AIDA Framework: Real-Time Correlation and Prediction of Intrusion Detection Alerts

Martin Husák
Institute of Computer Science
Masaryk University
Brno, Czech Republic
husakm@ics.muni.cz

Jaroslav Kašpar
Institute of Computer Science
Masaryk University
Brno, Czech Republic
kaspar@ics.muni.cz

ABSTRACT

In this paper, we present AIDA, an analytical framework for processing intrusion detection alerts with a focus on alert correlation and predictive analytics. The framework contains components that filter, aggregate, and correlate the alerts, and predict future security events using the predictive rules distilled from historical records. The components are based on stream processing and use selected features of data mining (namely sequential rule mining) and complex event processing. The framework was deployed as an analytical component of an alert sharing platform, where alerts from intrusion detection systems, honeypots, and other data sources are exchanged among the community of peers. The deployment is briefly described and evaluated to illustrate the capabilities of the framework in practice. Further, the framework may be deployed locally for experimentations over datasets.

CCS CONCEPTS

• **Security and privacy** → *Intrusion detection systems*; *Network security*; • **Computing methodologies** → *Machine learning*;

KEYWORDS

intrusion detection, information sharing, alert correlation, prediction, data mining

ACM Reference Format:
Martin Husák and Jaroslav Kašpar. 2019. AIDA Framework: Real-Time Correlation and Prediction of Intrusion Detection Alerts. In *Proceedings of the 14th International Conference on Availability, Reliability and Security (ARES 2019) (ARES '19), August 26–29, 2019, Canterbury, United Kingdom.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3339252.3340513

1 INTRODUCTION

Intrusion detection was subject of intensive research and development but is still struggling with limitations, such as limited visibility into the networks and information systems, insufficient knowledge of the attack patterns, and lack of global situational awareness. Many approaches were proposed to overcome such limitations. Collaborative intrusion detection [36] allows for exchanging timely information between the intrusion detection systems, correlation of alerts [6] provides a deeper understanding of the security events, and predictive analytics [16] may provide early warning. However, a certain level of technical maturity needs to be achieved before the prospective approaches are applicable in practice, especially in large and heterogeneous infrastructures.

This work is motivated by the experiences with the development of an alert sharing platform. The platform was created and deployed to share alerts from a variety of intrusion detection systems, honeypots, and other data sources deployed in various organizations. Analysis of the data and their proper utilization arose as an interesting research problem. One of the problems we are facing is the volume of the data shared in the platform; typically, we receive more than one million alerts per day. The recipients of the data use the identifiers (IP addresses, hostnames, etc.) in the alerts for various purposes, such as blacklisting, traffic filtering, etc. However, using all of them is resource intensive, and not all the alerts and identifiers are relevant for a specific data recipient. This is often a reason for the data recipients to turn the data away as they do not have the resource or capacity to process the data and select the most relevant entries. We would like to significantly decrease the volume of the data and focus on entries of particular interest for the receiver in the form of "personalized" blacklisting.

In this work, we introduce AIDA[1], an analytical framework developed for the needs of the alert sharing platform, but also usable in other scenarios. The framework receives all the data (alerts) submitted to the sharing platform and performs statistical analysis, aggregation, and correlation of the alerts. We selected event prediction as a primary use case for the analytical framework as it illustrates various tasks performed during the analysis of intrusion detection alerts. The alerts are aggregated to eliminate duplicate alerts representing the same events, correlated with distilling attack scenarios in the form of alert sequences, and, finally, the alert sequences are matched against the newly arrived alerts to predict upcoming events, that are reported to the sharing platform. Thus, we are able to gain a deeper understanding of the security situation and provide personalized early warning to the organizations at risk.

The contributions of this paper are twofold. First, we introduce a framework for the analysis of intrusion detection alerts shared in an alert sharing platform. The framework uses modern approaches to data processing, such as real-time data analysis and selected methods of data mining and complex event processing. Our framework is deployed as an analytical component of alert sharing platform SABU[2]. Second, we introduce predictive analytics as a use case for the framework. We distill common sequences of alerts to generate predictive rules and use them to predict future security events. The

ARES '19, August 26–29, 2019, Canterbury, United Kingdom

[1] https://github.com/CSIRT-MU/AIDA-Framework
[2] https://sabu.cesnet.cz/en/start

predictive analysis illustrates the power of the framework, namely the Data Mining and Rule Matching components. Inferring alerts from attacker's recent activity and predicting the location of the upcoming security event allows for personalized blacklisting and reduced the volume of data sent to the recipients.

This paper is organized as follows. Section 2 presents the related work on sharing intrusion detection alerts, their correlation, and predictive analysis. Implementation of the framework for the analysis of intrusion detection alerts is provided in Section 3. The results of the executed empirical evaluation are presented and discussed in Section 4. Section 5 concludes the paper and pinpoints a few topics, which pave the way for future work.

2 RELATED WORK

Collaboration and information exchange has become a substantial part of cybersecurity practice, from the cooperation between intrusion detection systems [13] to nation-wide information sharing [32]. Research and development have focused on automating the process so that timely and important pieces of information would be exchanged to provide early warning [30], increase the precision of intrusion detection or simply to announce a threat. From a research perspective, a lot of attention was dedicated to collaborative intrusion detection, which is essentially a low-level information sharing between intrusion detection systems. The theoretical background of this topic was described in details by Fung and Boutaba [13], and surveyed by Meng et al. [25] and Vasilomanolakis et al. [36]. From the practitioner's perspective, many implementations of security information sharing platform exist as surveyed by ENISA [7, 8]. Real-world information exchange platforms focus rather on high-level pieces of information such as security alerts, formal representations of security events reported by intrusion detection systems. An overview of formats and protocols for the exchange of such information was presented by Steinberger et al. [33].

Formalization of security alerts allowed a whole new field of research to emerge, the security alert correlation [5]. The task of alert correlation is to put together corresponding alerts, find relations between alerts, and reconstruct the progress of an attack, but also to recognize the focus of an attacker and analyze the impact of potential security incidents. The detailed tasks and processes of security alert correlation were proposed by Valeur et al. [35], while Cuppens and Miège [3] discussed the problem from the perspective of collaborative intrusion detection. Briefly, the alerts from sensors need to normalized, fused (aggregated), and verified first. Subsequently, the attack session can be reconstructed for the purposes of attack focus recognition, multi-stage attack correlation, and impact analysis [35]. The three later stages of alert correlation overlap with the task of attack prediction. Elshoush et al. [6] surveyed methods of alert correlation in a collaborative environment.

When the security alerts are correlated, and certain relations between them are observed, we can use this knowledge to predict the behavior of future attackers and progress of future attacks [16]. In the past, we have seen attempts to predict the attack progression or the desired goal of an adversary (this is often referenced in literature as *attack projection* and *focus recognition*). Early approaches used predefined models of attack scenarios. Such models could be attack graphs [23, 34], Bayesian networks [28, 31], or Markov models [9, 27], to cite the most relevant contributions. If a series of detected events corresponds to a part of an attack scenario in the model, the remaining parts of the scenario can be predicted. However, due to high demands on creating such models manually and continuously changing threat landscape, researchers started using methods of data mining to (semi-)automatically create the patterns and models to match and project running attacks [9, 19, 22, 23, 34]. Farhadi et al. [9] proposed a real-time approach, Kim and Park [22] proposed using continuous data mining, and Jiang et al. [19] combined data mining with similarity search. Simultaneously, attack prediction methods based on machine learning were proposed [2, 37].

Although the attempts to predict attacks are almost two decades old, they still mostly focus on predicting the events for a single observation point or a single network and were rarely combined with collaborative approaches to cyber security and intrusion detection. Only a few works approached the problem yet. For example, collaborative predictive blacklisting has been a subject of research [12, 24] with promising results against specific attacks. Predictive analytics based on information exchange can be found in work on network entity characterization by Bartoš et al. [1]. However, a generic approach similar to early attempts to attack prediction is still an open research problem [16, 36].

The purpose of attack projection and prediction is to mitigate the attack preemptively. Thus, the attention of the researchers in the field is also focused on countermeasure selection, e.g., as surveyed by Nespoli et al. [26]. Similarly to preceding tasks, a large amount of heterogeneous data and events calls for research on selecting optimal countermeasures and development of reaction frameworks that can be combined with SIEM, early warning, and prediction systems [30].

Contrary to previous work, we do not focus on methods of alert sharing, correlation, and prediction, but on the design principles of an analytical framework that performs such tasks, especially in a collaborative environment. The methods implemented in our framework and preliminary evaluations of their capabilities were already presented in our previous works [14, 15, 18]. Only a few of the related works describe an implementation of an analytical tool or framework; notable exceptions are the framework by Farhadi et al. [9], who also proposed using stream-based data processing, and RTECA [29], a framework for early warning based on real-time episode correlation. However, tools to be used for processing data from alert sharing platforms are rather scarce.

3 AIDA FRAMEWORK

Herein, we provide the description and reasoning behind the implementation of the AIDA framework. First, we present an overall scheme and outline data paths. Subsequently, we describe particular components for alert processing. General concepts applied in the design of AIDA are modularity and focus on real-time stream-based processing of the data. It is important to keep in mind that the data might be modified or dropped by any component, which might influence the data processing downstream. The variety and order of the presented components are loosely based on the security event prediction use case. Nevertheless, using and developing additional

components is possible and encouraged; several components were used in our previous work for various purposes, such as identifying relations between the data and computing statistics [17, 18].

Schema of the AIDA framework is presented in Figure 1. The central component of the framework is Kafka message broker[3] that takes the data from inputs and distributes them among the other components while ensuring the correct order of processing of the alerts. Particular components can then be found in two component groups based on their underlying software platform. On the left side, we can see a group of components implemented as applications in Spark analytical engine[4]. These applications typically receive a stream of alerts from Kafka, process them, and return them to Kafka. Some applications, such as the Data Mining component, have another output, such as a database for long-term storage of results of data processing. On the right side of the schema, we can see an application based on Esper[5], another event stream processing framework. The component implemented using Esper uses its features, such as a powerful query language, for the benefits of the whole AIDA framework. The Rule Matching component is also one of the few that provide outputs outside the framework. Outputs are in the same form as inputs, i.e., intrusion detection alerts, so the output channel is implemented analogously. More details on particular components are discussed further in this section.

3.1 Inputs and Data Distribution

Data distribution in the AIDA framework is resolved using Apache Kafka, a stream-processing message broker. Kafka receives the alerts from the inputs and distributes them via named topics in the publish-subscribe pattern. In our implementation, we have several topics in which we distribute alerts in different phases of processing. For example, alerts taken from the input are distributed in the *input* topic, where they are accessed by components that require raw data. The Alert Aggregation component reads alerts from the *input* topic, modifies the data, and writes to a different topic, named *aggregated*, where preprocessed data are accessible. Similar topics exist for predicted events and so on.

Inputs may vary depending on the deployment scenario, see Section 4. If the framework is deployed as a service, it receives the alert via a client of the sharing platform. In a standalone deployment, data can be read from a socket or a file system. AIDA framework takes advantage of the popularity of JSON data serialization in current stream-processing tools and, thus, expects the input alerts to be formatted in IDEA (Intrusion Detection Extensible Alert) format[6]. IDEA is inspired by IDMEF, a popular alert exchange format [4]. Contrary to IDMEF, IDEA is extensible, includes a classification of alerts based on taxonomies from CSIRT communities [21], and is adjusted to suit practical needs [20]. However, if needed, conversion from IDMEF to IDEA is rather straightforward.

The inputs may be further checked using the Data Sanitizer component. The Data Sanitizer is a stream-processing tool for syntactic and semantic checks of the alerts on the input. It checks for the validity of the JSON, in which the alert is serialized, checks if all the mandatory entries are present, and uses a set of rules for semantic checks. Alerts matching those rules may be modified or filtered out. The motivation for using semantic checks is to eliminate useless alerts from being processed in the framework. For example, alerts of phishing incidents typically do not contain any IP addresses that we use for correlating alerts. Further, it is possible to delete all the unnecessary entries from the alert to speed up data processing in the whole framework.

3.2 Alert Aggregation Component

Alert Aggregation component is the first application in the processing pipeline. Its task is to find and mark redundant alerts so that these are not included in further analyses. Alert sources often report the same information in several alerts, the alert of the same event may also be generated by multiple sources [18]. The presence of multiple alerts describing the same event may slow down and influence the result of the subsequent operations, especially the data mining process [15].

The component is implemented as a Spark application. It processes the stream of alerts, keeps the vectors of their main features (timestamp, source and target IP addresses, source and target ports, alert source, and event category), and matches the alerts with similar features according to a set of rules. The first alert with such features is left untouched; all the other are marked as duplicates of the first alert. The vectors are kept in a predefined time window, in which the alerts can be matched.

The rules for alert matching were inferred from our previous work [17, 18]. There are two rules to eliminate duplicates and repeated alerts of the same events. If all the features in the vector are the same, or only the timestamp is different, all alerts but the first one are marked as duplicates and are omitted from further processing. Further, there are rules to match alerts of the same event from multiple sources, e.g., intrusion detection systems deployed in different places. These alerts are marked differently as they may or may not be used in further analyses. Further aggregation rules and markings may be applied.

3.3 Data Mining Component

The Data Mining component fulfills the needs of alert correlation. Its purpose is to infer frequent patterns in the alerts so that the patterns may later be used to match related alerts and predict upcoming security events. We used the techniques of sequential pattern and rule mining, which reflect the use cases well [15]. The component consists of two parts, database generator and miner.

The first software part, the database generator, is implemented as a Spark application. It continuously extracts sequences from the alerts and saves them to a database. A sequence is a vector of n-tuples that share a common property. In our implementation, we chose a source IP address as common property. We also implemented a similarly behaving alternative that is using a pair of source and target IP addresses. The n-tuple consists of an alert type, target port, and name of the node (e.g., IDS). Thus, a sequence describes the behavior of a certain IP address in a time window. We chose this combination of keys and values as most promising concerning the number and usability of the results. However, many other combinations can be used for different purposes, e.g., inferring patterns of distributed attacks.

[3] https://kafka.apache.org/
[4] https://spark.apache.org/
[5] http://www.espertech.com/esper/
[6] https://idea.cesnet.cz/en/index

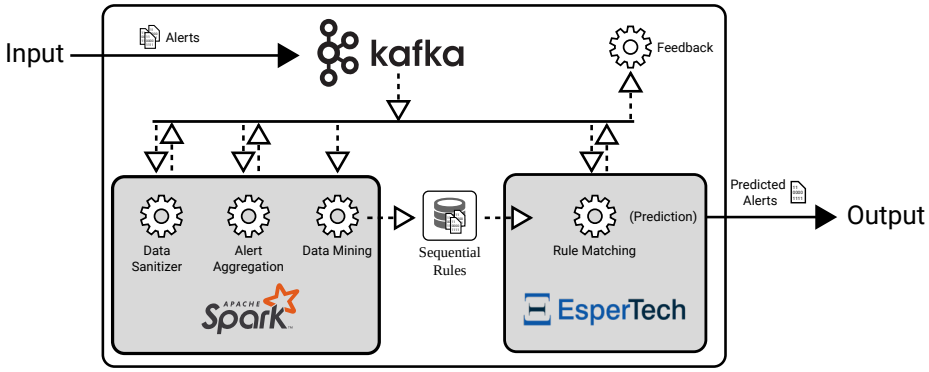

Figure 1: The schema of AIDA, framework for correlation and analysis of intrusion detection alerts.

```
OrganizationA.Honeypot1_Recon.Scanning_22,
OrganizationB.IDS1_Attempt.Login_22
==>
OrganizationA.IDS1_Attempt.Login_22
#SUPP: 0.0011 #CONF: 0.6111
```

Figure 2: Example of a sequential rule.

All the sequences are being continuously stored in a sequence database for the data mining that is conducted by the second part of the component, the rule miner. Contrary to all the other components, the miner is a batch script that is executed periodically, typically once in a day in our case, when enough sequences are extracted by the stream-based database generator. The miner is based on SPMF library [10] that contains implementations of sequential pattern and rule mining algorithms. The script processes the sequences collected by the database generator and mines frequent patterns or rules, depending on the selected algorithm. For the purposes of security event prediction, we chose Top-K sequential rule mining algorithm [11] to mine rules that are directly applicable for predictions. Once the mining is completed, the sequential database is cleared for the next collection period.

An example of an output of the miner, and the Data Mining component, in general, is presented in Figure 2. It consists of ordered n-tuples (sensor name, type of event, destination port) and support and confidence values. The n-tuples fold two groups; the first group implies the second group. Support value indicates the number of sequences in the sequential database that conform to the rule divided by the total number of all the sequences. The confidence value is a conditional probability of the second part of the rule given the first part of the rule.

3.4 Rule Matching Component

The Rule Matching component is the final application in the alert processing pipeline. It processes the stream of alerts and queries them for the presence of predefined patterns (sequences). If a predefined pattern is found, an action is performed. In case of prediction use case, the Rule Matching component takes sequential rules from the previous components and converts them into the queries over the stream of alerts. If the first part of a predictive rule is found in the stream of alerts, a new alert is generated using the second part of the predictive rule. Thus, the predictions are performed in this component.

The component was implemented as a stand-alone application using Esper software. Esper is a tool for Complex Event Processing (CEP) over a stream of data, even streams of large volumes of data such as network flows [38]. Its main advantage is EPL (Event Processing Language) that allows for SQL-like queries over a stream of data. Once an Esper application is running, we provide it with a list of queries and a stream of data. Then, we catch the matches of the query and react with a predefined action. In our case, we convert the first parts of predictive rules into EPL queries so that the matches (query results) trigger the generation of predicted alerts.

An EPL query generated from a sequential rule can be seen in Figure 3. Although the rules are quite simple, there is a minor inconvenience in converting the rules into EPL queries. We had to include permutations in the patterns so that the query behaves correctly. In the example, we have a rule consisting of two items denoted as A and B. Basic query would match only situations where alert A is directly following alert B. However, we want to match any occurrence of alerts A and B, regardless of their order and other alerts between them. Thus, the permutations and the wildcarded alert C were added into the query. The rest of the query is simply matching the items in the alerts.

An example of predicted alert can be seen in Figure 4. The predicted event is derived from the second part of the rule and contains IDs of rules on which the prediction was based, i.e., alerts that were matched by EPL query. The source IP address is taken from the matched alerts, and the remaining features are taken from the rule. Further, the predicted alert contains the current timestamp and descriptions, including the rule itself.

3.5 Feedback Component

The last alert processing component serves to collect feedback on event prediction and similar tasks. The Feedback component checks if a predicted event appeared in the alerts and if so, after what time after the prediction. Thus, we may log and measure the precision of predictions and collect feedback on the predictive rules. Using the logs of the Feedback component, we may calculate minimal, maximal, and average time between the prediction and the

```
select * from Idea.ext:time_order(detectTime.getTime(), 1
      hour)
  match_recognize (
    partition by source[0].IP4[0]
    measures A as a, B as b
    pattern ( match_recognize_permute(A, B, C*?) )
    define
      A as A.category.contains('Recon.Scanning')
        and A.node.countOf(v => v.name = 'OrganizationA.
             Honeypot1') > 0
        and A.target.countOf(v => v.port.countOf(i => i =
              22) > 0) > 0,
      B as B.category.contains('Attempt.Login')
        and B.node.countOf(v => v.name = 'OrganizationB.
             IDS1') > 0
        and B.target.countOf(v => v.port.countOf(i => i =
              22) > 0) > 0
  )
```

Figure 3: Predictive rule converted into EPL query.

```
{
  "Format": "IDEA0",
  "ID":"f62537c2-77b8-49c7-a0a2-24c4b81b20f8",
  "CorrelID": [ % IDs of preceding alerts
        "3688762d-2efa-44a8-9ea5-34a57b3ae0c7",
        "ae6d9ac6-6389-407f-9d7e-58b9692c6eaa"
      ],
  "DetectTime": "2019-03-16T12:17:21.609+00:00",
  "Category": ["Attempt.Login"],
  "Confidence": "0.6111",
  "Description": "The source IP address follows a known
       pattern that is expected to continue with the
       event described in this message.",
  "Note": "OrganizationA.Honeypot1_Recon.Scanning_22,
       OrganizationB.IDS1_Attempt.Login_22 ==>
       OrganizationA.IDS1_Attempt.Login_22",
  "Source": [{"IP4": ["10.11.12.13"]}],
  "Target": [{"Port":[22]}],
  "Node": [
    { % Node referencing the AIDA Framework
      "Name": "OrganizationX.AIDA",
      "SW": "AIDA",
      "Type": ["Correlation", "Statistical"]
    },
    { % Node derived from the rule
      "Name": "OrganizationA.IDS1"
    }
  ]
}
```

Figure 4: Example of predicted security event.

predicted event, which might serve as indicators of usability of the rules. For example, predicting events that will happen immediately is not useful, while predicting the events several minutes or hours in advance leaves enough time to spread the predicted alerts and set up proper countermeasures [14].

We decided to implement feedback collection as a standalone component rather than a part of the Rule Matching component. Thus, we may conduct experiments with various implementations of event predictions while keeping a single feedback component that calculates the necessary metrics. The functionality of the component is simple, in essence, so there is no need to implement it using frameworks like Spark or Esper as in other components. Instead, we used a script in Python that saves unique identifiers of predicted events in an array and checks if newly arrived alerts have similar content. If a match is found, the component logs a match and time difference between prediction and arrival of predicted alert. This information is later used for calculating prediction metrics, including true and false positive rates, precision, and recall. Calculation of the minimal and average time between prediction and predicted alert is beneficial for future use of predictive rules. However, the requirements on the Feedback component may vary depending on a particular use, so we leave space for future work.

3.6 REST API and Web Interface

The AIDA framework is equipped with a REST API and web interface so that the runtime information is easily accessible. The REST API serves as the main communication point between the AIDA framework and the web interface. It provides the information on the status of the framework, its components, and underlying systems, including the runtime, system load, and last five log entries. Further, there are endpoints enabling manipulation with the database of sequential rules that allow for actions such as listing the rules, (de)activating rules, deleting rules, and inserting custom rules.

The web interface is designed in the form of a dashboard. The initial screen shows the schema of the AIDA framework, where each component signalizes its status with a green or red light. Further screens show detailed information on particular components.

4 DEPLOYMENT AND EVALUATION

In this section, we describe how the AIDA framework was deployed in the SABU alert sharing platformand how it performed. First, we include the schema of deployment and important configuration options. Subsequently, we provide observations and comment on experiences gained while running the framework. Finally, we briefly describe an alternative deployment option for experimentation and repeatable measurements.

4.1 Deployment in the Alert Sharing Platform

The position of AIDA within an alert sharing platform is reflecting the fact the sharing platform is centralized with a single sharing hub. Thus, AIDA needs to interact with only the central sharing hub. The central hub receives the alerts from various sending connectors, including third-party alert sharing systems, and distributes them to various receiving connectors, including reporters. There is no need for AIDA to communicate directly with other components of the sharing platform. Otherwise, AIDA acts as both receiving and sending connector, in essence, indistinguishable from other connectors in terms of communication interface and implementation. The schema of AIDA and its placement within the sharing platform is depicted in Figure 5. The other components and data flow are described in this section.

The data flow in the sharing platform relevant for AIDA goes as follows. The alerts raised by intrusion detection systems and other sensors are collected by sending connectors and sent to the central sharing hub. Alternatively, alerts are transferred from a third-party alert sharing platform and shared via a specialized sending connector. In both cases, the incoming data are formatted in IDEA alert

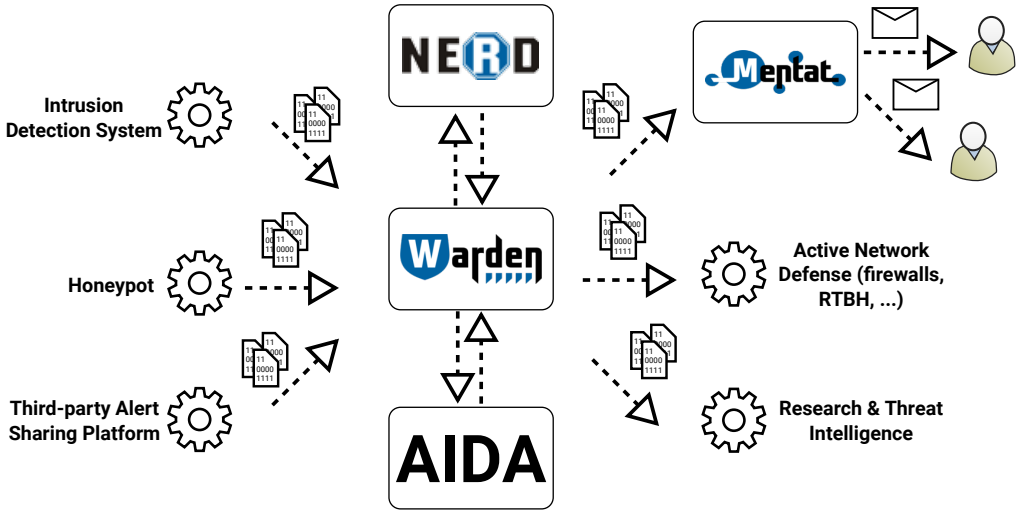

Figure 5: The deployment of AIDA in the SABU alert sharing platform.

format. The central hub stores a queue of alerts that are ready to be dispatched to receiving connectors. Receiving connectors, including AIDA, can access all the alerts in the central hub queue in real time. AIDA receives the data from the central hub, process them, and generates predicted alerts. Again, the alerts are formatted in IDEA format, although with appropriate marks that distinguish predicted alert from an ordinary alert. Predicted alerts are sent to the central hub due to AIDA's ability to act also as a sending connector. The remaining receiving connectors may receive all the data from the central hub, now including the predicted alerts from AIDA. It is up to receiving connectors what to do with such information; potential mitigation and incident response [26] are out of the scope of this work.

To summarize the data flows, there are three paths of alerts. First, there is a direct path from sending connectors to receiving connectors via the central sharing hub. Potentially, every receiving connector may receive all the data from sending connectors, but due to the amount of data and the fact that only a part of them are usable at some point, it is not often the case. Thus, the second path for alerts is from sending connectors via the central hub to AIDA, where the data are consumed during the analysis. Then, there is a path for predicted alerts from AIDA via the sharing hub to the receiving connectors. In this path, only the predicted alerts are transferred, which significantly reduces the volume of the data.

4.2 Configuration and Discussion

Herein, we are going to briefly comment on experience gained while running AIDA framework as described above. The AIDA framework was deployed on a dedicated server based on commodity hardware (Intel Xeon E5520, 8 threads, 16 GB RAM) and running Debian 8.11. Particular components were based on Kafka version 2.2.0, Spark version 2.2.0, and Esper version 8.0.0, most of them running on Java 8. As of the first half-year of 2019, we receive around 1.7 million alerts per day. We did not encounter any significant performance issues or insufficient memory issues during the experimental deployment with the presented configuration. Batch-processing tasks, such as data mining, caused spikes in hardware load, but it did not slow down the rest of the framework. However, we are aware of configurations with poor performance. Luckily, such tasks are not the ones with valuable results [15].

Strict input filtering rules had to be applied to ensure the correct functioning of the framework and reasonable outputs. The connector to the sharing platform allows for limited filtering of the alerts, but the filtering rules for AIDA turned out to be more complex. Apart from the validity and completeness of IDEA-formatted alerts, we had to check for semantic issues and particular content. First, we had to drop all the alerts that contained no source IP address or destination port, because our use case relies on these identifiers. Similarly, alerts of selected categories were also dropped from processing, including alerts of misconfigurations and vulnerability presence, which are not very useful for our use case. Subsequently, encounter numerous alerts of questionable quality. For example, some peers in the sharing platform contributed with alerts of questionable quality, such as alerts of network scanning raised in response to only one or very few packets, while the majority of peers understand network scanning as an event consisting of hundreds or thousands of packets. Similarly, we encountered misleading alerts of brute-force password attack raised in response to a single failed authentication. In response, we set a threshold for the minimal number of packets in scans and attempts at brute-force password attacks. Another problem that we faced was with alerts containing too many sources or targets. Peers in the platforms crafted alerts with thousands of source IP addresses instead of putting IP address range as recommended. Such alerts, in some cases, caused performance issues in some components, where particular IP addresses and ports were correlated. Overall, input filtering dropped approximately 1 % of the alerts.

The amount of aggregated alerts loosely follows our previous measurements [18]. Around 1.5 % of alerts are duplicates of another alert, i.e., they contain the same information. Around 54 % of alerts are consecutive duplicates of another alert, i.e., alerts by the same data source in response to the same event over an extended time.

Other deduplication and aggregation methods were not examined as they were not necessary for the given use case. Time windows, in which the alerts were aggregated, were set to 5 minutes for duplicates and 70 minutes for continuations because some contributing data sources report alerts once per hour.

The Data Mining component performed adequately; the sequential database was collected continuously and, once a day, the data mining was executed, and the database was cleared. From the 1.7 million alerts per day, we distilled around 650,000 sequences that had the same source IP address and around 170,000 sequences that shared the same source and target IP address. The run time of the data mining using Top-k algorithm from the SPMF library was typically around 20 seconds in both cases. From the various combinations of item sets, we choose to use items consisting of alert category, source IP address, and target port. Given the data, this combination led to the most valuable results and reflected the behavior of an attacker fairly well. Further, we experimented with a *null* value for port numbers. Not many alerts contained target port number, so we added the *null* value instead, and let the framework process it as any other port number. There were situations where part of the rule (attackers' common behavior) was a vertical scan or another event that is often reported without a port number. However, due to a large number of alerts of various categories without a port number, we saw an increase in the number of rules with low information value, such as correlations of unspecified network scans with other unspecified network scans.

The framework is designed to store newly found rules in the database, where it can be checked by a human operator and activated for the needs of the Rule Matching component. During the experimentation, we updated the rules every day with the rules mined on the previous day. We did not perform manual filtering. However, mining 10 alerts per day, we noticed 2-3 rules per day that would not be suitable for practical use, mostly because of their obviousness or low informative value, such as permutations of other rules or rules based on detection of the same event by different intrusion detection systems overlapping in network scope. As we illustrated earlier [14], the rules are stable over time, so on average 8 rules out of 10 mined every day are the same as the day before, the only difference is in slightly changing values of support and confidence. However, long-term monitoring of mined rules, at least over several months, would be beneficial.

The Rule Matching and Feedback components provided expected results since the rules are stable over several days [14]. Thus, the number of matched rules and percentage of successful predictions loosely followed the values of support and confidence of the rule. Although we set the minimal threshold for confidence value in a rule to 0.5, the actual values in Top-10 rules rarely dropped below 0.6 and often reached values up to 0.8 and 0.9 [14, 15]. Time differences between the prediction and observation of the predicted event were in the order of minutes or tens of minutes on average, while minimal time differences in approximately half of the cases dropped to only a few seconds. We considered this when selecting suitable predictive rules, and we would recommend to set up rule matching first, estimate minimal and average time differences between predictions and predicted events, and then activate or inactivate the rule in the matching component.

On the contrary, the performance of the Rule Matching component was not always satisfactory. Given the fact that every query in Esper requires approximately the same available resources, it is not very scalable. We were running ten queries in parallel, one for every Top-10 rules mined on the previous day. Running more parallel queries caused quick depletion of memory. However, we optimized the query by introducing a time window in which the alerts and queried and by dissecting the query. All the items from the rule are detected separately, and the matching of the rule is performed on the outputs of these queries, i.e., on a higher level of abstraction. Thus, we achieved a significant reduction of memory consumption for the cost of a slight increase in the demands on CPU, which allowed us to run more than a hundred queries in parallel. The optimized query is too long to be included in the text and not as comprehensible as the original query in Figure 3. Nevertheless, the principles remain the same.

4.3 Stand-alone Deployment

An alternative deployment of the AIDA framework is a stand-alone application. In such a scenario, AIDA is started locally, and the input data are provided in a batch. This is convenient for testing and processing batches of data such as datasets. The whole event prediction use case discussed throughout this paper might not be the most suitable for a stand-alone deployment scenario. However, particular components and tasks, such as relationship discovery and mining attack sequences, are worth performing over datasets and other batch inputs. Further, stand-alone deployment, along with a proper dataset, allows for experimentation with the data using different settings of the components and used methods.

There are several specifics of the stand-alone deployment scenario. First, the data must be sent manually to the framework. We implemented an interface that allows Kafka to automatically read alerts from files in a given directory so that the users only have to copy the dataset to the appropriate location. Some components are dependant on timing, which means that, in a stand-alone mode, a signal needs to be sent to the components to trigger actions such as running the data mining. Users are allowed to trigger such actions via the web interface.

5 CONCLUSION AND FUTURE WORK

In this paper, we presented AIDA, a framework for analyzing cybersecurity alerts with a special focus on alert correlation and prediction of security events. The source codes are available on GitHub. The framework allows for stream processing of security alerts in near real time and is based on current technologies for Big Data processing, such as Apache Spark and Esper. We deployed the AIDA framework in the SABU alert sharing platform, where alerts from various intrusion detection systems and honeypots from multiple networks are exchanged. The Alert Aggregation, Data Mining, and Rule Matching components were used in a scenario of security event prediction.

In our future work, we are going to develop the framework further and investigate the methods used in its components. For example, the Rule Matching component is based on Esper so that it can run complex queries over the stream of data. This approach is reliable and highly extensible. However, certain tasks, such as security

event prediction, might be optimized using alternative approaches and implementations. In related work, we can see many approaches to security event prediction based on Attack graphs, Bayesian networks, and Markov models [16, 26]. These models provide a solid background for efficient computation, and it might be interesting to compare the performance of CEP-based and graph-based matching components. Finally, we are going to publish an annotated dataset of alerts so that the users have a chance to familiarize themselves with the framework on sample data.

ACKNOWLEDGMENTS

This research was supported by ERDF "CyberSecurity, CyberCrime and Critical Information Infrastructures Center of Excellence" (No. CZ.02.1.01/0.0/0.0/16_019/0000822).

REFERENCES

[1] Václav Bartoš, Martin Žádník, Sheikh Mahbub Habib, and Emmanouil Vasilomanolakis. 2019. Network entity characterization and attack prediction. *Future Generation Computer Systems* 97 (2019), 674 – 686.

[2] Casey Cipriano, Ali Zand, Amir Houmansadr, Christopher Kruegel, and Giovanni Vigna. 2011. Nexat: A History-based Approach to Predict Attacker Actions. In *Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC '11)*. 383–392.

[3] Frédéric Cuppens and Alexander Miège. 2002. Alert correlation in a cooperative intrusion detection framework. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*. 202–215.

[4] H. Debar, D. Curry, and B. Feinstein. 2007. The Intrusion Detection Message Exchange Format (IDMEF). RFC 4765 (Experimental). (March 2007). http://www.ietf.org/rfc/rfc4765.txt

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

[6] Huwaida Tagelsir Elshoush and Izzeldin Mohamed Osman. 2011. Alert correlation in collaborative intelligent intrusion detection systems – A survey. *Applied Soft Computing* 11, 7 (2011), 4349–4365.

[7] ENISA. 2013. Detect, SHARE, Protect – Solutions for Improving Threat Data Exchange among CERTs. https://www.enisa.europa.eu/activities/cert/support/information-sharing/detect-share-protect-solutions-for-improving-threat-data-exchange-among-certs/at_download/fullReport. (Oct. 2013).

[8] ENISA. 2014. Standards and tools for exchange and processing of actionable information. https://www.enisa.europa.eu/publications/standards-and-tools-for-exchange-and-processing-of-actionable-information/at_download/fullReport. (Nov. 2014).

[9] Hamid Farhadi, Maryam AmirHaeri, and Mohammad Khansari. 2011. Alert Correlation and Prediction Using Data Mining and HMM. *ISeCure* 3, 2 (2011).

[10] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Antonio Gomariz, Ted Gueniche, Azadeh Soltani, Zhihong Deng, and Hoang Thanh Lam. 2016. The SPMF Open-Source Data Mining Library Version 2. In *Proc. 19th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 2016)*. Part III, Springer LNCS 9853, 36–40.

[11] Philippe Fournier-Viger and Vincent S. Tseng. 2011. Mining top-k sequential rules. In *International Conference on Advanced Data Mining and Applications*. Springer, 180–194.

[12] Julien Freudiger, Emiliano De Cristofaro, and Alejandro E. Brito. 2015. *Controlled Data Sharing for Collaborative Predictive Blacklisting*. Springer International Publishing, Cham, 327–349.

[13] Carol Fung and Raouf Boutaba. 2013. *Intrusion Detection Networks: A Key to Collaborative Security*. CRC Press.

[14] Martin Husák and Jaroslav Kašpar. 2018. Towards Predicting Cyber Attacks Using Information Exchange and Data Mining. In *2018 14th International Wireless Communications Mobile Computing Conference (IWCMC)*. 536–541.

[15] Martin Husák, Jaroslav Kašpar, Elias Bou-Harb, and Pavel Čeleda. 2017. On the Sequential Pattern and Rule Mining in the Analysis of Cyber Security Alerts. In *Proceedings of the 12th International Conference on Availability, Reliability and Security*. ACM, Reggio Calabria, "22:1–22:10".

[16] Martin Husák, Jana Komárková, Elias Bou-Harb, and Pavel Čeleda. 2019. Survey of Attack Projection, Prediction, and Forecasting in Cyber Security. *IEEE Communications Surveys & Tutorials* 21, 1 (Firstquarter 2019), 640–660.

[17] Martin Husák and Milan Čermák. 2017. A Graph-based Representation of Relations in Network Security Alert Sharing Platforms. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, Lisbon, 891–892.

[18] Martin Husák, Milan Čermák, Martin Laštovička, and Jan Vykopal. 2017. Exchanging Security Events: Which And How Many Alerts Can We Aggregate?. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, Lisbon, 604–607.

[19] Ci-Bin Jiang, I-Hsien Liu, Yao-Nien Chung, and Jung-Shian Li. 2016. Novel intrusion prediction mechanism based on honeypot log similarity. *International Journal of Network Management* 26, 3 (2016), 156–175.

[20] Pavel Kácha. 2013. IDEA: Designing the Data Model for Security Event Exchange. In *17th International Conference on Computers: Recent Advances in Computer Science*.

[21] Pavel Kácha. 2014. IDEA: Security Event Taxonomy Mapping. In *18th International Conference on Circuits, Systems, Communications and Computers*.

[22] Yong-Ho Kim and Won Hyung Park. 2014. A study on cyber threat prediction based on intrusion detection event for APT attack detection. *Multimedia Tools and Applications* 71, 2 (2014), 685–698.

[23] Jie Lei and Zhi tang Li. 2007. Using Network Attack Graph to Predict the Future Attacks. In *Communications and Networking in China, 2007. CHINACOM '07. Second International Conference on*. 403–407.

[24] Xiaobo Ma, Jiahong Zhu, Zhiyu Wan, Jing Tao, Xiaohong Guan, and Qinghua Zheng. 2010. Honeynet-based collaborative defense using improved highly predictive blacklisting algorithm. In *2010 8th World Congress on Intelligent Control and Automation*. 1283–1288.

[25] Guozhu Meng, Yang Liu, Jie Zhang, Alexander Pokluda, and Raouf Boutaba. 2015. Collaborative Security: A Survey and Taxonomy. *ACM Comput. Surv.* 48, 1, Article 1 (July 2015), 42 pages.

[26] Pantaleone Nespoli, Dimitrios Papamartzivanos, Félix Gómez Mármol, and Georgios Kambourakis. 2018. Optimal Countermeasures Selection Against Cyber Attacks: A Comprehensive Survey on Reaction Frameworks. *IEEE Communications Surveys Tutorials* 20, 2 (Secondquarter 2018), 1361–1396.

[27] Dick Ourston, Sara Matzner, William Stump, and Bryan Hopkins. 2003. Applications of hidden Markov models to detecting multi-stage network attacks. In *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*. 10.

[28] Xinzhou Qin and Wenke Lee. 2004. Attack plan recognition and prediction using causal networks. In *Computer Security Applications Conference, 2004. 20th Annual*. 370–379.

[29] Ali Ahmadian Ramaki, Morteza Amini, and Reza Ebrahimi Atani. 2015. RTECA: Real time episode correlation algorithm for multi-step attack scenarios detection. *Computers & Security* 49 (2015), 206 – 219.

[30] Ali Ahmadian Ramaki and Reza Ebrahimi Atani. 2016. A survey of IT early warning systems: architectures, challenges, and solutions. *Security and Communication Networks* 9, 17 (2016), 4751–4776.

[31] Ali Ahmadian Ramaki, Massoud Khosravi-Farmad, and Abbas Ghaemi Bafghi. 2015. Real time alert correlation and prediction using Bayesian networks. In *2015 12th International Iranian Society of Cryptology Conference on Information Security and Cryptology (ISCISC)*. 98–103.

[32] Florian Skopik. 2018. *Collaborative cyber threat intelligence: detecting and responding to advanced cyber attacks at the national level*. CRC Press.

[33] Jessica Steinberger, Anna Sperotto, Mario Golling, and Harald Baier. 2015. How to exchange security events? Overview and evaluation of formats and protocols. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 261–269.

[34] Zhi tang Li, Jie Lei, Li Wang, and Dong Li. 2007. A Data Mining Approach to Generating Network Attack Graph for Intrusion Prediction. In *Fuzzy Systems and Knowledge Discovery, 2007. FSKD 2007. Fourth International Conference on*, Vol. 4. 307–311.

[35] Fredrik Valeur, Giovanni Vigna, Christopher Kruegel, and Richard A. Kemmerer. 2004. Comprehensive approach to intrusion detection alert correlation. *IEEE Transactions on Dependable and Secure Computing* 1, 3 (July 2004), 146–169.

[36] Emmanouil Vasilomanolakis, Shankar Karuppayah, Max Mühlhäuser, and Mathias Fischer. 2015. Taxonomy and Survey of Collaborative Intrusion Detection. *ACM Comput. Surv.* 47, 4, Article 55 (May 2015), 33 pages.

[37] Kalyan Veeramachaneni, Ignacio Arnaldo, Vamsi Korrapati, Constantinos Bassias, and Ke Li. 2016. AI^2: Training a Big Data Machine to Defend. In *2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS)*. 49–54.

[38] Petr Velan, Martin Husák, and Daniel Tovarňák. 2018. Rapid prototyping of flow-based detection methods using complex event processing. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*.