Modeling and Describing Misuse Scenarios Using Signature-Nets and Event Description Language

Ulrich Flegel, Hochschule für Technik Stuttgart, Michael Meier, Fraunhofer FKIE Wachtberg/Technische Universität Dortmund

Summary In the area of intrusion detection the misuse detection approach assumes that relevant activity violating security policies is known a priori and it provides for fast intrusion detection with low false alarm rate, thereby complementing the anomaly detection approach. Hence, misuse detection is an indispensable ingredient to a suitable strategy for intrusion detection. Misuse detection calls for a comprehensive framework for modeling and analysing attack activity. This contribution presents the highly expressive and modular Signature-net framework. Signature-nets allow for visual modeling and simulation of attack detection signatures, as well as for formal analysis. The framework lends itself to highly optimized implementation and may be flexibly deployed, e.g., for network-based and host-based intrusion detection or alarm correlation.

Keywords K.6.5 [Computing Milieux: Management of Computing and Informatik Systems: Security and Protection]; security, intrusion detection, misuse detection, signature, signature net, EDL

1 Models of Misuse Scenarios
In the area of computer security intrusion detection systems (IDSs) play an important role for the automatic identification of attacks. In addition to preventive security mechanisms they provide post-mortem detection capabilities. A main problem for the detection of security violations using misuse detection systems is the modeling and description of misuse scenarios to be detected.

In the following we consider services in the sense of a running software system that provides some service to its users. Such a service is assumed to contain an audit component that observes the service activity.
Misuse scenarios are defined as activity considered to violate the security policy of the service organization. Observations of activity are manifested in the form of events, which are embedded in audit records. An ordered set of events is also denoted as the manifestation of certain activity, if these events are symptomatic to be observed when this activity takes place. An ordered set of audit records is denoted as audit data.

An event exhibits an event designator allowing event-discrimination, an event type specifying the number and types of the included features, and as many pairs of (feature designator, feature) as specified by the event type. A feature designator specifies the meaning and the format of the corresponding feature. The meaning usually is the role that an entity assumes in the activity described by the event, whereas the feature is the actual name or value of the entity. We define a feature type as the combination of an event type and a feature designator, hence a feature can be considered to be an instance of a feature type.

Misuse detection is the process of detecting manifestations of misuse scenarios in audit data based on models of manifestations of misuse scenarios. This is done under the assumptions that a manifestation is actually observed, when a misuse scenario takes place (completeness), and that a misuse scenario actually takes place, when its manifestation is observed (correctness). In the following, we assume that we have audit components that can observe misuse scenarios completely and correctly. This assumption simplifies the terms used for describing the concepts of misuse detection, because we do not need to strictly distinguish activity and observations or manifestations of activity. We may then talk about events, as if they are actual activity, instead of merely observations of activity. Also, it is then possible to talk about models of misuse scenarios that describe (misuse) activity, if we actually mean models that describe misuse scenario manifestations. A given model of manifestations of a given misuse scenario should also completely describe the manifestations of misuse activity, such that the model does not miss a manifestation of the misuse scenario (false negative). Likewise, the model should describe the manifestations of the misuse scenario correctly, such that the model does not apply to manifestations of other activity (false positive).

Under the assumption of correct and complete audit components and correct and complete models (of manifestations) of misuse scenarios, one can conjecture from a model matching a manifestation in audit data that the corresponding misuse scenario activity took place. Also, if no model matches any manifestation in the audit data, one can conjecture that no misuse scenario activity is taking place.

A prerequisite to specifying complete and correct models of misuse scenarios is a sufficiently expressive modeling framework. The framework should also support human intuition during model specification in order to reduce the potential for human error. To be processed by an IDS, the models of misuse scenarios usually are translated to or directly expressed in a purely textual language. The textual representation of models of misuse scenarios are commonly denoted as misuse detection signatures, intrusion detection signatures or just signatures in the literature about misuse detection or intrusion detection systems.

In this article we provide a brief overview of related work on misuse scenario modeling frameworks and signature languages in Sect. 2, motivating the need for a new modeling framework. In Sect. 3 the semantic requirements for misuse scenario modeling are summarized, and, accordingly our framework for modeling misuse scenarios is presented in Sect. 4. A corresponding modeling language in a textual representation is proposed in Sect. 5, and we conclude with a summary of some recent applications of the framework in Sect. 6.

2 Related Work on Signature Languages
Several pertinent languages for specifying intrusion detection signatures have been proposed, such as RUSSEL for ASAX [2], P-BEST for EMERALD [3], LAMBDA [4], ADeLe [5], and SHEDEL [6]. Studying and engineering signatures requires a notation that allows for an intuitive understanding of the considered misuse scenarios. We argue that in contrast to a textual representation for complex signatures a graphical representation provides a more intuitive view on what misuse scenario(s) a signature describes. For some languages, a graphical representation has already been proposed: MuSigs for ARMD [7], STATL for the STAT Framework [8], SUTHEK [9] and IDIOT [10]. While MuSigs, STATL and SUTHEK use their own notions of finite state automata (FSA) to model misuse scenarios, IDIOT uses colored Petri-nets (CPN). For a more comprehensive and detailed comparison of various signature languages refer to [11].

For a deeper understanding of the required semantics and expressiveness of a general modeling framework for misuse scenarios, we analyzed existing signature languages and signatures. In the next step we identified a modeling approach that can be easily adapted to satisfy all of the requirements and which has already been used in the intrusion detection domain. We adopted the proposal of Kumar [10] to model signatures using CPNs and adapted and extended it w.r.t. modeling elements and semantics to accommodate the required expressiveness and semantics. It was necessary to define our own notion of a modeling framework based on CPNs, because [Vigna et al. distinguish several (attack) languages that are involved in the process of reproducing (exploit languages), documenting (event languages), detecting (detection/correlation languages), and responding to attacks (response/report languages) [1]. According to this classification, signature (specification) languages are detection languages.]

[1] The loose concept of role here includes, but is not limited to subjects and objects of the activity.
Kumar’s model suffered from severe shortcomings [10]: Modeling the garbage collection of partial matches of misuse scenarios using invariants was error-prone and the implementation necessarily inefficient. The value of token variables could never change, such that different modes of repetition and step instance selection could not be expressed. Consumption of system state was modeled contra-intuitively as a property of the places, limiting intuitive modeling and also required expressiveness (see Sect. 3 for definitions of the terms ‘step instance selection’ and ‘consumptivity’).

When compared to existing work on FSAs for misuse detection, CPNs provide three major advantages:
1. CPNs allow to specify partial orders in a compact way.
2. During signature engineering the CPN tokens can be used to illustrate partial detections of misuse scenarios along with their current variable bindings.
3. CPNs provide the required expressiveness to engineer signatures for the use with any existing IDS by translating the CPN models into the target signature language.

Our previous analysis of existing IDS signature languages and signatures fused with Zimmer’s meta-model [12–13] provides strong indications that we postulate a complete set of requirements. It can be constructively shown that CPNs satisfy all of these requirements [11]. Consequently, the CPNs we use provide the deterministic semantics required for misuse detection and can express all misuse scenarios that can be expressed in any currently existing signature language.

3 Semantic Requirements

In the following, the semantic requirements for modeling misuse scenarios are shortly summarized. The presented requirements were obtained by analyzing domain knowledge (existing misuse detection signature languages and signature bases of IDSs) as well as by analyzing the semantics of events in a similar problem domain (triggers in active DBMSs) [12–14]. For details and examples refer to [11; 14].

We denote a model of a given misuse scenario as a complex event. A complex event consists of inter-related events, where each of the events is denoted as a step of the complex event. For a complex event to occur, matching events must have occurred and must be bound to each step of the complex event. Events that can be bound to a step are basic events and complex events. Basic events represent the basic observable unit, as provided by the audit component. When an event is bound to a step, the step is also said to be instantiated, and the instantiated steps are considered to be the (partial) instance of the complex event. If all steps of the complex event are instantiated, the (complete instance of the) complex event is said to occur.

Summarizing, a complex event models a misuse scenario by specifying how to identify the basic events that can be observed while the misuse scenario takes place. Since several misuse scenarios of the same type may be executed simultaneously, the modeling framework allows to model manifestation-specific states (e.g., bind features to variables, e.g., a file handle), to be able to distinguish distinct (partial) instances of a given (type of) complex event.

The semantics of complex events (for triggers in the domain of active DBMSs) can be partitioned in three dimensions [13] that are described in more detail in the following sections: Event pattern, step instance selection, and step instance consumption.

3.1 Event Pattern

An event pattern defines the complex event to look for. The frame of a complex event is formed by the steps (event types) and their order.

Many signatures describe simply consecutive events (sequence). Alternative activity can be modeled disjunctively, allowing to represent variants of misuse scenarios in a compact way (disjunction). Concurrent threads of activity can be modeled in a conjunctive way, such that all interleavings of the event sequences of the threads are accepted by the model (conjunction). Simultaneous events may occur in parallel systems and can be correlated using their time stamps (simultaneous). In the context of a complex event, certain basic events prohibit completion of the complex event. Such events can be modeled to be not allowed to occur within parts of the manifestation of a misuse scenario (negation).

It is useful to be able to specify the number of times a step must occur for a misuse scenario to complete (repetition), e.g., for dictionary attacks or denial of service attacks (exactly, at least or at most n times, or at least n and at most m times).

The continuity semantics of a misuse scenario model defines whether between three consecutive steps of the event types A, B and C an event c is allowed to occur between events a and b, and whether event a may occur between b and c. The continuous semantics allows for such occurrences and is most useful for the purposes of misuse detection.

When composing more complex patterns from a number of less complex patterns, it is necessary to decide about the concurrency of the less complex patterns. For concurrent composition the threads may overlap, i.e., interleave, and for sequential composition this is not the case (non-overlap).

A very important aspect of the semantics is the ability to specify constraints on the context in which the steps of a complex event occur. Constraints that can be evaluated by merely inspecting the features of the current event are denoted as intra-event conditions and can be used for example to select events that affect a certain user, host or file. Inter-event-conditions can only be evaluated by inspecting at least two events, which implies to create state. For example, inter-event conditions can be used to correlate events that affect the same user, host or file.
3.2 Step Instance Selection
While a given event pattern specifies when a complex event occurs, the step instance selection defines which of the possibly more than one matching events is bound to each step of the complex event. This is an important decision, if we are not only interested in the fact that a complex event occurred, but if we also need to document the events that lead to the complex event for further correlation and response. We adopt three of the instance selection modes proposed by Zimmer [13; 14]: selecting the first or the last event or all events that match the given step. These modes can be used for example to detect when a performance parameter exceeds a threshold, capturing the parameter value when the threshold was exceeded the first time, the current (last) value right before the next step occurred, or all values for further statistics.

3.3 Step Instance Consumption
The current system state that is relevant for a given complex event is reconstructed by binding events to the steps of the complex event, such that the partial instance of the complex event represents the reconstructed relevant system state. After an event has been bound to a given step of a complex event, the resulting partial instance of the complex event represents the occurrence of the event as well as the relevant system state that has been modified by the step.

Some events describe activity that changes features of system state, which are relevant in the context of the considered complex event, for example the destruction of system objects, e.g., process termination and file deletion, or the change of object features, e.g., renaming a file and changing access privileges. Such activity is said to consume the relevant system state, which has been created by previous activity, and which is represented by the matching partial instance of the complex event. Other (non-consuming) activity does not change relevant features of system state, e.g., reading from a file.

Step instance consumption defines, whether a given partial instance of a complex event can evolve into one or more partial instances by binding a consuming or a non-consuming event to a given step, respectively. Since a consuming event modifies the system state represented by the partial instance, the partial instance is evolved by binding the event to the step, effectively consuming the old partial instance. However, the partial instance is evolved for each occurrence of a given non-consuming event type, creating new partial instances representing each occurrence.

4 Modeling Framework
While the semantic requirements from Sect. 3 specify, what semantic aspects are relevant for the detection of manifestations of misuse scenarios, the modeling framework described here focuses on how the relevant aspects can be captured and modeled. We have shown that the expressiveness of the framework meets all seman-
occur, and which adjusted system state is the result if the event occurs. Hence, in a signature-net places and transitions are connected by directed edges to specify the order of the basic events to look for (cf. order in Sect. 3.1). We denote edges that are directed from places to transitions as input edges and edges that are directed from transitions to places as output edges.

Additionally, input edges are characterized by the consumptivity (see Sect. 3.3) of the transition event type w. r. t. the state represented by the place connected to the transition (see Sect. 1).

Consuming edges correspond to standard edges in Petri-nets, whereas non-consuming edges are similar to test edges in Petri-nets [19].

4.2 Places

The modeling framework distinguishes four types of places: initial, interior, escape and final places (see Fig. 2). Each signature-net contains one or more initial places. A signature-net describes one complex event and possibly also variants of the complex event. For the complex event and for its variants the signature-net contains one or more final places. If a token reaches a final place, then a manifestation of the corresponding (variant of the) complex event has been identified in the audit data, i.e., the respective (instance of the) (variant of the) complex event occurs.

Escape places characterize system state that a partial complex event instance has caused by means of the last transition, such that the complex event cannot be completed. Consequently, tokens reaching an escape place are removed from the signature-net. Escape places have the important function of garbage collection by removing obsolete tokens.

Places that are neither initial, final nor escape places are denoted as interior places, which are visited by tokens on their path from an initial place to a final or escape place.

Moreover, places connected to a transition via input edges are denoted as input places of the transition, whereas places connected to a transition via output edges are denoted as output places of the transition.

4.3 Transitions

Transitions describe the observable events changing the system state along some path in the signature-net until some partial instance of some complex event is completed or discarded. Transitions are characterized by an event type, by intra-event conditions, by inter-event conditions, by token bindings (cf. type, intra-event conditions, inter-event conditions and state in Sect. 3.1) and by actions. Moreover, a given transition is influenced by tokens in its input places and it influences tokens in its output places, representing the required system state such that the transition can occur and the adjusted system state after the transition has occurred, respectively (see Sect. 4.5). Each transition is associated with a transition label, which specifies the characteristics of the transition (see Fig. 3).

The event type of a transition identifies the type of event that the transition models as a step of the complex event (see event type ‘FailedLogin’ in Fig. 3).

Intra-event conditions specify additional restrictions w. r. t. the modeled event by requiring certain features of the event (see value ‘root’ required for the feature designated ‘user’ in Fig. 3).

Inter-event conditions relate the current event to events that were observed earlier. Atoms of inter-event conditions are features of the event or variable values of earlier token bindings (see the comparison of the value of the token variable named ‘h’ and the feature designated ‘hostname’ of the current event in Fig. 3).

Token bindings assign values to token variables, where the values are a function of constants, features of the event and values of token variables (see the assignment of the feature designated ‘file’ to the token variable named ‘f’). As an example for the correlation of two distinct events a and b, an inter-event condition may compare a feature of the current event b with a feature of an
earlier observed event \( a \), which has been stored in a token variable (e.g., in Fig. 3 the token variable named ‘hostname’ may contain the feature designated ‘hostname’ of an event \( a \) observed earlier). Actions allow for the (early) response to (partial) complex events (see the action ‘alert(“...”)’ in Fig. 3). In actions the features of the current event, variable values or constant values can be used.

The concepts specified in transition labels are mapped to events in audit data as follows: the event type is directly specified, features are referenced using feature designators, and variable values are referenced by the variable name. A feature type is the combination of an event type and a feature designator. Escape places are used for expressing that the current event makes it impossible for the complex event represented by the signature-net to occur. Consequently, the respective tokens are removed from the marking of the signature-net. In the following, we denote transitions that are connected to an escape place as escape transitions. An escape transition has the meaning that a complex event will not be completed and that no response will be necessary.

To be able to express conjunctively occurring events properly (see conjunction in Sect. 3.1), spontaneous transitions are introduced, which can occur independently from any events. They are therefore characterized by the fictitious event type \( \epsilon \), and they always have empty intra-event conditions. The inter-event conditions of spontaneous events do not refer to event features, but only to variable values and constant values. Transitions that are not spontaneous, are denoted as regular transitions.

### 4.4 Tokens

A signature-net containing only places and transitions connected by edges can describe the causal relationships of events. To be able to describe instances of (partial) complex events, i.e., partially matched misuse scenarios, the marking of signature-nets needs to be defined, as well as the rules how a given marking can be transformed into a new marking.

The marking of a signature-net assigns a (positive and finite) number of tokens to each place of the signature-net. The initial marking is the state of the signature-net, where exactly one token is assigned to each initial place of the signature-net.

Tokens describe the system state w.r.t. the corresponding instance of a (partial) complex event, as it was reconstructed from the events observed so far. The system state is characterized by the place where a token resides as well as by the variable bindings of the token. Token variables are conceptually analogous to the color of tokens in colored Petri-nets [20]. If a value is assigned to a token variable, the variable is said to be initialized, otherwise it is said to be uninitialized. Uninitialized variables are not depicted in the examples.

Figure 4 depicts a partial signature-net, its marking and a current event. The bindings of the token at the initial place are empty, i.e., all variables are uninitialized. Occurrences of transition \( t_1 \) have already generated three tokens in the output place of \( t_1 \). The token variable \( v \) has been bound to different values for each instance of the partial complex event that is represented by each token. Note, that all features designated \( oid \) of the events that activated \( t_1 \) (see the token binding of ‘\( t_1 \)’ in Fig. 4) satisfied the intra-event condition of \( t_1 \).

Moreover, the current event with event type \( e_2 \) is depicted, where the feature designator \( oid \) corresponds to Feature 5. The inter-event condition of transition \( t_2 \) requires that the feature designated \( oid \) equals the value of the token variable named \( v \). In Fig. 4 this condition is satisfied only for the input token, where value 5 is assigned to the token variable named \( v \), i.e., the partial complex event represented by this token will evolve.

In signature-nets a set of tokens is assigned to a place, i.e., if a marking would have identical tokens in the same place, the identical tokens are unified (merged) to a single token. This is being done because identical tokens represent the same instance of a misuse scenario. The identical tokens would simultaneously take the same path through the signature-net, i.e., trigger redundant responses.

### 4.5 Transition Activation and Occurrence

For misuse detection purposes it is useful that transitions in signature-nets occur in a deterministic way. In the following, rules are presented, which result in all activated transitions to occur for all activating unified tokens (see below).

To describe the rules for transition occurrence, some terms are defined. An activating token set of a given transition \( t \) contains exactly one token from each input place of \( t \). A set of tokens is unifiable, if for all initialized variables of all tokens in the set the following holds: the value assigned to an initialized variable named \( v \) from token \( t_i \) equals the assigned value of all variables named \( v \) from all other tokens \( t_j, j \neq i \). Each activating token
set $s$ that is unifiable can be represented by a unified token $u$. The unified token $u$ contains all variables of all tokens from $s$. That is, $u$ contains variables with the same names of all variables from all tokens in $s$, where the variables of $u$ receive the values from the initialized variables from the tokens in $s$ and the other variables of $u$ remain uninitialized. A unified token that satisfies all inter-event conditions of $t$ with respect to a given event $e$ is denoted as activating unified token. The union of all activating unifiable token sets that are represented by the activating unified tokens is denoted as the set of input tokens of $t$ with respect to $e$.

**Transition activation**

The marking of a signature-net is said to be unstable if spontaneous transitions are activated. Conversely, the marking of a signature-net is denoted as stable, if no spontaneous transitions are activated.

A given spontaneous transition $t$ is activated independently from any events occurring (see Sect. 4.3), i.e., if at least one activating unified token exists for $t$ in the marking of the signature-net.

For the current event $e$ and a given marking $m$ of the signature-net, a regular transition $t$ is activated if $m$ is stable, the event type of $e$ equals the event type of $t$, $e$ satisfies the intra-event condition of $t$, and at least one activating unifying token exists for $t$ with respect to $m$ and $e$. As a result of these activation rules, as long as the marking of the signature-net is unstable, only the activated spontaneous transitions may occur until the new marking is stable. Then, regular transitions may be activated.

**Transition occurrence**

In the following, the rules for the occurrence of transitions are described. After the initialization of a given signature-net with the initial marking $m_0$, spontaneous transitions may occur if $m_0$ is unstable. After the spontaneous transitions have occurred according to the rules, the resulting marking $m$ is stable.

Owing to the rules, it can be assumed that the current marking $m$ of the signature-net is stable, whenever a current event $e$ is provided to the signature-net.

**Scope.** All activated transitions occur virtually simultaneously for a given marking. Each activated transition $t$ occurs for all activating unified tokens. When a transition occurs, one or more tokens are generated in its output places (see below: token generation) and tokens may be removed from some of its input places (see below:

---

5.1 Places

Analogously to signature nets EDL differentiates between four types of places, initial, interior, escape and final places. An EDL description contains at least one initial and exactly one escape place. Each place has a unique name. In addition interior and final places are characterized by features. For tokens assigned to a place, the features of the place define which variables of the tokens are bound. For all tokens assigned to the same place the same variables are bound, regardless whether they reached the place on alternative paths. The data types **Bool**, **Integer**, **Float** and **String** are supported for features. Within an EDL description the features of place can be
referred to by the name of the place and the name of the feature. Figure 5 illustrates features of places for a sketch of a signature-net while Fig. 6 shows the corresponding EDL description declaring the respective places. The initial place \( p_1 \) does not have any feature. Interior place \( p_2 \) is assigned with feature \( uid \) of type String and feature \( pid \) of type Integer. Exit place \( p_3 \) has a feature \( uid \) of type String as well as the features \( pid \) and \( pid2 \) of type Integer.

5.2 Transitions

Transitions of an EDL description posses all properties of transitions of signature-nets, namely an event type as well as conditions, variable or feature bindings, and actions which can refer to place features and event features. Feature bindings of a transitions define, how the features of its output places are assigned. For each transition a list of names of input places and output places can be given which specifies the input and output edges of the transition. Thereby each input place can be marked with the symbol + to declare the respective input edge as non-consuming. If no symbol or – is given the input edge is considered consuming. Figure 7 illustrates transition labels of a signature-net and Fig. 8 shows the corresponding description of the transitions in EDL. The first transition \( t_1 \) is connected to its input place \( p_1 \) via a non-consuming edge and to its output place \( p_2 \). This transition is associated with event type \( X \) and an intra-event condition that requires the feature \( event \) of the associated event to be equal to the value "create". Further two token variables are bound by the transition. This corresponds to an EDL description which assigns features of its output places. Transition \( t_1 \) assigns the values of the event features \( userID \) and \( procID \) to the features \( uid \) resp. \( pid \) of its output place \( p_2 \). Transition \( t_2 \) is connected to its input place \( p_2 \) via a consuming input edge and to its output place \( p_3 \). It is associated with event type \( Y \) and describes two conditions. An intra-event condition requires the event feature \( event \) to be equal to "fork". The inter-event condition of \( t_2 \) requires the value of token variable \( uid \) to be equal to the value of the event feature \( userID \). The corresponding EDL description specifies a condition for the input place feature \( p_2.uid \) and the event feature \( userID \) accordingly. Transition \( t_2 \) of the signature-net also describes the binding of event feature \( procID \) to token variable \( pid2 \) which is mapped to a respective assignment of an output place feature in the corresponding EDL description.
Further the EDL description assigns the values of input place features \((p_2.uid\) and \(p_2.pid\)) to output place features \((p_3.uid\) and \(p_3.pid\)). In contrast to signature-nets where token variables are cumulated while a token is passing transitions, EDL requires to specify explicitly which input place feature should be adopted to which output place feature.

### 5.3 Module Concepts

EDL adopts the proved hierarchy concept of SHEDEL \([6]\) which supports several abstractions. An EDL description of a misuse scenario defines an accordingly named event type. Other EDL descriptions may contain transitions associated with such an event type. The features of an event of such an event type are defined by the features of the final place of the EDL description. Figure 9 illustrates these concepts. Usage of such an event hierarchy realizes a concurrent composition of complex events which in regard to the semantic aspect \textit{concurrency} corresponds to the \textit{overlap} mode (see Sect. 3).

For the purpose of reuse of partial EDL descriptions for misuse scenarios sequential composition of complex events, corresponding to \textit{concurrency} mode \textit{non-overlap}, is often required. In order to support this mode the second module concept of \textit{macros} was integrated into EDL. A macro corresponds to a partial signature-net which is delimited by places and may have multiple \textit{macro entry places} and \textit{macro exit places}. Macros specified in an EDL description can be used as event type. Macros are used via macro transitions, which are transitions that are associated with a macro event type.

Figure 10 exemplifies the definition of a macro \(B\) and its usage in the EDL description of event \(E\). For using a macro transition a mapping of its input places to its entry places and of its output places to its exit places needs to be specified. In addition it needs to be specified which features of the input places are assigned to entry places and which features of the exit places are assigned to output places. This form of parameterability allows for development of reusable macros, independent of concrete usage scenarios. Direct mappings of features of macro input places to features of macro output places can also be defined. For speci-
fying these mappings an EDL description of a macro transition contains additional sections marked with the respective keywords `PLACE_MAPPINGS`, `MACRO_IN_MAPPINGS`, `MACRO_OUT_MAPPINGS` and `MACRO_THROUGH_MAPPINGS`. Figure 11 shows the EDL description of the macro transition shown in Fig. 10. The meaning of a macro usage results from replacing the macro transition by the macro definition, whereby entry places and exit places are replaced by input and output places as specified in the description of the macro transition. Figure 12 shows a description of event $E$ which is equivalent to that in Fig. 10 after replacing macro transition $t_1$ with macro $B$.

6 Conclusion

Signature-nets provide a comprehensive modeling framework for describing signatures for intrusion detection, such that existing signatures formulated in languages of existing intrusion detection systems may be translated into Signature-nets. As a consequence Signature-nets are a suitable framework for further investigation of practical and research problems in the area of signature-based intrusion detection. Some of these directions are summarized in the following.

Signature-nets and EDL have already proved to effectively support systematic analysis and engineering of attack signatures. For example signature-nets have been used for studying structural properties of attack signatures. Based on gained insights a couple of optimizing strategies increasing the signature analysis efficiency have been proposed [22]. These optimizations were implemented in the matching engine JSAM for analyzing EDL, which is publicly available and, e.g., used by a malware early warning system for matching behavioral malware signatures against program execution reports [23].

First approaches for systematic signature development and engineering were also based on signature-nets and EDL. We proposed an approach for systematic engineering of EDL signatures based on re-use of existing signatures [24]. Schmerl et al. [25] introduce respective methods for testing EDL attack signatures systematically. In addition verification of EDL signatures using the SPIN model checker has been proposed [26; 27]. Owing to EDL’s relation to colored Petri nets EDL signatures can be transformed to PROMELA models allowing specification errors to be found by the SPIN model checker. EDL, as a version of Petri Nets, also lends itself to visual signature modeling and editing, as well as simulation of signature matching based on given audit trails. As a proof of concept Schmerl implemented such a graphical editor/simulator.

Intrusion detection is necessarily catered by audit trails that constitute monitoring data. Such monitoring data contains features enabling the identification of the actors of the monitored system, where these actors often are natural persons. As a consequence, audit trails in many systems are considered personal data, which is protected by pertinent privacy law. A suitable protection strategy for personal data is pseudonymization [28]. Based on the framework of Signature-nets it has been shown that it is possible to conduct signature matching on pseudonymized audit data, i.e., audit data where identifying features are replaced by suitable pseudonyms [29]. In addition the pseudonyms may be equipped with cryptographic data allowing to account signature matches to exactly those actors hidden behind the pseudonyms that are responsible for the activity modeled by the matched signatures. The described approach requires static analysis of the given signature-nets, as well as dynamic analysis of the audit data during run-time, considering the structural properties of the given signature-nets [29].

References


Received: January 1, 2012, accepted: January 15, 2012

---

**Prof. Dr. Ulrich Flegel** is a full professor at Hochschule für Technik Stuttgart. He holds a PhD in Computer Science from Technische Universität Dortmund. His research interests are the analysis and privacy-respecting detection of fraud, intrusions, and malware. He is founder member and former chair of the special interest group on Security – Intrusion Detection and Response (SIDAR) of the German Informatics Society and co-chairs the steering committee of the international conference on Detection of Intrusion & Malware and Vulnerability Assessment (DIMVA).

Address: Hochschule für Technik Stuttgart, P.O.B 101432, 70013 Stuttgart, Germany, Tel.: +49-711-8926-2801, Fax: +49-711-8926-19-2801, e-mail: ulrich.flegel@hft-stuttgart.de

**Dr. Michael Meier** is senior researcher with the Computer Science Department at Technische Universität Dortmund and the Cyber Defense research group at Fraunhofer FKIE and currently on substitute professorship with the University of Bonn. He holds a PhD in Computer Science from Brandenburg University of Technology Cottbus. His research focuses IT early warning, malware analysis, intrusion and malware detection and privacy respecting log data analysis. He is founder member and chair of the special interest group on Security – Intrusion Detection and Response (SIDAR) of the German Informatics Society and co-chairs the steering committee of the international conference on Detection of Intrusion & Malware and Vulnerability Assessment (DIMVA).

Address: Fraunhofer FKIE, Forschungsgruppe Cyber Defense, Neuenahrer Str. 20, 53343 Wachtberg/Technische Universität Dortmund, Computer Science VI, 44221 Dortmund, Germany, Tel.: +49-231-755-6481, Fax: +49-231-755-2405, e-mail: michael.meier@udo.edu