Collaborative Incident Handling Based on the Blackboard-Pattern

Nadine Herold
herold@net.in.tum.de

Holger Kinkelin
kinkelin@net.in.tum.de

Georg Carle
carle@net.in.tum.de

Technische Universität München
Boltzmannstr. 3, 85748 Garching bei München, Germany

ABSTRACT

Defending computer networks from ongoing security incidents is a key requirement to ensure service continuity. Handling incidents in real-time is a complex process consisting of the three single steps: intrusion detection, alert processing and intrusion response. For useful and automated incident handling a comprehensive view on the process and tightly interleaved single steps are required. Existing solutions for incident handling merely focus on a single step leaving the other steps completely aside. Incompatible and encapsulated partial solutions are the consequence.

This paper proposes an incident handling systems (IHS) based on a novel execution model that allows interleaving and collaborative interaction between the incident handling steps realized using the Blackboard Pattern. Our holistic information model lays the foundation for a conflict-free collaboration. The incident handling steps are further segmented into exchangeable functional blocks distributed across the network. To show the applicability of our approach, typical use cases for incident handling systems are identified and tested with our implementation.

Keywords

Incident Handling, Collaborative Knowledge Sharing, Blackboard Pattern

1. INTRODUCTION

Computer networks are permanently affected by attack attempts. When hardening and prevention mechanisms fail, active *incident handling* is required to ensure service continuity during an on-going attack. Incident handling is a complex process consisting of comprehensive steps: The underlying infrastructure to be protected (target system) is monitored continuously to observe its healthiness. Intrusion detection systems (IDS) search for signs of intrusions and raise alerts if a security incident is detected. Those alerts are processed to find the root cause of an intrusion. This root cause is mitigated by intrusion response systems (IRS) with suitable responses and identified security flaws are closed.

As today's computer networks are complex and the number and variety of security incidents is constantly rising, incident handling has to be automated. Automation allows quicker intervention than manual interaction, stops attacks and prevents the target system from further damage.

Considerable effort has been made to improve the single steps of incident handling. However, a comprehensive solution interleaving single steps and gaining a broad view is missing. Hence, information required in all steps has to be recollected instead of being shared in advance. Further, intermediate results helpful in subsequent steps are lost as only final results are passed. To enable automated, useful incident handling a collaborative execution model is required.

The **main contribution** of this paper is a collaborative and comprehensive *incident handling system (IHS)*. The collaboration between the individual steps of incident handling is realized by a dedicated component for information sharing, called blackboard. The central elements our IHS are a broad *information model* for the blackboard and a collaborative *execution model* to access information on the blackboard. To achieve our goals, we segment the single steps of incident handling into tasks representing functional units of each step. Those tasks are aligned with our information model building the foundation of collaboration. The tasks are designed to be free of interferences and conflicts to avoid inconsistencies in shared information. Additionally, we identify typical but challenging use cases for an IHS. Our implementation is evaluated against those use cases to show the applicability and usefulness of our approach.

The reminder of this paper is organized as follows. In Sect. 2 we identify drawbacks of existing solutions preventing a comprehensive IHS. In Sect. 3 we introduce our IHS and introduce the utilized design pattern. Our information model as the foundation of our system is introduced in Sect. 4. The tasks utilizing this information model are examined as functional modules in Sect. 5. We use our implementation examined in Sect. 6 to evaluate the applicability our approach in different use cases in Sect. 7.

2. RELATED WORK

Our work is influenced by different fields of research related to incident handling in particular: 1) systems providing a partial solution by focusing on a single step of incident handling, 2) execution models of existing solutions and, 3) information models for the security domain.

1) Partial Solution: A lot of effort has been made to

WISCS'16, October 24 2016, Vienna, Austria

ISBN 978-1-4503-4565-1/16/10...$15.00

DOI: http://dx.doi.org/10.1145/2994539.2994545

investigate single steps of incident handling. Even if no comprehensive solution is provided a closer look into the single steps, intrusion detection, alert processing and intrusion response, allows determining tasks of a step.

In [2, 3] an IDS based on learning strategies is introduced. This work focuses on the pure detection process and collaboration between different detection techniques. In [12] an IDS called *SPIDeR* is introduced. SPIDeR is based on multiple agents using self-organizing maps, a neural network technique used in machine learning. The IDS modules perform tasks independently and store their results on an information sharing component. In [7], a generic *Intrusion Detection and Diagnosis System (ID^{2}S)* is proposed. The system supports alert correlation for detecting and analyzing intrusions in large-scale critical infrastructures.

The *CIDS* framework [6] and work done in [23], describe tasks in the alert processing step. Tasks are executed in sequence to minimize the amount of alerts raised by IDS. The *ONTIDS* framework [18] proposes an alert correlation framework combining different information sources. *OutMet* [20] combines alert correlation and prioritization and represents correlated alerts as directed acyclic graphs.

[13, 29, 27, 9] structure and examine the intrusion response process in detail. They determine suitable responses and provide methods for assessing and selecting responses.

These systems cannot be distributed easily across the network as tasks are not decoupled. The functionality and communication between tasks is tightly coupled to the proposed system. Hence, single tasks cannot be used as standalone components. Information sharing and collaboration of those systems are limited and no possibilities of integrating missing steps of incident handling are proposed.

2) Execution Models: A suitable execution model has to provide the ability to combine all tasks of incident handling. This includes information sharing and collaboration.

Most execution models are pipelined [6, 16, 18, 20, 13, 29, 27, 9], i.e., the sequence of all tasks for detection, processing and response is fixed. Every task is done independently and the result of the previous task is input of a subsequent task. This is problematic as information is discarded between tasks. Hence, subsequent tasks have less information available. Additionally, the order of tasks is not deterministic, as alerts or aggregated alerts can be prioritized. Another severe issue is processing a virtually endless stream of alerts in a pipelined process expecting a finite input.

To cope with endless streams of alerts, complex event processing (CEP) is used [7]. Systems based on CEP are capable of processing streams of alerts, and offer a continuous way of alert processing. The drawback of CEP is the need of predefined alert windows defining how long alerts last in the system. As it is a priori unknown how long alerts are relevant, determining the window size is challenging.

More collaborative approaches use agent-based systems [17, 19]. The single steps are done by autonomous agents but a central master component controlling, managing, and providing information to the agents is required.

In [3, 2, 12] a dedicated component for information sharing is introduced. All modules in the systems work independently without being managed by a central master.

Pipelined processes and CEP-based systems suffer from limited possibilities of information sharing between steps in incident handling. Agent-based systems heavily rely on a master component controlling the agents and distribute information. Existing solutions providing a component for information sharing are immature and do not cover the whole incident handling process.

3) Information Models: A suitable information model for an IHS has to cover all aspects of incident handling. This comprises information about the underlying target system, alerts and responses.

In [24] an ontology for attacks and intrusion detection is given. Information about the target system and responses are not modeled. In [14] the first steps for an ontology for the cyber security domain are presented. Similarly to [24], the target system and responses are not modeled. In [22] an ontology for IDSes is presented. The authors provide limited information about the target system, and the concepts of attacks and consequences are introduced. But the authors do not model alerts and responses.

Available information models cover only a dedicated aspect of incident handling, but provide a basis for a subsumed and comprehensive information model.

3. SYSTEM DESIGN

We identified the *Blackboard Pattern* as a suitable design pattern for our IHS as it provides an information sharing component and autonomous modules collaboratively working on shared information. As this pattern lays the foundation of our system design, we give a short introduction in Sect. 3.1. The main components of our IHS are examined in Sect. 3.2. Information distribution and control structures of our IHS are presented in Sect. 3.3.

3.1 Blackboard Pattern

The *Blackboard Pattern* [1, 8] is based on the idea to introduce a shared data structure acting as a global memory. This data structure is called *blackboard*. Modules, called *domain knowledge sources* with different application and domain-specific knowledge work on the blackboard independently. The goal is to divide a complicated problem into more feasible subproblems, which can be solved by those modules in a collaborative manner. Modules store intermediate and partial solutions on the blackboard, or combine and delete information from it. The final result is produced by the collaboration of those modules. Communication between modules is exclusively done on the blackboard.

The processing of information is data-driven. That means the control flow depends on the blackboard's current state which is monitored by a *controller*. When a change on the blackboard occurs, the controller can activate additional modules for further execution. Task decomposition and dependency analysis are used to improve the controller [26].

The controller follows an *execution plan* to determine modules. The execution plan can be provided manually or automatically by so-called *control knowledge sources* [10]. Control knowledge sources can adapt the control plan during execution. Using the Reflection Pattern, the separation between controller with its control structures and modules with their data can be improved [21].

To ensure that only authorized modules access the blackboard, basic security mechanisms are introduced. The verification of modules is done by the controller [15].

3.2 Incident Handling System Overview

In this section, we introduce the four main components depicted in Fig. 1 needed to handle incidents automatically,

namely 1) the newly introduced blackboard as information sharing component to enable collaboration between following established IHS subsystems: 2) monitoring and intrusion detection (dotted boxes), 3) alert processing (solid boxes), and 4) intrusion response (dashed boxes).

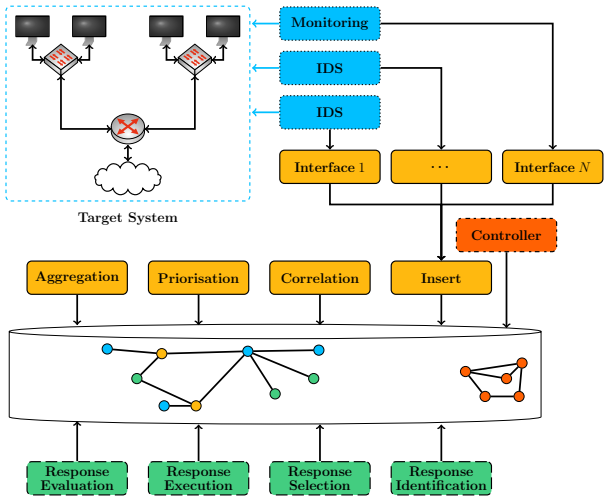


Figure 1: System Overview

1) Blackboard: The *blackboard* of the proposed system is the central information sharing component storing all required information. All modules can access the blackboard either reading or writing information on the blackboard. The blackboard can be realized as a distributed database system accessible from different points across the network.

The blackboard has to provide an information model to enable information sharing. We propose a graph-based information model where all information elements are represented as nodes. Connections between different elements are modeled as relations. Both, nodes as well as relations, may have attributes further describing the entity, see Sect. 4.

A further important component of a blackboard is the *controller*. Regarding its control plan, the controller can intervene the control flow. Modules can be activated or deactivated to handle changes in the system state, see section 3.3.

2) Monitoring and Intrusion Detection: The target system is continuously monitored by host and network monitoring systems like Zabbix[1] or Nagios[2]. Those systems collect so called *infrastructure information* describing the current state of the target system.

Additionally, IDS like OSSEC[3] or Snort[4] monitor the target system. In case an incident is detected they produce an alert message. Their capabilities and detection methods vary heavily and provide different insights in the target system. As it is common to use multiple IDS in parallel [12, 2, 28, 11] the different views of the IDS are consolidated.

The infrastructure and alert information is inserted on the blackboard using different interfaces to support multiple systems, see Sect. 5.1.

2) Alert Processing: Collected alerts are handled by alert processing. First, alerts are associated to infrastructure information on the blackboard to locate and identify the incident in the target system. An important goal of alert processing is to reduce the amount of alerts done by multiple aggregation methods [4]. Alert correlation allows identifying relationships between alerts like attack paths or root causes. Prioritizing alerts enables the system to identify urgent incidents to be handled immediately.

For the single tasks in the alert processing steps different tools and methods exist that can be applied independently and simultaneously, see Sect. 5.2.

3) Intrusion Response: Intrusion response utilizes the infrastructure data collected by monitoring systems and the alerts. First, suitable responses available on the target system are determined as candidate responses. From the candidate responses an optimal set is selected to counteract the intrusion. The selected responses are executed in the target system and evaluated by that time. The response evaluation is needed as response assessment to determine the optimal set of responses, see Sect. 5.3.

Modules can be distributed across the network as long as they are connected to the blackboard. Each module can be available in different variants, e.g. multiple correlators can be used, as long as they follow the information model.

3.3 Controller and Control Plan

The controller in the proposed design notifies appropriate modules regarding its control plan in case information on the blackboard is added or updated by other modules. A simplistic controller activates all modules and passes detected changes to all modules no matter what task they implement.

A more sophisticated controller can use *publish-subscribe* mechanisms to reduce the amount notifications. A change on the blackboard is noticed by the controller that publishes the change only to subscribed modules. As the controller needs to select the subscribers *symmetric publish/subscribe* can be used. Thus, the controller can specify which modules get a notification of the blackboard's change. This avoids deactivation of modules that must not get informed. Hence, no inconsistencies on the blackboard occur as aborted modules wrote preliminary results on the blackboard.

Putting the decision logic of responsibilities for particular information into the modules instead of the controller has certain disadvantages. The decision logic has to be distributed over multiple modules leading to complicated maintenance. However, a single and isolated module with a locally limited view cannot decide about its own responsibilities. Furthermore, overlapping executions of modules within the same domain might result in inconsistencies.

One challenge using a central control mechanism is to select suitable modules. The control plan defines under which conditions a module has to be used. Therefore, the controller determines and interprets the situation in a first step. Second, the controller maintains knowledge about the capabilities and properties of modules to select the correct ones.

For example, if only a few alerts arrive on the blackboard the work load will be low. The controller can select modules with extensive analyzing capabilities. If the alert rate increases, a faster decision is needed. In this example the controller has to be able to determine the rate of incoming alerts and needs methods to assess the work load for the system. With respect to the assessed expected workload appropriate modules have to be selected.

[1] http://www.zabbix.com/
[2] https://www.nagios.org/
[3] http://ossec.github.io
[4] https://www.snort.org

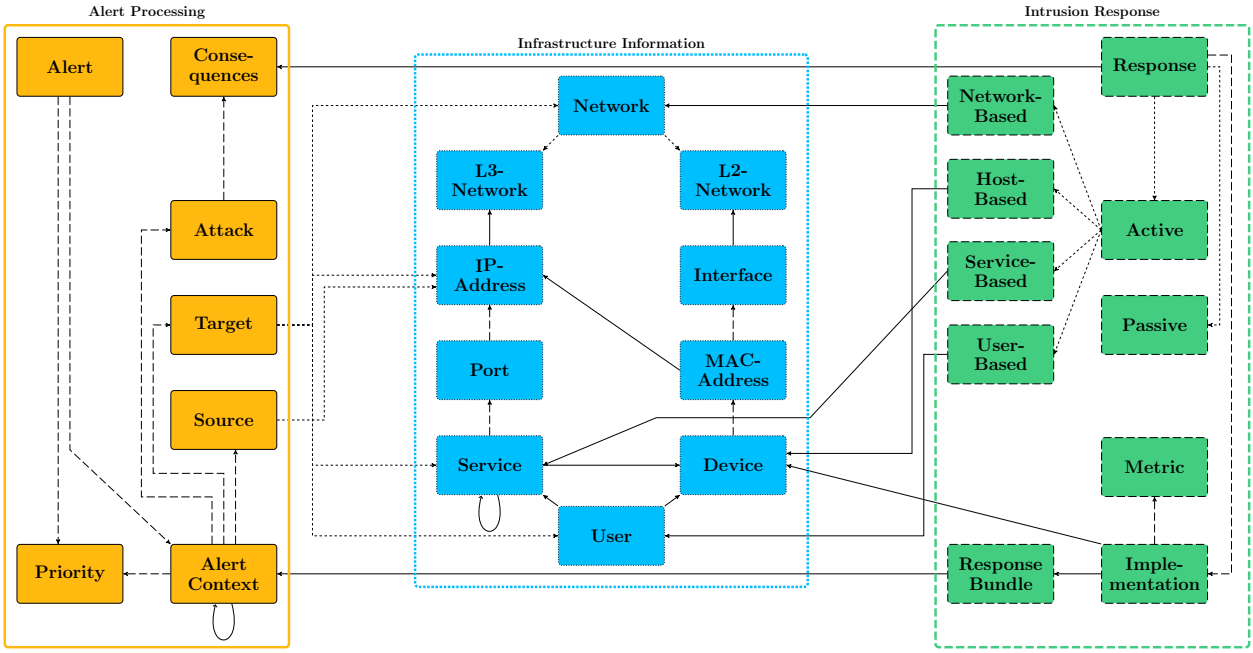


Figure 2: Information Model of the Blackboard

4. INFORMATION MODEL OF THE BLACKBOARD

For our information model we identified the following requirements that have to be fulfilled:

R1 - Separation: All modules have to update or add information independently without creating conflicts. Hence, the information stored on the blackboard needs to be segmented in disjoint parts.

R2 - Completeness: The information model has to provide the required infrastructure information for all tasks of incident handling.

R3 - Compatibility: Our information model has to be compliant to standards related to incident handling, e.g. the IDMEF [5] alert message format.

R4 - Traceability: The information model has to provide the ability to trace all tasks of incident handling. Hence, alerts and intermediate results like selected, and executed responses have to be stored in the blackboard.

The information model, depicted in Fig. 2, is graph-based. An information element is a high-level concept with a defined meaning and is depicted as *node*. A relation between information elements is depicted as *edge*. Nodes and edges have properties further describing the element. Dotted edges are an *Is-A* (generalization) relation, while solid edges are associations and dashed edges are *has*-relations.

To reflect the three types of modules, our information model is split into three groups of nodes: 1) infrastructure nodes, 2) alert processing related nodes, and 3) response related nodes. To avoid write conflicts (cf. R1), modules only write nodes related to their task. But, read access is allowed to all nodes to enable collaboration across tasks (cf. R2).

Infrastructure nodes (dotted box in Fig. 2) reflect infrastructure information collected by monitoring. A typical network consists of devices (`Device` node), e.g. hosts or routers, and subnetworks (`Network` node). Networks can be switched (`Layer2 Network` node) or routed (`Layer3 Network` node). A device can be connected to several layer2 networks via multiple physical interfaces (`Interface` node) having a MAC address (`MAC` node). An IP address (`IP` node) is assigned to a MAC address providing access to a layer3 network. Services (`Service` nodes) are running on a device and are associated with a port (`Port` node) of a specific IP. Services may depend on another service, indicated by the self-reference. Users (`User` node) are logged on devices and use services.

Alert processing related information is marked as solid box in Fig. 2. Each alert (`Alert` node) is associated with an alert context (`Alert Context` node) that bundles semantically related alerts. Each `Alert Context` node provides a `Solved` attribute to indicate whether the incident was handled successfully (`TRUE`) or not (`FALSE`). An alert or alert context can be prioritized using a `Priority` node. Prioritization can alternatively be realized as an attribute to improve performance by reducing the depth of the graph.

An alert context is associated with a `Source`, `Target`, and `Attack` node (cf. R3). The source describes the supposed intruder, normally determined by its IP. The target describes the victim in the target system and can be further described by creating references to infrastructure nodes such as `IP`, `User`, `Device`, `Service`, or `Network` nodes. An `Attack` node classifies the type of an intrusion. Attacks are associated to consequences (`Consequence` node) happening in case of success. `Attack` and `Consequence` nodes describe which incidents our system can handle and are provided by experts.

The last category is response-related information marked as dashed box in Fig. 2. A response (`Response`) can mitigate certain consequences resulting from an attack. Responses can either be `Active` or `Passive`. Passive responses (sending alerts or notifications) do not directly mitigate or decrease the damage to the target system, while active responses (blocking traffic, shutting down hosts) do. As we focus on active responses, we divide them with respect to the target they influence during execution. We identify

among others `User-Based`, `Network-Based`, `Service-Based`, and `Host-Based` responses. This list is not complete but exceeds the scope of the paper to identify all possible fine-grained targets. Information about available responses has to be provided to our system by an expert, knowing the capabilities of the target system.

For each response different implementations (`Implementation` node) can be available deployed on a device. Those implementations have different metrics (`Metric` node) describing the implementation. The metrics are used for response selection. A `Bundle` node is associated with an alert context and indicates responses associated with the incident. This association means that implementations within this bundle can be potentially applied to the incident. A bundle has two attributes used for communication between different modules in the intrusion response process. The `Executing` attribute indicates if the execution of this bundle is ongoing. The `Active` attribute indicates if this bundle is still in use. The relation between an implementation and a bundle carries additional information. A `Selected` attribute indicates whether or not a response implementation was selected to be executed. A `Executed` attribute indicates whether or not a response is yet executed. This design fulfills R4.

5. INCIDENT HANDLING MODULES

In this section we examine the tasks of incident handling in detail and present the modules implementing the tasks. The modules of our system reflect the knowledge sources of the Blackboard Pattern. The modules are independent from each other and provide functional disjoint tasks of incident handling. This segmentation has several advantages: Modules can be distributed across the network as no direct communication between modules is required. Modules can be exchanged easily as they are self-contained tasks. Collaboration between modules can be managed easily using a controller. In the following, we describe modules in detail.

5.1 Monitoring and Intrusion Detection

These modules collect information about the target system (e.g., IP or MAC addresses of devices) and store the current state of the target system on the blackboard. If new information is found, a new instance of the infrastructure node is added on the blackboard. Edges between the new node and existing nodes are added if needed. If a change is detected, the corresponding existing node or edge is updated. A continuous monitoring of the target system ensures that infrastructure information is up-to-date.

Multiple monitoring tools can be used in parallel distributed across the target system. For example dedicated tools processing log files, monitoring packet flows or scanning subnetwork can be deployed. So far, we incorporated a limited set of infrastructure information. However, the abilities of the IHS can be extended and adapted easily to other needs.

5.2 Alert Processing

Following alert processing tasks are adapted from [6, 23]:

Initial Insert: Once an IDS raises an alert, this information is inserted on the blackboard. The insertion module provides one or more interfaces for available IDSes receiving their alerts. The insertion proceeds as follows. A new `Alert` node is inserted first. The source, target, and attack classification are extracted from the alert. A suitable `Alert Context` node is searched. This means source, target and attack classification match. If exists, a new edge between the alert context and the alert is inserted. Otherwise, a new `Alert Context` node is created and linked to corresponding `Target`, `Source` and `Attack` nodes.

Within this step an implicit normalization is done. The attack classification itself is not stored in the `Alert Context` node but linked to available `Attack` nodes. Additionally, an implicit first aggregation is done. Alerts with the same source, target and attack classification are combined into an alert context [4]. This aggregation is lossless as alerts remain on the blackboard, but additional information is added.

Aggregation: Typical systems aggregate alerts in a lossy process. The raw alert information that is the input for aggregation is discarded, but aggregates are passed. In our IHS, aggregation is creating edges between existing `Alert Context` nodes with similar features. That may be the same source, target, attack classification, or a combination of those features. Aggregation with the same target and attack classification are especially helpful for distributed attacks like distributed denial of service attacks. As distributed attacks have the same target and attack classification, many alerts are combined into a single `Alert Context` node. Because of the lossless aggregation, additional aggregation features are possible, e.g., alerts from the same IDS.

The aggregation module is triggered after a new `Alert Context` node is inserted. The module checks, if the new node can be aggregated with existing `Alert Context` nodes. If so, the new node either belongs to an existing aggregate or a new `Alert Context` node as aggregate is needed. An edge between the newly inserted `Alert Context` node and the `Alert Context` node representing the aggregate is inserted.

Correlation: This module combines multiple different `Alert Context` nodes. In contrast to aggregation, correlation does not focus on feature similarity of alerts or alert contexts, but tries to find more high-level relations. A typical example is an attack path: a host is first the target of an attack and then becomes a source of a subsequent attack.

This module is triggered after a new `Alert` or `Alert Context` node is inserted. The processing procedure is analogous to aggregation, but using correlation rules instead.

Prioritization: This module calculates or updates the priority of an `Alert` or `Alert Context` node. If a new priority is added, this new priority value has to be propagated through connected `Alert Context` nodes.

Prioritization can be separated into two modules. One module calculates the priority value of a single alert, for example, considering available infrastructure information like the importance of effected hosts. This module is triggered after a new `Alert` node is inserted. Another module propagates for newly inserted or updated priorities. For propagation, different calculations methods can be applied, e.g., using the weighted average of all incoming priority values. The priority propagation module is triggered after updating or adding priority information.

5.3 Intrusion Response

To implement automated intrusion response, specialized functionalities from a completely different domain are needed. However, intrusion response needs the same infrastructure information as required by alert processing. Additionally, intermediate solutions from alert processing are helpful for intrusion response.

Response Identification: This module examines a set

of possible responses mitigating a specific and open (meaning unsolved) alert context. This is done by combining the available information as follows. The module follows the edge from the `Alert Context` node to the `Target` node and further to a concrete network entity, e.g., a `User`, `Network`, `Service`, or `IP` node. From this entity edges to `Response` nodes exists. As the responses are split up into the targets they influence, only applicable responses are reached, e.g., a `User` node is only related to `User-Based` response nodes.

Next, this set of candidate responses is further limited to useful responses. The module follows the edge from the `Alert Context` to the `Attack` node and further to the `Consequence` nodes. Finally, we determine responses that mitigate a certain consequence by following all edges between the identified `Consequence` and `Response` nodes. The result is a second set of candidate `Response` nodes.

The intersection of both calculated sets of candidate responses is calculated. The result is a set of candidate responses helpful against the consequences of an attack and effect the target of the incident. Afterwards, this component follows the edge between a `Response` and `Implementation` nodes that are available on a device associated with the target. For example, a user is logged in on a certain device means that only response implementations available on this host remain in the candidate set. After determining candidate `Response` nodes a new `Bundle` node is added on the blackboard. The `Bundle` node is connected to the corresponding `Alert Context` node and the previously identified `Implementation` nodes.

To trigger response identification multiple options are available. Different infrastructure information can be utilized as trigger, e.g., a highly valuable element in the target system is affected by an attack. Additionally, alert processing information can be utilized, e.g., an alert context exceeds a certain priority or number of connected alerts.

Response Selection: This module selects the most suitable subset of the candidate responses. It is triggered after insertion or updates of a `Bundle` node where the `Active` attribute is `True`. This module can be implemented using optimization techniques to identify the optimal response set or heuristics in case of less relevant threats. For the selection process, metrics associated with the implementations of the candidate responses are utilized. This module updates the edges between the `Bundle` and the `Implementation` node by setting the `Selected` attribute to `True`. The `Active` attribute of the `Bundle` node is set to `False`.

In case the selected responses were not efficient against a certain attack, the existing bundle can be reused as candidate responses are already selected. The response selection only takes responses into account not yet tried (the `Selected` attribute is `False`). Our approach has the advantage that all intermediate steps are still documented on the blackboard, which allows accountability of the system.

Response Execution: This module examines suitable actors able to execute the responses, generates an execution plan and starts cooperative response execution. It is triggered by updates on the `Selected` property of the relation between response implementation and bundle. All selected not yet executed responses and the correct sequence of execution are determined. The resulting execution plan is than executed on the target system. Questions like scheduling and coordination of responses are done by this module. As the execution starts the `Executing` attribute in the `Bundle` node is set to `True`. If the execution is finished, the `Executing` attribute is set back to `False`. Additionally, the `Executed` attribute of the edges between `Bundle` and `Implementation` node are set to `True`.

Response Evaluation: This module measures response implementations during their execution and updates related `Metric` nodes, e.g. effectiveness or duration. Those metrics are stored on the blackboard in form of additional nodes related to `Implementation` nodes. Those metrics are used for response selection. This module examines if a response bundle was successful to mitigate an intrusion. If so, the `Alert Context` node is closed by setting the `Solved` attribute to `True`. Otherwise, the `Active` attribute in the `Bundle` node is set to `True` to trigger a new response selection. A response bundle is interpreted as successful, if no update or insert on the `Alert Context` node is done after a defined time span. This time span is the duration a response needs to be effective. The response evaluation is started after the `Executing` attribute in the `Bundle` node is set to `True` and stopped after this attribute is set to `False`.

5.4 Garbage Collector

All modules described so far only insert or update nodes. Hence, the underlying blackboard will be soiled by outdated or no longer required information over time. Therefore, a single dedicated module for garbage collection is needed. The *garbage collector* uses a fine-grained ruleset to determine which information is still required on the blackboard and for how long. The ruleset depends on the use case as it determines how long a view into the past is possible. As the garbage collector is the only module allowed to delete information, no conflicts with other modules occur.

6. IMPLEMENTATION

As a proof of concept and basis for our evaluation, we implement a prototype in Python 3, available on GitHub[5]. Our implementation requires an underlying database system as blackboard to store information persistent. We support two different types of databases. As the structure of the blackboard is graph-based, a graph database is naturally highly suited. We chose OrientDB[6] as it provides hooks and triggers. As an alternative, we implement our blackboard based on PostgreSQL[7], a widely used relational database. To interface with OrientDB pyorient[8] is used. To interface with PostgreSQL psycopg2[9] is used.

Our information model is represented as Python classes. Each information element (node) and relation (edge) is represented as a class. Those classes are translated automatically to the database scheme for both database systems.

After successfully deploying the schema, the modules are started as separate processes. Our implementation includes for both databases an aggregator, a correlator, and two prioritization modules. As this is only a proof-of-concept implementation, the functionality of the modules is limited. The aggregator fuses alerts to an `Alert Context` node based on equal sources, targets or attack classifications. The correlator supports checking for attack paths by searching for

[5]https://github.com/Egomania/BlackboardIDRS
[6]http://orientdb.com/orientdb/
[7]https://www.postgresql.org/
[8]https://github.com/mogui/pyorient
[9]http://initd.org/psycopg/

attacks launched by sources that were previously a target of another attack. The first prioritization module randomly allocates a priority to an `Alert` node. The second propagates priority information through connected `Alert Context` nodes.

To receive notifications about changes of the blackboard, two different controllers – one for each database type – are implemented. Those controllers wait for incoming notifications from the database and notify modules subscribed for the particular information. For this purpose, Python's built-in queues are utilized. All modules are triggered to start their task by receiving notifications from the controller.

To enable change notification from the blackboard, the underlying database has to support a notification or trigger mechanism. For OrientDB a *native Java Hook* is utilized triggering after a new node or edge is added. To reduce the amount of information pushed from the database to the controller, this hook is configurable to decide which nodes and edges are needed. The database sends the change information on updates, inserts and deletes using a `POST` method to push the information to the controller. As OrientDB does not support after-commit hooks directly, this feature has to be implemented within the modules. The modules have to actively query the database to wait for the commit.

In case of PostgreSQL, the built-in `NOTIFY` command is utilized. An *after-commit hook* triggers a function preparing the information to be sent over the notification channel. The information is prepared using built-in functionality to generate `JSON` files at `pg_notify call`. The controller on the other side only needs a connection to the database to listen on the channel the notifications are sent to.

7. EVALUATION

We evaluate our system design in two dimensions. First, a qualitative analysis showing the access behavior of components is given in Sect. 7.1. In Sect 7.2, we demonstrate the applicability of our approach to the problem domain.

7.1 Qualitative Analysis

The access behavior of the modules is most essential for our IHS as data consistency has to be ensured. The blackboard has been designed such that most of the writing behavior is additive and only in some cases updating existing information is necessary. This approach facilitates the access control and reduces the amount of potential states where inconsistencies can appear.

Infrastructure nodes are updated or inserted by corresponding modules responsible for scanning and generating the infrastructure data. In case of static data (e.g. `IP`, `Device` or `Interface` nodes) nodes and edges are updated infrequently. Consequently, the access to those elements is rare. Besides, modules are separated such that only one updates information, while others only read information.

The same applies to `Attack`, `Response` and `Implementation` nodes. The nodes' information is only updated in case of a policy refinement; otherwise their information is only read as input for other the modules.

In case of `Alert` and `Alert Context` nodes we designed an additive behavior for updates or inserts. If a new alert is raised by an IDS, a new `Alert` node is inserted. Aggregation and correlation only add `Alert Context` nodes or relations between `Alert` and `Alert Context` nodes respectively. The prioritization module updates (re-prioritization) or adds initial priority information. As only a single module is allowed to change this information and others are only permitted to read, no inconsistencies can occur.

One crucial action on the blackboard is the deletion of outdated information. This operation is done on a large amount of nodes and relations. In case an alert processing module wants to access this data during deletion, inconsistencies may occur. The decision of deleting information is done by a separate module, the garbage collector. This allows defining fine-grained strategies for removing information. Additionally, this garbage collector can store outdated information on long-term storage.

7.2 Quantitative Analysis

Within this quantitative analysis, the applicability of the proposed approach is shown. First, the test setup is introduced in Sect. 7.2.1. In order to evaluate our approach different datasets representing typical but challenging use cases are identified (see Sect. 7.2.2). Sect. 7.2.3 finally shows the results of the evaluation for different test cases.

7.2.1 Evaluation Methodology

The usefulness of alert processing has already been proven [23] with common datasets. Hence we provide a simulation to evaluate our approach in typical use cases providing a challenge to an IHS. For this purpose, we generated different datasets containing `IDMEF` messages with specific and challenging properties (see Sect. 7.2.2). Needed infrastructure data is provided by preprocessing those datasets.

Those datasets are read by our *simulator module* transferring single alerts one by one into the underlying database. After insertion, the simulator waits until all modules finished their processing. All implemented modules are executing their specific tasks as describes previously. Each dataset was generated in different variants containing 1000 to 5000 alerts using a step size of 500.

All variants of each dataset were evaluated under the following three different test cases.

Test Case 1 – Insert Nodes: For this test case the alerts are pushed on the blackboard. This includes adding an `Alert` was well as an `Alert Context` node if needed. Additionally, relations between the `Alert Context` node and the `IP` and `Attack` nodes are inserted. As first only the insertion behavior of our system should be tested, no further analysis steps were performed.

Test Case 2 – Prioritize Nodes: This test case includes the prioritization module, to investigate the update behavior of our system. Alerts are added as describes above, but beyond alert and context information is prioritized.

Test Case 3 – Combining Nodes: For this test case all implemented modules are activated, to investigate the analysis behavior of our IHS. Alerts are added as describes above and all modules (prioritization, aggregation, and correlation) are working simultaneously on the data.

For the evaluation we used a machine equipped with an Intel Xeon E3-1275 3.4 GHz 4-core CPU, 16 GB of RAM and a SSD. The operating system was Ubuntu 15.10 64-Bit with Python 3.4.3, pyorient 1.5.2 and psycopg2 2.6.1.

7.2.2 Generated Datasets

To describe the properties of our generated datasets, we use the following notation. The number of unique `Alert Context` nodes (# unique) describes the number of `Alert`

`Context` (# context) nodes not having outgoing edges to other `Alert Context` nodes. The number of possibilities to aggregate same sources ($same_s$), targets ($same_t$) and attack classifications ($same_c$) and the number of attack paths ($paths$) are given. For each dataset a database is created storing used attacks and IP-addresses. The generated datasets are available on GitHub[10].

DoS Dataset: The DoS dataset simulates a distributed denial of service attack (DDoS) with a huge number of sources, i.e., attackers to a small number of targets, i.e., victims. The most challenging aspect of this dataset is to build the structure of resulting `Alert Context` nodes pointing to a huge number of sources. This results in many edges between the nodes. We use two different attacks, namely, a port scan as preparation attack and a DoS attack against the target system. So, $same_c = 2$ for all variants of this datasets. Within this dataset, frequently the same 18 targets are attacked ($same_t = 18$). The dataset does not contain an attack path ($paths = 0$). 2018 IP-addresses are stored in the database. The not yet described properties of the dataset are listed in Tab. 1.

Table 1: Properties of the DoS Dataset

Setting	D1	D2	D3	D4	D5	D6	D7	D8	D9
$\#alerts$	1000	1500	2000	2500	3000	3500	4000	4500	5000
$\#context$	1198	1834	2497	3175	3828	4455	5089	5694	6273
$\#unique$	207	351	534	717	895	1046	1196	1328	1435
$\#same_s$	187	331	514	697	875	1026	1176	1308	1415

Flooding Dataset: Within this dataset, a small number of sources execute three different attacks on a small number of targets. The attacks are port scan, DoS and buffer-overflow, so $same_c = 3$. The challenge of this dataset is the frequent use of the same properties, namely source, target and attack classification. This occurs when multiple IDSes detect the same attack at once.

This dataset did not contain any attack path, so $paths = 0$. 35 IP-addresses are stored in the database. Only 3 attackers continuously attack the target system ($same_s = 3$). The target system consists of 32 hosts ($same_t = 32$). The number of unique context nodes is 38 for all dataset variants as all aggregations and correlations are the same for all variants. The properties of the dataset are listed in Tab. 2.

Table 2: Properties of the Flooding Dataset

Setting	D1	D2	D3	D4	D5	D6	D7	D8	D9
$\#alerts$	1000	1500	2000	2500	3000	3500	4000	4500	5000
$\#context$	315	325	326	326	326	326	326	326	38

Attack Path Dataset: This dataset simulates an intrusion spreading across the network. First, a small number of sources attacks a huge number of targets to compromise them. The compromised entities finally launch additional attacks. The challenge of this dataset is identifying a spreading intrusion, like malware infections. To detect this attack huge portions of the database have to be investigated by the correlator.

The dataset's properties are listed in Tab. 3. The launched attacks are port scans as preparation attacks and buffer overflows, so $same_c = 2$. A buffer-overflow will result in the take-over of the target that becomes than the source of a new attack. The initial number of attackers is three, so $paths = 3$. 2003 IP-addresses are stored in the database.

Table 3: Properties of the Attack Path Dataset

Setting	D1	D2	D3	D4	D5	D6	D7	D8	D9
$\#alerts$	1000	1500	2000	2500	3000	3500	4000	4500	5000
$\#context$	1378	2124	2883	3727	4516	5271	5901	5894	5797
$\#unique$	379	626	883	1228	1516	1773	1965	1978	1939
$\#same_s$	260	372	489	634	745	844	985	990	963
$\#same_t$	114	249	389	589	766	924	975	983	971

7.2.3 Evaluation Results

Within the evaluation different aspects are of interest. First, we will have a look on the stability of our measurements by examining the standard deviation of our measurements. Second, we will examine the alert per second rate of our test cases for all datasets. Lastly, we will examine the new nodes per second rate as every dataset differs in terms of new `Alert` and `Alert Context` nodes to be inserted.

Measurement Stability: In Tab. 4 the average standard deviation of all measurements for both database backends given in seconds is shown. The table denotes the dataset[11] as well as the test case ($t1$ to $t3$). Each measurement was done 5 times.

Table 4: Average Standard Deviation

Dataset	d_{t1}	d_{t2}	d_{t3}	ap_{t1}	ap_{t2}	ap_{t3}	f_{t1}	f_{t2}	f_{t3}
PostgreSQL	0.15	0.34	4.11	0.09	0.45	3.82	0.05	0.05	0.09
OrientDB	1.02	1.48	15.79	1.06	1.59	205.53	0.74	0.99	0.84

The table shows that in general PostgreSQL is more stable than OrientDB, as the PostgreSQL measurements show small deviations. With raising computational effort of the test case, the average standard deviation is increasing.

Alerts per Second Rate: The number of alerts that can be processed per second are shown in Tab. 5. Hereby, the minimum (min), maximum (max) and average (avg) values for PostgreSQL (p) and OrientDB (o) are given. The table denotes the dataset [11] as well as the test case ($t1$ to $t3$).

Table 5: Alerts per Second Rate

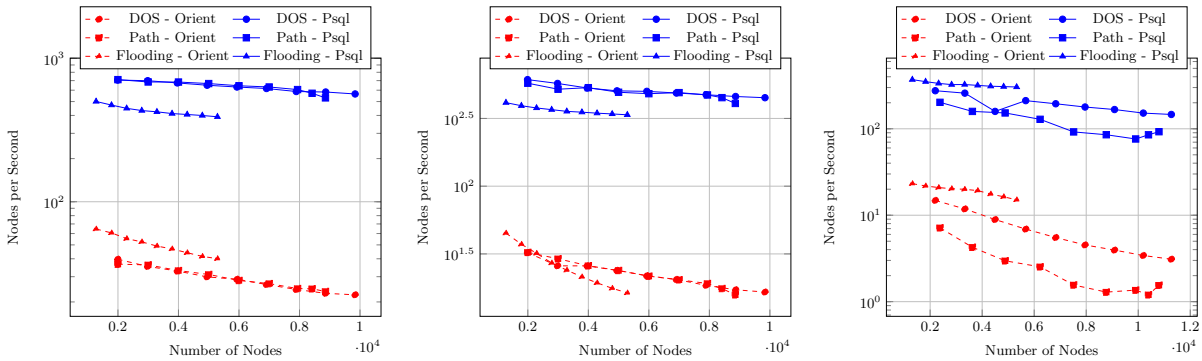
Dataset	p_{min}	p_{max}	p_{avg}	o_{min}	o_{max}	o_{avg}
d_{t1}	287.09	354.72	320.75	11.4	19.72	14.73
d_{t2}	228.61	307.27	257.8	8.4	16.24	11.55
d_{t3}	64.97	125.44	86.15	1.37	6.75	3.12
ap_{t1}	299.4	355.76	324.76	12.5	19.35	15.13
ap_{t2}	230.36	287.86	250.71	8.91	16.23	11.62
ap_{t3}	30.80	85.12	49.59	0.51	3.01	1.1
f_{t1}	370.32	396.63	384.58	37.88	50.87	44.77
f_{t2}	318.1	330.31	325.04	15.4	35.29	23.38
f_{t3}	281.78	293.31	287.73	14.13	18.00	16.97

First, it can be observed that the number of alerts that can be processed per second is decreasing with a higher number of activated modules. While insertion is quite fast using PostgreSQL, prioritizing nodes is noticeable. More comprehensive analysis like aggregation and correlation leads to drop down the throughput.

Secondly, it is obvious that the processing rates between OrientDB and PostgreSQL significantly differ. While PostgreSQL is still able to process alerts within an adequate

[10] https://github.com/Egomania/BlackboardIDRS

[11] dos = d, attack path = ap, flooding = f



(a) Test Case $t1$ – Insertion (b) Test Case $t2$ – Prioritization (c) Test Case $t3$ – Combining

Figure 3: Evaluation of different Datasets and Test Cases

time, OrientDB is not suitable for an IHS. We assume, that is because OrientDB is actively queried to wait for the committed results, as OrientDB only provides an after-create hook in contrast to PostgreSQL. Additionally, for OrientDB the notifications are sent via `HTTP Post` while for PostgreSQL inbound channels are used.

Additionally, it is shown that the processing rates differ between the datasets. While the flooding dataset has a good throughput, the path and DoS dataset seam to cause performance issues. That is because of their structure leading to a higher number of `Alert Context` nodes to be insert.

For test case $t1$ the number of alerts processed per second slightly drops for the all dataset. For test case $t2$ and $t3$ the number of alerts processed per second slightly drops for the flooding dataset while the rate drops significantly with a rising number of alerts for the other two datasets.

New Nodes per Second: As the datasets results in a different number of new `Alert Context` nodes that have to be added, we investigate the rate of new nodes added to the database per second. Hereby, new nodes are the sum of `Alert` and `Alert Context` nodes to be added. The results for this part of the evaluation are shown in Fig. 3a. Whereat, the x-axis shows the number of new nodes and the y-axis shows the evaluated rate in log scale.

This representation shows again, that OrientDB cannot combat with PostgreSQL within our use case. Additionally, the drop down of the throughput with a rising number of alerts is clearly visible through all datasets and test cases. This representation shows that the number of new nodes within the flooding dataset is significantly lower than for the DoS and path dataset, as those are structured such that a high number of `Alert Context` nodes are produced.

Comparing Fig. 3a and Fig 3b a drop down of the throughput is noticeable. The prioritization of alerts produces therefore, a visible impact on the system. In both figures the flooding dataset seams to perform worse than the DoS and path dataset. This is because most of the alerts to insert are unique in terms of source, target and attack classification. The result is a low number of additionally added `Alert Context` nodes, but the checks for uniqueness are the same effort as in the other two datasets.

Comparing Fig. 3b and Fig. 3c another throughput drop is noticeable. The additionally added modules produce the main effort within the system. The flooding dataset is now performing better than the other two datasets as the checks for new `Alert Context` nodes take more time in the DoS and path dataset as, for example the number of IPs to check is higher in those two datasets.

8. LIMITATIONS AND FUTURE WORK

The infrastructure information included in the information model is limited so far, but provides an insight into the capabilities of our collaborative IHS. Future work will extend our information model with respect to infrastructure information to increase the system's capabilities.

Furthermore, additional modules need to be implemented to cover all steps of incident handling, e.g., root-cause analysis and a more featured correlator. Modules manipulating the control plan and a more advanced controller supporting symmetric publish subscribe are also part of future work.

Finally, response capabilities need to be extended by providing concrete expert knowledge about responses, consequences and attacks. For this purpose, we plan to integrate various tools from previous work, i.e., our response selection mechanism[12] and *GPLMT* [25] for response execution.

9. CONCLUSION

In this paper we contribute a comprehensive and collaborative intrusion handling system (IHS) that interleaves the single steps of incident handling instead of considering them as isolated systems. As most steps of incident handling rely on the same information, our system presents a novel and broad information model covering required information elements. Interconnecting the information elements from different knowledge areas, allows intertwining information sharing. This information model lays the foundation for the information sharing component of our system.

Additionally, we provide an execution model enabling collaboration between modules fulfilling different tasks of the steps of incident handling. Due to the segmentation of the incident handling process into manageable tasks, we provide modules that are autonomous, distributable and interchangeable. Those modules are designed to be free of interference and conflicts to enhance collaboration between these modules. Furthermore, these modules are closely aligned to our information model to closely interleave the modules.

Lastly, we give an overview of typical but challenging use cases an IHS has to cope with. To evaluate our approach we

[12]https://github.com/Egomania/ResponseSelection

provide an implementation supporting two information sharing components the identified use cases are applied to. Our comparison shows that PostgreSQL is a suitable database system to provide a capable and useful IHS.

Acknowledgment

This work has been supported by the German Federal Ministry of Education and Research (BMBF) under support codes 16KIS0145 (SURF) and 16KIS0538 (DecADe).

10. REFERENCES

[1] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-oriented software architecture, Volume 1: A system of patterns*. Wiley, 1996.

[2] M. Dass, J. Cannady, and W. D. Potter. A blackboard-based learning intrusion detection system: a new approach. In *Developments in Applied Artificial Intelligence*, 2003.

[3] M. Dass, J. Cannady, and W. D. Potter. Lids: A learning intrusion detection system. In *FLAIRS Conference*, 2003.

[4] B. T. Dave and S. Jimit Mahadevia. Application profiling based on attack alert aggregation. *Global Journal of Computer Science and Technology*, 2014.

[5] H. Debar, D. Curry, and B. Feinstein. The Intrusion Detection Message Exchange Format (IDMEF). Technical report, Internet Requests for Comments, 2007.

[6] H. Elshoush. An innovative framework for collaborative intrusion alert correlation. In *Science and Information Conference (SAI)*, 2014.

[7] M. Ficco and L. Romano. A generic intrusion detection and diagnoser system based on complex event processing. In *1st International Conference on Data Compression, Communications and Processing (CCP)*, 2011.

[8] S. Kaisler. *Software Paradigms*. Wiley, 2005.

[9] W. Kanoun, N. Cuppens-Boulahia, F. Cuppens, S. Dubus, and A. Martin. Intelligent response system to mitigate the success likelihood of ongoing attacks. In *6th International Conference on Information Assurance and Security (IAS)*, 2010.

[10] P. Lalanda. Two complementary patterns to build multi-expert systems. In *Pattern Languages of Programs*, 1997.

[11] F. Manganiello, M. Marchetti, and M. Colajanni. Multistep attack detection and alert correlation in intrusion detection systems. In *Information Security and Assurance*, 2011.

[12] P. Miller and A. Inoue. Collaborative intrusion detection system. In *22nd International Conference of the North American Fuzzy Information Processing Society (NAFIPS)*, 2003.

[13] S. Mitropoulos, D. Patsos, and C. Douligeris. On incident handling and response: A state-of-the-art approach. *Computers & Security*, 2006.

[14] L. Obrst, P. Chase, and R. Markeloff. Developing an ontology of the cyber security domain. In *7th International Conference on Semantic Technologies for Intelligence, Defense, and Security (STIDS)*, 2012.

[15] J. L. Ortega-Arjona and E. B. Fernandez. The secure blackboard pattern. In *Proceedings of the 15th Conference on Pattern Languages of Programs*, 2008.

[16] S. Ossenbühl, J. Steinberger, and H. Baier. Towards automated incident handling: How to select an appropriate response against a network-based attack? In *9th International Conference on IT Security Incident Management IT Forensics (IMF)*, 2015.

[17] D. Ragsdale, C. Carver, J. Humphries, and U. Pooch. Adaptation techniques for intrusion detection and intrusion response systems. In *IEEE International Conference on Systems, Man, and Cybernetics*, 2000.

[18] A. Sadighian, J. Fernandez, A. Lemay, and S. Zargar. Ontids: A highly flexible context-aware and ontology-based alert correlation framework. In *Foundations and Practice of Security*, 2014.

[19] D. Schnackenberg, K. Djahandari, and D. Sterne. Infrastructure for intrusion detection and response. In *DARPA Information Survivability Conference and Exposition (DISCEX)*, 2000.

[20] R. Shittu, A. Healing, R. Ghanea-Hercock, R. Bloomfield, and R. Muttukrishnan. Outmet: A new metric for prioritising intrusion alerts using correlation and outlier analysis. In *IEEE 39th Conference on Local Computer Networks (LCN)*, 2014.

[21] O. Silva, A. Garcia, and C. Lucena. The reflective blackboard pattern: Architecting large multi-agent systems. In *Software Engineering for Large-Scale Multi-Agent Systems*, 2003.

[22] J. Undercoffer, A. Joshi, and J. Pinkston. Modeling computer attacks: An ontology for intrusion detection. In *Recent Advances in Intrusion Detection*, 2003.

[23] F. Valeur, G. Vigna, C. Kruegel, and R. Kemmerer. Comprehensive approach to intrusion detection alert correlation. *IEEE Transactions on Dependable and Secure Computing*, 2004.

[24] R. van Heerden, L. Leenen, and B. Irwin. Automated classification of computer network attacks. In *International Conference on Adaptive Science and Technology (ICAST)*, 2013.

[25] M. Wachs, N. Herold, S.-A. Posselt, F. Dold, and G. Carle. Gplmt: A lightweight experimentation and testbed management framework. In *17th International Conference on Passive and Active Measurement (PAM)*, 2016.

[26] S. Yongyong. Decision algorithm for multi-agent intelligent decision support system based on blackboard. *Information Technology Journal*, 2013.

[27] Z. Zhang, P.-H. Ho, and L. He. Measuring ids-estimated attack impacts for rational incident response: A decision theoretic approach. *Computers & Security*, 2009.

[28] B. Zhu and A. A. Ghorbani. Alert correlation for extracting attack strategies. *International Journal of Network Security*, 2006.

[29] S. Zonouz, H. Khurana, W. Sanders, and T. Yardley. Rre: A game-theoretic intrusion response and recovery engine. *IEEE Transactions on Parallel and Distributed Systems*, 2014.