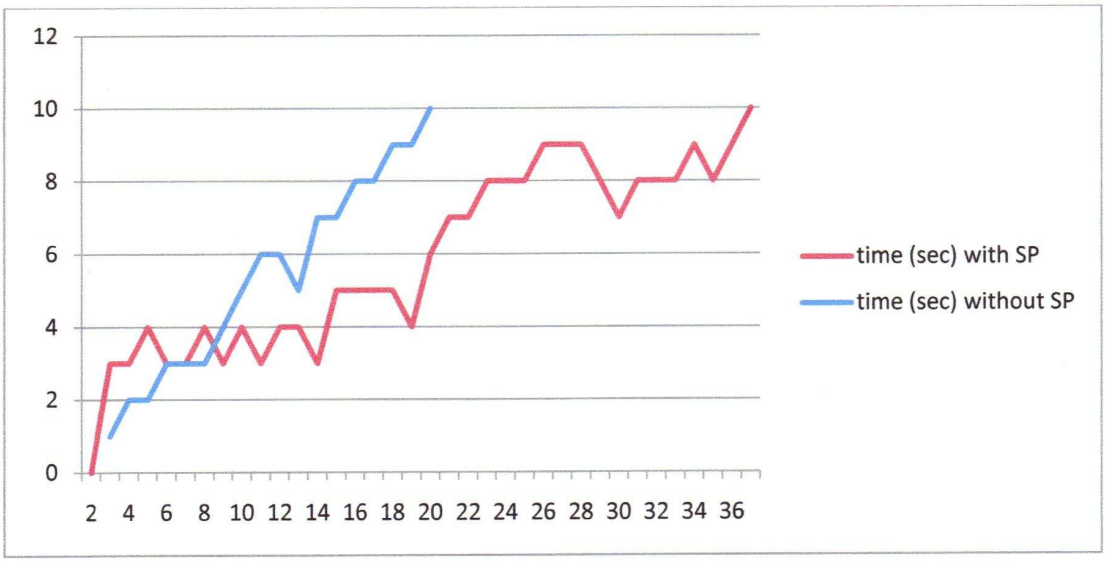

Figure 9: Frequency Calculation Time as a Function of Window Size

episodes. Slow attacks, for example, may be detected by accumulating the number of specific rare episodes (even of only one event) over several days. When the threshold (common for a given attack type) is reached, the attack is confirmed. To have overview of all episodes, both frequent and rare, we set the frequency threshold on 1. Thus, no episodes are filtered out. We noticed that the number of windows, containing a given episode, is dependent on the window size. We come back to it later.

A too large window leads to enormous number of episodes, redundant data, expansive use of resources, and low efficiency, while too small window limits episode variation since the time interval between the first and the last event in an episode is not larger than the window size. We do not try to detect slow and distributed attacks in this work. So, we do not need large windows. We start with $win = 2$ to calculate the actual number of events of various kind. Then, we set $max(win) = 37$ to fit 2 FTP dictionary series of 18 packets (the largest attack series in our packet trace), increment the window size by 1, and register the time required to calculate all the episodes' frequencies for each window size. The timing results are shown on Figure 9. For the complete feature set, we observe a fast growing of the time on the interval from $win = 20$ to $win = 26$. This fact made us to stop the episode analysis on $win = 20$. The experiment on the reduced feature set was also limited by $win = 20$.

Large windows give us episodes that do not fit into smaller windows. Thus, the episode variety is dependent on the window size. Figure 10 shows the dependency of the episode variety and the window size. The main observations of the curves are the following:

- The variation of episodes for the complete feature set is higher than for the reduced set.
- The variations of order 3 episodes is higher than the variations of order 2 episodes, and the difference is much bigger for larger windows.

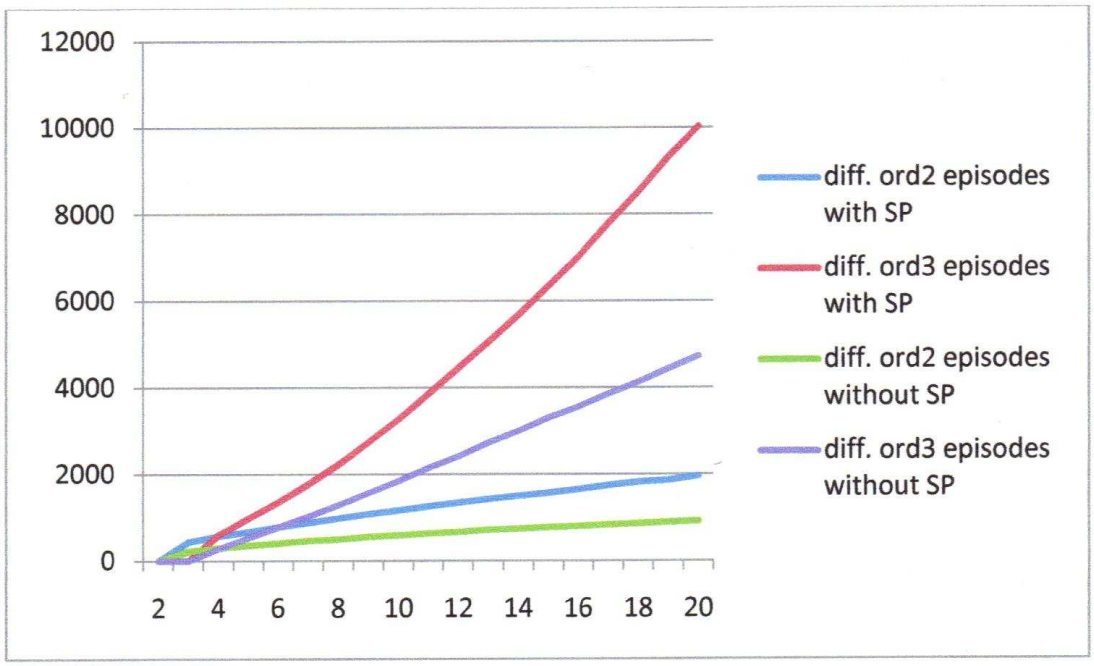

Figure 10: Variety of Episodes as a Function of Window Size

Our packet trace consists of 760 packets. The variety of episodes of order 2 exceeded the packet number already for $win = 6$ and the variety of episodes of order 3 – for $win = 5$. But some intrusions have period larger than 6, and we should not limit ourselves by such a small window size. This means that unfiltered episode collections do not reduce the work load of intrusion detection and we need effective filtering and pruning methods in addition to episode discovery.

The total number of episodes of order 2 for each of the subexperiments is presented as a function of window size in Figure 11. We noticed that the total number of episodes with source port (for the complete feature set) is higher than without it (for the reduced feature set). We explain it by the counting method's main principle: we count the windows, containing the episodes, but not the episodes. One window may contain several episodes. The probability of this increase with increasing the window size. The reduced feature set decreases the episode variation, and the probability of several episodes in one window is higher than for the complete set.

The difference between the total numbers of episodes with and without source port can say us something on the number of close similar events that differ only in source port. The form of the corresponding curve is very interesting. The slow growth for small and large windows and medium growth for medium windows (from $win = 9$ to $win = 15$) we explain by the experimental observation (which was made under the traffic analysis and described below) of the distance between attack series, where the series were run from different ports on the attacker's machine. The distance measured in packets between the key events of 2 series varies inside the

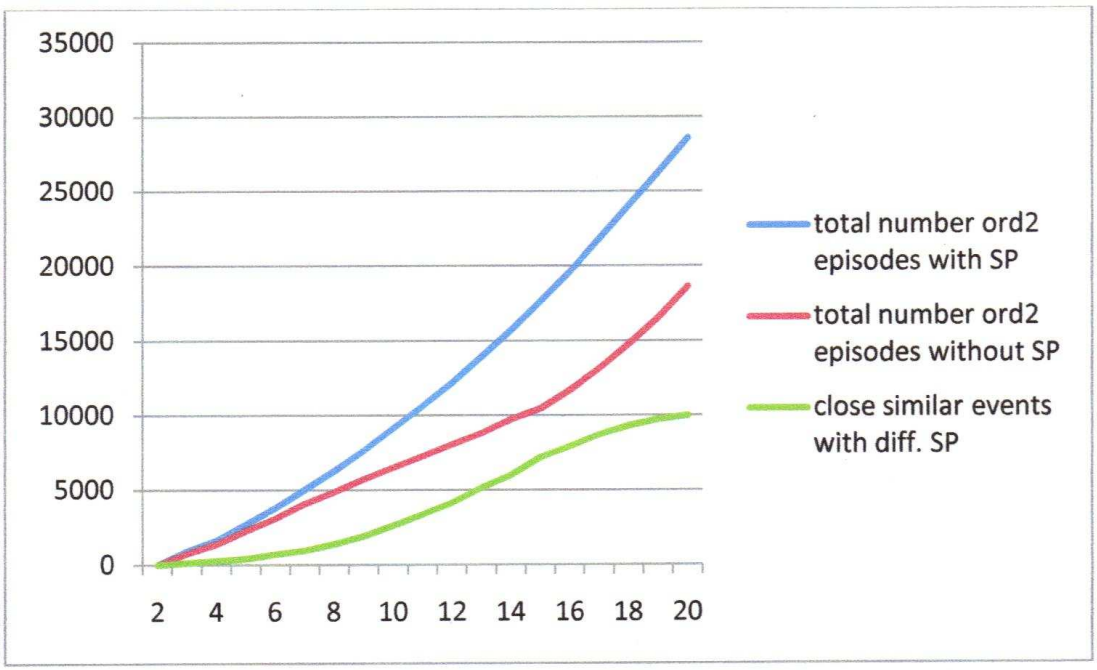

Figure 11: Total Number of Episodes as a Function of Window Size

medium windows' interval for most of the analyzed attacks.

4.4 Traffic Analysis

In this section, we describe our observations of attacks in order to define the attacks' signatures. We also analyze corresponding episode structures that we get from the output of our episode discovery program. Unfortunately, we cannot recommend episodes that describe normal activity, because the normality degree depends on the actual security policy. But we discuss how such episodes may look like and say something on the ranges of frequencies that the normal episodes vary most likely in. We assume that there were no distributed and complex attacks to our host at the time we captured packets for our experiment.

4.4.1 Remote-to-User

FTP Dictionary

Let us look at the event structures generated by an FTP Dictionary attack. From our packet trace we choose 3 series of FTP Dictionary: packets between 510 and 527 from port 58747, between 663 and 681 from port 60997, and between 719 and 737 from port 33581. Our host was attacked by only one FTP Dictionary attacker: IP = 62.193.225.104. The packets have the same destination port: ftp. Obviously, the same program was used to generate all the attack packets. So, we can expect similarity in the event structures. We are interested in this similarity and variations in the attack series, because we want to define the attack's signature in form of one or several episodes.

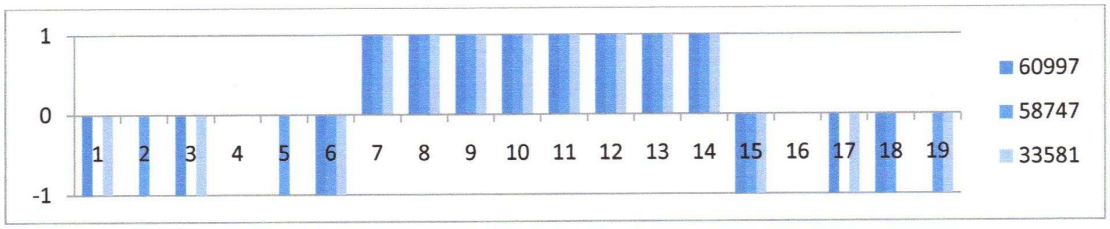

Figure 12: FTP Dictionary Series (FTP packets coded by 1, TCP – by (-1); other – by 0)

Every period of the attack contains 13 packets in the following constant order: TCP SYN, TCP ACK, TCP ACK, FTP USER, FTP PASS, FTP USER, FTP PASS, FTP USER, FTP PASS, FTP USER, FTP USER, TCP FIN-ACK, TCP RST, and TCP RST. The periods cover each other in some degree because of the attack algorithm the attacker applied: If the current FTP session began to close itself (it happens after 3 unsuccessful login attempts on our host; and that rule depends on the local FTP setup), start a new FTP session from another port. Consequently, TCP packets, closing an FTP session to one port, are mixed with other TCP packets, opening a new FTP session.

In the Figure 12, we can see that the packet sequence from the 6th to the 15th – TCP, 8 FTP, TCP without any other packets in between – is common for the FTP series. The absence of other packets can be explained by very small time intervals. For example, the average duration of one FTP session in our experiment is 0,49 seconds and the average time interval between the last TCP packet before the FTP packet sequence and the first TCP packet after it is 0,35 seconds. Obviously, the packets from the 6th to the 15th can serve as an FTP Dictionary signature, especially if the period duration is short. A usual FTP login (when the login and password are correct under the first attempt) can be presented by packets from 1 to 8. Besides, the time intervals between 6th, 7th and 8th are much larger for any human user.

We can also define the attack signature by episodes, combining the 2 events: FTP and TCP packets to port ftp. There are 4 possible episodes of order 2 in this case: TCP-TCP, TCP-FTP, FTP-FTP, and FTP-TCP. All the episodes are relevant to the normal behavior pattern when the packet number is prior to the timestamp. The following episodes of order 3 for a small window size mean a short (either in packets or in time) FTP session: FTP-FTP-FTP and TCP-FTP-TCP. If a short session is classified as anomaly in the security policy, we can use the episodes as the attack's signature. Results of the experiment show that the episodes FTP-FTP-FTP from the same port to port ftp are the most frequent for any window size higher than 3. This means that the attack was successfully detected.

MS SQL Exploit

Another Remote-to-User attack in form of login attempts on MS SQL Server we called MS SQL Exploit. The login is known to be "sa", but the passwords were sent encrypted. It could be a Dictionary attack to MS SQL Server or an automated exploit directed to the default login setup: login = password = "sa". The Exploit attack is represented by the following packet sequence to port ms-sql-s: TCP SYN, TCP ACK, TDS, TDS, TDS, TCP ACK, TCP FIN-ACK. Each login attempt was made from different port on the attacker's host. Figure 13 shows 3 login attempts from

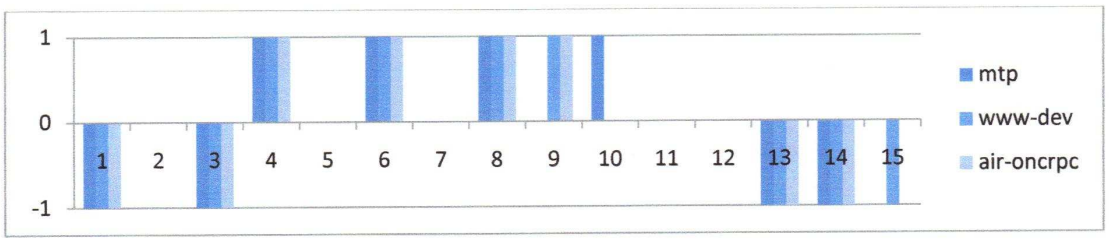

Figure 13: MS SQL Exploit Series (TDS packets coded by 1, TCP – by (-1); other – by 0)

ports mtp, www-dev and air-oncrpc. Obviously, the packets number 1, 3, 4, 6, 8, 13 and 14 are common for the attempts. There are only 3 TDS packets, and a normal login packet sequence is similar to those from Figure 13.

Experimenting on the feature set reduced by the source port feature provides the complete picture of all login attempts on MS SQL Server. From our packet trace, we detected that there were 15 packets between the first TDS in one attempt and the first TDS in the next attempt. Our episode-based method for event distribution analysis can help us to detect this and similar Dictionary attacks.

4.4.2 Probe

Portscan

A commonly-known Probe attack Portscan is also presented in our audit trail. The first packet (nr.69) from 202.120.43.208 (China, source port 10267) came 23.03.2008 21:31:09 to destination port ndl-aas. It was a simple TCP SYN packet. During the next 1 second, some more similar packets had come from ports 10267 and dell-rm-port on other our ports: http, socks, http-alt, cpq-wbem, ddi-tcp-1. Our host responded with either (RST, ACK) or (SYN, ACK). In the second case, the attacker established TCP connection with final (ACK) or cancelled the handshake by (RST). Each opened TCP connection (from dell-rm-port to http) was scanned by HTTP requests like www.scanproxy.com.

7 minutes after that, the attacker began to respond on our (RST, ACK) packets by “TCP Port number reused” because several half-opened connections were initiated from the same port. It is interesting that such bad TCP-packets also contained a new SYN for the same handshake. At the same time, another TCP-connection was successfully established: from izm to http, and http port had been scanned again. At 21:44, 7 minutes later, the same sequence of packets was repeated and http was scanned from source port 4202; at 21:51 from itose; at 21:58 from simple-push-s. Such repeatability of portscans allows us to discover the attack with the full feature set. The Portscan episodes’ frequency is the next highest after the FTP Dictionary.

There are several portscan periods shown in Figure 14. TCP packets from port 10267 to different ports and HTTP packets from different ports are coded by positive numbers; TCP packets to port http are coded by negative numbers; other - by 0. We applied the following destination port coding for TCP packets from 10267: 1 - http, 2 - ndl-aas, 3 - socks, 4 - http-alt, 5 - cpq-wbem, 6 - ddi-tcp-1; - and source port coding for packets TCP and HTTP to port http: 1/-1 - dell-rm-port,

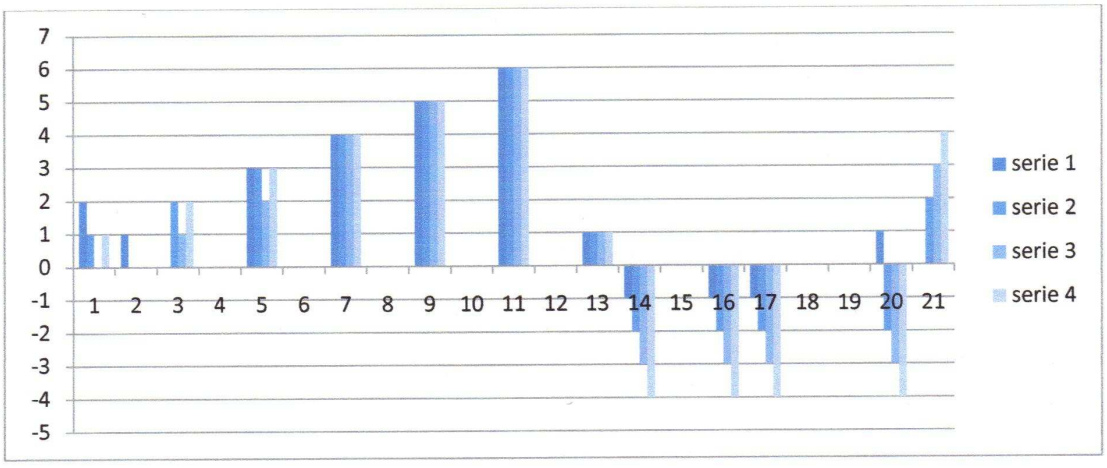

Figure 14: TCP Portscan Series

2/-2 - izm, 3/-3 - 4202, 4/-4 - itose. We can see that the variations between the Portscan periods are very small. Packets 7, 9, 11, 13, 14, 16, and 17 are common for all periods.

A Portscan attack can be detected in form of a sequence of close packets from one port to various ports. Hence, episodes of order 3 for a fixed source IP and different ports can indicate a Portscan attack. Usually, the episodes are rare. For example, the following episode is one of the rarest episodes ($\mathtt{frequency} = 2$) for $win = 20$: (202.120.43.208_dell-rm-port_http_TCP; 202.120.43.208_10267_http-alt_TCP; 202.120.43.208_izm_http_TCP). The analysis of rare episodes requires a fixed structure of the attack series to define the window size, in which the episode is visible at the first time ($\mathtt{frequency} > 0$).

The order relationships between the packets do not give us any information useful for the Portscan detection. But the episode structure can be applied to event distribution analysis, and we can discover regularities of the Portscan. For example, there are about 20 packets between the 11th packets of 3 close Portscan periods in our audit trail. For $win = 23$ we can notice the periodical nature of this Portscan attack by counting the episodes.

Normal TCP handshakes from one IP are, usually, sent to one port, and the next TCP SYN is sent only when the previous TCP handshake is finished (by TCP FIN, timeout, or other finalization methods). Hence, we can make a normal behavior pattern of two or more TCP SYN sent from the same IP to the same port. But, as our experiment showed, several TCP SYN to the same port can be sent periodically to hide the Portscan attack. Since the TCP SYN packets were sent from one port, only the first handshake was accepted, and all other were declined with error message "TCP Port numbers reused", containing a new TCP handshake initiation between the same ports.

We can also assume that several episodes (TCP SYN, TCP FIN) with the same pair (source port, destination port) are normal. We should not forget that several TCP FIN to different ports are a Portscan as well. So, if the number of episodes (TCP SYN, TCP SYN) – as well as (TCP FIN, TCP FIN) – is not higher than the number of episodes (TCP SYN, TCP FIN), there is no Portscan there. The same technique can be used to detect TCP SYN Flood attacks, but we should compare

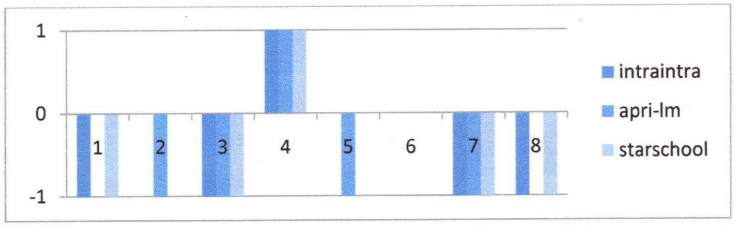

Figure 15: PHP Probe Series (HTTP packets coded by 1, TCP – by (-1); other – by 0)

the number of episodes (TCP SYN, TCP SYN) to one port with the total number of episodes (TCP ACK, TCP ACK) and (TCP RST, TCP RST) to the same port.

PHP Probe

Another Probe attack (we called it PHP Probe) was directed to our host under the data capturing. The attack consisted of many series with the following structure: TCP handshake (TCP SYN, TCP ACK), HTTP GET (/administrator/phpMyAdmin-2.8.0/main.php or other possible .../main.php and .../login.php) from different ports, and final TCP handshake (TCP ACK, TCP FIN). By such requests, the attacker collects information on PHP administration services on the host. The packets have the same destination port: http. Various PHP Probe series are shown in Figure 15.

The common packets are 3, 4, and 7, but the episodes TCP-TCP, TCP-HTTP, HTTP-TCP, TCP-HTTP-TCP are rare for the complete feature set, because the packets were sent from different IP addresses and different ports – it was a low (distributed) Probe attack or several parallel attacks. The series initiated from the same IP address are identical because they were generated by the same attack tool. Such series give use frequent episodes if we exclude source port from the feature set. The event distribution analysis can give us interesting results, especially if we exclude the source IP as well. Or we can simply pay attention to the episodes with frequency 1 or 2 for $win = 6$, like TCP-HTTP-TCP. They mean a short http-session and, if they are regular, indicate anomalies.

4.4.3 Spam

SPIM

The last attack we want to discuss is SPIM, which is supposed to generate a pop-up window on the host's desktop with an instruction on downloading and installing a program from the Internet, i.e. an advertisement. The attack is represented by several Messenger packets from one IP, but from different ports to port cap or 1027 with the following content: "NetrSendMessage Request: STOP! WINDOWS REQUIRES IMMEDIATE ATTENTION. Windows has found 55 Critical System Errors. To fix the errors please do the following: 1. Download Registry Update from: www.regfixit.com. 2. Install Registry Update. 3. Run Registry Update. 4. Reboot your computer. FAILURE TO ACT NOW MAY LEAD TO SYSTEM FAILURE!" Figure 16 shows a sequence of some SPIM packets.

And again we have a lot of rare packets, but not so regular at this time. Variations in periods decrease the effectiveness of the event distribution analysis, when only the source port feature

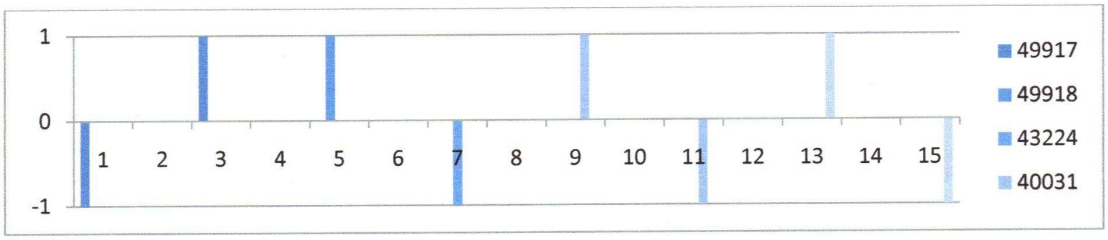

Figure 16: SPIM Series (Packets to port 1027 coded by 1, to port cap – by (-1); other – by 0)

is ignored. Excluding both the source and destination ports from the feature set makes the automated attack visible: the Messenger packets from any port to any port are equally distributed with the period of 2 packets (see Figure 16).

But even one SPIM period is an attack. So, we better leave the SPIM discovery task to the misuse module. The attack signature – a Messenger packet from the outside – is an event (atomic episode), which must be included into the attack signature database. The important moment here is that, if an episode was defined as intrusive by the misuse module, it should be presented to the anomaly module together with the other episodes. For this SPIM example, the anomaly module is supposed to discover the attack's regularity, and the decision module should pay attention to the intrusion discovery results from both misuse and anomaly modules and produce only one alert for all series of the SPIM.

4.5 Frequency Analysis

We have discussed the event sequences generated by several attackers under the experiment. Now we want to present the results of the episode discovery and counting provided by our Winepi implementation and shown in Table 2. The table contains only episodes, consisting of events from the same IP. It was our pruning method based on the fact that a separate attack attempt is always initiated from one IP, especially, if it is represented by a protocol session.

4.5.1 TCP Portscan

We study TCP Portscan on the episodes of type TCP-TCP, where the TCP packets (events) have different destination port. Such episodes indicate close portscan attempts and reflect the nature of the portscans – regular switches of the destination port.

The results for the complete and reduced (without SP) feature set coincide. It means that the attacker did not changed source port under the attack. The maximal frequency for $win = 3$ is equal to 5. It is the frequency of an episode with fixed features. The high frequency for $win = 3$ indicates that the attacker launched similar sequences of portscans 5 times. The only explanation of such behavior is that the attack was generated by a computer.

The frequency increases regularly by 5 with the window growing by 1. The largest window in the table is win=8, hence, there are more than 8-3=5 packets between the port switches. The regularity of event distribution and similarity of the attack series are demonstrated. The rare episode analysis is not possible for this attack kind.

Episodes of order 2 for win=	3	4	5	6	7	8	delta
FTP Dictionary							
FTP-FTP from one port to ftp	5	9	23	31	42	49	7
TCP-TCP from one port to ftp	4	6	12	16	20	24	4
TCP-TCP without SP to ftp	65	169	260	325	390	455	65
TCP Portscan							
TCP-TCP from one port to diff. ports	5	10	15	20	25	30	5
TCP-TCP without SP to diff. ports	5	10	15	20	25	30	5
PHP Probe							
TCP-TCP from one port to http	4	8	11	14	17	20	3
TCP-TCP from diff. ports to http	4	6	8	10	12	14	2
HTTP-HTTP from diff. ports to http	0	0	0	1	2	3	1
TCP-TCP without SP to http	20	33	53	73	103	120	17
HTTP-HTTP without SP to http	0	0	0	8	16	24	8
HTTP-TCP from one port to http	2	3	4	5	6	7	1
HTTP-TCP from diff. ports to http	1	2	3	4	5	6	1
TCP-HTTP from one port to http	2	3	4	5	6	7	1
TCP-HTTP from diff. ports to http	0	1	2	3	4	5	1
MS SQL Exploit							
TCP-TCP from one port to ms-sql-s	5	8	11	14	17	20	3
TDS-TDS from one port to ms-sql-s	4	7	10	13	16	19	3
TCP-TCP without SP to ms-sql-s	20	31	56	75	92	109	17
TDS-TDS without SP to ms-sql-s	19	28	47	64	79	94	15
TCP-TDS without SP to ms-sql-s	10	15	20	25	30	35	5
TDS-TCP without SP to ms-sql-s	0	1	6	11	16	21	5
TCP-TDS from one port to ms-sql-s	2	3	4	5	6	7	1
TCP-TDS from diff. ports to ms-sql-s	0	0	0	1	2	3	1
SPIM							
Messenger from one port to cap-1027	2	3	4	5	6	7	1
Messenger without SP to cap-1027	2	3	4	5	6	7	1
Messenger from diff. ports to cap-cap	0	0	1	2	3	4	1
Messenger without SP to cap-cap	0	0	1	2	3	4	1
Messenger from diff. ports to 1027-cap	1	2	3	4	5	6	1
Messenger without SP to 1027-cap	1	2	3	4	5	6	1

Table 2: Episode Frequencies Estimated by Winepi ($\mathtt{delta} = \mathtt{fr}(win = 8) - \mathtt{fr}(win = 7)$)

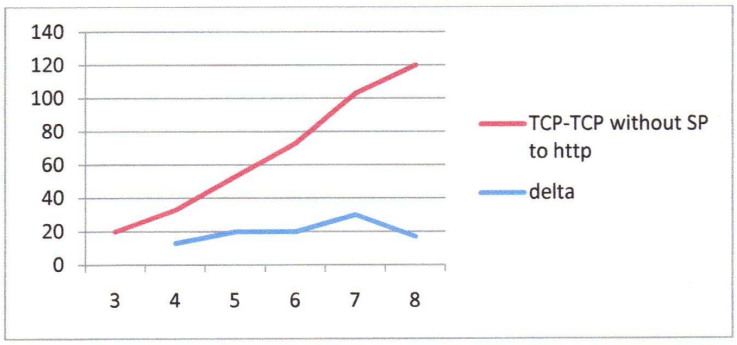

Figure 17: TCP-TCP to http Episodes: Frequency as a Function of win for RFS

4.5.2 PHP Probe

The most interesting episodes of our PHP Probe are TCP-TCP and HTTP-HTTP from the same or different ports to http port. The analysis for the complete feature set is similar to the TCP Portscan's episode analysis, and we do not describe it. TCP-TCP episodes from the same port to http indicate overlap of attack series, where several PHP queries were sent from one port close together.

The most useful observation was made under comparison of the results from the two feature sets. The frequencies are much higher for the reduced set; for example, the frequencies of the episode HTTP-HTTP from different ports are precisely 8 times higher than for the complete set. It means that the attacker changed the source port at least 8 times. The rare episode analysis of these episodes is based on the fact that the episodes appeared for win=6 at the first time both for the complete and reduced feature set (RFS). So, the distance between the attack series is at least 6-3=3 packets. The rare episode analysis of the episodes related to a PHP Probe attack shows that there is at least one intersection of the attack periods, which results in a relatively small window size $win = 4$ for $\mathrm{fr} = 1$.

To analyze the events' distribution, we look at the frequency of the episode TCP-TCP to http port without source port (SP) in Figure 17. We see that the increase (delta) is highest for $win = 7$. The explanation is the 4 (=7-3) packets distance between the last TCP of one period and the first TCP of the next attack period. We provide a detailed description of the event distribution analysis technique in the next section.

4.5.3 MS SQL Exploit

We neglect the frequent episode analysis and start with rare episodes. The episodes TCP-TDS from different ports have frequency $\mathrm{fr} = 1$ earliest at $win = 6$. So, the minimal distance between a TCP of one MS SQL Exploit period to a TDS of the next period on another port is 3 packets (5, including these 2 packets, and the 6th packet we do not count by the window definition). And it is true. The event distribution analysis of the attack can be based on episodes TCP-TCP to ms-sql-s port without SP. The corresponding frequencies are shown in Figure 18. The highest frequency increase was for $win = 5$. Hence, the attack periods are close together: only 5-3=2

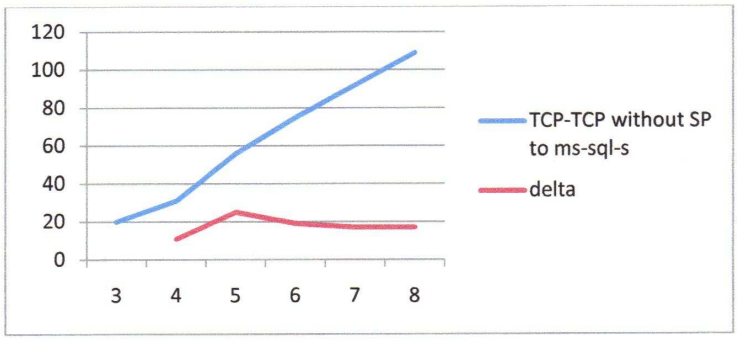

Figure 18: TCP-TCP to ms-sql-s Episodes: Frequency as a Function of win for RFS

packets in between.

4.5.4 SPIM

The episodes of Messenger packets differ from the other attack episodes by low frequencies for the reduced feature set. We knew that the source port was changes often in our SPIM attacks. Thus, we expected high frequencies for the reduced feature set. The results made us to look closer at the SPIM attack. We discovered that our traffic contains SPIM attacks from many IP addresses. We could not notice it before, because the addresses vary only in the last 3 numbers, excluding one absolute different IP. This late discovery shows how precise an attack picture may became with the event distribution technique. Moreover, it can simply be automated.

So, there were 2 SPIM attackers under the experiment: IP=218.10.137.140 and 202.97.238.xxx. The first attacker varies only source port, but runs only 2 attempts, which resulted in the episodes and frequencies presented in Table 2. The analysis of rare episodes of type Messenger-Messenger from different ports to cap port gives us the distance between the attempts: 5-3=2 packets. The results for the complete and reduced feature sets are equal because we have only 1 pair of attempts, and we should have at least 3 periods to talk on the attack's regularity.

The second attacker varies both IP and source port. Obviously, it is a distributed attack, which is run by a computer program. The attack is relatively slow. All these attack properties allow the attacker to remain undetected. It means that our techniques are not always effective, and our IDS must be supplied by other techniques or, may be, other data structures.

4.6 Detailed Event Distribution Analysis

A pair of close events is contained by more windows than a pair of events with a large time interval in between. At the same time, the difference in the number of windows is equal to the difference in time or packets' number between the events. Relative difference is insignificant for large windows. A possible solution can be using only small window sizes or special weights for windows counting dependent on the intervals: the closer the events are, the higher the weight is. Such weights may increase proportionally to the number of similar events following each other and, so, help to detect automated packet generation, which, usually, is a sign of an attack.

Our Winepi implementation calculates windows, containing episodes of several equal events,

win	FTP ord2 with SP 3	series of FTP ord2	FTP ord3 with SP 3	series of FTP ord3
2	0	0	0	0
3	7	21	6	18
4	8	24	7	21
5	9	27	8	24
6	10	30	9	27
7	11	33	10	30
8	12	36	11	33
9	13	41	12	36
10	14	42	13	43
11	15	43	14	44
12	16	44	15	45

Table 3: The FTP Dictionary's Episodes Counting Results

with such weights, corresponding to the number of equal events without any packets in between, in cases of very close event occurrences, like FTP Dictionary attack. Thus, the FTP-FTP and FTP-FTP-FTP episodes are the most frequent for every win, because the number of windows with FTP-FTP increases by the number of close FTP-FTP from the same port (7), when win increases by 1. The same weights are applied for order 3 episodes FTP-FTP-FTP: the number of corresponding windows increases by 6, when win increases by 1. All this is valid for $win > 6$, when the whole FTP period can be covered by one window of the next window size (which is minimum 8 packets).

The linear dependency between frequency and the window size is obvious in Table 2. For small window sizes, it follows from the counting method description. Let us represent the frequency as a linear function of the window size: $\mathrm{fr} = \mathrm{k} * win + \mathrm{b}$. Coefficient k depends on the intensity of the attack: $\mathrm{k} = delta(\mathrm{fr})/delta(win)$. Constant b depends on the attack signature. For high win and k, frequency is very high. Intense attacks are visible for even low win values, while slow attacks have low frequency even for high win. So, win threshold influences frequency threshold and the thresholds can be avoided in intrusion detection by operating, for example, with relative frequencies. We propose another method: fix the frequency and observe the windows size. For serial attacks, coefficient k is a function of win: $\mathrm{k} = \mathrm{k}(win)$ and $\mathrm{k} \to 1$ if $win \to \infty$. In other words, $\mathrm{k} = 1$ if the window is so large that all attack series can be covered by one window. We show it on an example.

Let us calculate the real (non-weighted) frequencies of our FTP Dictionary episodes counted with sliding windows of different sizes. We remember that the attack was organized periodically and in many close series. We try to find evidence of this fact in the counting results shown in Table 3. Columns "port_FTP_ord2" and "port_FTP_ord2" contain frequencies of the episodes (FTP, FTP) and (FTP, FTP, FTP) from the same port. Columns "3 FTP series_ord2" and "3 FTP series_ord3" refer to the same episodes but from any port. The results are visualized in Figure 19.

In the figure we can see the difference between the results with attention to the source port (fixed port results) and without it (any port results). The difference between the results caused by the episode order is insignificant. The curves for the any-port results are not linear, and below we explain why. For a better result visualization, we present the growth of the episodes' frequency

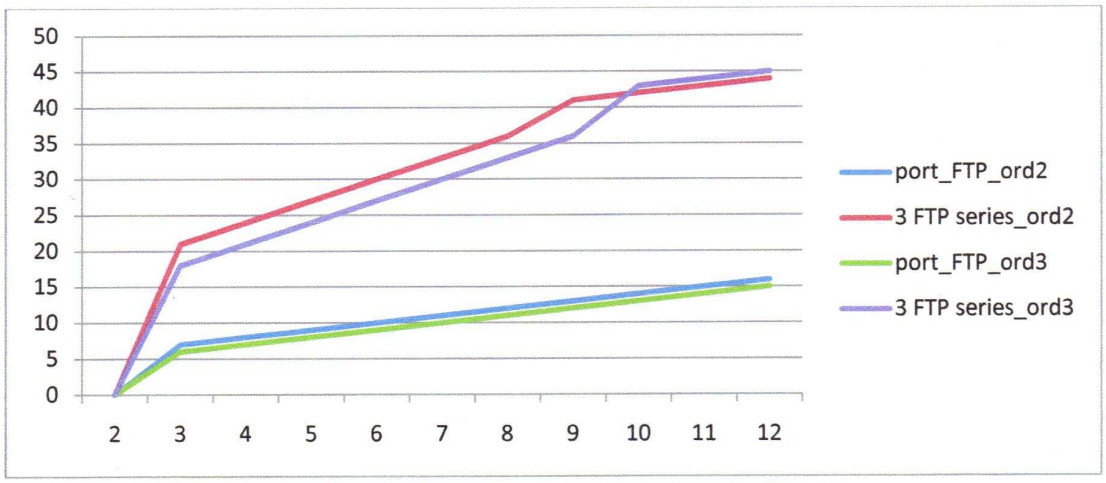

Figure 19: Visualization of the FTP Dictionary's Episodes Counting Results

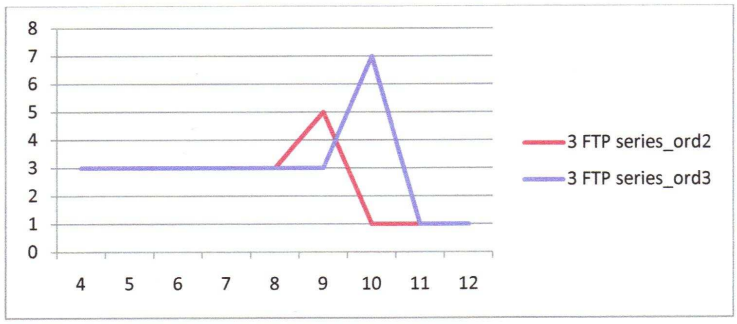

Figure 20: $\mathrm{Delta}(\mathrm{fr})$ as a Function of win for the FTP Dictionary Episodes

$\mathrm{delta}(\mathrm{fr})$ as a function of the window size win in Figure 20.

There are 6 packets between the last FTP of an attack period and the first FTP of the next period. Hence, any window smaller than 9 packets ($win < 9$) does not cover the two FTP packets from different periods at the same time. For those windows and the episodes (FTP, FTP) from any port we have: $\mathrm{delta}(\mathrm{fr}) = \mathrm{N}$ for $\mathrm{delta}(win) = 1$, where N is the number of the attack periods. For $win = 9$, the frequency of the episodes increases by $2 * \mathrm{N} - 1$ in comparison to $win = 8$. The frequency of the similar episodes (FTP, FTP, FTP) increases on $win = 10$ by $\mathrm{N} + 2 * (\mathrm{N} - 1)$. After those gaps, $\mathrm{delta}(\mathrm{fr}) = \mathrm{delta}(win)$ for $win > 9$ for the second order episodes and $win > 10$ for the third order episodes.

Now we have described a possible application of episodes, in general, and the Winepi counting method, in particular, to event distribution analysis. The observation of frequency growth allows to discover regularity in event sequences and to define the period of the discovered event series.

4.7 Quality of the Experiment

According to the [58], there are 4 general criteria for a research project: universality, replication, control and measurement. Our research is universal and replicable because any competent person can repeat the experiment and get similar results. Here we want to notice that the probability of capturing the same intrusions is low, but event counting of the input sequence (even by using a different Winepi implementation) give precisely the same results. The lack of control does not allow us to eliminate the uncertainty and generalize our results, but we introduced control by isolation of the intrusive behavior under the capturing, using the independency of the HoneyWall and manual isolation of some attacks under the input sequence construction.

The measurement aspect is complex. In our project, we measure intrusions by event sequences, event sequences – by episodes, and episodes – by number of windows containing them. At first, we want to capture intrusive and only intrusive behavior and, therefore, use Honey tools. Then, we register all network packets by Wireshark and generate event sequence from them. Finally, we construct and count episodes with our own Winepi implementation. The measurement instruments must be reliable and valid [58].

Reliability of an experiment is a consistency of the results of measuring instruments [58]. In our case the reliability question may be divided into internal consistency reliability and test-retest reliability. The variety of events, attracted by a HoneyPot, depends on the HoneyPot's setup. The repeatability cannot be achieved because of the stochastic nature of network events on a given host. Thus, the test-retest reliability of the capturing methods is difficult to check. Our capturing equipment (HoneyWall and Wireshark), as their documentation claims, are consistent and capture all events, related to the host, equally. The internal consistency reliability we build on these claimes. The reliability of the counting algorithms and our program was proved by several simple input tests and repeating of the actual events counting. Both timing and counting results remained the same, and it proves the reliability of the counting method.

The experiment validity is a correspondence between the actual events and the measurement results [58]. Again, we refer to the HoneyWall and Wireshark documentation and conclude that the capturing method is valid. The validity of the algorithm and its implementation was proved by their testing on a specially constructed input sequences. Many repeating events of several event types were mixed in various ways, counted first manually, based on the Winepi idea, and then we input the sequences to the program. The results were compared and the validity of the counting procedure was proved.

4.8 Experiment Summary

The experiment showed that several good attack patterns can be built on episodes. Frequent episode counting was demonstrated to mine information essencial useful in intrusion detection from long event sequences. So, the episode structure is an efficient instrument. Despite the attack variations, frequent episodes could detect most of the attacks during the experiment. It means that the episode structure is accurate enough to be applied to intrusion detection.

5 Conclusions

In our literature study, we demonstrated the lack of systematization of episode improvements and episode discovery algorithms. The main reason of the variety of episodes and algorithms we see in limitations of the episode structure. The episode definition requires generalization to be flexible enough for several application areas, including intrusion detection. The universality of the structure and discovery algorithms will result in deepen episode research and turn the episode structure into a powerful instrument for data mining.

There are few IDS on episodes, and their testing demonstrates outstanding results. It means that episode discovery is useful for intrusion detection. But, because of the lack of research and weak discussion, the IDSs are only examples and test objects; they are not a framework. We studied a recently-proposed hybrid IDS critically and discussed its architecture. The problem of hidden assumptions, limited view, and anchored decisions can be solved by attention of several researchers to the possibilities of episodes and their compatibility to intrusion detection. Our IDS model is an attempt to break the frame, but, of course, it is only a new subjective look at IDS construction.

Another difficulty is caused by the attacks variety. The today's episode structure is not flexible enough to be the only structure for IDS building. But, before we fortify it by other data structures, we should investigate the strengths and weaknesses of this structure in detail and define its applicability area. Our experiment and its analysis are the first steps in this direction. We analyzed some attack behavior and its influence on corresponding episodes and their counts. A systematical analysis of this kind is of enormous importance in intrusion detection.

As we have shown, the episode structure is useful in detection of various intrusions, which can be represented by either frequent or rare patterns. The similarity and regularity of most of the episodes related to an attack allow us to discover and exploit hidden possibilities of episode counting based on sliding window method (Winepi). The three pattern discovery techniques that we proposed in the thesis can be combined to analyze the attacks in depth.

The effectiveness of episodes in pattern discovery we demonstrated by the experiment. Data mining efficiency and intrusion detection accuracy were shown. The efficiency problem was out of the focus in our thesis and needs to be be investigated in detail. The episode analysis of the results helped to develop and test the new pattern discovery techniques. But, obviously, there are many unsolved problems and questions, considering the episode applicability to intrusion detection.

6 Future Work

In April 2009 an article on complex-event processing (CEP) [59] was published. The author points on the lack of general real-time event analysis and recommend CEP systems that collects events and, using some algorithms and rules, constructs complex events. Episodes represent a particular case of complex events, and episode discovery is a part of CEP. The CEP systems build databases of profiles and patterns on floods of data. The author suggests reuse of the patterns, but refer to the lack of standards on event analysis and libraries of rules and patterns of complex events. So, standardization of pattern discovery and libraries of useful episodes are directions of the future work related to episodes.

The future of intrusion detection is described in [9] as coordinated distributed monitors on the network and protocol stack levels. Thus, the possibility of including domain information into the episodes is significant for the future use of episodes in intrusion detection. Then, we will be able to use not only network data, but include the whole spectrum of data sources into the event analysis.

After the general future tasks, we come back to our IDS. The most important aspect of IDS analysis is performance testing. Testing results depend on the input event sequences, and we could not test our modules properly to compare our IDS with other ones. So, capturing and simulating of normal and intrusive events in order to build a representative traffic for training and testing the IDS is the next task.

As we have mentioned above, the detection module of an IDS must contain discovery, matching (inter-matching), and, preferably, pruning (intra-matching) algorithms, which should be efficient enough. Which algorithms fit FE discovery, pruning, and matching best in intrusion detection context? Time complexity analysis is required to decide if the real-time intrusion detection using episodes is possible. Another task is to check if a hardware implementation of the algorithms may be more efficient than software implementations.

Another future work on our IDS is implementation and testing of episode discovery algorithms for sequences with simultaneous events. In the next experiment, the packet numbers should be replaced by the timestamps. The traffic and episode analysis can be made analogously. Experiments on the feature set are also important, especially, analysis of the influence of various reductions and extensions of the set on the set of discovered episodes.

7 Acknowledgements

I would like to thank Prof. Slobodan Petrovic for supervising and support. The experiment was successful thanks to the Gjøvik University College's IT Service and my co-student Lars Olav Gigstad. In addition, I want to notice that the thesis would not be so rich and interesting without the inspiration I got from my family and friends.

Bibliography

[1] Qin, M. 2007. Hybrid intrusion detection with weighted signature generation over anomalous internet episodes. *IEEE Trans. Dependable Secur. Comput.*, 4(1), 41–55. Fellow-Hwang,, Kai and Member-Cai,, Min and Student Member-Chen,, Ying.

[2] Qin, M. & Hwang, K. 2004. Frequent episode rules for internet anomaly detection. In *NCA '04: Proceedings of the Network Computing and Applications, Third IEEE International Symposium*, 161–168, Washington, DC, USA. IEEE Computer Society.

[3] Mannila, H., Toivonen, H., & Verkamo, A. 1995. Discovering frequent episodes in sequences. In *KDD '95: In Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, 210–215, Montreal, Canada.

[4] www.wireshark.com.

[5] Co., J. P. A. Computer securtity threat monitoring and surveillance. Technical report, James P. Anderson Co., Fort Washington, PA 19034, February 1980. Contract 79F296400.

[6] Allen, J., Christie, A., Fithen, W., McHugh, J., Pickel, J., & Stoner, E. State of the practice of intrusion detection technologies. Technical Report CMU/SEI-99-TR-028, ESC-99-028, Carnegie Mellon University, 1999.

[7] Denning, D. E. 1987. An intrusion-detection model. *IEEE Transactions on Software Engineering*, 13, 222–232.

[8] Zhou, J., Heckman, M., Reynolds, B., Carlson, A., & Bishop, M. 2007. Modeling network intrusion detection alerts for correlation. *ACM Trans. Inf. Syst. Secur.*, 10(1), 4.

[9] Anderson, R. J. 2008. *Security Engineering: A Guide to Build Dependable Distributed Systems*. Wiley Publishing, Inc., 2nd edition.

[10] Bosworth, S. & Kabay, M. E., eds. *Computer Security Handbook*, chapter 37. Vulnerability assessment and intrusion detection systems by Rebecca Gurley Bace. John Wiley & Sons, 4th edition, 2002.

[11] Qin, M. & Hwang, K. 2004. Frequent episode rules for intrusive anomaly detection with internet datamining.

[12] Lee, W. 2002. Applying data mining to intrusion detection: the quest for automation, efficiency, and credibility. *SIGKDD Explor. Newsl.*, 4(2), 35–42.

[13] http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/index.html.

[14] Lee, W. & Stolfo, S. J. 2000. A framework for constructing features and models for intrusion detection systems. *ACM Trans. Inf. Syst. Secur.*, 3(4), 227–261.

[15] www.snort.org.

[16] Gates, C. & Taylor, C. 2007. Challenging the anomaly detection paradigm: a provocative discussion. In *NSPW '06: Proceedings of the 2006 workshop on New security paradigms*, 21–29, New York, NY, USA. ACM.

[17] Mannila, H. & Toivonen, H. 1996. Discovering generalized episodes using minimal occurrences. In *KDD'96: In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 146–151, Portland, OR.

[18] Mannila, H., Toivonen, H., & Verkamo, A. I. 1997. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3), 259–289.

[19] Srivastava, J., Wijesekara, D., & Guralnik, V. Pattern directed mining for frequent episodes. Technical report, University of Minnesota, 1998.

[20] Harms, S. K., Deogun, J., Saquer, J., & Tadesse, T. 2001. Discovering representative episodal association rules from event sequences using frequent closed episode sets and event constraints. *icdm*, 00, 603.

[21] Wojciechowski, M. 2000. Discovering frequent episodes in sequences of complex events.

[22] Casas-Garriga, G. 2003. Discovering unbounded episodes in sequential data. In *In Proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'03)*, 83–94, Cavtat-Dubvrovnik, Croatia.

[23] Laxman, S., Sastry, P. S., & Unnikrishnan, K. P. 2007. A fast algorithm for finding frequent episodes in event streams. In *KDD'07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, 410–419, New York, NY, USA. ACM Press.

[24] Bettini, C., Wang, X. S., Jajodia, S., & Lin, J.-L. 1998. Discovering frequent event patterns with multiple granularities in time sequences. *IEEE Transactions on Knowledge and Data Engineering*, 10(2), 222–237.

[25] Laxman, S., Sastry, P. S., & Unnikrishnan, K. P. 2005. Discovering frequent episodes and learning hidden markov models: A formal connection. *IEEE Transactions on Knowledge and Data Engineering*, 17(11), 1505–1517.

[26] Katoh, T., Hirata, K., & Harao, M. *Mining Sectorial Episodes from Event Sequences*, 137–148. Springer Berlin / Heidelberg, 2006.

[27] Katoh, T., Hirata, K., & Harao, M. 2007. Mining frequent diamond episodes from event sequences. In *MDAI '07: Proceedings of the 4th international conference on Modeling Decisions for Artificial Intelligence*, 477–488, Berlin, Heidelberg. Springer-Verlag.

[28] Mannila, H. & Meek, C. 2000. Global partial orders from sequential data. In *KDD'00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, 161–168, New York, NY, USA. ACM Press.

[29] Lee, W. *A data mining framework for constructing features and models for intrusion detection systems (computer security, network security)*. PhD thesis, Columbia University, New York, NY, USA, 1999. Adviser-Stolfo„ Salvatore J.

[30] Mielikainen, T. 2004. Discovery of serial episodes from streams of events. In *Proceedings. 16th International Conference on Scientific and Statistical Database Management*.

[31] Fang, M., Shivakumar, N., Garcia-Molina, H., Motwani, R., & Ullman, J. D. 1998. Computing iceberg queries efficiently. In *VLDB '98: Proceedings of the 24rd International Conference on Very Large Data Bases*, 299–310, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[32] Gibbons, P. B. & Matias, Y. 1999. Synopsis data structures for massive data sets. In *SODA '99: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, 909–910, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

[33] Karp, R. M., Shenker, S., & Papadimitriou, C. H. 2003. A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst.*, 28(1), 51–55.

[34] Golab, L., DeHaan, D., Demaine, E. D., Lopez-Ortiz, A., & Munro, J. I. 2003. Identifying frequent items in sliding windows over on-line packet streams. In *IMC '03: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, 173–178, New York, NY, USA. ACM Press.

[35] Huang, K. Y. & Chang, C. H. 2005. A general model for mining synchronous periodic pattern in temporal database. *IEEE Transaction on Knowledge and Data Engineering (TKDE)*, 17(6), 776–785.

[36] Huang, K.-Y. & Chang, C.-H. *Efficient Mining Strategy for Frequent Serial Episodes in Temporal Database*, 824–829. Springer Berlin / Heidelberg, 2005.

[37] Cheng, J., Ke, Y., & Ng, W. 2008. A survey on algorithms for mining frequent itemsets over data streams. *Knowl. Inf. Syst.*, 16(1), 1–27.

[38] Lee, W. & Stolfo, S. J. 1998. Data mining approaches for intrusion detection.

[39] Lee, W., Stolfo, S. J., & Mok, K. W. 1999. Mining in a data-flow environment: experience in network intrusion detection. In *KDD'99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, 114–124, New York, NY, USA. ACM Press.

[40] Hätönen, K., Klemettinen, M., Mannila, H., Ronkainen, P., & Toivonen, H. 1996. Knowledge discovery from telecommunication network alarm databases. In *In 12th International Conference on Data Engineering (ICDE '96)*, 115–122, New Orleans, LA.

[41] Hätönen, K., Klemettinen, M., Mannila, H., Ronkainen, P., & Toivonen, H. 1996. Tasa: Telecommunication alarm sequence analyzer or: How to enjoy faults in your network.

[42] Hätönen, K. & Klemettinen, M. *Domain Structures in Filtering Irrelevant Frequent Patterns*, 289–305. Springer Berlin / Heidelberg, 2004.

[43] Devitt, A., Duffin, J., & Moloney, R. 2005. Topographical proximity for mining network alarm data. In *MineNet '05: Proceeding of the 2005 ACM SIGCOMM workshop on Mining network data*, 179–184, New York, NY, USA. ACM.

[44] Mooney, C. H. & Roddick, J. F. 2006. Marking time in sequence mining.

[45] Zaki. 1998. Efficient enumeration of frequent sequences. In *CIKM: ACM CIKM International Conference on Information and Knowledge Management*. ACM, SIGIR, and SIGMIS.

[46] Zaki, M. J. 2000. Sequence mining in categorical domains: Incorporating constraints. In *CIKM*, 422–429.

[47] Luo, J. & Bridges, S. M. 2000. Mining fuzzy association rules and fuzzy frequency episodes for intrusion detection. In *International Journal of Intelligent Systems*, volume 15.

[48] Luo, J., Bridges, S. M., & Rayford B.Vaughn, J. 2001. Fuzzy frequent episodes for real-time intrusion detection. In *The 10th IEEE International Conference on Fuzzy Systems*, volume 1, 368–371, Melbourne, Vic., Australia.

[49] Guan, J., xin Liu, D., & Wang, T. 2004. Applications of fuzzy data mining methods for intrusion detection systems. In *Computational Science and Its Applications–ICCSA 2004*, 706–714. Springer Berlin / Heidelberg.

[50] Julisch, K. & Dacier, M. 2002. Mining intrusion detection alarms for actionable knowledge. In *KDD'02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 366–375, New York, NY, USA. ACM Press.

[51] Huang, K.-Y. & Chang, C.-H. 2008. Efficient mining of frequent episodes from complex sequences. *Inf. Syst.*, 33(1), 96–114.

[52] Qin, M. & Hwang, K. Anomaly intrusion detection from internet traffic datamining. technical report. Technical report, Internet and Grid Computing Labortory, University of Southern California, Los Angeles, 2004.

[53] http://www.tracking-hackers.com/papers/honeypots.html.

[54] http://old.honeynet.org/papers/honeynet/.

[55] http://old.honeynet.org/papers/gen2/.

[56] https://projects.honeynet.org/honeywall/.

[57] http://old.honeynet.org/papers/cdrom/roo/.

[58] Leedy, P. D. & Ormrod, J. E. 2005. *Practical Research: Planning and Design*. Merrill Prentice Hall, Pearson Education International, 8th, international edition.

[59] Leavitt, N. 2009. Complex-event processing poised for growth. *Computer*, 42(4), 17–20.

[60] http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/docs/attackdb.html.

[61] Cheswick, W. R., Bellovin, S. M., & Rubin, A. D. 2003. *Firewalls and Internet Security: repelling the wily hacker*. Addison Wesley, 2nd edition.

A Attack Description

A.1 Remote-to-User Attacks

The goal of a Remote-to-User (R2U) attack is to login to a service on a local machine without having an account there. Such unauthorized access can be gained by various attacks like buffer overflows and weak policy exploits, for example, Dictionary and Spoofing attacks.

Systematical password guessing is the key idea of the Dictionary attack. It is effective for weak passwords and usual usernames. According to [60], the most popular services, supporting username- password authentication and, therefore, suitable for dictionary attacks, are telnet, ftp, pop, rlogin, and imap. The attack is usually represented by many periodical login attempts. [60] mentions that, to effectively detect dictionary attacks, an IDS needs a list of session protocols of every password authentication-based service, and that the attack signature may be build on failed login attempts. Depending on the failed login frequency for normal behavior, the attacks could be discovered if the threshold (for the number of failed login attempts in a fixed time period) is passed.

A.2 Probe

Mapping of services on many machines in a network provides the attackers with information, useful for other attacks that exploit vulnerabilities of the available services. An automated probing results in a map over weaknesses of the scanned machine. Portscan (discovery of listening ports) and Ping scanning are common Probe examples.

Many network scans are organized by the Nmap program. According to [60], the program allows various Portscans: TCP with SYN, FIN and/or ACK flags, UDP, and ICMP (Ping). Nmap also has several options, such as time intervals between packets/ports (from one second to several hours) and scanning order (sequential or random). 1999 DARPA Evaluation Dataset [13], for example, includes different Portscan attacks and the number of scanned ports per host varies from three to one thousand.

Another port scanning tool, called QueSO, sends a set of 7 TCP packets (4 usual and 3 rare) to a port, sometimes with some minutes delay. Then, the program compares the responses against a special database (classification of response types depending on OS and machine configurations) and allows the attacker to find out which operating system and computer is represented by the scanned IP-address. The common feature of all Probe attacks is the presence of several incoming network packets (TCP SYN, TCP FIN or UDP) to different ports within a period of time, from a minute to some days (slow Portscan). Detection of a slow attack requires a lot of resources, especially memory. Source hosts and source ports may be different if the attack is run by a group of attackers (low Portscan). [60] mentions that detection of low/slow Portscans is a challenge.

A.3 Denial of Service Attacks

The goal of a Denial of Service (DoS) attack is blocking of any legitimate activity on a service by flooding the service's resources. It may be achieved by various methods, commonly by exploiting bugs in TCP/IP implementations and network daemons. A distributed DoS, when many spurious hosts flood the real victim simultaneously, is especially powerful. According to [61], it is difficult to prevent and stop DoS attacks. The authors suggest filtering out the attack packets as an effective counteraction. It supposes following up strange packets and accurate feature and threshold choice.

SYN Flood is a denial of the TCP handshake service, which floods the half-open connections table of a limited size. When a half-open connection, initiated by the attacker, expires, the attacker may repeat the request. [60] refers to the SYN Flood program Neptune, which has different sending options like the source address, the number of packets, and the destination ports. It allows sending to all ports on a chosen machine as well. The authors suggest using an attack signature based on the number of SYN packets, sent simultaneously to the same IP from an unreachable IP. They also mention that, since a host-based IDS can monitor the data structure of tcpd connections, it can also alert when the connection table is close to its size limit.

A.4 Spam

Spam is commonly known as sending of undesirable messages. It is not harmful, but can be annoying and disturbing. Sometimes, it may cause denial of service: the user, who has received a lot of spam messages, starts to ignore even normal (legitimate) messages. A network layer spam example is SPIM – spam by Instant Messaging (IM). IM packets are supposed to be used by system administrators for sending information in form of pop-up messages to local network users. The Messenger service is turned on by default and may be misused by spammers from the Internet. When the service cannot be turned off, packet filtering is a possible solution. Source IP-address may help to distinguish normal packets from spam, as well as number and frequency of the Messenger packets.

B WINEPI Algorithms for Serial FED

B.1 Algorithm 2

Input: A set E of event types, an event sequence S over E, a set E of episodes, a window width win, and a frequency threshold min_fr.

Output: The collection $F(S; win; min\_fr)$ of frequent episodes.

Algorithm 2 generates a collection of frequent episodes from a given set of episodes epsilon from an event sequence S over a set of event types E. The window width win and the frequency threshold min_fr are also given. The levelwise FE discovery consists of the repeating of two steps: candidate generation and frequency check. We start with $n = 0$ and put all trivial episodes, consisting of 1 event, into the candidate collection. Algorithm 3, described below, generates candidates of order $n + 1$. The frequency checks for every collection are accomplished by Algorithm 5.

B.2 Algorithm 3

Input: A sorted array $F(l)$ of frequent serial episodes of size l.

Output: A sorted array of candidate serial episodes of size $l + 1$.

Algorithm 3 for generation of candidate serial episodes is based on the $Property1$ (on frequency in subepisode relations): frequent episodes of order $n + 1$ can be found only in the set of superepisodes of frequent episodes of order n. The algorithm operates with a new data structure – block, which is a sequence of all consecutive episodes, which differs only by the last event, from a given collection. Candidate episodes of the next order are generated from such blocks by appending various pairs of last events of block's episodes to the block base (common for all). For example, episode collection AA, AB, BA, BC, CC has 2 blocks: AA, AB and BA, BC. So, we have the following potential candidates of order 3: AAB, ABA, BAC and BCA. A value block_start is assigned to each episode to simplify block identification.

B.3 Algorithm 5

Input: A collection C of serial episodes, an event sequence $S = (s; T_s; T_e)$; a window width win; and a frequency threshold min_fr.

Output: The episodes of C that are frequent in S with respect to win and min_fr.

Algorithm 5 creates a database of frequent serial episodes from a given collection of candidates (generated by Algorithm 2). Before we describe the algorithm, we focus on the "incremental update" method proposed by [18]. Each window shift results in small changes in the window content – at the beginning and end,– while the main part stays unchanged. By using this fact, we can scan windows efficiently.

The key idea of the algorithm is the use of automata to present serial episodes. The first event of the episode, coming into the window, initializes an instance of the corresponding automaton.

1. $\mathcal{C}_1 := \{\alpha \in \mathcal{E} \mid |\alpha| = 1\}$;
2. $l := 1$;
3. *while* $\mathcal{C}_l \neq \emptyset$ *do*
4. /* Database pass (Algorithms 4 and 5): */
5. compute $\mathcal{F}_l := \{\alpha \in \mathcal{C}_l \mid fr(\alpha, \mathbf{s}, win) \geq min\_fr\}$;
6. $l := l + 1$;
7. /* Candidate generation (Algorithm 3): */
8. compute $\mathcal{C}_l := \{\alpha \in \mathcal{E} \mid |\alpha| = l$ and for all $\beta \in \mathcal{E}$ such that $\beta \prec \alpha$ and
9. $|\beta| < l$ we have $\beta \in \mathcal{F}_{|\beta|}\}$;
10. *for* all l *do* output $\mathcal{F}_l$;

Figure 21: Algorithm 2

1. $\mathcal{C}_{l+1} := \emptyset$;
2. $k := 0$;
3. *if* $l = 1$ *then for* $h := 1$ *to* $|\mathcal{F}_l|$ *do* $\mathcal{F}_l.block\_start[h] := 1$;
4. *for* $i := 1$ *to* $|\mathcal{F}_l|$ *do*
5. $current\_block\_start := k + 1$;
6. *for* $(j := i; \mathcal{F}_l.block\_start[j] = \mathcal{F}_l.block\_start[i]; j := j + 1)$ *do*
7. /* $\mathcal{F}_l[i]$ and $\mathcal{F}_l[j]$ have $l - 1$ first event types in common,
8. build a potential candidate α as their combination: */
9. *for* $x := 1$ *to* l *do* $\alpha[x] := \mathcal{F}_l[i][x]$;
10. $\alpha[l + 1] := \mathcal{F}_l[j][l]$;
11. /* Build and test subepisodes β that do not contain $\alpha[y]$: */
12. *for* $y := 1$ *to* $l - 1$ *do*
13. *for* $x := 1$ *to* $y - 1$ *do* $\beta[x] := \alpha[x]$;
14. *for* $x := y$ *to* l *do* $\beta[x] := \alpha[x + 1]$;
15. *if* β is not in $\mathcal{F}_l$ *then* continue with the next j at line 6;
16. /* All subepisodes are in $\mathcal{F}_l$, store α as candidate: */
17. $k := k + 1$;
18. $\mathcal{C}_{l+1}[k] := \alpha$;
19. $\mathcal{C}_{l+1}.block\_start[k] := current\_block\_start$;
20. output $\mathcal{C}_{l+1}$;

Figure 22: Algorithm 3

Active states of the instances represent incomplete occurrences of the episode in the current window. When an instance falls into the accepting state (it means that the window contains the whole episode) and there are no other accepted instances, the window start time value is stored into the corresponding episode's variable. When the event leaves the window, the instance is removed and, if it was the only accepted instance, the counter is increased by the number of previous windows, containing the episode.

The redundant instances (several automata in the same state) are removed by the last-in-last-out principle. Hence, the maximum number of instances is the episode order. The latest initialization time is stored for each instance. Efficient automata access is realized by lists $waits(event)$ containing all automata, waiting for the event, plus the number of the events in the automata (episodes). To avoid overwriting of important data, all transitions in form of $(episode, eventnumber, init.time)$ after a window shift are stored in a list. Old states are updated directly. Automata removal is simplified by the list $beginsat(t)$, containing all automata with initialization time t.

```
1.  /* Initialization: */
2.  for each α in C do
3.      for i := 1 to |α| do
4.          α.initialized[i] := 0;
5.          waits(α[i]) := ∅;
6.  for each α ∈ C do
7.      waits(α[1]) := waits(α[1]) ∪ {(α, 1)};
8.      α. freq_count := 0;
9.  for t := T_s − win to T_s − 1 do beginsat(t) := ∅;
10. /* Recognition: */
11. for start := T_s − win + 1 to T_e do
12.     /* Bring in new events to the window: */
13.     beginsat(start + win − 1) := ∅;
14.     transitions := ∅;
15.     for all events (A, t) in s such that t = start + win − 1 do
16.         for all (α, j) ∈ waits(A) do
17.             if j = |α| and α.initialized[j] = 0 then α.inwindow := start;
18.             if j = 1 then
19.                 transitions := transitions ∪ {(α, 1, start + win − 1)};
20.             else
21.                 transitions := transitions ∪ {(α, j, α.initialized[j − 1])};
22.                 beginsat(α.initialized[j − 1]) :=
23.                                 beginsat(α.initialized[j − 1]) \ {(α, j − 1)};
24.                 α.initialized[j − 1] := 0;
25.                 waits(A) := waits(A) \ {(α, j)};
26.     for all (α, j, t) ∈ transitions do
27.         α.initialized[j] := t;
28.         beginsat(t) := beginsat(t) ∪ {(α, j)};
29.         if j < |α| then waits(α[j + 1]) := waits(α[j + 1]) ∪ {(α, j + 1)};
30.     /* Drop out old events from the window: */
31.     for all (α, l) ∈ beginsat(start − 1) do
32.         if l = |α| then α.freq_count := α.freq_count − α.inwindow + start;
33.         else waits(α[l + 1]) := waits(α[l + 1]) \ {(α, l + 1)};
34.         α.initialized[l] := 0;
35. /* Output: */
36. for all episodes α in C do
37. if α.freq_count/(T_e − T_s + win − 1) ≥ min_fr then output α;
```

Figure 23: Algorithm 5

C The Algorithms' Implementation

```
//Collection of all possible atomic episodes—for the first round
for(int i=0; i<n; i++)
{
Episode e = new Episode(true,1);
        events.Add(new  Event(s.GetEventTypesList().GetValue(i).
            ToString(),0));
        e.SetEventAt(0,(Event)events[i]);
        C.Add(e); //atomic episodes—of one event only
}
//begin loop for a non–epmty collection
while(C.Count>0)
{
//Algorithm 5 for creating a serial episode database
n = C.Count;
        //initialization
        for(int j=0; j<n; j++)
                for(int i=0; i<ord; i++)
                {
                        ((Episode)C[j]).initialized[i]=–1;
                        ClearWaits(((Episode)C[j]).GetEventAt(i).
                            GetName());
                }
        for(int j=0; j<n; j++)
        {
                SetWaits(((Episode)C[j]).GetEventAt(0).GetName(),(new
                     Transition((Episode)C[j],0)));
                ((Episode)C[j]).freq_count = 0;
        }
        for(int t=Ts–win; t<Ts; t++)
                ClearBegs(t); //array of t lists of (episode,int)
        int k=1;                                    while((s.GetAt(k).
            GetData().GetTimeInstance()<Ts)&&(k<s.GetNumNodes()))
        k++;
        //recognition
        for(int start=Ts–win+1; start<Te+1; start++)
        {
                //bring in new events to the window
                ClearBegs(start+win–1);
                transitions.Clear();
                while(s.GetAt(k).GetData().GetTimeInstance()==start+
                    win–1)
                {
```

```
                        string tempA = s.GetAt(k).GetData().GetName()
                            ;
                        ArrayList temp_waits = GetWaits(tempA);
                        int cc = temp_waits.Count;
                        int deleted=0;
for(int i=0; i<cc; i++)
{
        temp_trans = ((Transition)temp_waits[i-deleted]);
        temp_alpha = temp_trans.GetEpisode();
        temp_j = temp_trans.GetVar1();
        if(temp_j == temp_alpha.GetNumEvents()-1)
        {
                if(temp_alpha.initialized[temp_j]==-1)
                        temp_alpha.inwindow = start;
                if(temp_j==0)
                        transitions.Add(new Transition(temp_alpha,0,
                            start+win-1));
                else if(temp_alpha.initialized[temp_j-1]>-1)
                {
                        transitions.Add(new Transition(temp_alpha,
                            temp_j, temp_alpha.initialized[temp_j-1]))
                            ;
                        DeleteBegs(temp_alpha.initialized[temp_j-1],
                            new Transition(temp_alpha,temp_j-1));
                        temp_alpha.initialized[temp_j-1] = -1;
                        DeleteWaits(tempA,i-deleted);
                        deleted++;
                }
        }
        if(k<s.GetNumNodes())
                k++;
        else
                break;
}
for(int i=0; i<transitions.Count; i++)
{
        temp_j = ((Transition)transitions[i]).GetVar1();
        temp_alpha = ((Transition)transitions[i]).GetEpisode();
        temp_t = ((Transition)transitions[i]).GetVar2();
        temp_alpha.initialized[temp_j] = temp_t;
        SetBegs(temp_t,(Transition)transitions[i]);
        if (temp_j<temp_alpha.GetNumEvents()-1)
        SetWaits(temp_alpha.GetEventAt(temp_j+1).GetName(),new
            Transition(temp_alpha,temp_j+1));
}

ArrayList temp_al = (ArrayList)GetBegs(start-1);
bool has_more;
for(int i=0; i<temp_al.Count; i++)
```

```
{
        if(((Transition)temp_al[i]).GetVar1()==((Transition)temp_al[i
            ]). GetEpisode().GetNumEvents()-1)
        {
                ((Transition)temp_al[i]).GetEpisode().freq_count = ((
                    Transition)temp_al[i]).GetEpisode().freq_count -
                    ((Transition)temp_al[i]).GetEpisode().inwindow +
                    start;
                has_more=false;
if(((Transition)temp_al[i]).GetEpisode().next_inwindow.Count>0)
                {
                        ((Transition)temp_al[i]).GetEpisode().
                            inwindow = ((int)((Transition)temp_al[i]).
                            GetEpisode().next_inwindow[0]);
                        ((Transition)temp_al[i]).GetEpisode().
                            next_inwindow.RemoveAt(0);
                }
        }
        else
                has_more = RemoveWaits(((Transition)temp_al[i]).
                    GetEpisode().GetEventAt(((Transition)temp_al[i]).
                    GetVar1()+1).GetName(),new Transition(((Transition
                    )temp_al[i]).GetEpisode(), ((Transition)temp_al[i
                    ]).GetVar1()+1));
        if(!has_more)
                ((Transition)temp_al[i]).GetEpisode().initialized[((
                    Transition)temp_al[i]).GetVar1()] = -1;
}
}
                //end of Algorithm 5

                //separate the frequent episodes
                F.Clear();
                int curr_bs=0;
                for(int j=0; j<n; j++)
                {
                        if(((Episode)C[j]).block_start==j)
                                curr_bs=F.Count;
                        if (!(((Episode)C[j]).freq_count < min_fr))
                        {
                                ((Episode)C[j]).block_start=curr_bs;
                                F.Add((Episode)C[j]);
                        }
                }
                C.Clear();
ord++;
                int l=ord-1;
if((ord<4)&&(ord<win))
{
```

```
        // Algorithm 3 for generation of candidate serial episodes
        int kk = 0;
        int current_block_start;
        Episode alpha;
        Episode beta;
bool go;
        if(l==1) //no block exists for episodes of order 1
                for(int h=0; h<F.Count; h++)
                        ((Episode)F[h]).block_start = 0;
                for(int ii=0; ii<F.Count; ii++)
//for all episodes from the given collection do
                {
                        current_block_start = kk;
        go = ((Episode)F[((Episode)F[ii]).block_start]).block_start
           ==((Episode)F[ii]).block_start;
//is block_start set right?
//for all episodes from the block do
        for(int jj=((Episode)F[ii]).block_start; go; jj++)
        {
                alpha = new Episode(true,ord);
                beta = new Episode(true, l);
                for(int x=0; x<l; x++)
//for all events of the block base do
                alpha.SetEventAt(x,((Episode)F[ii]).GetEventAt(x));
alpha.SetEventAt(l,((Episode)F[jj]).GetEventAt(l-1));
                for(int y=0; y<l-1; y++)
//for all events of the block base do
                {
                        for(int x=0; x<y; x++)
                                beta.SetEventAt(x,alpha.GetEventAt(x)
                                    );
                        for(int x=y; x<l; x++)
                                beta.SetEventAt(x,alpha.GetEventAt(x
                                    +1));
                        for(int z=0; z<F.Count; z++)
                                if(beta.Equals((Episode)F[z]))
                                {
                                        go=false;
                                        y=l-1;
                                        break;
                                }
                }
                if(go) //if all subepisodes of alpha are in
                    collection
                {
                        C.Add(alpha);
                        ((Episode)C[kk]).block_start =
                            current_block_start;
                        kk++;
```

```
                        if(jj<F.Count-1)
go = ((Episode)F[jj+1]).block_start ==((Episode)F[ii]).block_start;
                        else go = false;
                }
        }
//end of the Algorithm 3
        }
} //end loop for the collection
                }
        }
}
```

D Experimental Event Sequence

1 202.97.238.195_60064_cap_Messenger
2 128.39.44.100_60064_cap_ICMP
3 202.97.238.194_43455_1027_Messenger
4 128.39.44.100_43455_1027_ICMP
5 133.9.8.22_48778_ssh_TCP
7 218.10.137.141_33984_cap_Messenger
8 128.39.44.100_33984_cap_ICMP
9 218.10.137.141_34043_cap_Messenger
10 128.39.44.100_34043_cap_ICMP
11 65.203.50.5_intraintra_http_TCP
13 65.203.50.5_intraintra_http_TCP
14 65.203.50.5_intraintra_http_HTTP
17 65.203.50.5_intraintra_http_TCP
18 65.203.50.5_intraintra_http_TCP
20 65.203.50.5_netwatcher-db_http_TCP
22 65.203.50.5_netwatcher-db_http_TCP
23 65.203.50.5_netwatcher-db_http_HTTP
26 65.203.50.5_netwatcher-db_http_TCP
27 65.203.50.5_netwatcher-db_http_TCP
29 65.203.50.5_isns_http_TCP
31 65.203.50.5_isns_http_TCP
32 65.203.50.5_isns_http_TCP
34 65.203.50.5_isns_http_TCP
41 218.9.66.130_us-gv_ms-sql-m_UDP
42 128.39.44.100_us-gv_ms-sql-m_ICMP
43 61.136.186.46_esbroker_ms-sql-m_UDP
44 128.39.44.100_esbroker_ms-sql-m_ICMP
45 218.10.137.142_60596_1027_Messenger
46 128.39.44.100_60596_1027_ICMP
47 218.10.137.142_60596_cap_Messenger
48 128.39.44.100_60596_cap_ICMP
49 218.10.137.142_60597_cap_Messenger
50 128.39.44.100_60597_cap_ICMP
51 203.94.243.191_ardus-trns_ms-sql-m_UDP
52 128.39.44.100_ardus-trns_ms-sql-m_ICMP
53 125.147.35.235_34327_ssh_TCP
55 221.208.208.86_44675_cap_Messenger
56 128.39.44.100_44675_cap_ICMP
57 218.10.137.140_57916_cap_Messenger
58 218.10.137.140_57916_1027_Messenger
59 128.39.44.100_57916_cap_ICMP

60 128.39.44.100_57916_1027_ICMP
61 218.10.137.140_57917_cap_Messenger
62 128.39.44.100_57917_cap_ICMP
63 218.64.237.219_buddy-draw_ms-sql-m_UDP
64 128.39.44.100_buddy-draw_ms-sql-m_ICMP
65 202.97.238.199_52246_cap_Messenger
66 128.39.44.100_52246_cap_ICMP
67 201.70.98.198_33757_ssh_TCP
69 202.120.43.208_10267_ndl-aas_TCP
70 202.120.43.208_10267_http_TCP
73 202.120.43.208_10267_socks_TCP
75 202.120.43.208_10267_http-alt_TCP
77 202.120.43.208_10267_cpq-wbem_TCP
79 202.120.43.208_10267_ddi-tcp-1_TCP
81 202.120.43.208_10267_http_TCP
82 202.120.43.208_dell-rm-port_http_TCP
84 202.120.43.208_dell-rm-port_http_TCP
85 202.120.43.208_dell-rm-port_http_TCP
88 202.120.43.208_dell-rm-port_http_HTTP
90 202.120.43.208_10267_http_TCP
92 202.120.43.208_10267_ndl-aas_TCP
94 202.120.43.208_10267_socks_TCP
96 202.120.43.208_10267_http-alt_TCP
98 202.120.43.208_10267_cpq-wbem_TCP
100 202.120.43.208_10267_ddi-tcp-1_TCP
102 202.120.43.208_10267_http_TCP
103 202.120.43.208_izm_http_TCP
105 202.120.43.208_izm_http_TCP
106 202.120.43.208_izm_http_TCP
109 202.120.43.208_izm_http_TCP
110 202.120.43.208_izm_http_HTTP
112 221.208.208.212_33450_cap_Messenger
113 128.39.44.100_33450_cap_ICMP
114 202.120.43.208_10267_http_TCP
116 202.120.43.208_10267_ndl-aas_TCP
118 202.120.43.208_10267_http-alt_TCP
120 202.120.43.208_10267_cpq-wbem_TCP
122 202.120.43.208_10267_ddi-tcp-1_TCP
124 202.120.43.208_10267_http_TCP
125 202.120.43.208_4204_http_TCP
127 202.120.43.208_4204_http_TCP
128 202.120.43.208_4204_http_TCP
131 202.120.43.208_4204_http_TCP
132 202.120.43.208_4204_http_HTTP
134 202.120.43.208_10267_http_TCP
136 202.120.43.208_10267_ndl-aas_TCP
138 202.120.43.208_10267_socks_TCP
140 202.120.43.208_10267_http-alt_TCP
142 202.120.43.208_10267_cpq-wbem_TCP

144 202.120.43.208_10267_ddi-tcp-1_TCP
146 202.120.43.208_10267_http_TCP
147 202.120.43.208_itose_http_TCP
149 202.120.43.208_itose_http_TCP
150 202.120.43.208_itose_http_TCP
153 202.120.43.208_itose_http_TCP
154 202.120.43.208_itose_http_HTTP
156 61.188.38.13_kyoceranetdev_ms-sql-m_UDP
157 128.39.44.100_kyoceranetdev_ms-sql-m_ICMP
158 202.120.43.208_10267_http_TCP
160 202.120.43.208_10267_ndl-aas_TCP
162 202.120.43.208_10267_http-alt_TCP
164 202.120.43.208_10267_cpq-wbem_TCP
166 202.120.43.208_10267_ddi-tcp-1_TCP
168 202.120.43.208_10267_http_TCP
169 202.120.43.208_simple-push-s_http_TCP
171 202.120.43.208_simple-push-s_http_TCP
172 202.120.43.208_simple-push-s_http_TCP
175 202.120.43.208_simple-push-s_http_TCP
176 202.120.43.208_simple-push-s_http_HTTP
178 60.249.22.224_36764_ssh_TCP
180 202.173.11.34_x11_ssc-agent_TCP
182 221.208.208.93_55683_cap_Messenger
183 128.39.44.100_55683_cap_ICMP
184 125.211.214.141_http_24667_TCP
185 138.100.10.251_vtu-comms_gds_db_TCP
187 138.100.10.251_vtu-comms_gds_db_TCP
189 138.100.10.251_vtu-comms_gds_db_TCP
191 218.61.192.73_http_24667_TCP
192 60.28.175.36_http_24667_TCP
193 218.61.192.73_http_24667_TCP
194 203.197.178.171_35830_ssh_TCP
196 125.211.214.141_http_24667_TCP
197 202.103.180.105_swrmi_ms-sql-m_UDP
198 128.39.44.100_swrmi_ms-sql-m_ICMP
199 221.209.110.13_40031_1027_Messenger
200 128.39.44.100_40031_1027_ICMP
201 221.209.110.13_40031_cap_Messenger
202 128.39.44.100_40031_cap_ICMP
203 221.208.208.212_55233_1027_Messenger
204 128.39.44.100_55233_1027_ICMP
205 221.208.208.212_55233_cap_Messenger
206 128.39.44.100_55233_cap_ICMP
207 64.34.169.174_starschool_http_TCP
209 64.34.169.174_starschool_http_TCP
210 64.34.169.174_starschool_http_HTTP
213 64.34.169.174_starschool_http_TCP
214 64.34.169.174_starschool_http_TCP
216 122.116.112.162_10391_socks_TCP

218 64.34.169.174_dna_http_TCP
220 64.34.169.174_dna_http_TCP
221 64.34.169.174_dna_http_HTTP
224 64.34.169.174_dna_http_TCP
225 64.34.169.174_dna_http_TCP
227 136.1.7.55_ds-mail_ms-sql-m_UDP
228 128.39.44.100_ds-mail_ms-sql-m_ICMP
229 202.101.235.100_socks_ms-sql-m_UDP
230 128.39.44.100_socks_ms-sql-m_ICMP
231 218.38.34.226_x11_ms-sql-s_TCP
233 218.38.34.226_x11_ms-sql-s_TCP
234 218.38.34.226_netopia-vo2_ms-sql-s_TCP
236 218.38.34.226_netopia-vo2_ms-sql-s_TCP
237 218.38.34.226_netopia-vo2_ms-sql-s_TDS
239 218.38.34.226_netopia-vo2_ms-sql-s_TDS
241 218.38.34.226_netopia-vo2_ms-sql-s_TDS
243 218.38.34.226_netopia-vo2_ms-sql-s_TDS
246 218.38.34.226_netopia-vo2_ms-sql-s_TCP
247 218.38.34.226_netopia-vo2_ms-sql-s_TCP
249 218.38.34.226_mtp_ms-sql-s_TCP
251 218.38.34.226_mtp_ms-sql-s_TCP
252 218.38.34.226_mtp_ms-sql-s_TDS
254 218.38.34.226_mtp_ms-sql-s_TDS
256 218.38.34.226_mtp_ms-sql-s_TDS
258 218.38.34.226_mtp_ms-sql-s_TDS
261 218.38.34.226_mtp_ms-sql-s_TCP
262 218.38.34.226_mtp_ms-sql-s_TCP
264 218.38.34.226_bcs-lmserver_ms-sql-s_TCP
266 218.38.34.226_bcs-lmserver_ms-sql-s_TCP
267 218.38.34.226_bcs-lmserver_ms-sql-s_TDS
269 218.38.34.226_bcs-lmserver_ms-sql-s_TDS
271 218.38.34.226_bcs-lmserver_ms-sql-s_TDS
273 218.38.34.226_bcs-lmserver_ms-sql-s_TDS
276 218.38.34.226_bcs-lmserver_ms-sql-s_TCP
277 218.38.34.226_bcs-lmserver_ms-sql-s_TCP
279 218.38.34.226_netop-school_ms-sql-s_TCP
281 218.38.34.226_netop-school_ms-sql-s_TCP
282 218.38.34.226_netop-school_ms-sql-s_TDS
284 218.38.34.226_netop-school_ms-sql-s_TDS
286 218.38.34.226_netop-school_ms-sql-s_TDS
288 218.38.34.226_netop-school_ms-sql-s_TDS
291 218.38.34.226_netop-school_ms-sql-s_TCP
292 218.38.34.226_netop-school_ms-sql-s_TCP
294 218.38.34.226_stun-p2_ms-sql-s_TCP
296 218.38.34.226_stun-p2_ms-sql-s_TCP
297 218.38.34.226_stun-p2_ms-sql-s_TDS
299 218.38.34.226_stun-p2_ms-sql-s_TDS
301 218.38.34.226_stun-p2_ms-sql-s_TDS
303 218.38.34.226_stun-p2_ms-sql-s_TDS

306 218.38.34.226_stun–p2_ms–sql–s_TCP
307 218.38.34.226_stun–p2_ms–sql–s_TCP
309 210.51.47.31_x11_ms–sql–s_TCP
311 210.51.47.31_x11_ms–sql–s_TCP
312 200.118.111.244_60250_ssh_TCP
314 64.34.169.174_pc–telecommute_http_TCP
316 64.34.169.174_pc–telecommute_http_TCP
317 64.34.169.174_pc–telecommute_http_HTTP
320 64.34.169.174_pc–telecommute_http_TCP
321 64.34.169.174_pc–telecommute_http_TCP
323 221.208.208.99_49917_cap_Messenger
324 128.39.44.100_49917_cap_ICMP
325 221.208.208.99_49917_1027_Messenger
326 128.39.44.100_49917_1027_ICMP
327 221.208.208.99_49918_1027_Messenger
328 128.39.44.100_49918_1027_ICMP
329 202.97.238.227_43224_cap_Messenger
330 128.39.44.100_43224_cap_ICMP
331 61.34.63.19_56881_telnet_TCP
333 211.174.58.166_x11_ms–sql–s_TCP
335 211.174.58.166_x11_ms–sql–s_TCP
336 211.174.58.166_tqdata_ms–sql–s_TCP
338 211.174.58.166_tqdata_ms–sql–s_TCP
339 211.174.58.166_tqdata_ms–sql–s_TDS
341 211.174.58.166_tqdata_ms–sql–s_TDS
343 211.174.58.166_tqdata_ms–sql–s_TDS
345 211.174.58.166_tqdata_ms–sql–s_TDS
348 211.174.58.166_tqdata_ms–sql–s_TCP
349 211.174.58.166_tqdata_ms–sql–s_TCP
351 211.174.58.166_radwiz–nms–srv_ms–sql–s_TCP
353 211.174.58.166_radwiz–nms–srv_ms–sql–s_TCP
354 211.174.58.166_radwiz–nms–srv_ms–sql–s_TDS
356 211.174.58.166_radwiz–nms–srv_ms–sql–s_TDS
358 211.174.58.166_radwiz–nms–srv_ms–sql–s_TDS
359 211.174.58.166_radwiz–nms–srv_ms–sql–s_TDS
363 211.174.58.166_radwiz–nms–srv_ms–sql–s_TCP
364 211.174.58.166_radwiz–nms–srv_ms–sql–s_TCP
366 211.174.58.166_desktop–dna_ms–sql–s_TCP
368 211.174.58.166_desktop–dna_ms–sql–s_TCP
369 211.174.58.166_desktop–dna_ms–sql–s_TDS
371 211.174.58.166_desktop–dna_ms–sql–s_TDS
373 211.174.58.166_desktop–dna_ms–sql–s_TDS
374 211.174.58.166_desktop–dna_ms–sql–s_TDS
378 211.174.58.166_desktop–dna_ms–sql–s_TCP
379 211.174.58.166_desktop–dna_ms–sql–s_TCP
381 211.174.58.166_www–dev_ms–sql–s_TCP
383 211.174.58.166_www–dev_ms–sql–s_TCP
384 211.174.58.166_www–dev_ms–sql–s_TDS
386 211.174.58.166_www–dev_ms–sql–s_TDS

388 211.174.58.166_www–dev_ms–sql–s_TDS
389 211.174.58.166_www–dev_ms–sql–s_TDS
393 211.174.58.166_www–dev_ms–sql–s_TCP
394 211.174.58.166_www–dev_ms–sql–s_TCP
395 211.174.58.166_www–dev_ms–sql–s_TCP
397 211.174.58.166_aic–oncrpc_ms–sql–s_TCP
399 211.174.58.166_aic–oncrpc_ms–sql–s_TCP
400 211.174.58.166_aic–oncrpc_ms–sql–s_TDS
402 211.174.58.166_aic–oncrpc_ms–sql–s_TDS
404 211.174.58.166_aic–oncrpc_ms–sql–s_TDS
405 211.174.58.166_aic–oncrpc_ms–sql–s_TDS
409 211.174.58.166_aic–oncrpc_ms–sql–s_TCP
410 211.174.58.166_aic–oncrpc_ms–sql–s_TCP
412 202.97.238.204_44572_1027_Messenger
413 128.39.44.100_44572_1027_ICMP
414 202.97.238.204_44572_1027_Messenger
415 128.39.44.100_44572_1027_ICMP
416 209.216.205.250_4961_timbuktu_TCP
418 61.141.5.10_gte–samp_ssh_TCP
420 195.154.63.19_murray_ssc–agent_TCP
422 195.154.63.19_murray_ssc–agent_TCP
424 195.154.63.19_murray_ssc–agent_TCP
426 195.154.63.19_commlinx–avl_enpp_TCP
428 195.154.63.19_commlinx–avl_enpp_TCP
430 195.154.63.19_commlinx–avl_enpp_TCP
432 195.154.63.19_de–cache–query_vnc–server_TCP
434 195.154.63.19_de–cache–query_vnc–server_TCP
436 195.154.63.19_de–cache–query_vnc–server_TCP
438 195.154.63.19_ewall_5901_TCP
440 195.154.63.19_ewall_5901_TCP
442 195.154.63.19_ewall_5901_TCP
444 210.51.44.199_9065_5491_TCP
445 85.114.130.179_30718_telnet_TCP
447 221.208.208.96_51053_cap_Messenger
448 128.39.44.100_51053_cap_ICMP
449 221.208.208.96_51310_1027_Messenger
450 128.39.44.100_51310_1027_ICMP
451 91.121.72.120_ncl_65101_TCP
453 91.121.72.120_ncl_65101_TCP
455 91.121.72.120_ncl_65101_TCP
457 221.209.110.50_45187_cap_Messenger
458 221.209.110.50_45187_1027_Messenger
459 128.39.44.100_45187_cap_ICMP
460 128.39.44.100_45187_1027_ICMP
461 71.248.114.105_31707_5901_TCP
463 71.248.114.105_31707_5901_TCP
465 71.248.114.105_31707_5901_TCP
467 219.238.41.155_x11_ssc–agent_TCP
469 202.97.238.195_58786_1027_Messenger

470 128.39.44.100_58786_1027_ICMP
471 61.184.104.106_50814_ssh_TCP
473 61.184.104.106_50814_ssh_TCP
475 64.34.169.174_simple-tx-rx_http_TCP
477 64.34.169.174_simple-tx-rx_http_TCP
478 64.34.169.174_simple-tx-rx_http_HTTP
481 64.34.169.174_simple-tx-rx_http_TCP
482 64.34.169.174_simple-tx-rx_http_TCP
484 202.97.238.194_54766_cap_Messenger
485 128.39.44.100_54766_cap_ICMP
486 202.97.238.199_55085_1027_Messenger
487 128.39.44.100_55085_1027_ICMP
488 202.97.238.199_55086_cap_Messenger
489 128.39.44.100_55086_cap_ICMP
490 206.193.196.188_40156_ndmp_TCP
492 221.209.110.50_59025_cap_Messenger
493 128.39.44.100_59025_cap_ICMP
494 210.51.44.199_9065_5491_TCP
495 218.83.152.251_distinct_31925_TCP
509 62.193.225.104_58548_ftp_TCP
510 62.193.225.104_58747_ftp_TCP
511 62.193.225.104_58548_ftp_TCP
512 62.193.225.104_58548_ftp_TCP
513 62.193.225.104_58747_ftp_TCP
514 62.193.225.104_58747_ftp_TCP
515 62.193.225.104_58747_ftp_FTP
516 62.193.225.104_58747_ftp_FTP
517 62.193.225.104_58747_ftp_FTP
518 62.193.225.104_58747_ftp_FTP
519 62.193.225.104_58747_ftp_FTP
520 62.193.225.104_58747_ftp_FTP
521 62.193.225.104_58747_ftp_FTP
522 62.193.225.104_58747_ftp_FTP
523 62.193.225.104_58747_ftp_TCP
524 62.193.225.104_58895_ftp_TCP
525 62.193.225.104_58895_ftp_TCP
526 62.193.225.104_58747_ftp_TCP
527 62.193.225.104_58747_ftp_TCP
528 62.193.225.104_58895_ftp_TCP
529 62.193.225.104_58895_ftp_FTP
530 62.193.225.104_58895_ftp_FTP
531 62.193.225.104_58895_ftp_FTP
532 62.193.225.104_58895_ftp_FTP
533 62.193.225.104_58895_ftp_FTP
534 62.193.225.104_58895_ftp_FTP
535 62.193.225.104_58895_ftp_FTP
536 62.193.225.104_58895_ftp_FTP
537 62.193.225.104_58895_ftp_TCP
538 62.193.225.104_59035_ftp_TCP

539 62.193.225.104_58895_ftp_TCP
540 62.193.225.104_59035_ftp_TCP
541 62.193.225.104_58895_ftp_TCP
542 62.193.225.104_59035_ftp_TCP
543 62.193.225.104_59035_ftp_FTP
544 62.193.225.104_59035_ftp_FTP
545 62.193.225.104_59035_ftp_FTP
546 62.193.225.104_59035_ftp_FTP
547 62.193.225.104_59035_ftp_FTP
548 62.193.225.104_59035_ftp_FTP
549 62.193.225.104_59035_ftp_FTP
550 62.193.225.104_59035_ftp_FTP
551 62.193.225.104_59264_ftp_TCP
552 62.193.225.104_59035_ftp_TCP
553 62.193.225.104_59264_ftp_TCP
554 62.193.225.104_59035_ftp_TCP
555 62.193.225.104_59035_ftp_TCP
556 62.193.225.104_59264_ftp_TCP
557 62.193.225.104_59264_ftp_FTP
558 62.193.225.104_59264_ftp_FTP
559 62.193.225.104_59264_ftp_FTP
560 62.193.225.104_59264_ftp_FTP
561 62.193.225.104_59264_ftp_FTP
562 62.193.225.104_59264_ftp_FTP
563 62.193.225.104_59264_ftp_FTP
564 62.193.225.104_59264_ftp_FTP
565 62.193.225.104_59264_ftp_TCP
566 62.193.225.104_59537_ftp_TCP
567 62.193.225.104_59264_ftp_TCP
568 62.193.225.104_59537_ftp_TCP
569 62.193.225.104_59264_ftp_TCP
570 62.193.225.104_59537_ftp_TCP
571 62.193.225.104_59537_ftp_FTP
572 62.193.225.104_59537_ftp_FTP
573 62.193.225.104_59537_ftp_FTP
574 62.193.225.104_59537_ftp_FTP
575 62.193.225.104_59537_ftp_FTP
576 62.193.225.104_59537_ftp_FTP
577 62.193.225.104_59537_ftp_FTP
578 62.193.225.104_59537_ftp_FTP
579 62.193.225.104_59537_ftp_TCP
580 62.193.225.104_59750_ftp_TCP
581 62.193.225.104_59537_ftp_TCP
582 62.193.225.104_59537_ftp_TCP
583 62.193.225.104_59750_ftp_TCP
584 62.193.225.104_59750_ftp_TCP
585 62.193.225.104_59750_ftp_FTP
586 62.193.225.104_59750_ftp_FTP
587 62.193.225.104_59750_ftp_FTP

588 62.193.225.104_59750_ftp_FTP
589 62.193.225.104_59750_ftp_FTP
590 62.193.225.104_59750_ftp_FTP
591 62.193.225.104_59750_ftp_FTP
592 62.193.225.104_59750_ftp_FTP
593 139.142.58.248_zion-lm_http_TCP
594 139.142.58.248_zion-lm_http_TCP
595 139.142.58.248_zion-lm_http_HTTP
596 139.142.58.248_zion-lm_http_TCP
597 139.142.58.248_rgtp_http_TCP
598 139.142.58.248_zion-lm_http_TCP
599 139.142.58.248_rgtp_http_TCP
600 139.142.58.248_rgtp_http_HTTP
601 139.142.58.248_rgtp_http_TCP
602 139.142.58.248_rgtp_http_TCP
603 139.142.58.248_eicon-x25_http_TCP
604 139.142.58.248_eicon-x25_http_TCP
605 139.142.58.248_eicon-x25_http_HTTP
606 139.142.58.248_eicon-x25_http_TCP
607 139.142.58.248_eicon-x25_http_TCP
608 139.142.58.248_apri-lm_http_TCP
609 139.142.58.248_apri-lm_http_TCP
610 139.142.58.248_apri-lm_http_HTTP
611 139.142.58.248_apri-lm_http_TCP
612 139.142.58.248_interhdl_elmd_http_TCP
613 139.142.58.248_apri-lm_http_TCP
614 139.142.58.248_interhdl_elmd_http_TCP
615 139.142.58.248_interhdl_elmd_http_HTTP
616 139.142.58.248_interhdl_elmd_http_TCP
617 139.142.58.248_interhdl_elmd_http_TCP
618 139.142.58.248_ibm_wrless_lan_http_TCP
619 139.142.58.248_ibm_wrless_lan_http_TCP
620 139.142.58.248_ibm_wrless_lan_http_HTTP
621 139.142.58.248_ibm_wrless_lan_http_TCP
622 139.142.58.248_ibm_wrless_lan_http_TCP
623 139.142.58.248_csdm_http_TCP
624 139.142.58.248_csdm_http_TCP
625 139.142.58.248_csdm_http_HTTP
626 139.142.58.248_csdm_http_TCP
627 139.142.58.248_csdm_http_TCP
628 139.142.58.248_taligent-lm_http_TCP
629 139.142.58.248_taligent-lm_http_TCP
630 139.142.58.248_taligent-lm_http_HTTP
631 139.142.58.248_taligent-lm_http_TCP
632 139.142.58.248_taligent-lm_http_TCP
633 139.142.58.248_miteksys-lm_http_TCP
634 139.142.58.248_miteksys-lm_http_TCP
635 139.142.58.248_miteksys-lm_http_HTTP
636 139.142.58.248_miteksys-lm_http_TCP

637 139.142.58.248_miteksys-lm_http_TCP
638 200.93.217.173_45598_ssh_TCP
639 122.30.179.186_attachmate-g32_ms-sql-m_UDP
640 128.39.44.100_attachmate-g32_ms-sql-m_ICMP
641 136.1.7.55_univision_ms-sql-m_UDP
642 128.39.44.100_univision_ms-sql-m_ICMP
643 202.97.238.203_52499_1027_Messenger
644 128.39.44.100_52499_1027_ICMP
645 202.97.238.203_52499_cap_Messenger
646 128.39.44.100_52499_cap_ICMP
647 202.97.238.204_51455_cap_Messenger
648 128.39.44.100_51455_cap_ICMP
649 61.152.76.195_http_24079_TCP
650 84.19.187.119_ssrip_mysql_TCP
651 84.19.187.119_ssrip_mysql_TCP
652 84.19.187.119_ssrip_mysql_TCP
663 62.193.225.104_60997_ftp_TCP
664 62.193.225.104_60768_ftp_TCP
665 62.193.225.104_60997_ftp_TCP
666 62.193.225.104_60768_ftp_TCP
667 62.193.225.104_60768_ftp_TCP
668 62.193.225.104_60997_ftp_TCP
669 62.193.225.104_60997_ftp_FTP
670 62.193.225.104_60997_ftp_FTP
671 62.193.225.104_60997_ftp_FTP
672 62.193.225.104_60997_ftp_FTP
673 62.193.225.104_60997_ftp_FTP
674 62.193.225.104_60997_ftp_FTP
675 62.193.225.104_60997_ftp_FTP
676 62.193.225.104_60997_ftp_FTP
677 62.193.225.104_33014_ftp_TCP
678 62.193.225.104_60997_ftp_TCP
679 62.193.225.104_33014_ftp_TCP
680 62.193.225.104_60997_ftp_TCP
681 62.193.225.104_60997_ftp_TCP
682 62.193.225.104_33014_ftp_TCP
683 62.193.225.104_33014_ftp_FTP
684 62.193.225.104_33014_ftp_FTP
685 62.193.225.104_33014_ftp_FTP
686 62.193.225.104_33014_ftp_FTP
687 62.193.225.104_33014_ftp_FTP
688 62.193.225.104_33014_ftp_FTP
689 62.193.225.104_33014_ftp_FTP
690 62.193.225.104_33014_ftp_FTP
691 62.193.225.104_33014_ftp_TCP
692 62.193.225.104_33260_ftp_TCP
693 62.193.225.104_33014_ftp_TCP
694 62.193.225.104_33260_ftp_TCP
695 62.193.225.104_33014_ftp_TCP

696 62.193.225.104_33260_ftp_TCP
697 62.193.225.104_33260_ftp_FTP
698 62.193.225.104_33260_ftp_FTP
699 62.193.225.104_33260_ftp_FTP
700 62.193.225.104_33260_ftp_FTP
701 62.193.225.104_33260_ftp_FTP
702 62.193.225.104_33260_ftp_FTP
703 62.193.225.104_33260_ftp_FTP
704 62.193.225.104_33260_ftp_FTP
705 62.193.225.104_33260_ftp_TCP
706 62.193.225.104_33459_ftp_TCP
707 62.193.225.104_33260_ftp_TCP
708 62.193.225.104_33459_ftp_TCP
709 62.193.225.104_33260_ftp_TCP
710 62.193.225.104_33459_ftp_TCP
711 62.193.225.104_33459_ftp_FTP
712 62.193.225.104_33459_ftp_FTP
713 62.193.225.104_33459_ftp_FTP
714 62.193.225.104_33459_ftp_FTP
715 62.193.225.104_33459_ftp_FTP
716 62.193.225.104_33459_ftp_FTP
717 62.193.225.104_33459_ftp_FTP
718 62.193.225.104_33459_ftp_FTP
719 62.193.225.104_33581_ftp_TCP
720 62.193.225.104_33459_ftp_TCP
721 62.193.225.104_33581_ftp_TCP
722 62.193.225.104_33459_ftp_TCP
723 62.193.225.104_33459_ftp_TCP
724 62.193.225.104_33581_ftp_TCP
725 62.193.225.104_33581_ftp_FTP
726 62.193.225.104_33581_ftp_FTP
727 62.193.225.104_33581_ftp_FTP
728 62.193.225.104_33581_ftp_FTP
729 62.193.225.104_33581_ftp_FTP
730 62.193.225.104_33581_ftp_FTP
731 62.193.225.104_33581_ftp_FTP
732 62.193.225.104_33581_ftp_FTP
733 62.193.225.104_33581_ftp_TCP
734 62.193.225.104_33754_ftp_TCP
735 62.193.225.104_33581_ftp_TCP
736 62.193.225.104_33754_ftp_TCP
737 62.193.225.104_33581_ftp_TCP
738 62.193.225.104_33754_ftp_TCP
739 62.193.225.104_33754_ftp_FTP
740 62.193.225.104_33754_ftp_FTP
741 62.193.225.104_33754_ftp_FTP
742 62.193.225.104_33754_ftp_FTP
743 62.193.225.104_33754_ftp_FTP
744 62.193.225.104_33754_ftp_FTP

745 62.193.225.104_33754_ftp_FTP
746 62.193.225.104_33754_ftp_FTP
747 62.193.225.104_33754_ftp_TCP
748 62.193.225.104_33991_ftp_TCP
749 62.193.225.104_33754_ftp_TCP
750 62.193.225.104_33754_ftp_TCP
751 62.193.225.104_33991_ftp_TCP
752 62.193.225.104_33991_ftp_TCP
753 62.193.225.104_33991_ftp_FTP
754 62.193.225.104_33991_ftp_FTP
755 62.193.225.104_33991_ftp_FTP
756 62.193.225.104_33991_ftp_FTP
757 62.193.225.104_33991_ftp_FTP
758 62.193.225.104_33991_ftp_FTP
759 62.193.225.104_33991_ftp_FTP
760 62.193.225.104_33991_ftp_FTP