A Generic Intrusion Detection and Diagnoser System Based on Complex Event Processing

Massimo Ficco  
Department of Information Engineering  
Seconda Universita’ di Napoli (SUN)  
via Roma 29, 81031 Aversa, Italy  
Email: ficco@unina.it

Luigi Romano  
Department of Technologies  
Università degli Studi di Napoli “Parthenope”  
Centro Direzionale di Napoli, IT  
Email: luigi.romano@uniparthenope.it

Abstract—This work presents a generic Intrusion Detection and Diagnosis System (ID²S), which implements a comprehensive alert correlation workflow for detection and diagnosis of complex intrusion scenarios in Large scale Complex Critical Infrastructures. The on-line detection and diagnosis process is based on an hybrid and hierarchical approach, which allows to detect intrusion scenarios by collecting diverse information at several architectural levels, using distributed security probes, as well as perform complex event correlation based on a Complex Event Processing. The escalation process from intrusion symptoms to the identified target and cause of the intrusion is driven by a knowledge-base represented by an ontology. A prototype implementation of the proposed ID²S framework is also presented.

Keywords—intrusion detection system; diagnosis; attack scenario recognition; complex event processing;

I. INTRODUCTION

Large scale Complex Critical Infrastructures (LCCI), play a key role into several fundamental human activities. The consequences of an outage can be catastrophic in terms of efficiency, economical losses, consumer dissatisfaction, and even indirect harm to people.

In recent years, complex, distributed and large software systems have increasingly been used in scenarios, often critical, such as transport infrastructures (e.g., airports, seaports), or plants for the provision of water and energy. They have been used for LCCI interconnection, control, and management. The complexity of such software systems reflects the distributed nature of modern business and the diversity and independence demanded by its computational processes.

Software complex systems from the point of view of security are of increasing importance in both industrial and corporate. In particular, LCCI size has significantly grown, crossing national boundaries, and their operational environment, originally planned to be ‘closed’, becomes more and more ‘open’ to allow interoperability among LCCIs and remote accesses and control. This implies malicious attacks should be taken into account. Therefore, the criticality of LCCIs poses new challenges for software engineers, which must develop systems to ensure a high level of protection, and at the same time they must keep low costs and development time.

The security of LCCI is hard to achieve for complexity and heterogeneity of its components. Moreover, attacks against a complex software system manifest themselves in terms of symptoms, which can be of differing nature, at different system architectural levels, and in different systems components. Therefore, security administrators have to use several IDSs to monitor such symptoms, which produce a multitude of alerts most of which are false positives, or do not constitute actual intrusions. The effects of these attacks can involve different components, and they can also occur at different times. Moreover, attacks can be complex scenarios, which consist of multiple stages performed to achieve a final malicious objective.

In this work is presented a generic ID²S framework, which implements an ‘hybrid’ and ‘hierarchical’ correlation process for detection of complex intrusion scenarios. The correlation capability is driven by a knowledge-base represented by an ontology, which is used to capture the causal relationships among the intermediate attacks that are detected.

In particular, this work presents the different components necessary to implement the ‘hybrid’ approach, which is based on diversity both in information sources and methods used to detect malicious activities. The ID²S collects streams of information at several architectural levels (i.e., Middleware, operating system, data base, and application), using multiple security probes, which are deployed as a distributed architecture in the LCCI.

In order to recognize complex intrusion patterns and enrich the semantics of diagnosis, a ‘hierarchical’ correlation approach based on a Complex Event Processing (CEP) is adopted. It captures the causal relationships among the resulting alarms (which may represent intermediate attacks of a more complex attack scenario), by correlating them on the base of temporal and logical constraints.

The presented ID²S architectural model consists of a collection of software components that transform raw attack symptoms into high level intrusion scenario alarms/reports to
alert the administrator or/and to perform a recovery action. Each component focuses on different aspects of the overall correlation process.

Finally, this work shows how the proposed solution is able to detect complex attack scenarios that consist of specific sequence of malicious activities (called ‘intermediate attacks’) performed by the attacker in order to discover system’s vulnerabilities.

The rest of the paper is organized as follows: Section II presents the adopted correlation process; Section III describes the proposed knowledge-base; in Section IV we present the ID$^2$S architectural model; Section IV-A provides a prototype implementation of the proposed ID$^2$S framework, which is entirely based on open source components; the system operation based on a CEP is presented in Section V; finally, some conclusions and lessons learned are presented in Section VII.

II. THE CORRELATION PROCESS

The adopted correlation process consists of a collection of components that transform raw attack symptoms into high level intrusion scenario alarms to alert the administrator or/and to perform a recovery action. In particular, each component focuses on different aspects of the overall correlation process:

- **Monitoring**: Distributed probes are used to observe relevant symptoms at different architecture levels (Middleware, operating system, application) by using both anomaly- and misuse-based monitoring methods. The use of multiple heterogeneous probes potentially improves detection performance through the generation of different views of the same security incident.

- **Classification**: During the classification process the symptoms can be aggregated into categories. In particular, they are divided into abuses, misuses, and suspicious acts. Abuses represent actions which change the state of a system’s component. They can further be divided into anomaly- and knowledge-based. The former represent anomaly behaviors (e.g., unusual application load, anomalous input requests) [1]; the latter, are based on the recognition of signatures of previously known attacks (e.g., brute force attacks). Misuses represent out-of-policy behaviors, in which the state of the components is not affected (e.g., authentication failed, failure queries). Suspicious acts are not violations of policy, but however, they are events of interest to the detection process (e.g., commands which provide information about the state of the system), which are not signaled by explicit messages.

- **Normalization**: Every monitored symptom is coded and normalized into a standardized format, as well as augmented with additional information, such as time-stamps and source address of the attacker.

- **Fusion**: Events generated by different probes detecting the same attack are merged into a single alarm. Such symptoms are aggregated by using clustering-based aggregation rules [2]. They combine symptoms based on the ‘similarity’ among their attributes.

- **Diagnosis**: For each attack reporting by the ranking and filtering phase, the diagnosis determines whether the attack was either a successful attack or a no relevant attack, i.e., an attack that did not lead to intrusion. In the case of intrusion, it has to identify the intrusion effects, i.e., the extent of the damages and the faulty components. During the on-line diagnosis activities, different diagnosis mechanisms can be adopted. For example, several threshold-based over-time diagnostic mechanisms using either heuristic or probabilistic approaches [4] [5].

- **Correlation**: The goal of correlation is to identify complex attack scenarios. In the intrusion detection literature, attack scenario (or attack pattern) is a sequence of explicit attack steps, which are logically linked and lead to an objective. Correlation receives the detected intermediate attack alarms and tries to capture the causal relationships among them.

In this work, an attack scenario is modeled by causal-based correlation rules [6]. A rule consists of a set of prerequisites (pre-conditions) and consequences (post-conditions), which are logical conditions on the intermediate attacks. The pre-conditions are logical conditions that specify the requirements to be satisfied for the attack to succeed. The post-conditions are logical conditions that specify the effects of the attack when this attack succeeds.

The causal-based correlation approach correlates different attacks by matching consequences of an intermediate attack with the prerequisites on another intermediate
Figure 1. A simplified view of the proposed knowledge-base

attack. Post- and pre-conditions are linked together by temporal constraints. Each condition is described by predicates, which are conditions to be verified. The pre-conditions are verified by means of information inferred during the ranking phase, whereas the post-conditions are verified by information reported by the diagnosis output. A deadline is fixed for the recognition of each scenario.

- **Recovery**: If intrusion effects are identified, it is determined the less costly and most effective remediation based on the severity of the error affecting the system components.

### III. A KNOWLEDGE-BASE SCHEMA

The detection and diagnosis process is driven by a knowledge-base schema represented by an ontology. Figure 1 presents a simplified view of our defined knowledge-base. It is used to recognize complex attack scenarios, and to aggregate symptoms produced by different probes with the final goal of identifying one or more intermediate attacks as likely causes of observed symptoms. Moreover, the knowledge-base is also used to automate the process of deriving queries for diagnostic analysis (to identify the cause and the effect of the discovered intrusion) as well as determine the remediation based on the error affecting the system components.

As Figure 1 shows, each kind of attack can be related to a set of potential symptoms. More specifically, an attack is described by using `AttackIndicator` that has the following properties: (i) `hasTrustworthiness`, which is defined by the concept of likelihood that the observed feature is a symptom of the considered attack, (ii) `isAssociatedTo`, which is defined by the concept `Probe`, and (iii) `indicates`, which is defined by the concept `Symptom`.

Symptoms are classified into Abuses, Misuses and Suspicious Acts. Each Symptom is characterized by the property `hasIntensityScore`, which reflects the probability of occurrence of the given symptom with regards to the specific monitoring method, and other properties, such as `Timestamp`, `Origin`, and `Destination`.

Probes are characterized by `MonitoringMethod` property, which defines the method used to monitor the symptom. Moreover, they are also classified depending on the specific architectural level to which they belong, namely Operating System Level Probe, Middleware Level Probe, Database Level Probe and Application Level Probe. A similar classification is adopted for Target that refers to anything that should be protected: Node, Software and Data. In particular, each Software component (i.e., Service, Application Component, System Component) is characterized by Port Number, and a Process Identifier.
The Attack Scenario identifies the correlation rule for the attack scenario recognition. It has proprieties hasDeadline, hasPrecondition and hasPostcondition. The former defines the time window by which monitoring the attack scenario. The latter defines the pre-conditions and the post-conditions, which describe the conditions required for including an intermediate attack in the considered scenario. Pre- and post-conditions have the properties: Time, which is the temporal condition for detection of an intermediate attack, and Predicate, which corresponds to a numerical or logical condition over the attack instance to be verified.

Attacks are also defined by hasEffect propriety, which represents a possible effect on the state of the monitored target component as well as the threat on a security objective (i.e., availability, integrity, confidentiality, authentication), if the attack is successful.

Effects may be characterized by: (i) Security Objective, and (ii) Effect Threshold, that identifies operation point beyond which is necessary to active the reaction. Moreover, Effect has several subclasses, including Illegal Access, Data Destruction and Information Leakage.

Finally, Reaction is related with a specific attack and a specific target. It can be exploited to mitigate attack effects. It is classified into Backward Recovery, Forward Recovery, and Reconfiguration, which indicate the type of remediation action.

Given the extensibility of our ontology, new classes can be easily added. For example, as shown in Figure 1 (by the concepts depicted through dashed lines), we could inherit the similar structure as in the class Attack of the security ontology extracted from [7]. Attack has propriety isCausedBy, which is defined by the concept Treat Agent (i.e., the agent that exploits the vulnerability). Moreover, Threat Agent is connected to the concept Vulnerability through exploits propriety.

IV. INTRUSION DETECTION AND DIAGNOSIS SYSTEM ARCHITECTURE

The architecture of the implemented Intrusion Detection and Diagnosis System (ID²S) is shown in Figure 2. It is characterized by several logical entities hierarchically organized: Probes, Agents, Decision Engine, Adjudicator, Remediators, and Monitors [8].

- **Probes:** They are entities deployed in strategic points in the system, and at different system architectural levels, which ensure the diversity with respect to information sources. They infer attack symptoms from real-time traffic and specific logs. Each probe monitors specific features of a single node or a software component. For example, a host monitor measures CPU utilization, user logins, disk activity, a software observes monitors the unauthorized access to application data, and so on. Each probe can assess if an anomalous activity is underway on the base of its local view, and send detailed security messages to nearby Agent. They can adopt different detection methods (both anomaly- and misuse-based), which triggers different security message formats.
- **Agents:** They are autonomous software components that analyze and forward probe messages to the Decision Engine. Each Agent converts raw security data into attributes usable by the Decision Engine (e.g., timestamps, probe identifier, source and destination of the anomalous activity).

To decouple Decision Engine from the specific format of the Probe messages, Agents perform a normalization process, that enables different kind of Probes to generate messages using an unique language. In this work is adopted a standard representation of symptoms based on the Intrusion Detection Message Exchange Format (IDMEF) data model [9].

- **Decision Engine:** It performs both attack monitoring and intrusion diagnosis activities. In particular, it allows (i) to aggregate symptoms of monitored behaviors, (ii) to recognize what attack can be hypothesized to be the cause of the anomalous symptoms, (iii) to diagnose the intrusions, and (iv) to detect complex attack scenarios.

As Figure 2 shows, the Decision Engine uses the defined knowledge-base in order to decide whether the observed behaviors represent potential attacks or not, as well as to perform the diagnosis of intrusions.
The enabling components are the Stream Processor, the Filter, the Diagnoser, and the CEP.

- The Stream Processor aggregates continuous events data streams in real-time or near real-time fashion. It uses a correlation schema defined by the knowledge-base to infer a set of queries to be performed on the incoming messages streams. For each monitored attack type it extracts the corresponding monitored symptoms, and forwards them to the Filter.

- The Filter is a software component which performs the ranking and filtering process of the produced events, in order to determine if the aggregated symptoms represent a real attack in progress in the system. Moreover, Filter leads to identify what attacks can be hypothesized to be the cause of the identified symptoms. Finally, for each detected attack, an alarm is forwarded to both the Diagnoser and the CEP.

- The Diagnoser is a component that performs the diagnostic process. It determines whether the monitored attack was either an intrusion or a no relevant attack, and what parts of the system are supposed to be the target of such attacks. It is responsible to assess the intrusion effects on the target components for each received security alarm. Finally, the Diagnoser forwards the diagnostic results to the CEP.

- The CEP is in charges of performing real-time complex queries on different data streams in order to detect attack scenario. Knowledge-base is used to automate the process of deriving roles for recognition of attack scenarios.

- Adjudicator: It is responsible to identify the best action to take in order to mitigate the intrusion effects on the target components for received security alarms. The information produced during this activity is used to instigate the remediation.

- Remediators: They are components distributed across the system infrastructure, which are designed to recovery to a specific attack or intrusion. In particular, they receive control information from the Adjudicator concerning a particular fault treatment, and perform the associated actions designed to react to the attack (e.g., system reconfiguration, revoke permits access).

- Monitors: They are central analysis servers, which receive data from a Detection Engine. They provide an interface that allows: (i) to see the current attacks status; (ii) to make interactive querying of attack data for analysis, and (iii) to identify attack patterns.

A. Implementation details

This Section presents some details of implemented components:

- Agents gather information from multiple data sources distributed across the system infrastructure. To cope with the heterogeneity of the formats of data generated by probes, a structured data organization can be exploited by adopting grammar-based parsers. They translate raw events to an intermediate format, so that they can be merged in a single data stream for further processing. Depending on specific implementation choices and/or deployment requirements, such components may be either implemented/deployed as relatively autonomous software modules or integrated in specific subsystems of the detection framework. The current implementation of the Adaptable Parser relies on Java Compiler Compiler (JavaCC) technology [10].

- Stream Processor is based on a software implementation described in a previously work [11]. It operates a first level screening of events, and splits the main stream into a set of individual streams consisting of events which represent symptoms of the same attack.

- The Diagnoser is based on a Simple Event Correlator (SEC) [12]. It is an open-source rule-based event correlation tool, created in an academic context. SEC can be seen as a complex, context-aware filter that selects and correlates relevant information based on matching rules defined using regular expressions; rather complex matching patterns can be defined in a compact way that would otherwise be quite awkward to express. Rules are defined in specific configuration files (text format), which can be refreshed at run-time keeping the status of the ongoing correlations [13].

- The CEP engine is based on Borealis [14], an open source framework for query processing of event streams. It performs SQL-style processing on the incoming events streams, without necessarily storing them. It uses specialized primitives and constructs (e.g., time-windows, logical condition) to express stream-oriented processing logic. Queries consist of merging streams produced by Ranker and Diagnoser components via Join constructs. In particular, on the base of information described by the knowledge-base (Time-Condition and Predicate proprieties), different temporal and logic conditions can be applied to every stream.

- An open source framework, named Prelude [15] has been adopted, which acts as an event bus between the Agents and the Decision Engine (Fig. 3). In particular, Agents report events in centralized fashion using local connections to a Prelude ‘local-manager’ server. The Prelude local-manager server processes received events and delivers them to both a specified media (mysql database, postgresql database, XML file) and the Prelude manager of the Decision Engine. Relaying is a feature that allows a Prelude-manager to ‘forward’ received events to another ‘prelude-manager’
as represented in Figure 3. For this reason, Prelude provides a library (called Libprelude), that guarantees secure connections (i.e., TLS connections) both between the Agents and the local-manager and among Managers them-self. Moreover, Libprelude provides the necessary functionality for generating and emitting IDMEF events and automates the saving and retransmission of data in times of temporary interruption of one of the components of the system.

Finally, a ‘Prelude console’ has been extended to implement Web Monitors used to view the detected attacks and the system status.

V. Operation of the ID\textsuperscript{2}S

Understanding the security status of a complex system needs combining observations performed across several entities distributed within the system. In particular, in order to increase the detection coverage of intermediate attacks as well as reduce the time and the cost of managing the large number of false positives, an hybrid event correlation approach based on diversity both in information sources and methods used to detect malicious activities is adopted. Then, in order to recognize complex intrusion patterns and enrich the semantics of diagnosis, a CEP-based correlation approach is adopted, which captures the causal relationships among intermediate attacks, by correlating them on the base of temporal and logical constraints.

A. Intermediate attack recognition

In order to detect the intermediate attack symptoms, in a previously work [3], we defined different probes, which used either anomaly detection methods (AD) or misuse detection methods (MD).

As AD methods, a ’score’ is assigned to the generated events, which reflects the intensity of the given anomaly with regard to an established profile. For each observed feature, the AD probe can perform in one of three phases: training, validation, and testing. In the training phase, data sets are used to parameterize the detection method (necessary to determine the characteristics of expected behavior). The suspicious requests are manually extracted in order to guarantee that the data sets are (as much as possible) attacks free. The validation phase aims at validating the detection models and to establish thresholds, that are used to distinguish between regular and anomalous behaviors during the testing phase. In the testing phase, the AD method is used to monitor anomaly behaviors with respect to the desired profile computed during the training phase. The choice of a proper threshold value is the main problem in this process, since there is a trade-off between the number of false positives and the expected detection accuracy: a low threshold can result in many false positives, whereas a high threshold can result in many false negatives. Once the profiles and thresholds have been derived, the testing phase is operated. If the computed ’score’ exceeds the fixed threshold a message is triggered.

As for MD methods, since they also produce false positives, the ’score’ of the generated messages is fixed to a value from 0 to 1. It is estimated during a validation phase, and represents the likelihood to monitor correctly an attack symptom.

Finally, in order to increase the detection coverage and reduce the time and the cost of managing the large number of false positives generated by probes, a “clustering” correlation approach is adopted to aggregate attack symptoms (based on similarity among symptoms attributes [16]).

B. Attack scenario recognition

During an attack scenario, it is more likely that the attacker first performs some knowledge gathering steps, which consist of a set of commands that enable him/her to gain knowledge about the target system, and then performs the intrusion. For example, during an attack scenario, a port scan can be adopted to trace the open ports on the target server. It can be only the first step of a more complex attack, which should point with an intrusion in the system data base.

Figure 4 shows examples of possible attack scenarios. They consist of sequence of more common intermediate attacks followed by an attacker to discover and exploit a ‘SQL Injection’ vulnerability [17]. In particular, the considered intermediate attacks are:
**A1** - **Port scanning**: A scanning procedure is adopted to trace the open ports on different system nodes (e.g., distributed port scan in space and in time).

**A2** - **Telnet**: A discovered port 80 (on a specific system node) is used to determine if a web server is listening on that port (e.g., by a telnet request).

**A3** - **Directory traversal**: By sending to the Web server a custom request, consisting of a long path name created artificially by using numerous dot-dot-slash, the attacker gains a listing of the web server directory contents. During this step, a hidden web page (no public link to that page is published on the web) is discovered.

**A4** - **Policy violation & buffer overflow**: The attacker enters strings with variable length as GET parameters of the requests to the web page discovered. With very long strings an error message is displayed within the Web page. The page contains the structure of a query performed by the application in the database (e.g., login query).

**A5** - **SQL injection**: Known the query structure (e.g., in terms of tables in which are located confidential information), the attacker performs a SQL Injection attack by HTTP POST requests.

Figure 4 shows some of symptoms and effects for each intermediate attack. As described in Section II, the pre-conditions specify the context to be satisfied for attack to succeed. They are verified by the outcomes of monitoring phase (attack symptoms). The post-conditions specify the effects of intrusion. They are verified by means of data inferred by information reported by the diagnosis outcomes. Pre- and post-conditions are specified by logical and temporal conditions (predicates), which are combined by using logical connectives (conjunctions and negations) among them. Table I presents an example of the pre- and post-conditions that have to be verified in order to recognize Scenario3 represented in Figure 4.

In general, for each known scenario, it is possible to define a complex query that can be performed by CEP in order to correlate symptoms and effects of such an attack. The query operate on the two input streams. ‘SYM’ is the input stream associate with the symptoms, whereas ‘EFF’ is the input stream associate with the effects. The query execution generates a new tuple that is added to a corresponding output stream.

For example, the following query filters the pre- and post-conditions of a generic attack ‘Ax’, within a temporal window.

```sql
// Attack pre- and post-conditions matching
Stream In SYM
Stream In EFF
Stream Out Ax

SELECT COUNT(*) AS eventID // Assign event identifier
SYM.IP_Source AS PRE.IP_Source, ...
EFF.IP_Target AS POST.IP_Target, ...
EFF.time AS POST.time, ...
FROM SYM, EFF
WHERE (SYM.attackType = EFF.attackType)
and ABS(EFF.time - SYM.time) < T)
```

Using events received by output streams associated with the attacks, other queries are performed to recognize the attack scenarios. For example, the following symbolic query is used to recognize Scenario3 (we supposed that the intermediate attack sources are the same):
RECOGNITION OF KNOWN SCENARIOS
Scenario3 detection

Stream In A3, A4, A5, A6
Stream Out SCENARIO3

INSERT INTO SCENARIO3
SELECT A6.POST.time, A6.POST.queryResult, ...
FROM {{

// Post-conditions of A3 and pre-conditions of A4
// have to be verified. Moreover, the source and
// the target of both attacks have to be the same
// (i.e., the same IP address and port)
A3 JOIN A4 ON
(A3.POST.keyword = true)
and (A4.PRE.pageDiscovered = true)
and (A3.POST.target.IP_addr = A4.POST.target.IP_addr)
and (A3.POST.target.port = A4.POST.target.port)
and (A3.PRE.source.IP_addr = A4.PRE.source.IP_addr)

// Post-conditions of A4 and pre-conditions of A5
// have to be verified. The source and the target of both
// attacks have to be the same (i.e., the same web page)
JOIN A5 ON
(A4.POST.violation = true)
and (A5.PRE.violation = true)
and (A4.POST.target.IP_addr = A5.POST.target.IP_addr)
and (A4.PRE.source.IP_addr = A5.PRE.source.IP_addr)

// Some query keywords used in A6 is the same
// discovered during A5
JOIN A6 ON
(A5.POST.keywords = A6.PRE.keywords)
and (A5.PRE.source.IP_addr = A6.PRE.source.IP_addr)
WHERE {
// Temporal conditions
(A3.PRE.time < A4.PRE.time)
and (A4.PRE.time < A5.PRE.time)
and ((A6.PRE.time - A3.PRE.time) < deadline)
}

// A new tuple is added to the output stream only if it
// is not already included in SCENARIO1 or SCENARIO2
EXCEPT (SCENARIO1 OR SCENARIO2)

In general, an attack scenario can be described by a
state-transition-based language used to describe sequence
of malicious actions performed by attacker [19], and it can
be learned from training datasets using data mining and
machine learning techniques [20]. On the other hand, this
approach is restricted to known scenarios, which have to be
described by a human expert or learned from training data
sets.

The pre- and post-conditions technique (adopted in this
work) has the potential of discovering unknown attacks
patterns by matching the consequence of some previous
alerts with the prerequisite of some later ones.

In particular, in order to detect unknown attack patterns,
a more complex query schema has to be defined and
performed. For example, if we consider four type of attacks
\{A, B, C, D\}, and assuming that (in order to simply the
considered example):

1) \( (A, B) \) is the generic query used to verify the match-
ing between the A post-conditions and the B pre-
conditions;
2) for each scenario, A can only precede the other attack
types \{B, C, D\}, B can only precede the attack types
\{C, D\}, and C can only precede the attack type D,

the following decision tree has to be implemented by CEP:

<table>
<thead>
<tr>
<th>Program 1 Recognition of unknown attack scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>if ((A,B) and ((B.time - A.time) &lt; deadline))</td>
</tr>
<tr>
<td>if ((B,C) and ((C.time - A.time) &lt; deadline))</td>
</tr>
<tr>
<td>if ((C,D) and ((D.time - A.time) &lt; deadline))</td>
</tr>
<tr>
<td>if ((A,B,C,D) and ((D.time - B.time) &lt; deadline))</td>
</tr>
<tr>
<td>Scenario(A,B,C,D)</td>
</tr>
<tr>
<td>else Scenario(A,B)</td>
</tr>
<tr>
<td>else Scenario(A,B)</td>
</tr>
<tr>
<td>if ((A,C) and ((C.time - A.time) &lt; deadline))</td>
</tr>
<tr>
<td>if ((C,D) and ((D.time - A.time) &lt; deadline))</td>
</tr>
<tr>
<td>if ((A,C) and ((C.time - A.time) &lt; deadline))</td>
</tr>
<tr>
<td>if ((C,D) and ((D.time - A.time) &lt; deadline))</td>
</tr>
<tr>
<td>Scenario(A,C)</td>
</tr>
<tr>
<td>else Scenario(A,C)</td>
</tr>
<tr>
<td>if ((B,C) and ((C.time - B.time) &lt; deadline))</td>
</tr>
<tr>
<td>if ((C,D) and ((D.time - B.time) &lt; deadline))</td>
</tr>
<tr>
<td>if ((A,D) and ((D.time - A.time) &lt; deadline))</td>
</tr>
<tr>
<td>Scenario(A,D)</td>
</tr>
<tr>
<td>else Scenario(B,C)</td>
</tr>
<tr>
<td>if ((C,D) and ((D.time - C.time) &lt; deadline))</td>
</tr>
<tr>
<td>Scenario(C,D)</td>
</tr>
</tbody>
</table>

Figure 5. The monitored symptoms and effects
The previous algorithm can be implemented by performing three sequential query sets, in which an output stream of a query set is the input stream for the next query sets:

1) \((A,B); (A,C); (A,D); (B,C); (B,D); (C,D)\);
   with the general query \((X,Y)\) equal to:

```
SELECT X.PRE.time AS PRE.time, X.PRE..., Y.POST.time AS POST.time, Y.POST...
FROM X,Y
// Conditions to be verified
WHERE (X.POST = Y.PRE)
```

2) \((A,B) \Join (B,C); (A,B) \Join (B,D); (B,C) \Join (C,D)\);
   with the general query \((X,Y) \Join (Y,W)\) equal to:

```
SELECT XY.PRE.time AS PRE.time, XY.PRE..., YW.PRE.time AS POST.time, YW.POST...
FROM XY,YW
WHERE (XY.PRE = YW.POST) and (XY.eventID = YW.eventID)
```

3) \((A,B) \Join (B,C) \Join (C,D)\);
   with:

```
SELECT XYW.PRE.time AS PRE.time, XYW.PRE..., WZ.PRE.time AS POST.time, WZ.POST...
FROM XYW,WZ
WHERE (XYW.PRE = WZ.POST) and (XYW.eventID = WZ.eventID)
```

If the second assumption is not true, a more complex algorithm has to be defined and performed by CEP.

VI. RELATED WORK

In order to improve the attack detection rate, enrich the semantics of alerts and reduce the overall number of false alerts, different work propose explicit alarm correlation techniques [18], [24], [25]. In particular, [26] proposes a correlation workflow intended to unify the various steps of the correlation process. They adopt an approach that combines events that represent the independent detection of the same attack occurrence by different probes. Our work is strongly inspired by this framework, but we proposed and implemented a ID²S prototype that combines different correlation techniques according to an hybrid approach and employs open source CEP technology in order to process events and discovers attack patterns among multiple streams of event data.

Several open source IDSs have been proposed. The most widely known and used is Snort [21]. It is a Network IDS, which performs real-time traffic analysis and packet logging on IP networks. Even though some works addressing distributed IDSs [22], [23], to the best of our knowledge no concrete application seems to have been developed which addresses the issues of gathering intermediate attacks from distributed nodes in order to recognize complex attack scenario. Moreover, these works propose the correlation of alerts produced by network sensors only. This could bring IDSs to miss crucial details necessary for gaining the knowledge required to recognize relations among intermediate attacks. Our solution addresses precisely this issue through (i) the collect of streams of information at several architectural levels (i.e., Middleware, operating system, data base, and application), and (ii) the use of a CEP that offers great flexibility to the management of a complex correlation logic.

VII. CONCLUSION AND LESSONS LEARNED

This work a generic Intrusion Detection and Diagnosis System (ID²S) is presented. It implements an hybrid and hierarchical correlation approach, which captures the causal relationships among the alarms that represent intermediate attacks of a more complex attack scenario, by correlating them on the base of temporal and logical constraints. The correlation capability is based on a CEP and driven by knowledge-base, which is used to recognize attack scenarios.

A common weakness of this correlation technique is that it requires specific knowledge about the attacks in order to identify their prerequisites and consequences. Moreover, it is difficult to define all prerequisites and all of their possible consequences. Therefore, the major limitation of this technique is that it cannot correlate unknown attacks, since their prerequisites and consequences are not defined. Furthermore, for a multi-step attack, usually there is a time gap between executing two consecutive stages. Therefore, during a training phase should be necessary to extract time duration between steps of attack scenarios and use the results as a threshold in the framework to estimate the required time for observing the next stage of a specific attack scenario. On the other hand, it is hard to specify quantitative time constraints in scenarios, because the time gaps among each stage may vary a lot, depending on how hurried the attacker is. Moreover, attack profiles to be used for training are hard to find. Therefore, in this work alarms that cannot be merged to any scenario, within a deadline, are provided to the administrator individually.

In future work, we will aim to define approaches to automatically estimate the more probable attack scenario in progress in the system in order to active a reaction before that the attacker archives its final objective.
ACKNOWLEDGMENT

This work has been partially supported by the Italian Ministry for Education, University, and Research (MIUR) in the framework of the Project of National Research Interest DOTS-LCCI (PRIN-2008-LWRBHF) as well as by the European Community’s Seventh Framework Programme as part of the ICT MASSIF project (grant agreement no. 257644) and ICT mOSAIC Project (grant agreement no. 256910).

The final version of this paper is available at: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=6061035

REFERENCES


[10] JavaCC, a Java Compiler Compiler library, at https://javacc.dev.java.net/JavaCC.


