Introduction to Misuse Intrusion Detection Systems (IDS)

- Two categories
  - Single event IDS
    - Each event is compared with each known signature
    - Specifying signatures simple
  - Multi-event IDS
    - Do not have a uniform abstract algorithm because they do not propose the same operators to combine events
    - Can be further split into two categories (next slide)

Multi-event IDS Categories

- Transition based
  - What are the significant traces of attacks is hidden by how they should be detected
  - Very tricky to write signatures
- Declarative
  - Signatures only contain what are the significant traces of attacks
  - How they are detected is addressed by an algorithm
  - Easier to write signatures
  - Problem: algorithm is a black box and detects all instances of an attack, allowing attackers to choke the IDS by launching many incomplete instances of an attack
Focus of Paper

- Refine the declarative approach
  - Formally specify the algorithm in two stages
    - Classify the signature instances
    - Give a set of detection rules which detects in an audit trail a representative of each class
    - Rules are formally specified using a “parsing schemata”
    - Algorithm defined by the rules are proved sound and complete
  - What and the how still separated, but security officer can parameterize the detection by choosing a class for each signature

Contribution of Paper

- Two main contributions
  - Less instances of signatures are tracked
    - More resistant to choking attacks
  - Detection algorithm is specified in a high level formal way
    - Easy to understand and reason about
    - Essential operation features are made explicit

Specification of Signatures

- Intrusion signatures describe combinations of events

- A filter is a signature on one event

- Complex signatures can be focused on two ways
  - Sequence
  - Conjunction
Specification of Signatures

• Definition 1 (Event): An event is a collection of values identified by field names. We represent an event as a set of pairs (field_name, value). We assume that in an event, a field name belongs to one pair.

\[ E = \{ (\text{eventID}, \text{EXEC}) \} \]

• Definition 2 (Trail): A trail is a totally ordered sequence of events. Given a trail \( T \), we note \( T[i] \) the \( i \)th event in the trail.

• Definition 3 (Filter): A filter is a set of constraints between event field names, constant values and variable names. In this paper, we consider that constraints involving variable names can only be equality constraints.

\[ F = \{ \text{eventID} = \text{EXEC} \} \]

\[ \text{userID} \neq 0 \]

\[ \text{userID} = \text{User} \]

\[ \text{path} = \text{ProgName} \]
**Specification of Signatures**

- **Definition 4 (Signatures):** A signature is defined by a 5-tuple \((V, F, N_T, S, P)\)
  - \(V\) is a set of variables
  - \(F\) is a set of filters that use variables in \(V\)
  - \(N_T\) is a set of non-terminal elements
  - \(S\in N_T\) is the axiom,
  - \(P\) is a set of production rules \(N_T \rightarrow \text{Prod.}\), where \(p\in \text{Prod.}\) can be:
    - \(\text{Filter}(f)\) where \(f\in F\)
    - \(\text{Seq}(A, B)\) where \((A, B)\in N_T \times N_T\)
    - \(\text{And}(A, B)\) where \((A, B)\in N_T \times N_T\)

**Specification of Signature**

Several signatures rules can be summed up into a single one. For example:

\[
\begin{align*}
N_T & \rightarrow \text{Seq}(N_{T+1}, N_{T+2}) \quad | \quad f_1[A, B] = (...) \\
N_{T+1} & \rightarrow \text{Filter}(f_2[A]) \\
N_{T+2} & \rightarrow \text{Filter}(f_3[B])
\end{align*}
\]

is equivalent to:

\[
\begin{align*}
N_T & \rightarrow \text{Seq}(N_{T+1}, N_{T+2}) \\
N_{T+1} & \rightarrow \text{Filter}(f_2[A]) \\
N_{T+2} & \rightarrow \text{Filter}(f_3[B]) \\
\end{align*}
\]

**Example of Signature**

<table>
<thead>
<tr>
<th>Const</th>
<th>Define</th>
</tr>
</thead>
<tbody>
<tr>
<td>(N_{T+1})</td>
<td>(N_{T+1} = \text{Seq}(N_{T+2}, N_{T+3}))</td>
</tr>
<tr>
<td>(N_{T+2})</td>
<td>(N_{T+2} = \text{Filter}(f_{T+1}[A]))</td>
</tr>
<tr>
<td>(N_{T+3})</td>
<td>(N_{T+3} = \text{Filter}(f_{T+2}[B]))</td>
</tr>
<tr>
<td>(N_T)</td>
<td>(N_T = \text{Seq}(N_{T+1}, N_{T+2}))</td>
</tr>
<tr>
<td>(f_{T+1})</td>
<td>(f_{T+1} = \text{Filter}(A, B))</td>
</tr>
<tr>
<td>(f_{T+2})</td>
<td>(f_{T+2} = \text{Filter}(A, B))</td>
</tr>
</tbody>
</table>
Semantics of Signature

• A concrete instance of a signature is a collection of events that
  – Fulfill the constraints in filters
  – With respect to the correlation specified by the logical variables
  – And are in a correct order according to temporal constraints

• This is denoted \( \{p_1, \ldots, p_n\} \) where \( p_i \) are positions in the trail

• Proposed semantics:
  – \( (T, i, j) \) where \( T \) is a signature valuation, \( i \) is the position of the first event of the instances, and \( j \) is the position of the last event of the instance

Semantics of Signatures (Signature Constraints)

• Definition 5 (Signature constraints): A signature constraint is a set of constraints that force some properties on the possible values of the variables used in a signature. Can not contain reference to a field name.

\[
\theta = \{ \gamma_1 = 42, \gamma_5 \neq 21, \gamma_5 \neq 34, \gamma_5 \neq 17 \}
\]

Semantics of Signatures (Signature Valuation)

• Definition 6 (Signature valuation):
  A signature constraint is said to be a signature valuation if it forces a unique value for each variable that appears in the signature.

\[
\theta = \{ \gamma_1 = 42, \gamma_5 = 28 \}
\]
Semantics of Signatures
(Event Matching)

- Definition 7 (Event Matching):
  - We define the predicate match(E, F, T) where E is an event, F is a filter, and T is a valuation of F.
  - This predicate holds iff the constraint set \( (F_{E_T}) \) admits at least one solution.
  - It does not hold if an event field name used in F is not present in E.

Semantics of Signature

- Semantics of the language given by means of the relation \( \mathcal{D} \). Given a signature S, a trail T, and an abstract instance \( I=(T, i, j) \), \( I \) is an instance of S in T iff \( T, T, I, j \mathcal{D} S \).

Specification of Signature Instances

- Many approaches to ID strive to be both sound and complete.
- Authors argue that completeness is not necessary and is sometimes a drawback.
- Authors propose an approach to specify what instances are relevant for detection.
  - Specifications are expressed as equivalence relations between instances of each signature.
  - Once classified, the IDS can report only a particular instance of each class.
Specification of Signature Instances

- Given a signature, equivalence relation is specified by choosing an element in the lattice of all the subsets of variable of the signature.

- Two instances are equivalent if they contain the same values for the variables in this subset.

```
<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>{U1, U2, T1, T2}</td>
<td>{U3, T1}</td>
<td>{T2}</td>
</tr>
<tr>
<td>{U1, T2, T3}</td>
<td>{U1}</td>
<td></td>
</tr>
<tr>
<td>{T1}</td>
<td>{T3}</td>
<td></td>
</tr>
</tbody>
</table>
```

Specification of Signature Instances

- Each element e in this lattice corresponds to an equivalence R(e) between instances:
  - \( R(\top) = \lambda x_1 \ldots \lambda x_n . \text{true} \)  
    All instances are in the same class.
  - \( R(T_i) = \lambda x_1 \ldots \lambda x_n . x_i = \text{false} \)  
    Each instance is in its own class.
  - For any other element \( e = \{ T_1, \ldots, T_i \} \) of the lattice:
    \[
    R(e) = \lambda x_1 \ldots \lambda x_n . \begin{cases} \text{true} & \text{if } T_i(x_i) = \text{false} \\ \text{false} & \text{otherwise} \end{cases}
    \]

where \( T_i(x_i) \) is the value of variable \( T_i \) in the instance \( T \). Here, \( e \) is a sequence of \( T_i \)s present in \( e \).

Specification of Signature Instances

- Two motivations:
  - Want to be able to prune search paths on the fly
  - Don’t want to miss relevant instances

- For \( T_i \neq \top \):
  \[
  R(\{ U_1, U_2, T_1, T_2 \}) = R(\{ U_3, U_2, T_3 \})
  \]

- For \( T_i = \top \):
  \[
  R(\{ U_1, U_2, T_1, T_2 \}) = \lambda x_1 \ldots \lambda x_n . \text{true}
  \]

- For \( T_i = \bot \):
  \[
  R(\{ U_1, U_2, T_1, T_2 \}) = \lambda x_1 \ldots \lambda x_n . \text{false}
  \]
Specification of Signature Instances

• After instances are classified, must decide which instance to report to the IDS
• Three strategies (Chakravarthy et al.):
  – Report the instance that starts first and ends first
  – Report the one that starts last among all the ones that end first
  – Report the shortest instance for each event that starts an instance
• This paper selected the first strategy:
  – Finding the instance that ends first is required for analyzing an infinite trail
  – Easier to constrain further search as opposed to canceling previous results

First Strategy

• The predicate First is defined as
  – First(S, ?, T, a, (i, j, T))
  
  • S is a signature
  • ? is an equivalence relation between instances of S
  • T is a trail
  • a is a position in T
  • (i, j, T) is an instance in S
  – Given an instance I = (T, i, j) with a = I, this predicate holds iff, among all instances equivalent to I according to ? that start after a, I is the one that starts first and ends first

First Algorithm

• Implements the First strategy
• Described with a formalism called parsing schemata
  – Specifies algorithms using a set of deduction rules
  – Gives a formal framework to describe and prove properties
  – Modular description (i.e. one does not need to know the whole specification to understand how a particular construct is searched for in the language)
Parsing Schemata

- Parsing algorithm is described as set of deduction steps
  - Hypothesis and conclusion of these steps are called parsing items
  - Parsing items are partial or complete parsing trees
  - Deduction starts with an item representing an empty parsing tree
  - Deduction ends when an item representing a complete parsing tree of the axiom grammar is produced

Parsing Schemata
(Defining the Domain of Items)

- Uses the form:
  - $$[i, a?ß, j]T$$
  - $$(i, j)$$ are positions in the trail
  - $$a?ß$$ is the right-hand side of a grammar production where $$a$$ has been inserted
  - $$T$$ is a signature constraint

Description of First Algorithm

- Assumptions on specifications:
  - Signatures that use the $$[]$$ notation have to be expanded
  - Non-terminal elements can be used only once in all grammar rules
  - All filters must be labeled with the equivalence relation associated to the signature (Ex. Filter,$$_r$$(F) where $$?$$ is an equivalence relation)
Operators (Propag)

• Propag operator unifies the variables in the signature constraint with the values of an event (Definition 8)
  – Denoted as Propag(E, F, T)
    • E is an event
    • F is a filter
    • T is a signature constraint
  – This constraint is obtained by:
    • Copying F into F’ and removing all constraints with no variable in F
    • Substituting all field names of F according to E
    • Making the union of F’ and T

Operators (Restrict)

• The Restrict operator creates a new constraint which causes some paths in the search to be pruned (Definition 9)
  – Denoted as Restrict(?, T)
    • T is a valuation of a given signature S
    • ? is an equivalence relation
    • Defined as:

\[
\text{Restrict}(\rho, ?) = \text{case } \rho \text{ in } \begin{cases} 
R(()) : \text{return } VALUE \\
R(T) : \text{return } T \\
R((T_1, T_2)) : \text{return } \bigvee_{i=1}^n \{ T_i \neq ?(T_i) \}
\end{cases}
\]

Operators (Constraint Comparison)

• =s compares signature constraints
  – Given a signature S and two signature constraints T_1 and T_2
  – T_1 =s T_2 iff the set of possible values for each element of Var(S) described by T_1 includes the one described by T_2

\[
\text{S} = \{ (\text{Seq}(A \star B), \theta), (\text{Seq}(A \star B), \theta) \} : \{ j \leq k \}
\]
**Deduction Rules for Filters**

- Rule **Filter 1** specifies that if event $T[i]$ cannot be used to match the filter, then the algorithm goes one step forward in the trail.

- Rule **Filter 2** handles the other case. The first item memorizes an instance of $F$ found in position $i$. Propag takes into account that some variables can be instantiated here. The second item starts the search for a new instance of $F$ in the remaining part of the trail. Can be more constrained than the one that produced this item according to the result provided by Restrict.

\[
\text{Filter 1: } \frac{\text{match}(T[i], F, \theta) = \text{false}}{\text{match}(T[i+1], F, \theta) = \text{false}}
\]

\[
\text{Filter 2: } \frac{\text{match}(T[i], F, \theta) = \text{true}}{\theta_1 = \text{Propag}(T[i], F, \theta) \quad \theta_2 = \theta_1 \setminus \{\text{restrict}(T[i])\}}
\]

**Deduction Rules for Sequence**

- Rule **Seq 1** starts the search for the first part of the sequence.

- Rule **Seq 2** shows that once an instance of the first part is found, that item is replaced to find the next item. The second item added starts the search for $B$.

\[
\text{Seq 1: } \frac{\text{match}(A \cdot B, \theta)}{\text{match}(A, \theta)}
\]

\[
\text{Seq 2: } \frac{\text{match}(A, \theta) = \text{true}}{\text{match}(B, \theta)}
\]

**Deduction Rules for Sequence**

- Rule **Seq 3** triggers once $B$ is found:
  - Checks that $B$ is found after $A$ ($j = k$).
  - The constraint of the second part must refine the constraint of the first part.
  - Does not remove first item, because it may be needed later.
  - Second item added showing that it found an instance of Seq(AB).

\[
\text{Seq 3: } \frac{\text{match}(A \cdot B, \theta)}{\text{match}(A, \theta) \land \text{match}(B, \theta)}
\]

\[
\frac{\text{match}(A, \theta) = \text{true}}{\text{match}(A \cdot B, \theta) \land \text{match}(B, \theta) \land \theta_1 \land \theta_2}
\]

\[
\frac{\text{match}(A, \theta) = \text{true}}{\text{match}(A \cdot B, \theta) \land \text{match}(B, \theta) \land \theta_1 \land \theta_2 \land \text{match}(A, \theta)}
\]
Deduction Rules for Conjunction

- Rule And\(_1\) starts the search of both parts of the conjunction.
- Rule And\(_2\) states that when two parts of a conjunction are found, if their respective constraints are compatible, then a new item is created to notify that an instance of the conjunction is found.

\[
\text{And}_1: \bar{P}, \{\text{\textbullet}, \text{\textbullet}, (\bar{A}, \bar{B}), \bar{P}\} \\
\text{And}_2: \bar{P}, \{\text{\textbullet}, \text{\textbullet}, (\bar{A}, \bar{B}), \bar{P}\} \\
\]

Conclusion

- Described how to specify signatures with sequences and conjunctions of events correlated with logical variables.
- Presented a declarative semantics to these signatures.
- Introduced signature instance classes based on the valuation of variables of interest.
- Given a formal description of a detection algorithm.
- Parsing schemata makes it easy to understand and reason about while essential features are made explicit.