Development of a Network Alert Correlator and Framework

Rafael Fedler

July 2, 2014

Chapter 1

Introduction

Network alert correlation has been a topic of interest to the research community for at least 20 years [13]. Not much later, research towards a practical alarm correlation system was initiated [11], with a plethora of proof of concept and demonstration systems having been developed until today. However, there still exists no system which is reportedly used in a notable number of production networks, arguably due to shortcomings in detection capabilities of existing approaches. Thus, effective network alert correlation remains an unsolved problem. We aim to contribute to its solution with this software and report authored as part of an Interdisciplinary Project (IDP), a coursework requirement for acquiring the Master of Science degree in Computer Science at the Technische Universität München. The software will serve as a component in the ANSII research project for security in industrial information systems and networks [1].

Network alert correlation promises several benefits, depending on the capabilities of the respective approach:

- Combining several or many non-uniform alerts into one alert, thus reducing the overall number of network alerts which has become a big issue for security officers [30]
- Identifying multi-stage intrusions which consist of otherwise harmless or less critical steps [30]
- Providing network administrators with a global view of events in their network by combining local views of several monitors (network and host intrusion detection systems)
- The identification of a root cause for complex events and their effects
- Enabling operators to take effective and more efficient countermeasures against incidents

The objectives of this IDP are threefold:

1. Development of a software framework for easy development and deployment of correlator *modules*

2. Development of a correlator, to be deployed within said framework as a module

3. Design of test cases to facilitate (1) the validation of the correlator's efficacy and (2) aid in the authoring of new signatures/rules for the correlator

The software framework (1.) will be lightweight and take care of basic connection setup to data sources, such as intrusion detection system (IDS) monitors or a single data sink, for correlator modules at startup. It will receive and pass on incoming events in whatever format they are provided, potentially parsing or converting them if required. Correlated alerts generated by the correlator will be returned to the data sinks, data bases, or policy enforcers by the framework. Thus, any additional correlator modules to be plugged into the framework need only take care of correlation itself, and not of communication and configuration.

The correlator (2.) itself is the core part of this work and will be the sample correlator module to be plugged into the framework. It will process events received and passed on by the framework from one or many data sources in its environment. The correlator to be developed in this IDP pursues a "signature of signatures" approach: The signature or rule author will access fields of received events/alerts which in turn have been generated due to signature matches on IDS. Thus, signature matches are combined to deduce knowledge of *atomic events* that happened as part of more complex events, such as multi-stage or distributed attacks. Two possible and closely related approaches for implementing such a "signature of signatures" correlator have been envisioned by the author of this IDP. One is a largely completely novel solution making use of a new decision tree-like data structure. The second option is largely based on Esper, a readily available open source Complex Event Processing (CEP) engine. These two approaches, the one developed from scratch by the author and the one based on Esper, will be introduced and evaluated. The approach deemed most suitable will then be implemented as the correlator. The correlator framework alongside with the correlator module will contribute to the ANSII research project [1].

The test cases (3.) have two objectives: Firstly, they will, as indicated by the name, serve for testing the efficacy of the correlator. To this end, the test environment shall be able to simulate a series of potentially security-critical events which a correlator should be able to detect. In distributed attacks, for example, the test bed will forge source IP addresses to simulate many involved hosts, but also multi-stage attacks will be featured.
Secondly, the test environment will aid in the authoring of signatures/rules for the correlator. By enabling a signature developer to easily launch attacks, the developer can find out which events are reliably triggered and thus which events and event fields new signatures should be based upon.

1.1 Report Organization

This report is structured as follows: In Chapter 2, we will review existing related work and approaches at attack correlation. We will introduce the reader in Chapter 3 to the environment the correlator will run in and the framework which provides the interface to the environment for correlator modules. Chapter 4 will detail the two approaches at "signature of signatures"-based event correlation, one being completely novel and the other being based on the complex event processing engine Esper. The test environment will be covered in Chapter 5. We will evaluate our own correlation approach against existing correlators in Chapter 7, partially building upon the test environment of Chapter 5. Chapter 8 concludes the report.

1.2 Terminology

When not stated otherwise, *alarm* and *alert* will be used interchangeably. *Incidents* or *events*, also used interchangeably, trigger alerts or alarms.

Atomic events are events which trigger a single, non-correlated alarm by an IDS such as Snort or Samhain.

Chapter 2

Related Work

In this chapter, we review existing work in the field of alert correlation. Our review will be largely based on survey papers [29, 24] with detailed accounts of existing work, but will be expanded with our own findings. We will apply the same classification taxonomy as used in [29, 24], which is used by two independent papers, proving its suitability, and deemed fit by ourselves. Accordingly, in Section 2.1, we will cover scenario-based correlation approaches. Section 2.2 will deal with rule-based correlation, while Section 2.3 covers statistical and machine-learning based approaches. Time-based alerts tested for potential significance using statistical relevance tests will, only covered in [24], will be the topic of Section 2.4. We will mainly cover approaches already listed in above two papers, though with different focus and detail. Some additional papers will be included as well.

A survey of existing free and open source as well as commercial correlation solutions can be found in [18].

2.1 Scenario-based

Scenario-based approaches correlate alerts based on whether they can be combined to form a known attack scenario. To this end, even languages for scenario specification such as LAMBDA [3], STATL [5] and ADeLe [16] have been defined.

STATL [5] is a very powerful language, describing an attack's substeps and transitions from each stage of an attack to another using conditions. It allows for very fine-grained and flexible description of events and scenarios. It also allows for temporal conditions and even dynamic responses to scenario subevents through *code blocks*. While this approach is certainly very powerful and offers more precise description options, its strengths are also its weaknesses: Scenario definition is not trivial and potentially error prone; other formats can hardly be reused, and the language is very low-level. Also, it is not trivial to plug it into all systems while supporting all of its features.

ADeLe [16], while offering attack precondition formulation, is less powerful in dynamics and granularity. It also provides a number of response actions to counteract undergoing attacks at runtime. ADeLe, unlike STATL, can also describe preconditions, but these require in-depth knowledge about vulnerabilities present in a system. It is not specified how these are determined or fed to ADeLe.

The disadvantage of all of these three approaches is that they directly specify how the substeps of an attack can be detected [15]. This requires all substeps to be formulated explicitly for these systems, which is error prone, requires a lot of effort, and may sometimes be impossible due to shortcomings of the description language or a lack of knowledge. Also, these systems cannot easily cooperate with existing IDSs and use them as information sources.

The system devised in [4] establishes correlation based on duplicate matching (which would often be considered aggregation) and on consequence chains. The latter are similar to the activate/dynamic rules, now supported in standard Snort installations. Additionally, a number of "aggregation scenarios" are defined where certain fields of an alert match, implying different kinds of distributed attacks or precursors to attacks. As such, we do not consider it to be very powerful.

A more capable approached based on chronicles is described in [17]. It performs correlation based on temporal specifications of events that can be matched at runtime against chronicles of events. The temporal specifications contain event descriptors/names and specify in which order the atomic events can occur. This approach is capable of describing multi-stage attacks and their stages' single steps temporal relation. Its theoretical capabilities are similar to the ones of the approach later to be detailed in this work.

2.2 Rule-based

Rule-based approaches are a more flexible take at scenario-based correlation. Instead of defining specific scenarios, they describe preconditions and postconditions. This gives credit to the fact that subobjectives in courses of action with multiple steps can be achieved through different means. However, these systems often need precise knowledge and access to information in order to monitor whether conditions have been met. Measuring this is not trivial in practice. On the other hand, when such information can be assumed, these approaches are more flexible and yet less complex. Also, cooperation with established IDS systems is possible.

Examples for rule-based systems are PreludeIDS [21] or commercial systems as offered by McAfee [14] and others.

2.3 Statistical and Machine Learning

Statistical approaches measure properties to find deviations of statistical relevance. Techniques used from machine learning and pattern detection are widely used to this end, for example Bayesan networks. The advantage of this kind of approach is that it requires no expert knowledge or scenario/attack descriptions. On the other hand, it often requires a training set to function properly, and in case of "positivist" approaches, i.e., approaches trying to extract known patterns acquired through machine learning, will only recognize patterns learned during training. A "negativist" approach detecting everything not matching typical property distribution will not be very precise with high false positive rates, and establishing correlation in this case is even more difficult. Also, it will not be able to state *what* has been detected, but only *that* something has been detected. In general, these approaches are often not very precise.

Statistical and machine learning approaches have been put into practice in [22], [23] or [25], for example.

2.4 Classification of Our Approach

Our approach can be classified as *mostly rule-based*, with the possibility to specify concrete scenarios as well. In terms of correlated incident specification, it is almost identical to [20]. One major advantage of our system over [20] is that it works in real time.

Our approach does not specify how exactly the IDS should detect attack substeps, which (a) makes our approach more flexible, (b) scenario detection much easier and less error prone, and (c) allows for cooperation with existing host and network IDS. Our system only uses the detection capabilities of connected IDSs to detect attack substeps, which are finally correlated by our approach. These are advantages specifically over STATL, ADeLe and LAMBDA. On the other hand, in the trade-off of simplicity vs. complexity and granularity, these three systems outperform in the latter point. However, we argue that systems where scenario definition is extremely complex and error-prone are less likely to be deployed in production networks, and it is unknown whether substep descriptions as comprehensive as established IDSs' signature databases exist.

Chapter 3

Environment & Framework

In the following, we will introduce the reader to the scenario and environment the correlator will be deployed in. Requirements as well as system design and approach decisions are direct consequences of the scenario.

3.1 Correlator Environment

The correlator is dependent on knowledge about the occurrence of *atomic events*. This knowledge will be supplied by many IDS monitors. The more IDS monitors are positioned at different spots in a network, the more complete the correlator's view will become. The correlator will combine the atomic events observed by each monitor to form a global view, and check whether all atomic events forming one correlated event are present. The event feed is provided by an XMLBlaster instance in the backend. The correlator framework, which is intended to enable easy development and deployment of additional correlation modules, takes care of communicating with the XMLBlaster.

In summary, many IDS monitors, both network and host based, will stream their events to the central data store, an XMLBlaster instance. The combined feed is received by the framework, which will, after optional parsing and filtering, pass the alerts on to one or more correlator modules. Correlated alerts which have been newly generated will be passed back to the framework. The framework will then return them to the XMLBlaster. Optionally, the framework can be extended to supply correlated alerts to other entities, such as policy generators and/or enforcers, or evaluation and comparison components.

3.2 Framework

One objective of this IDP is to not only develop a correlator itself, but to integrate it into an extensible framework. This framework should allow for easily plugging in additional correlators and take care of input and output of event streams.

Common object-oriented modeling can define the input interface format of such correlator modules. However, as correlation happens at runtime and thus return data format is unknown at compile time, the return data format cannot be specified as part of the interface definition. This makes interface definition necessarily very lax. Correlator modules need to be trusted that they only insert objects of the expected format into the output stream.

Simple extensibility will be provided by a configuration file containing the names of correlator modules. The specified correlator modules will be located and loaded using reflection. Thus, the framework code does not need to be altered when new correlator modules are added. Only the correlator's class name needs to be added to a text-based configuration file. As stated above, however, interface definition cannot be strict enough to prevent loading misbehaved correlation modules.

For asynchronous and thus more efficient data processing and less communication overhead, the framework's public interface will receive callbacks upon each received new alert instead of busy-polling. These asynchronous callbacks will, after optional parsing and filtering steps, call into one ore more registered correlator modules. The correlator and its environment are depicted in Figure 3.1.

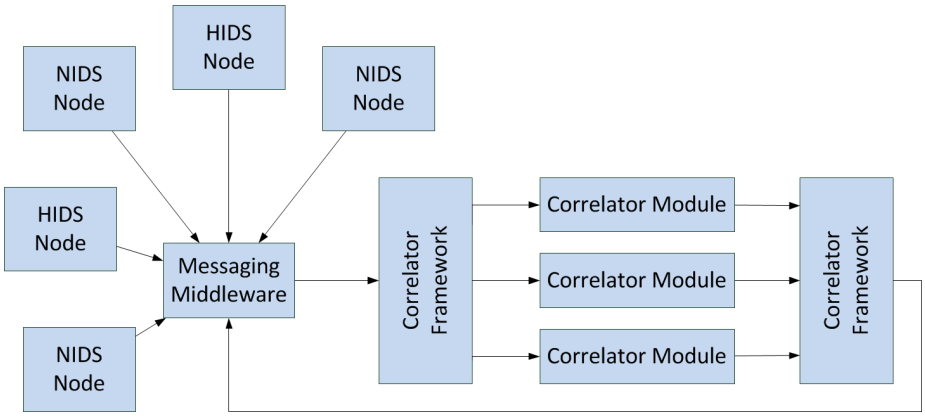

Figure 3.1: The correlator framework and its environment

3.2.1 Framework software architecture

The framework's software architecture is visualized in Figure 3.2. Asynchronous communication services with the XMLBlaster instance running on a remote machine are provided by the `Correlator_Input_Iface1` and `Correlator_Output_Iface1` classes which inherit from classes generated by the software environment the correlator framework is deployed it. The `Correlator_Input_Iface1` is asynchronously woken up when new messages arrive, and places a callback to the

`Policy_Correlator` class which acts as a mediator between the communication interface and proxy modules. This class keeps track of all registered and enabled correlation modules and passes data on to these modules. To allow for plugging in additional correlation modules, the class names of such modules are parsed from a configuration file and loaded at runtime through the class loader interface which is commonly used with reflection tasks. The correlation module detailed in Chapter 4 will, for instance, be listed in this configuration file. Once such a correlation module has generated an alert, it will pass it on to the `Correlator_Output_Iface1` class, which sends it to the XMLBlaster instance.

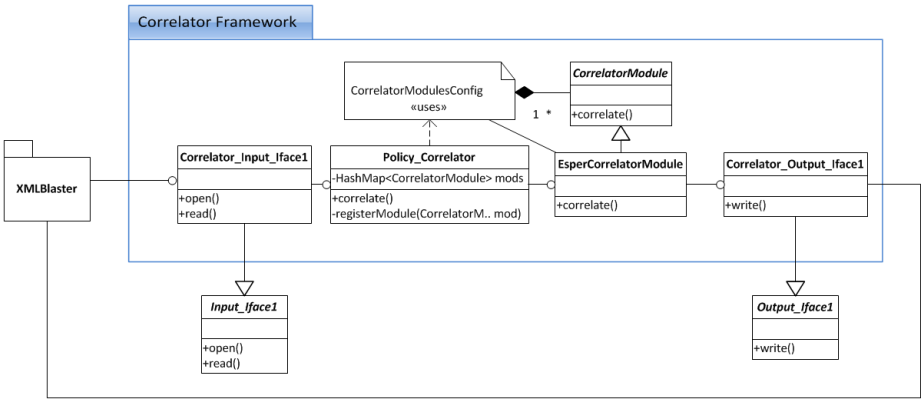

Figure 3.2: Framework software architecture class diagram

3.2.2 Module software architecture

Module software architecture is limited by the lack knowledge of return types at runtime. The processing model is asynchronous and needs to account for modules which most likely will only produce new alerts and send them to the output interface (cf. Figure 3.2) out of order.

Specific modules can be limited in their data's runtime return type through inheriting from interfaces they will use and putting checks such as `instanceof` operations in place. For example, if a module will use a certain API or library which accepts all objects, the interface class of this API or library can be inherited from to impose limitations on data inserted into this API or library.

Apart from that, interface definition is limited to methods provided and input data formats to be supported.

Chapter 4

Correlator

The author of this IDP has envisioned two potential approaches which differ significantly from the approaches presented in Chapter 1. Both of these are "signature of signatures" approaches in the sense that correlator signatures build upon *atomic events* which have been found by other IDS monitors, most likely by signature matches of their signature sets.

The major difference in the two approaches is how event processing and correlation is done in the back end: One approach, a novel development by the author, utilizes a new form of decision tree-like data structure which has been extended to persistently store intermediate results for such data that cannot be classified immediately, and to classify multiple events as groups. This kind of new decision-tree does not only classify one event at a time by sending it down the whole decision tree, but in case certain conditions dependant on other events have not been met, the event remains in a non-leaf node in the tree until the conditions for further traversal are met. This kind of decision tree has been named *persistent-storage enabled decision tree*, *stateful decision tree*, or *delayed decision tree* by the author.

The second approach uses a readily available complex event processing (CEP) engine called Esper [7]. It thus does not need to take care of how and in what kind of data structures the computations to combine events are carried out.

The basic assumption underlying both approaches is that event correlation can be expressed through the description of logical relation, as some kind of logical relation will always be given in a correlated attack. For example, in a distributed denial of service attack on one system, the common property, an equality relation, for all single events to be correlated will be the target system being identical. In a multi-stage attack, single steps will be linked transitively via the involved hosts serving as intermediates. We postulate that for any distributed attack or pre-attack activity, such a relation can always be found, as also from the attacker's perspective his course of actions is correlated. Otherwise, the adversary's actions would be arbitrary. Such logical relations can be used for classification and combination of events, and in turn to group them into a correlated event.

4.1 Approach Options

Two approaches are considered sensible by the author, both of which will be introduced in the following. The first approach is a custom data structure for non-immediate classification: A decision tree where input data can temporarily reside at inner nodes and only traverse to leaves, i.e. classes, at a later time when all conditions for traversal are met. This accounts for the temporal aspect of multi-stage attacks and the fact that the knowledge of a multi-stage attack is incomplete before each stage of an attack has been conducted. This option is effectively a completely new approach at CEP.

The second approach builds upon an already existing solution known by the name of Esper [7].

4.1.1 Persistent Storage-Enabled/Stateful Decision Trees

The first approach is an extension for decision trees. Decision trees are a well-known means for classifying data according to relations or criteria, such as "greater than", "less than", or "equal to". Most commonly, however, they have no notion of persistent storage. Any instance of data to be classified will traverse the tree until it reaches a leaf.

In our case, however, we want instances of data to be able to *remain* at a node in the tree. The reasons are the following:

1. To group multiple events with criteria referring to these multiple events, such as "at least n events with property x".

2. To account for the fact that atomic events comprising a correlated event will not arrive at the same time, but with a temporal discrepancy. This discrepancy can be huge, e.g., multiple hours in case of stealthy malware. Thus, we need to store atomic events that may be linked to events which might occur much later.

These requirements for (1) criteria and relations referring to multiple events and (2) accounting for temporal discrepancy lead to the design of *persistent storage-enabled*, *stateful*, or *delayed decision trees*, which temporarily store data in a node when no criteria for traversing to the next node are (yet) met.

To counteract memory consumption until depletion, a maximum lifespan for events has to be defined. This can be a global or a node specific maximum time to live value. A global limit will remove events from the data structure independent of the node they are currently stored in after a certain specified amount of time, while a node-local limit will only apply to events stored in that specific node.

Signatures

Signatures ought to be able to express any kind of relation which might occur in incoming events. We propose the following relational operators:

- $<, \leq$
- $>, \geq$
- $=, \neq$
- $\subset$ (e.g., for subnet definition or "contains" string operators)

Operands can be field values as provided by event data, or constants. Quantifiers are implicit: Whenever addressing two entities a and b, we assume a Cartesian product, i.e. we assume all pairs of a and b with matching criteria to be part of the solution set.

Fields can be addressed in typical "dot" notation: `a.dstip = 12.34.56.78`. This specifies that one event with destination IP 12.34.56.78 must be part of the whole correlated event. `a.dstip = b.dstip` means that one event a's destination IP address must be identical to another event b's destination IP address. This signature, as it refers to two events a and b, will trigger traversal of both events into the next node. Additionally, such a signature does not only apply to two specific events a and b, but to any pair of events a and b in the set of all received events and $a \neq b$. a always refers to the incoming event and all other alphanumerical identifiers to existing events, allowing for an arbitrary number of identifiers. It holds that any alphanumerical event identifier will be pairwise dissimilar to any event already in the solution set for another identifier. This, effectively, generates implicit Cartesian products on a set of criteria.

Additionally to prespecified operators, calls into Java methods are also, in general, possible using reflection, dynamic class loading, and method invocation. The signature writer needs to make sure, however, that the Java class he loads and the methods he invokes are existent and accessible from the scope of the query.

Examples A few example signatures will be given to facilitate a better understanding of syntax and expressivity:

- $a.dstip = b.dstip \ \wedge \ a.srcip \neq b.srcip \ \wedge \ a.time - 300 < b.time \ \wedge \ ''dos'' \subset a.description \ \wedge \ a.description = b.description$: This rule is a very basic signature for distributed denial of service (DDoS) incidents. By checking the time, which is assumed to be in Unix timestamp format, it considers all b within the last 300 seconds before the timestamp of a. Additionally, the signature queries the event's *description* field to look for a *denial of service* classification. This of course requires that the IDS monitors detect single-source denial of service attacks.

- $a.srcip = b.dstip \ \wedge \ b.srcip = c.dstip \ \wedge \ a.description = b.description \ \wedge \ b.description = c.description$: A very generic rule matching for transitively spreading events comprising three hosts. This signature would match any malware that spreads to one host A coming from a host B which has originally been infected by a host C.

Compilation of Signatures into a Single Tree

To receive a single data structure, the set of signatures needs to be compiled into a single tree.

For all atomic events comprising a signature to produce a correlated event according to each atomic event's specification, there needs to be a path to the same leaf for each of these atomic events. Tree compilation itself is trivial: For each signature, each relation (expressed by a subterm in the signature with conjuctions being the subterm's borders) not yet included will be given a new node and according edge. Already existing relations will just be ignored. Node rearranging, e.g. to have all relations and thus traversing options on one level of the tree, can help keep the tree small but is not a requirement. A single tree is necessary to keep track of conditions related to multiple atomic events memory-efficiently, so that every event can be potentially met in the same tree if required by a signature.

The organisation in a (potentially rearranged and thus minimized) tree erases the need for query optimization as the execution of the evaluation of subqueries follows a predefined order given a compiled tree.

Instead of compiling all signatures into a single tree, all of them can be maintained individually, i.e. as much smaller trees depicting one signature at a time. This would make the correlation process much more parallelizable and in extension much faster on modern multicore processors.

Ambiguity in Tree Traversal

We propose that every event shall not only traverse one path down the tree, but all paths whose signatures an event matches. This ensures that no decision towards one correlated event is made before all necessary information is available.

Sometimes, when being entered into the classification tree, an event may be ambiguous in regards to the branches to be taken: Multiple or no branches' conditions may apply. This can be the result of signatures which do not match for certain fields at all, because no signature properties have been defined.

Also, some events with completely identical signatures may belong to completely different correlated events. While the raw events may differ in some fields, the signatures might match for less fields. Thus, in these fields the events might seem identical.

Lastly, the aforementioned events matching multiple signatures and thus subtrees, may be part of completely different activities. What kind of activity they were part of can only be determined in the retrospective, however, once all other sub-events of the activity have been observed. Until then, it cannot be decided what kind of bigger event an event matching multiple signatures is part of.

Incomplete Signatures

Some signatures may be indifferent as to the value of certain fields, i.e., they do not match for certain constants or relations on these fields. Still, these signatures' matching events need to traverse down the tree.

The solution is to either skip fields, or to introduce "no-criteria" nodes and paths for each criterion. These are *only* traversed when there is no rule for this field (and the corresponding level in the tree), as the underlying signature does not specify only this field. Thus, the "all-matches-traversal" proposed in Section 4.1.1 should not apply to "no-criteria" nodes. They should only be entered/traversed when nothing else matches.

Example Tree

For a better understanding of the resulting data structure and classification process, we provide an example of a very simple tree for classifying different kinds of correlated events in Figure 4.1.

4.1.2 Esper

Esper [7] is a CEP framework readily available for Java and the .NET framework, the former being the language of choice for this IDP. Esper queries can be stated in an SQL-like language, decreasing the difficulty for the creation of new rules/signatures, as SQL is widely understood. Pattern matching for condition evaluation is performed using non-deterministic finite automata (NFA). Events and their conditions are mapped as trees with dynamic subtrees; dynamic meaning here that they create and destroy dynamically at runtime. NFA make pattern matching easily parallelizable, and dynamic trees quickly navigable and memory-efficient. [8]

Queries can be limited in the time window that needs to be considered, i.e., the events up until a maximum point in time in the past that will regarded for query evaluation.

Esper also allows for calling into Java/.NET methods at runtime and use their results for query evaluation. It can directly evaluate on Java objects as long as they follow the *Plain Old Java Object (POJO)/JavaBean* standard requiring getter and setter methods for all fields that need to be considered in queries according to a fixed naming scheme. Thus, parsing and conversion is not necessary if the data source is already POJO-style objects. This is the case in this project which is to be integrated into a larger architecture not detailed in this report.

Signatures

As Esper queries are formulated in Esper's SQL-like Event-Processing Language EPL, the signatures for the correlator will be as well. As stated in the introduction, our concept can be described as a "signature of signatures". Each event the correlator receives will be the result of a signature match on one of the

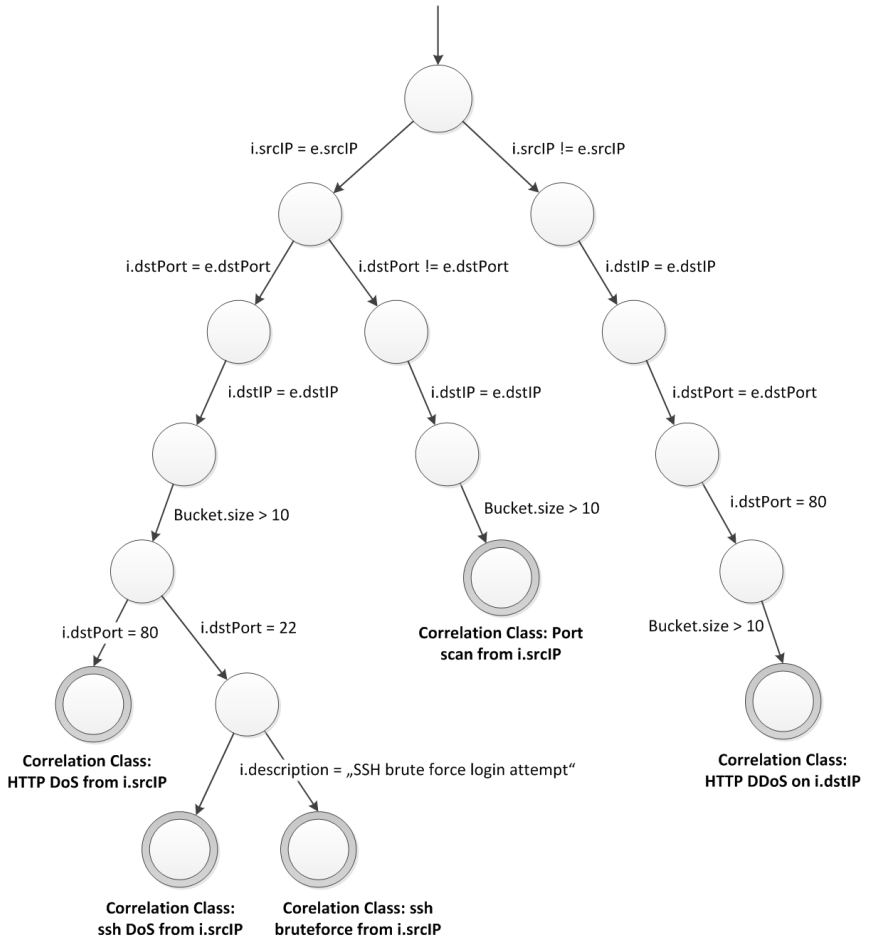

Figure 4.1: Example of a persistent-storage enabled classification tree

monitors. Each signature for a correlated event will match for the existence of all required atomic events as specified in a signature.

4.2 Approach Decision

Of the two approaches above, one will be chosen according to the following criteria:

- Effectiveness
- Development complexity and effort
- Signature format and complexity of signature generation

- Execution performance
- Signature expressiveness and flexibility

Effectiveness Both approaches are effective at solving the task of evaluating signatures for incoming events which do not only pertain to the event itself but also to future and past events. For persistent-storage enabled/stateful/delayed decision trees, it has been detailed how this is achieved. Esper, on the other hand, being a general purpose CEP engine, has already been deployed in a wide range of comparable scenarios for event processing.

Development complexity and effort The development effort when using Esper is obviously considerably lower than when having to implement our own, novel approach at event processing.

Signature format and complexity of signature generation The signature format of our own approach closely resembles typical first order logic syntax and the language used by `libpcap` filters as used by, e.g., Wireshark or tcpdump. Esper, on the other hand, uses a query language very similar to SQL. It is arguable whether our own language with implicit pairwise comparison/implicit Cartesian products or SQL with explicit joins is more difficult to write when complex signatures have to be created. We tend to assume that implicit pairwise comparisons reduce complexity, thus making signature authoring in our language a little bit easier.

Execution performance This criterion is hard to evaluate without benchmarks. In theory, both approaches should be equal in execution complexity: Given n events with m fields to evaluate on, $n * m$ single evaluations need to be carried out both in our persistent-storage enabled tree approach and in Esper. While not explained in great detail, query evaluation using Esper's SQL-like Event Processing Language is also tree-based in Esper as EPL statements are translated into trees. However, Esper uses many techniques known from database performance optimization to increase its throughput such as an indexed data store and query planning and optimization. It is not possible to evaluate execution performance of our own approach against Esper's, but it is highly likely that Esper's performance will be better in practice due to its various optimizations and long ongoing development.

Signature expressiveness and flexibility Though certain database operations, most notably different join types, are not available in our own language, these can usually be rephrased in statements without those joins. Both languages allow for calling into Java methods for statement evaluation. Esper is more convenient when formulating some queries through its support for SQL syntax which offers a broad range of operators. On the other hand, our own language provides implicit pairwise operations.

Decision

The author of this IDP report decides to use Esper. In theoretical aspects, the two approaches seem to be almost identical. Esper, however, is widely deployed, has a rich interface, allows for calling into Java methods and provides a proven and high performance engine with many optimizations. All of these would need to be implemented from scratch when pursuing the persistent storage-enabled/delayed/stateful decision tree approach. As there is no advantage in the storage-enabled/delayed/stateful decision tree approach, there is no reason not to use a readily available solution and to instead develop a complex system which does not offer any additional functionality or better suitability for this specific scenario. It might be more suitable for other scenarios, though, and is an option for potential future projects.

4.3 Correlator Module Software Design

This section describes software design aspects of the correlator module and how it integrates into the architecture and data flow of the correlation framework.

4.3.1 Data Flow

Following the well-known observer software pattern, developers building on top of the Esper engine can use classes implementing an `UpdateListener` interface to be notified by Esper once a query generates new data [6]. Each EPL query (i.e. a signature) corresponds to one `UpdateListener` which generates a specific classification text for the correlated alert.

As the results of a query are always provided as `EventBean` objects, correlated events need to be converted back into `Alert` objects which is also being passed as input to the correlator framework.

The resulting data flow model is depicted in Figure 4.2.

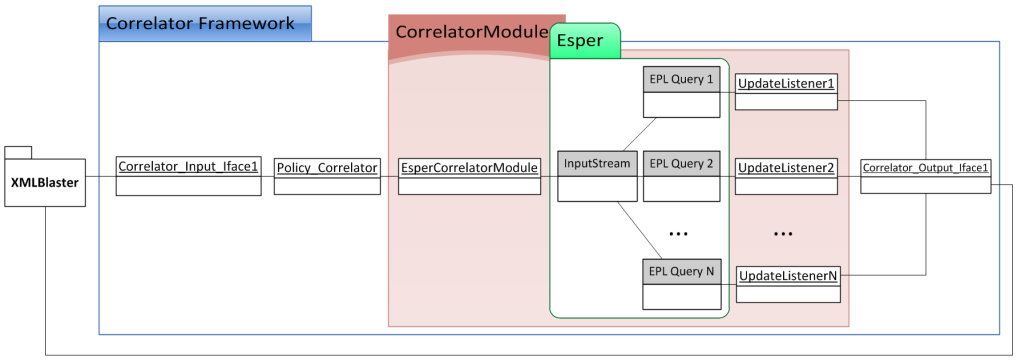

Figure 4.2: Data flow through the correlator framework and module

4.3.2 Signature Storage

Obviously, to eliminate the need to recompile the module when adding new signatures, these will be provided in a signature file. No integrity or sanity checks will be performed on this file, as checking compliance with a language that is very, but not completely similar to SQL is far from trivial. The module will rely on the signature set to be syntactically valid. This is the responsibility of the signature author.

A matter of interest in this regard has to be pointed out: This lack of checks of signatures is likely an attack vector in case an unauthorized third party gains write access to the signature store. An adversary can write signatures compromising general event processing in order to cover his activities.

4.4 Reuse of Correlated Events

Correlated events can themselves be part of other correlated events. Essentially, an attack can become arbitrarily complex and consist of arbitrarily many substeps. Thus, it can be meaningful to correlate already correlated events.

A simple example is a password brute force attack on the RDP service on a *Host X* preceding malware propagation after the RDP service's compromise. Then, the compromised host will start trying out passwords on other hosts where it has discovered a running RDP service. This course of action is taken, e.g., by the Morto worm.
In this example, multiple alerts for unsuccessful RDP password authentication can be correlated based on the target host, *Host X*. If now subsequently the same alert occurs for other hosts in the same network with *Host X* being the source host now, we can correlate these again to deduce that *Host X* has been compromised and either a multi-stage attack is underway or malware is spreading successfully. Additionally, this way we can reconstruct the whole path of propagation, if the means of propagation are constant among all involved hosts. Effectively, there will be a transitive relation over all hosts for which such a correlated event can be found.

In such a manner, we can recursively define arbitrarily complex distributed attacks without explicitly describing all minor substeps. Instead, we can reuse previously correlated events and correlate these again.

In our approach, the usability of already correlated events and their reinjection into the event input stream is covered by the infrastructural environment. All events of all monitoring IDSs are gathered in a central entity running XMLBlaster. New events generated by the correlator system designed and developed in this IDP will be transmitted to the XMLBlaster too, from where they can be fetched again.

4.5 Signature Stock

The presupplied signature stock shipped with the correlator module is intended to prove its effectiveness at detecting a wide range of distributed or composed incidents.

4.5.1 Network Incident Taxonomies

In order to provide proof that our approach is suitable for a wide range of complex or compound incidents, one needs to define this range and choose sample incidents distributed widely and equally across this range. To this end, we will use a network incident or attack taxonomy, categorizing incidents or attacks into several categories, and select sample incidents from each category that relates to our approach.

A multitude of attempts has been made at formulating network incident taxonomies. Process-driven taxonomies such as [12] or vulnerability-focused taxonomies like [2] will not be considered, as the system under development is focused on detecting incidents triggered over the network. This is a different level of abstraction, as we follow the "signature of signatures" approach explained before, and we have no information about the exact process for exploitation or the vulnerability being exploited. It is our intention to classify incidents or attacks only, and not attackers or their intention, or the systems under attack, the attacked vulnerability, et cetera. A classification with fewer dimensions is desirable for our objectives. The taxonomies we consider are:

- Hansman's taxonomy [10]: In its first and second dimension, this taxonomy is quite useful for our approach. The first dimension classifies according to the attack vector **or** the type of network attacks with some broad categories as used in casual attack descriptions. These include, for example, *virus*, *worm*, *trojan horse*, *denial of service attack*, *buffer overflow*. Its weakness lies in the inconsistency that arises from both considering attack vectors and broad terms for this dimension.
  The second dimension describes the target, though for our purposes, it is too specific: It describes whether hard- or software was attacked and what layer was target of an attack. In our case, a distinction according to classes of hosts would be more sensible: One host, multiple hosts in one subnet, multiple hosts in one organization, a series of hosts in a transitive connection, and so on.

- The taxonomy developed by the Swedish TS-CERT and, according to ENISA [9], popular with the European Computer Security Incident Response Team Network [19]: It is very flat, its categories apply universally to network incidents, and it does not require knowledge which cannot be gathered at our level of abstraction. Its *incident class* field representing its first dimension is intuitive and not inconsistent like the two ways of classifying in Hansman's taxonomy. Additionally, the *incident class* field roughly corresponds to information security goals such as confidentiality,

anonymity, authenticity, availability, integrity, and non-repudability. Oftentimes, the attacks trying to violate a certain security property proceed in a similar fashion. This way, attacks that work similarly are automatically grouped together in the *incident class* dimension of the European CSIRT/CERT security incidents taxonomy. The taxonomy's second dimension roughly equals Hansman's first dimension, minus the inconsistency. However, it lacks Hansman's taxonomy's dimension according to targets. This taxonomy is used by many European CERTs [9].

4.5.2 A Taxonomy Accounting for Correlation

While the two aforementioned taxonomies are the most suited out of the existing taxonomies for our purposes, they are not perfect: Hansman's taxonomy's first dimension allows for inconsistency and lacks the grouping of similar attacks. The European Computer Security Incident Response Team Network incident taxonomy lacks classification according to targets. Thus, we create a hybrid out of those two taxonomies to match our needs, combining the European CSIRT incident taxonomy's first two dimensions with Hansman's taxonomy's second dimension. The resulting taxonomy's dimensions are as follows:

1. Incident class
2. Incident type
3. Incident target (limited to network node abstraction, one of: local, one-to-one, many-to-one, one-to-many, one-to-one-transitively, many-to-many)

The last dimension's possible values as defined above enable us to classify incidents according to how they are correlated. Only correlated events (which cannot be single events by definition) are of relevance for this work. Correlation can occur in many ways, such as multi-stage attacks on one host (third dimension: "one-to-one"), multi-host attacks where exposed or less well protected hosts may be used to gradually work towards a target host or malware spreads transitively (third dimension: "one-to-one-transitively"), or attacks against whole (sub-)nets like BGP hijacking ("one-to-many") . The possible value "local" is irrelevant for us as we consider network incident correlation, but may be integrated into our correlator as future work incorporating host intrusion detection systems.

These values roughly represent two potential dimensions of correlation: Time and multiplicity. Events can be correlated in time if they represent a chain of events that are related by a shared objective that are carried out in sequence. Events can be correlated in multiplicity if more than one host acts is the source and/or more than one host is the target of related events.

4.5.3 Representative Incidents & Sample Signatures

Although we just described a full taxonomy, in order to demonstrate that our approach is effective at detecting all kinds of correlated attacks, it is sufficient

to show that our approach is capable of detecting attacks of all possible values in the third dimension. As stated above, the third dimension describes how a composed event are correlated on a network level: Is one host attacking many? Are many hosts attacking one host or one network, e.g., in a denial of service attack? Is one attacker trying to first gain access to unprivileged machines to finally compromise a target machine? Is malware spreading from host to host, infecting in increasing number of machines?

As a proof of concept, we want to show that and how signatures for all these kinds of composed and correlated events can be written, covering all possible ways of composition in the third dimension of the taxonomy. For obvious reasons, we will exclude *local* and single-event *one-to-one* events, as there is no correlation or composition in those kinds of events.

Our sample signatures, as well as all signatures of our approach, require that there are signatures for the sub- or atomic events of a correlated/composed incident in the connected IDS monitors. Otherwise, they would not trigger IDMEF messages to the correlator.

The sample incidents are as follows:

- *many-to-one*: Distributed denial of service attack on one host
- *one-to-many*: Portscan of multiple hosts on a net
- *one-to-one-transitively*: Malware trying to spread from an already infected host
- *many-to-many*: No idea yet

The sample signatures are as follows:

Many-to-one: Distributed denial of service

```
select a1.* from Alert.win:time(10 sec) a1
  where
     (classification.text.toLowerCase().contains('denial
     of service')
    or
       classification.text.toLowerCase().contains('dos'))
    and (select
       count(a2.source[0].node.address[0].address)
       from Alert.win:time(10 sec) a2 where
       a1.source[0].node.address[0].address =
       a2.source[0].node.address[0].address) > 3; --
       at least 3 different source IPs
```

Listing 4.1: Sample signature for a distributed denial of service attack

This query outputs all events within the last 10 seconds that had "denial of service" or "dos" in their classification and were generated by at least 3 different source IP addresses. We match for two different strings as input might not have necessarily been normalized. Note how data is extracted from the alert using nested field accesses as the alert is provided in nested POJO (plain old java object) format. Also note how Java methods are executed on the extracted data.

One-to-many: Portscan of multiple hosts

```
select a1.* from Alert.win:time(100 sec) a1
where
   classification.text.toLowerCase().contains("portscan")
  and (select
     count(a2.target[0].node.address[0].address) from
     Alert.win:time(100 sec) a2
       where a2.target[0].node.address[0].address =
          a1.target[0].node.address[0].address) > 2;
          -- at least 2 different destination IPs
```

Listing 4.2: Sample signature for a port scan of multiple target hosts

Port scans are a almost always one of the first steps of early reconnaissance of a target network for an attacker. This signature selects atomic port scan events from the same source IP address to at least two different target IP addresses.

One-to-one-transitive: Malware spreading from host to host

```
select a1.*, a2.detectTime, a2.source, a2.target,
   a2.classification
  from Alert.win:time(12 hour) a1, Alert.win:time(12
     hour) a2
  where a1.target[0].node.address[0].address =
     a2.source[0].node.address[0].address
    and a1.classification.text =
       a2.classification.text
    and a1.detectTime.ntpstamp <
       a2.detectTime.ntpstamp
```

Listing 4.3: Sample signature for a transitively occurring incident

This signature tracks arbitrary events that occur in transitive chains of hosts, for example malware spreading from one host to another.

Another instance of transitive correlated events are attack paths in general which can be reconstructed with according signatures. However, as each correlated attack can consist of a multitude of single sub steps, it is hard to formulate explicit signatures matching all those single sub steps.

Many-to-many

We leave this as an exercise to the reader.

4.5.4 Limitations of EPL

While SQL syntax is a subset of Esper's Event Processing Language EPL, SQL is not fully supported in every context. For example, subqueries inside another query's `where`-clause cannot contain the `group by` or `having` keywords. Furthermore, `join` operations cannot be performed on subqueries in a `from`-clause. Taken together, these are significant limitations on subqueries. This holds especially for queries that need to compute on data which depend on the data's relation to an aggregate function over all data inside the event stream.

There are possible workarounds for this. Firstly, one can create an additional Esper event stream to specifically provide the aggregated data that other queries might need. The problem with this approach is that in our case, we cannot determine beforehand what kind of data the dynamically loaded signatures might refer to, and we cannot assume that signature developers know the internal names of these extra streams. This would reduce signature creation flexibility greatly and we would need to create all kinds of aggregated data streams beforehand which might potentially be used. Thus, this workaround is not feasible for our approach.

Secondly, one can try to reformulate the semantics of subqueries using `having` or `group by` statements, or used as tables to be joined, as correlated subqueries in the `where`-clause. This retains the independence of signatures from data structures that are internal to the correlator and thus their flexibility and ease of writing. We used this approach for the example signatures above. However, reformulating uncorrelated subqueries using `group by`/`having` or used in `join` operations into a semantic equivalent might not always be possible.

4.5.5 Classification Keyword Nomenclature

Our approach needs to rely on the classifications of adjoined IDS monitors to know what kind of event occurred. Thus, keywords in signatures need to be identical to those of adjoined monitors. For this IDP, we choose to use keywords used in Snort signatures, as this IDP will be tested using Snort monitors later on and Snort is in widespread use in productive environments. For example, the port scan signature from above matches for the "portscan" keyword instead of "port scan", as Snort rules classify port scans using the former. Alternatively, one could also match for both cases or the existence of both "port" and "scan". For the simplicity of the examples, though, such measures are not deployed in our sample signatures.

Chapter 5

Test Environment

The objective of the test environment is to test the correlator whose implementation has been described in the previous chapter and its sample signatures. This serves as a general proof of concept demonstrating that our approach is capable of detecting the four different types of composed/correlated events defined in Section 4.5.3.

We will first describe the test environment setup which is based on the GNUnet Parallel Large-scale Management Tool (GPLMT). It allows for scripting and monitoring multiple hosts in a network. We will continue to describe the tests we run and lastly their results.

5.1 Environment Setup

The tests are conducted in our testbed. It contains a number of physical and virtual hosts, running mostly Linux configurations. Two sets of virtual hosts will take the attacker and target roles, respectively.

5.1.1 Hardware

Figure 5.1 shows the test bed. The three VMs running on Physical Host 1 have their incoming traffic scanned by a Snort instance. The attacking VMs are in another subnet on a different physical machine, Physical Host 2. Both machines are connected with a switch. Each physical machine corresponds to a different network in our setup. The Snort instance passes its alerts via IDMEF massages to an XMLBlaster instance, from where our correlator fetches them. If alerts are found to be correlated, the correlator will create an IDMEF correlated alert message and send it to the XMLBlaster.

5.1.2 Software

Tests are coordinated and automated using the GNUnet Parallel Large-scale Management Tool (GPLMT). It allows for host scripting via ssh and centralized

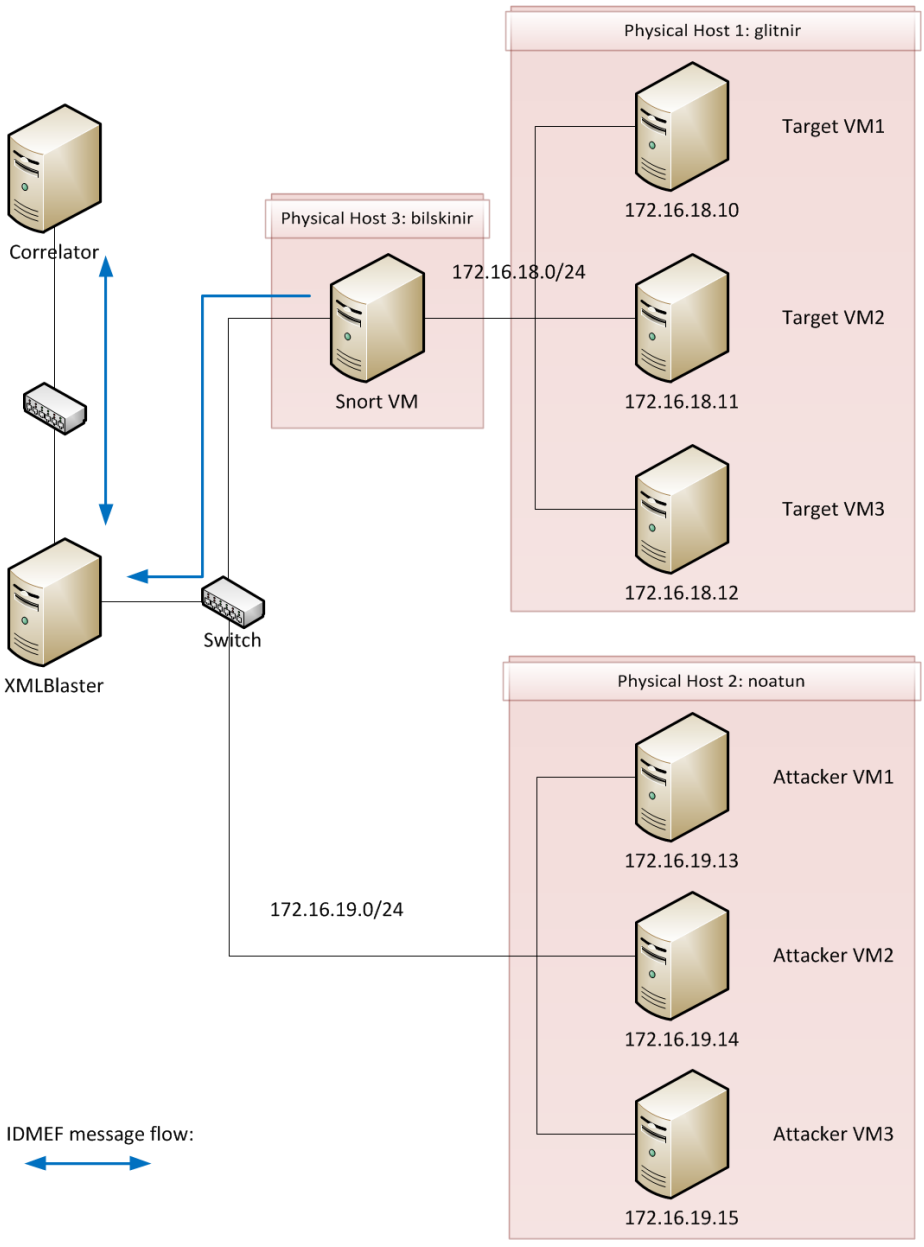

Figure 5.1: Network diagram of the testbed

task definition in an XML file. On the client side, we only require `echo`, `netcat`, and `nmap`.

5.2 Tests

The signatures described in Section 4.5.3 cover every form of correlation which we characterized in Section 4.5.2. The tests are intended to confirm the signatures' efficacy at this task. We formulate a test case for each signature and thus one test case for each form of correlation.

TC1: Many-to-One: Distributed Denial of Service

For this scenario, the three attack hosts in the test bed will flood one target host in the victim network.

It has to be noted that the Snort registered user ruleset does not contain rules for typical flooding denial of service attacks. The reason is likely that flooding is not primarily characterized by packet contents, but by the amount of packets per time and by state, or rather lack of the latter. It is still possible to write Snort rules covering rate-based denial of service with `detection_filter:track by_dst, count 50, seconds 1;` as part of the signature. However, for this test this is not necessary: It does not matter for our signature whether the underlying denial of service attack triggering the Snort instance to send out a "denial of service" alert is flooding-based or malicious payload-based.

As a consequence, to trigger atomic "denial of service" alerts by Snort, we created byte payloads that match existing Snort rules in the "denial of service" category. These payloads were stored in a script (Listing 5.1), which was uploaded to the attacker hosts in our testbed, and then executed via GPLMT.

```
echo -e "HTTP/ GET mi" | nc -n 172.16.18.10 443
echo -e "Cookie\x3A =\x0D\x0A\x0D\x0A" | nc -n
   172.16.18.10 80
echo -e "Range\x3Abytes\x3D0-\x2C" | nc -n
   172.16.18.10 80
echo -e "P=*?*?*?*?*?*?*?*?*?*?*?*?*?*?*?*?*?*?*?*?*?"
   | nc -n 172.16.18.10 80
echo -e "Range\x3A bytes\x3D" | nc -n 172.16.18.10 80
echo -e
   "\x80\xEB\x00\x00\x00\x00\x00\x0A\x00\x00\x06\x04"
   | nc -n 172.16.18.10 4569
echo -e "\x02SMB 2 \xFFSMBr" | nc -n 172.16.18.10 139
echo -e "\x02SMB 2 \xFFSMBr" | nc -n 172.16.18.10 445
```

Listing 5.1: Script to trigger atomic denial of service alerts

Due to the nature of `netcat` (TCP) connections which require a successful handshake before transmitting the payload, a listening server on the target port

is required. This can be accomplished through listening `netcat` instances on the server's side.

Trivially, though, one could also create test signatures that only contain prespecified test patterns. For example, if we created a test signature matching for the pattern `This is a Denial of Service attack`, we could generate an atomic denial of service alert just like this:

```
echo -e "This is a Denial of Service attack" | nc -n
    172.16.18.10 80
```

It would not matter for the correctness of our correlator-side signatures for distributed events in what way the atomic alerts they try to correlate are triggered. That is one of the strengths and weaknesses of our approach: We rely on the monitors to detect atomic events and trigger atomic alerts. For our correlation signatures, it is of no relevance how they were created, thus making them easy to test.

TC2: One-to-Many: Portscan of Multiple Hosts

These tests are carried out by running nmap on one attacking host. Port scans can be detected by the `sfPortscan` preprocessor of Snort.

To demonstrate our usage of GPLMT, we show one GPLMT tasklist in Listing 5.2.

```
<?xml version="1.0" encoding="UTF-8"?>
<tasklist name="Perform portscan of three target
    hosts"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="./tasklist_schema.xsd">
  <sequence>
    <run id="0" name ="Scan first host">
      <command>nmap</command>
      <arguments>-sT 172.16.18.10 -p 22,23</arguments>
      <timeout>30</timeout>
      <expected_return_code>0</expected_return_code>
      <expected_output></expected_output>
      <stop_on_fail>true</stop_on_fail>
    </run>
    <run id="1" name ="sleep">
      <command>sleep</command>
      <arguments>1</arguments>
      <timeout>2</timeout>
      <expected_return_code>0</expected_return_code>
      <expected_output></expected_output>
      <stop_on_fail>true</stop_on_fail>
    </run>
    <run id="2" name ="Scan second host">
```

```
      <command>nmap</command>
      <arguments>-sT 172.16.18.11 -p 22,23</arguments>
      <timeout>30</timeout>
      <expected_return_code>0</expected_return_code>
      <expected_output></expected_output>
      <stop_on_fail>true</stop_on_fail>
    </run>
    <run id="3" name ="sleep">
      <command>sleep</command>
      <arguments>1</arguments>
      <timeout>2</timeout>
      <expected_return_code>0</expected_return_code>
      <expected_output></expected_output>
      <stop_on_fail>true</stop_on_fail>
    </run>
    <run id="4" name ="Scan third host">
      <command>nmap</command>
      <arguments>-sT 172.16.18.12 -p 22,23</arguments>
      <timeout>30</timeout>
      <expected_return_code>0</expected_return_code>
      <expected_output></expected_output>
      <stop_on_fail>true</stop_on_fail>
    </run>
    </sequence>
</tasklist>
```

Listing 5.2: Sample GPLMT tasklist describing a testcase for a port scan on multiple target hosts

TC3: One-to-One-Transitively: Spreading of Malware

To test our signature for transitive malware infections, we alter the roles in our testbed a bit. Instead of considering the network 172.16.18.0/24 the target and 172.16.19.0/24 the attacker side, we now assume that both networks belong to the same company but may, for example, belong to different departments. We simulate the spreading malware infection as depicted in Figure 5.2.

Just like in Test Case 1 (TC1), we trigger atomic alerts by Snort via sending according payloads, similar to the script described in Listing 5.1.

5.3 Results

All three signatures successfully correlated the atomic events and lead to the generation of according correlated alerts. Thus, all three tests yielded a positive result.

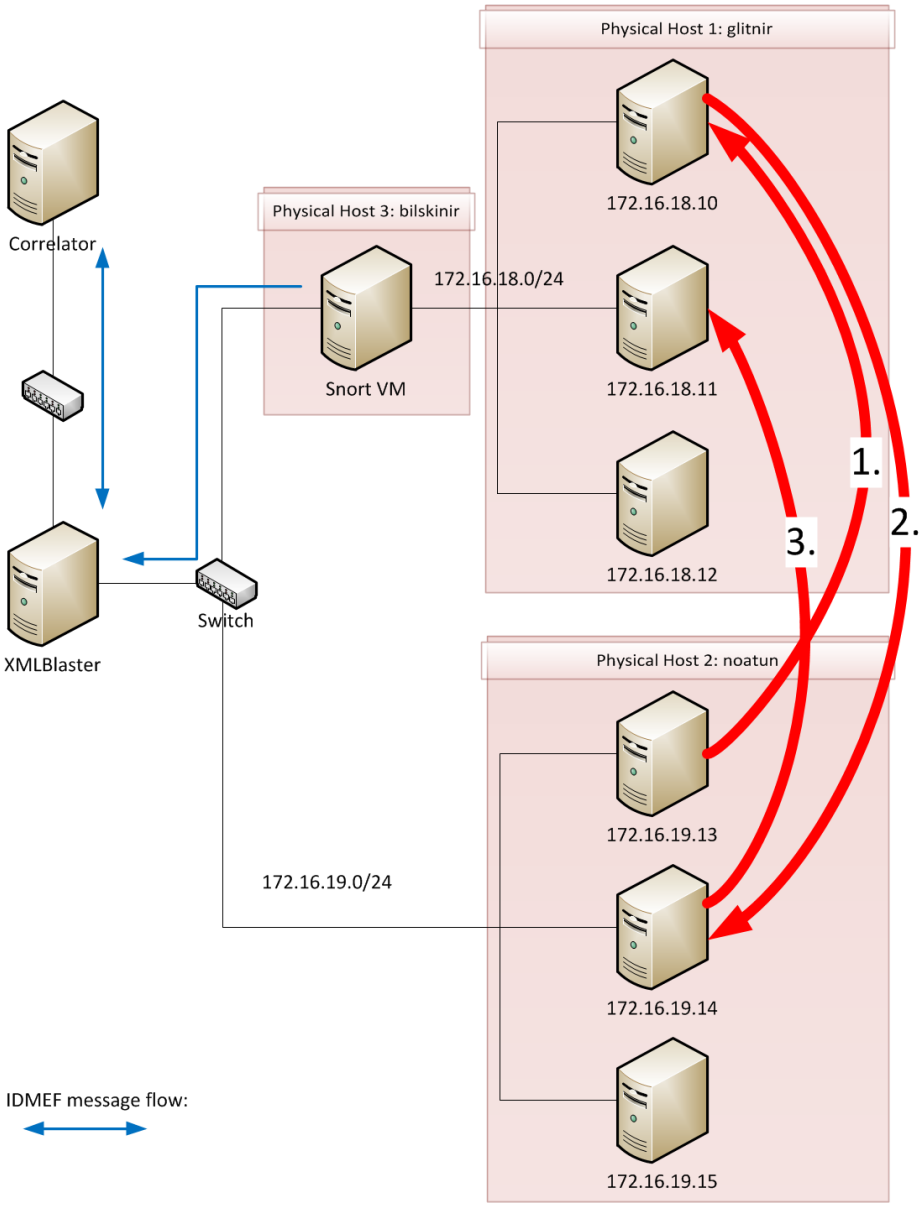

Figure 5.2: Simulated spreading of malware through our testbed

Chapter 6

Infrastructure Integration

While our correlator can be used as a standalone solution, part of the objectives of this IDP was the integration of our correlator as a component into the ANSII [1] infrastructure. ANSII, an acronym for *"Anomalieerkennung und eingebettete Sicherheit in industriellen Informationssystemen"* – Anomaly Detection and embedded security in industrial information systems – is a research project funded by the German Federal Ministry of Education and Research (Bundesministerium für Bildung und Forschung, BMBF), with its objectives being states as follows:

"The ANSII research project's aim is to develop of a procedure model for the integrating IT-security algorithms into industrial information systems. This model includes the selection of the assets that need to be protected, the threat analysis, risk assessment, and a selection of actions and implementations. Application-specific models and solution cores should be developed within the field of embedded systems, aiming at industrial information systems." [1]

The following subtasks were defined to integrate our correlator:

1. Development of a web user interface to be added to the central ANSII management for the configuration of the correlator, which is now an ANSII component

2. Integration of the correlator and its output into the Zabbix network monitoring solution to allow for central monitoring of correlation alerts

3. Integration of test cases for verification of the correlation rules' efficacy into Zabbix' web interface

Their successful completion will only be shortly reported on, as they are matters specific to the ANSII infrastructure and of no great value to the issue of correlation itself, which this report documents.

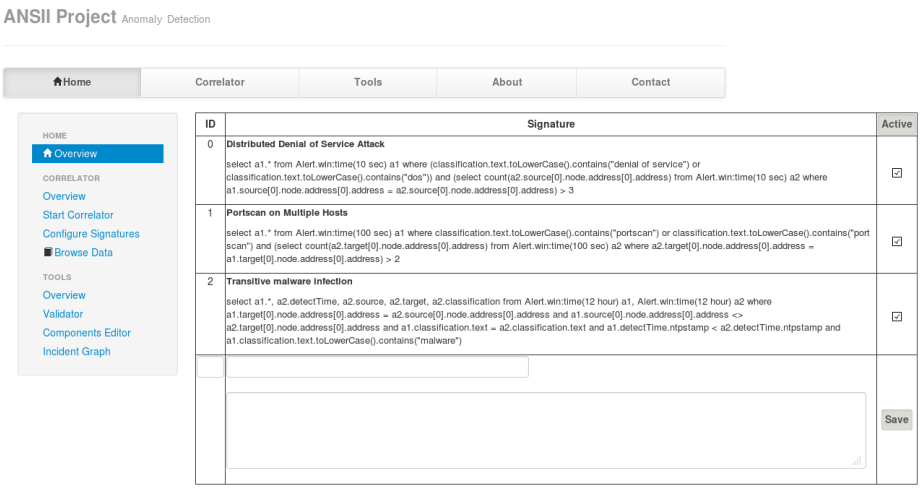

Figure 6.1: Web UI component for configuration of the correlator's signature database inside ANSII's management interface

6.1 ANSII Management Web UI Integration

ANSII's communication and persistent storage infrastructure is XML-based, as are the IDMEF messages that are provided by NIDS and HIDS nodes, processed by our correlator, and the alerts emitted as IDMEF in case a correlated event has been detected. Communication is performed asynchronously using the XMLBlaster middleware, while persistent storage, including that of configuration files and output generated by ANSII components, is supplied by the eXist XML database system. As a result, the central ANSII management web user interface was implemented as an eXist web app with server-side content generation and delivery being implemented in XQuery.

To allow for the central management of the correlator alongside with the other ANSII components, a tiny web interface, shown in Figure 6.1 with corresponding server-side XQuery code was developed. As of now, it allows for the configuration and extension of the correlator's signature database. Signatures can be added, activated, and deactivated. These are the only useful features for our correlator module; other modules, to be plugged into our framework, can add features accordingly.

6.2 Zabbix Network Monitoring Integration

Zabbix [28] is an open source network monitoring solution. It can gather vitality and resource usage data, perform active tests on monitored hosts using agent daemons, parse logs, and collect and visualize all data in one central monitoring interface.

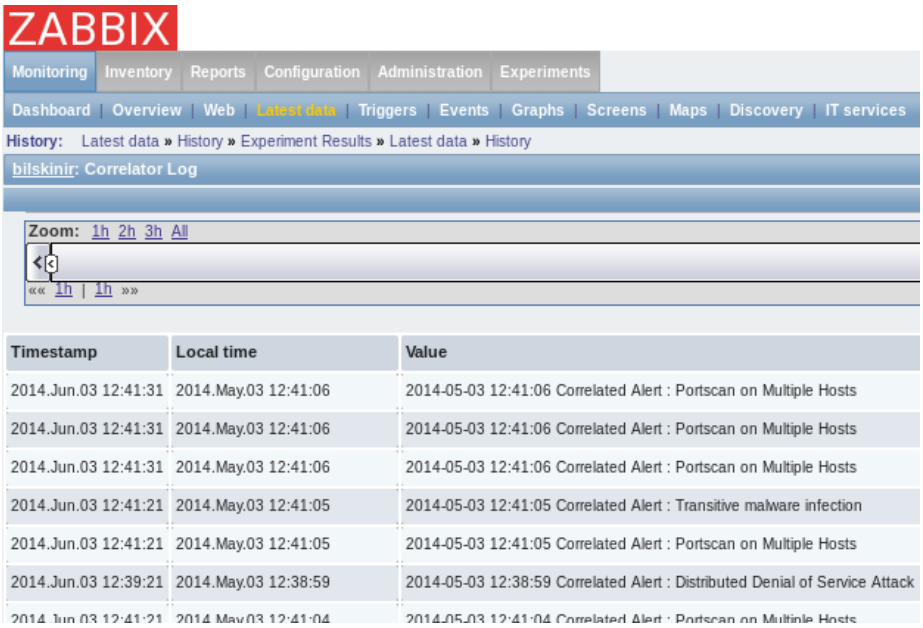

Figure 6.2: Log monitoring output on Zabbix server

To allow for centrally tracking correlation alerts alongside with the other monitoring data collected by Zabbix instead of requiring a separate interface just for tracking correlation alerts, our correlator's alerts were integrated into Zabbix accordingly. To this end, suitable monitoring target items [26] and triggers [27] were defined which evaluate a locally produced correlation alert log file.

The resulting log output on the Zabbix server is shown in Figure 6.2.

6.3 Test Case Integration into Zabbix

Zabbix is, by itself, only able to monitor and collect data, but not to carry out active tests. To add this functionality, a patch has been developed at the Chair for Network Architectures and Services of the Department of Computer Science of the Technische Universität München to integrate GPLMT into Zabbix. With this patch, GPLMT task lists like those specified in Section 5.2 can be uploaded to and executed from Zabbix.

The integration of tests directly into Zabbix has two advantages:

1. Easy verification of the correct configuration and deployment of the correlator as well as of the efficacy of its signatures: Tests can be launched from inside Zabbix and the results immediately show up in its monitored log data

2. Aiding in the development of new signatures: Complex attack scenarios or other network-level processes can be centrally specified and then be carried out by network nodes, making simulation of scenarios much simpler. Again, the correctness of new signatures can be verified as log data is available right in Zabbix. In summary, this allows for a "signature development lifecycle": (1) Scenario specification (2) Signature writing (3) Test execution (4) Result validation: If not satisfactory, return to Step 2.

For the sake of completeness and result documentation, a screenshot of test case integration into Zabbix is provided in Figure 6.3.

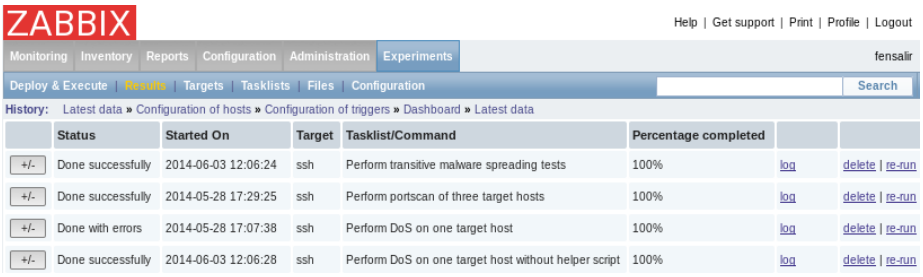

Figure 6.3: Test case integration into Zabbix using the GPLMT patch developed by the Chair for Network Architectures and Services, Technische Universität München

Chapter 7

Evaluation

As the previous chapter has shown, our approach is capable of correlating events related in the four different ways described in Section 4.5.3. As its efficacy has been demonstrated, we aim to compare it to other approaches for network event correlation in this chapter.

7.1 Comparison With Existing Approaches

System	Scenario-based	Rule-based	Statist./ML	Our Approach
Ease of Deployment	-	+	+	+
Ease of Sig. Def.	-	o	+	+
False Positives	+	o	-	+
False Negatives	-	o	-	-
Req. Expert Knowl.	-	-	+	o
Coop w/ ext. IDS	-	+	-	+

Explanations

Ease of deployment was chosen as low for **scenario-based approaches**, as they are often closely intertwined with their host systems. Their sophisticated, very complex, time-consuming, and error-prone signature definition process result in another low rating. On the other hand, these signatures prohibit false positives almost perfectly. False negatives, however, are high, unless a "complete" scenario signature set can be assumed, which is impossible in practice. Very high expert knowledge of all sub-aspects of an attack is required, resulting in another low rating. Finally, as these systems define how substeps are to be detected exactly, they also cannot cooperate with existing remote IDSs.

The ease of deployment has been rated as medium for **rule-based approaches**. They do not need close host system interaction for direct event detection. Signatures are easier to define as they do not necessarily require in-depth knowledge of all aspects of substeps. On the other hand, rule authors still

need to know about pre- and post-conditions. False positives can occur, though not to a high degree, as conditional rules only measure the effect of events, which might have been brought about by non-malicious activity. Flexibility in condition specification on the other hand can lead to lower false negative rates, as non-specific definition of all substeps will lead to more cases described by a rule. Again, expert knowledge is required, this time about system conditions. As these systems do not specify how exactly an event has to take place, they can be easily extended to cooperate with existing IDSs.

Statistical and machine learning approaches are very easy to deploy, as long as classification to specific attack types is not demanded, which would require training sets which correctly identify certain attack patterns. This is, however, not the common case. Blind approaches are much more common in this class. Thus, no expert knowledge is required, as signatures are unnecessary as well. False positive ratios and false negative ratios will naturally be quite high, though. Cooperative options with existing IDS systems are also not given.

Our approach is easy to deploy, as it only requires input from existing IDSs. Signatures are easy to write, as they do not require any in-depth knowledge of substep specifics which would need to be described. Only a listing of substep criteria is to be provided. False positive rates will be low, for the same reasons as with the scenario-based approaches. However, similarly, false negative ratios will be high as anything not described by signatures will necessarily be missed. Expert knowledge is required for signature definition, as one has to know at least the composing steps of an attack. This requirement is still lower than for other approaches which require more complete and exact knowledge about the detection process itself. Cooperation with existing IDS for a global view is not only possible, but is mandatory for our approach.

7.2 Detection Test Results Evaluation

It is very important to note that, due to our testing methodology, the evaluation of the detection capabilities is not to be understood comparative to other approaches in any way. As we test our correlator against a pre-defined set of events which we have previously used to author signatures/rules for the correlator in the first place, a detection rate of 100% is to be expected. Likewise, anything that is not covered by our signatures/rules will definitely not be detected. This is a general property of signature-based approaches. These tests, evaluating the true positive detections, were only meant as a proof of concept: We showed that our approach is capable of detecting a broad range of attacks through the signatures that can be authored and the event processing techniques that are used to process incoming atomic event alerts. In order to ensure that indeed a broad range of events is covered, we composed an attack taxonomy for correlated attacks in Section 4.5.2 and tested our correlator for the different kinds of correlation we characterized in Chapter 5.

7.2.1 False Positives

While signatures can detect exactly the kind of events they define and will succeed at that unless the properties of events change, it might be that the properties used in the signature are not only characteristic to malicious but also to neutral activity. Thus, imprecisely authored signatures/rules will trigger false positives, raising alerts for allegedly malicious events even though none took place.

The advantage of our approach in regards to false positives is that it relies on adjoined IDS monitors for the detection of atomic events. Under the assumption that adjoined IDS monitors do not produce false positives, the probability of the correlator producing false positives is significantly reduced as then it is certain that the atomic events indeed have happened. Thus, false positives – as long as no false positive atomic events are reported by IDS monitors – will only result from a badly authored signature, probably due to bad understanding of characteristic properties of composed events, but not due to detection mechanisms themselves, as they are offloaded to IDS monitors.

False positives are hence not a problem of our approach, but can only result from badly authored signatures or monitors falsely reporting atomic events.

To show that the very small signature set used for proving the general effectiveness of our approach does not produce a notable number of false positives or potentially even none at all, we placed a workstation computer in the test environment. From that workstation, we then simulated the following workloads, aiming to cover a broad range of different packet distributions:

- Browsing the web
- Searching the local AD domain for other PCs and printers
- Running a BitTorrent client and downloading a file
- Performing an nmap scan of a target host outside the test environment

Especially the last case, as nmap scans are often part of reconnaissance of potential target networks and machines, is of importance. It is also contained in our signatures as atomic events.

7.2.2 Results

False positives were indeed detected, but only as a consequence of false positive alerts of port scans generated by our monitoring hosts. As our approach depends on and completely trusts the IDS monitors' alerts, this issue cannot be resolved at our end.

7.3 Shortcomings of the "Signatures of Signatures"-Approach

As with any signature-based approach, each possible case that needs to be detected has to be formulated as a signature. While this is already a hard task for regular applications like network IDS signatures or anti-malware signatures, this becomes even more of a problem for correlated attacks, as now all possible combinations of atomic events need to covered. This results in an explosion of the event space signatures would need to be created for. A possible remedy is matching for generic keywords such as "worm" or "virus" or "exploit" which the atomic events' classifications might or might not contain. Either way, the correlated event space is greatly increased over atomic event space.

Chapter 8

Summary and Conclusion

In this report we presented a new signature-based approach at network incident alert correlation and put it into practice. We described design decisions and the implementation process. Furthermore, we composed a security incident/attack taxonomy from two existing taxonomies and extended it by adding a classification for the different ways in which atomic events can be related. The resulting taxonomy for correlated attacks, such as multi-stage attacks and attacks with multiple source or target hosts, accounts for two dimensions of correlation: Multiplicity relations of involved hosts (e.g., 1–n, n–1, n–m) and time. We formulated test cases covering those possible types of correlation and proved our correlator's efficacy at detecting them using proof-of-concept signatures.

We also introduced a new approach at complex event processing which we dubbed *persistent-storage enabled decision tree*, *stateful decision tree*, or *delayed decision tree*. This kind of new decision tree allows for delayed classification, postponing it until all information for classification is available and permitting multiple data instances in a tree at the same time. Thus, classification of one instance of data can depend on other data, giving it context and relation. While we concluded that a readily-available solution for complex event processing, Esper, is better suited for the specific task at hand, our new form of decision tree might have an advantage in other scenarios.

Evaluating our signature-based correlator, we found it has various advantages over existing approaches. Defining correlated events is easier than in other approaches, false positive rates are low, and cooperation with other correlators or intrusion detection systems is simple. On the downside, our correlator depends on adjoined intrusion detection system's capability of detecting atomic events, and shares the disadvantage of all signature-based approaches, i.e., it can only detect what has been explicitly defined in the ruleset before. Deployment and operation of our signature-based correlator is easy and has no productivity-reducing factors through false positives. It reliably detects what has been expressed as a signature before and does not depend on a learning phase with potentially huge discrepancies between the learning data and real-life data.

All in all, our signature-based correlator allows for easy deployment and extension of its detection capabilities. Especially, it reduces the effort for detecting correlated events to writing signatures in Event Processing Language (EPL) whose syntax is almost identical to SQL. With its advantages in practical deployment and operation, we consider it a considerable improvement over existing correlation systems. These suffer from high false positive and low true positive rates and require training data sets or high expert knowledge. They are hardly deployed in current production networks, arguably due to the aforementioned practicability issues which we intended to resolve with our correlation system.

Bibliography

[1] ANSII Research Project. ANSII: ANomaly and embedded Security in Industrial Information systems ("Anomalieerkennung und eingebettete Sicherheit in industriellen Informationssystemen"). Research project primarily funded by the German Federal Ministry of Education and Research (Bundesministerium für Bildung und Forschung) and carried out by Technische Universität München and the Fraunhofer Institute for Security in Information Technology (Fraunhofer SIT), among others. `http://www.ansii-projekt.de/`.

[2] Matt Bishop. A taxonomy of unix system and network vulnerabilities, 1995. Technical Report CSE-95-10.

[3] Frédéric Cuppens and Rodolphe Ortalo. Lambda: A language to model a database for detection of attacks. In Hervé Debar, Ludovic Mé, and S.Felix Wu, editors, *Recent Advances in Intrusion Detection*, volume 1907 of *Lecture Notes in Computer Science*, pages 197–216. Springer Berlin Heidelberg, 2000.

[4] Hervé Debar and Andreas Wespi. Aggregation and correlation of intrusion-detection alerts. In *Recent Advances in Intrusion Detection*, pages 85–103. Springer, 2001.

[5] Steven T Eckmann, Giovanni Vigna, and Richard A Kemmerer. Statl: An attack language for state-based intrusion detection. *Journal of Computer Security*, 10(1):71–103, 2002.

[6] Esper Project. Esper documentation: Processing model. `http://esper.codehaus.org/esper-4.11.0/doc/reference/en-US/html/processingmodel.html`.

[7] Esper Project. Esper Website. `http://esper.codehaus.org/`.

[8] Esper Project. FAQ. `http://esper.codehaus.org/tutorials/faq_esper/faq.html`.

[9] European Union Agency for Network and Information Security (ENISA). Existing taxonomies. `http://www.enisa.`

europa.eu/activities/cert/support/incident-management/browsable/incident-handling-process/incident-taxonomy/existing-taxonomies.

[10] Simon Hansman and Ray Hunt. A taxonomy of network and computer attacks. *Computers & Security*, 24(1):31–43, 2005.

[11] K. Houck, S. Calo, and A. Finkel. Towards a practical alarm correlation system. In *Proceedings of the Fourth International Symposium on Integrated Network Management*, pages 226–237, London, UK, UK, 1995. Chapman & Hall, Ltd.

[12] John Douglas Howard. *An analysis of security incidents on the Internet 1989-1995*. PhD thesis, Carnegie Mellon University, 1998.

[13] G. Jakobson and M. Weissman. Alarm correlation. *IEEE Network: The Magazine of Global Internetworking*, 7(6):52–59, November 1993.

[14] McAfee. McAfee Advanced Correlation Engine website. http://www.mcafee.com/au/products/advanced-correlation-engine.aspx.

[15] Michael Meier, Niels Bischof, and Thomas Holz. Shedel – a simple hierarchical event description language for specifying attack signatures. In *Security in the Information Society*, pages 559–571. Springer, 2002.

[16] Cédric Michel and Ludovic Mé. Adele: An attack description language for knowledge-based intrusion detection. In *In Proceedings of the 16th International Conference on Information Security (IFIP/SEC 2001*, pages 353–368. Kluwer, 2001.

[17] Benjamin Morin and Hervé Debar. Correlation of intrusion symptoms: an application of chronicles. In *Recent Advances in Intrusion Detection*, pages 94–112. Springer, 2003.

[18] Andreas Müller. Event correlation engine. Master's thesis, Department of Information Technology and Electrical Engineering, Eidgenössische Technische Hochschule Zürich, 2009.

[19] European CSIRT Network. The european csirt network. http://www.ecsirt.net/.

[20] Peng Ning, Yun Cui, and Douglas S Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 245–254. ACM, 2002.

[21] Prelude team. Prelude-IDS website. https://www.prelude-ids.org/.

[22] Xinzhou Qin. *A probabilistic-based framework for infosec alert correlation*. PhD thesis, Georgia Institute of Technology, 2005.

[23] Hanli Ren, Natalia Stakhanova, and AliA. Ghorbani. An online adaptive approach to alert correlation. In Christian Kreibich and Marko Jahnke, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment*, volume 6201 of *Lecture Notes in Computer Science*, pages 153–172. Springer Berlin Heidelberg, 2010.

[24] Reza Sadoddin and Ali Ghorbani. Alert correlation survey: framework and techniques. In *Proceedings of the 2006 International Conference on Privacy, Security and Trust: Bridge the Gap Between PST Technologies and Business Services*, PST '06, pages 37:1–37:10, New York, NY, USA, 2006. ACM.

[25] Reuben Smith, Nathalie Japkowicz, Maxwell Dondo, and Peter Mason. Using unsupervised learning for network alert correlation. In Sabine Bergler, editor, *Advances in Artificial Intelligence*, volume 5032 of *Lecture Notes in Computer Science*, pages 308–319. Springer Berlin Heidelberg, 2008.

[26] Zabbix 2.4 Documentation. Items. `https://www.zabbix.com/documentation/2.4/manual/config/items`.

[27] Zabbix 2.4 Documentation. Triggers. `https://www.zabbix.com/documentation/2.4/manual/config/triggers`.

[28] Zabbix Official Website. Zabbix: An Enterprise-Class Open Source Distributed Monitoring Solution. `http://www.zabbix.com/`.

[29] Tianning Zang, Xiaochun Yun, and Yongzheng Zhang. A survey of alert fusion techniques for security incident. In *Web-Age Information Management, 2008. WAIM '08. The Ninth International Conference on*, pages 475–481, 2008.

[30] Jingmin Zhou, Mark Heckman, Brennen Reynolds, Adam Carlson, and Matt Bishop. Modeling network intrusion detection alerts for correlation. *ACM Transactions on Information and System Security (TISSEC)*, 10(1):4, 2007.