Sebastian Roschke, Feng Cheng, Christoph Meinel: "A Flexible and Efficient Alert Correlation Platform for Distributed IDS"
in Proceedings of the 4th International Conference on Network and System Security (NSS 2010), IEEE Press, Melbourne, Australia, pp. 24-31, 9, 2010. ISBN: 978-0-7695-4159-4.

2010 Fourth International Conference on Network and System Security

A Flexible and Efficient Alert Correlation Platform for Distributed IDS

Sebastian Roschke, Feng Cheng, Christoph Meinel
Hasso Plattner Institute (HPI), University of Potsdam
P.O.Box 900460, 14440, Potsdam, Germany
{sebastian.roschke, feng.cheng, meinel}@hpi.uni-potsdam.de

***Abstract*—Intrusion Detection Systems (IDS) have been widely deployed in practice for detecting malicious behavior on network communication and hosts. The problem of false-positive alerts is a popular existing problem for most of IDS approaches. The solution to address this problem is correlation and clustering of alerts. To meet the practical requirements, this process needs to be finished as soon as possible, which is a challenging task as the amount of alerts produced in large scale deployments of distributed IDS is significantly high. We identify the data storage and processing algorithms to be the most important factors influencing the performance of clustering and correlation. We propose and implement the utilization of memory-supported algorithms and a column-oriented database for correlation and clustering in an extensible IDS correlation platform. The utilization of the column-oriented database, an *In-Memory Alert Storage*, and memory-based index tables leads to significant improvements on the performance. Different types of correlation modules can be integrated and compared on this platform. A plugin concept for *Receivers* provides flexible integration of various sensors and additional IDS management systems. The platform can be distributed over multiple processing units to share memory and processing power. A standardized interface is designed to provide a unified view of result reports for end users. The efficiency of the proposed platform is tested by practical experiments with several alert storage approaches, different simple algorithms, as well as local and distributed deployment.**

***Keywords*-Memory-based Correlation, Memory-based Clustering, Memory-based Databases, IDS Management**

I. Introduction

Intrusion Detection Systems (IDS) has been proposed for years as an efficient security measure and is nowadays widely deployed for securing critical IT-Infrastructures [27]. Lots of commercial and open source IDS implementations have emerged towards identifying malicious behaviors against protected hosts or network environments. An effective IDS should be capable of detecting different types of attacks as well as all the possible variants of a certain type of attack. An IDS should detect not only known attacks but also unknown attacks. Furthermore, the IDS itself needs to be robust against any evasion techniques. To simultaneously provide multiple benefits from various IDS sensors, an integrated IDS solution is required, which mostly relies on unified data and communication (e.g. IDMEF[2]).

The problem of false positive alerts is a well known problem for many IDS approaches [27]. Suboptimal patterns or insufficient thresholds for pattern-based and anomaly-based IDS approaches are the main reasons for a huge number of false-positive alerts. By deploying the IDS sensors in a distributed environment, the number of false positive alerts increases as a single event may be detected and reported multiple times by different involved sensors. The popular solution to address this problem is correlation and clustering of alerts. To improve the efficiency of clustering and correlation, several techniques have been proposed [1]. As high security requirements of networks need real-time reporting of attacks and malicious behavior, the performance of these techniques is a critical factor. In particular in large scale distributed IDS (DIDS), providing high-performance correlation and clustering remains to be a difficult challenge, due to a huge amount of alerts. The performance of alert correlation can be improved by using table indexes in main memory for hyper alerts [26], i.e., clusters of alerts with the same properties. Furthermore, correlation in real-time is often based on filtering and clustering of alerts to hyper alerts [25], which reduces the number of processed alerts significantly. The approach reaches a correlation rate on the order of $100,000$ alerts per second based on the massive reduction of alerts by clustering them. However, a general approach that can handle a higher amount of alerts per second without the need of alert reduction is considered to be useful.

In-Memory and column-based databases are usually used for costly analytical processing of huge amounts of data [15]. A column-based organization of the database improves analytical operations, which often consist of comparison of all values from a single or multiple columns. As described in [17], database systems can benefit from the use of main memory. In-memory and column-based databases are suitable for future computing paradigms, such as multi-core systems, which are supposed to have more CPUs and a huge amount of main memory. By storing a database in the main memory, analytical operations can be processed in parallel by several CPUs with direct access to the main memory. There are many implementations for column-based database systems, such as MonetDB [18], [19], which is an open-source database system for high-performance analytical operations, e.g., Online Analytical Processing (OLAP), Geographic Information Systems (GIS), XML Query, text and multimedia retrieval. MonetDB often achieves a significant speed improvement for SQL over other open-source systems, e.g., MySQL[20] or PostgreSQL[21]. The general benefits of in-memory and column-based databases can be useful to correlation and clustering of IDS alerts.

In large scale deployments of DIDS, huge number of alerts are produced in a short time. To fulfill the challenging

978-0-7695-4159-4/10 $26.00 © 2010 IEEE
DOI 10.1109/NSS.2010.26

Sebastian Roschke, Feng Cheng, Christoph Meinel: "A Flexible and Efficient Alert Correlation Platform for Distributed IDS"
in Proceedings of the 4th International Conference on Network and System Security (NSS 2010), IEEE Press, Melbourne, Australia, pp. 24-31, 9, 2010. ISBN: 978-0-7695-4159-4.

task of fast correlation and clustering, we identified the data storage and processing algorithms to be the most important among several other influential factors for the performance. To improve those factors, we propose the utilization of improved algorithms using a memory based index table and the deployment of In-Memory or column-oriented databases for correlation and clustering. A flexible correlation system is needed, which synchronizes, unifies, and analyzes all the security related events produced by the integrated sensors. To meet these requirements, we implement an extensible IDS correlation platform in this paper, which consists of several *Correlation Handlers*, a unified *Alert StorageController*, a unified *AlertUpdateController*, and a *RequestController* for visual presentation and network communication. The utilization of a *Column-oriented Database* and an *In-Memory Alert Storage* in connection with *Improved Algorithms* using *Memory-based Index Tables* for correlation and clustering lead to significant improvements of the performance. Different types of correlation modules can be easily integrated and compared on this platform. A new plugin concept for *Receivers* provides flexible integration of various sensors and additional IDS management systems. The platform can be distributed over several units to share memory and processing power. The IDMEF standard is used to represent and exchange the alert information. A standardized interface is designed to provide a unified view of result reports for users. The efficiency of the proposed platform is evaluated by practical experiments for various alert storage approaches and simple algorithms, within local or distributed deployment of the platform.

The rest of the paper is organized as follows. Section II introduces some related works in the field of clustering and correlation as well as memory-based databases. In Section III, the the performance problems of correlation are discussed and advanced algorithms are proposed and analyzed. Section IV presents the architecture and implementation of our correlation platform. In Section V, the performance of the proposed approach is compared and benefits as well as drawbacks are discussed. Section VI lists some possible future works and Section VII gives a short summary.

II. Related Work

A. Alert Correlation

The alert correlation framework usually consists of several components [1]: *Normalization*, *Aggregation (Clustering)*, *Correlation*, *False Alert Reduction*, *Attack Strategy Analysis*, and *Prioritization*. Over the last years, alert correlation research focused on new methods and technologies for these components. IDMEF s[2] and CVE [3] are important efforts in the field of *Normalization*. Approaches of aggregation are mostly based on similarity of alerts [5], [6] or generalization hierarchies [4]. The correlation algorithms [1] can be classified as: *Scenario-based correlation* [7], *Rule-based correlation* [8], *Statistical correlation* [9], and *Temporal correlation* [10]. False alert reduction can be done by using such techniques as data mining [11] or fuzzy techniques [12]. Attack strategy analysis often depends on reasoning and prediction of attacks missed by the IDS [13]. In terms of Prioritization, the alerts are categorized based on their severity, e.g., using attack ranks [14]. To solve problems of alert correlation, a variety of disciplines are used, e.g., machine learning, data mining [11], or fuzzy techniques [12]. Most of the efforts do not consider the aspect of performance, which is needed in case of huge amounts of alerts.

The work described in [25] considers the performance of alert correlation by using memory-based table indexes for hyper alerts. A hyper alert is a cluster of alerts with the same properties, e.g., the same source address and target address. The approach using index tables is introduced in [26]. To perform correlation in real-time, the approach is based on filtering and clustering of alerts to hyper alerts, which reduces the number of processed alerts significantly. However, this technique may lead to inexact results of the correlation, as multiple alerts are generalized to a single hyper alert. The approach reaches a correlation rate on the order of $100,000$ alerts per second based on the massive reduction of alerts by clustering in hyper alerts. As we use a more general approach (by modifying the data storage and using a distributed architecture), we claim that we can handle even more alerts per second without the need of alert reduction.

B. In-Memory and Column-oriented Databases

Memory-based column databases are used for costly analytical processing of huge amounts of data [15]. The column-based organization of the database improves the performance of analytical operations, which often include a comparison of all values from a single or multiple columns, e.g., the calculation of statistical values or the grouping and clustering of data. Furthermore, column-based databases show good results in compression of the data [16]. As described in [17], database systems can benefit from the use of main memory. Especially future computing paradigms, such as multi-core systems, can benefit from the utilization of *In-Memory Databases* and *Column-oriented Databases* with a huge amount of main memory. By storing a database into the main memory, analytical operations can be processed in parallel by several CPUs with direct access to the main memory. Known problems could be persistence and recovery. There are multiple implementations for column-based database systems, such as MonetDB [18], [19], which is an open-source database system for high-performance applications based on analytical operations, e.g., data mining, Online Analytical Processing (OLAP), Geographic Information Systems (GIS), XML Query, text and multimedia retrieval. Due to the column-based organization, MonetDB often achieves a significant speed improvement for SQL over other open-source systems, e.g., MySQL[20] or PostgreSQL[21].

III. Correlation and its Performance

The efficiency of the correlation depends on the quality of the algorithm and its performance as well as the storage and organization of original alerts. The quality is a measure of the correctness of the algorithm and depicts how many of the recognized correlations are correct, i.e., how many of the correlations found represent existing relations between alerts.

Sebastian Roschke, Feng Cheng, Christoph Meinel: "A Flexible and Efficient Alert Correlation Platform for Distributed IDS"
in Proceedings of the 4th International Conference on Network and System Security (NSS 2010), IEEE Press, Melbourne, Australia, pp. 24-31, 9, 2010. ISBN: 978-0-7695-4159-4.

Furthermore, it depicts how many of the existing relations between alerts are found by the algorithm. The performance of the correlation describes the amount of time needed to correlate a number of alerts. Due to complex and large scale networks, the amount of alerts increases significantly. Therefore, the performance of correlation algorithms is a major aspect of the efficiency of correlation.

A. Correlation Algorithms and Performance

The first considered algorithm is called *Simple Clustering*. The algorithm takes a specific column as input and returns the clusters regarding this column. An example application for this algorithm is the retrieving of all different alerts with IP_i as target IP from the data source, i.e., i different clusters with alerts. The second algorithm is called *Aggregated Clustering*. The algorithm takes a specific column $colA$ and a specific value $valB$ as input. It returns the clusters regarding $colA$ with all alerts that possess $valB$, which is a value from another column. An example application for this algorithm is the retrieving of all different alerts with IP_i as target IP from the data source, which have the source IP $IP_0 = 123.123.123.123$. The third algorithm is called *Simple Correlation*. The algorithm takes two columns $colA$ and $colB$ as well as one constraint $const$ to calculate correlations. It returns the correlations with the following simple semantic: all alerts have the values $valA = valB$ in the columns $colA$ and $colB$, and fulfill the constraint $const$. An example could be all alert sets, with alerts A_0 posessing the target IP $IP_T = 123.123.123.123$ and alerts A_1 posessing the source IP $IP_S = 123.123.123.123$ fulfilling the constraint $const(a, b) = T(a) < T(b)$, where $T(a)$ is the time-stamp of the alert.

The performance of correlation depicts how many alerts can be processed in a certain amount of time. The performance is influenced by numerous factors:

- Quality of data (original alerts)
- Hardware Resources
- Storage and organization of data (used DB-technology and schema)
- Algorithms

The quality of data describes how many duplicate alerts are in the database, whether the alerts in the database are correct, or if there is any wrong information in the database. A low quality of the data leads to increasing processing time and therefore affects the performance. The hardware (e.g. CPU, memory, etc.) directly influences the processing time of the correlation and is an important factor for its performance. The database technology (e.g. row-based, column-based, memory-based, etc) can influence the performance of the correlation depending on the used algorithms. The database scheme can influence the performance badly, if it is very complicated and leads to complex database queries in the correlation algorithm. The algorithm itself affects the performance of the correlation process, as it defines the number of necessary database queries or the usage of memory-internal data structures.

To improve the performance of the correlation, we decided to focus on the factors: database technology and algorithms. Known correlation approaches use row-oriented databases as basic alert source. Such databases show low performance for analyzing huge amounts of data. Therefore, we used column-oriented and memory-based databases to improve the performance of the correlation significantly. Additionally, known correlation and clustering algorithms do not make full use of the capabilities of the main memory. Therefore, these algorithms only provide medium performance. We introduce hash-index supported algorithms which heavily use the main memory and show high performance.

B. Advanced Algorithms and Data Structure

To improve the performance of the previously described algorithms, it makes sense to make extended use of main memory to store and process alerts. Two important modifications are made to the algorithms. First, we load basic information for all alerts into the memory. Second, all identified clusters are saved as hash tables for further processing. To save memory, we only store a subset of the alert data (called *CorrelationData*) in the main memory. All alerts can be retrieved from the database on demand by using a message Id which needs to be unique. The subset includes the values:

- messageId
- creationTime
- analyzerName
- sourceName
- sourceAddress
- targetName
- targetAddress
- classificationText
- reference

The subset consists of values from the IDMEF[2] Alert. The *messageId* is an attribute of the *Alert*, which can be used to identify it in the system. The *creationTime* holds the time the *Alert* was created. The *sourceName* and *sourceAddress* hold information on the source of the attack triggering the alert, i.e., the name and IP address of the source. The *targetName* and *targetSource* hold information on the victim of the attack triggering that alert, i.e., the name and IP address of the source. The *classificationText* holds the actual content of the *Alert*, i.e., the basic information to classify the *Alert*. The *reference* holds references to external databases related to that attack, such as CVE [3].

To store all alerts in the database, we use a simple database scheme with two tables: one table storing the *CorrelationData* and one table storing a reference to the full alert data. We use that simple structure to improve the performance of the correlation, as a complicated scheme would yield costly table joins for analytical operations. The In-Memory alert storage is organized in a configurable way. The simple list is the basic structure with hash tables for correlation. To perform more sophisticated and high-performance correlation, the platform offers the approaches described in [26] for organizing the main memory. In this way, the platform can provide optimal data structures for performing correlation in batch mode (i.e. correlation of a large fixed set of alerts) or in a streamed mode (i.e. correlation of continuously incoming alerts).

Sebastian Roschke, Feng Cheng, Christoph Meinel: "A Flexible and Efficient Alert Correlation Platform for Distributed IDS"
in Proceedings of the 4th International Conference on Network and System Security (NSS 2010), IEEE Press, Melbourne, Australia, pp. 24-31, 9, 2010. ISBN: 978-0-7695-4159-4.

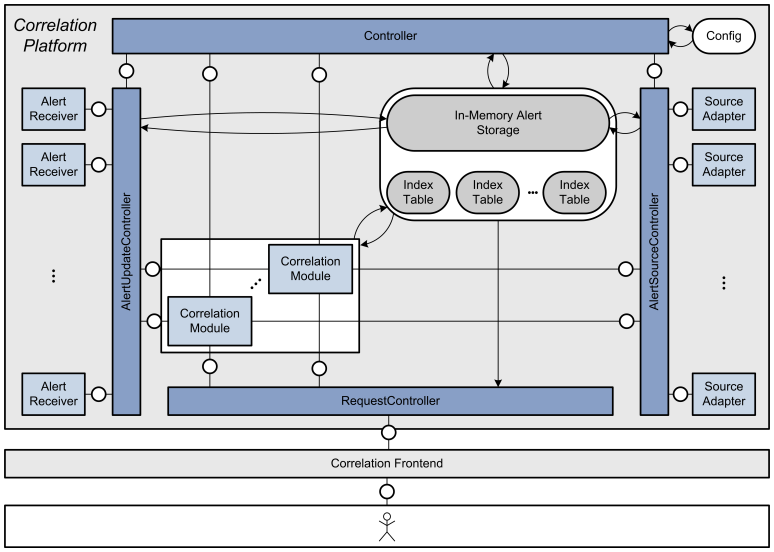

Fig. 1. IDS Correlation Platform

IV. An Extensible Correlation Platform

To test the performance of the above mentioned simple algorithms and advanced algorithms as well as column-oriented and in-memory databases for correlation, we propose a flexible correlation platform.

A. Architecture

The architecture of the correlation platform is shown in Figure 1. It consists of four major components: the *Controller*, the *RequestController*, the *AlertUpdateController*, and the *AlertSourceController*. The *Controller* is responsible for starting the platform with all other controllers, loading the *Correlation Modules*, and initializing the *In-Memory Alert Storage* and the *Index Tables*. The *AlertSourceController* provides the interface to the source storage, e.g., a row or column-oriented database. By using *Source Adapters* as plugins, the *AlertSourceController* provides an easy and flexible mechanism to connect different types of databases. The *UpdateSourceController* provides the interface to a running IDS management system. The plugin concept of *Alert Receivers* offers the possibility to connect different management systems as well as different IDS sensors directly [23]. The *RequestController* provides the interface to the *CorrelationFrontend*, which is responsible for presenting the correlation results to the user. The *Correlation Modules* implement the different correlation algorithms. Each module is working independently based on a data source, which is either the database provided by the *AlertSourceController*, or the *In-Memory Alert Storage* created at startup. The *In-Memory Alert Storage* can also be disabled if needed, as it is memory consuming. Furthermore, each module can update and read the *Index Tables* to cluster and correlate the alerts.

The platform basically supports two modes of operation: run-time mode and non-runtime mode. In non-runtime mode, the *AlertUpdateController* is disabled and the correlation is done based on the data source, e.g., a row- or column-based database. In runtime mode, the *AlertUpdateController* is enabled and new alerts are processed by the system at runtime. The initial correlation and clustering of the data source is performed as well during the startup of the platform. The *In-Memory Alert Storage* is updated regularly by adding incoming alerts and dropping old alerts. This procedure is necessary as the internal memory is always limited.

The *AlertSourceController* can easily integrate row-oriented as well as column-oriented databases. The *Source Adapters* can use different technologies to connect to databases, e.g., SQL or XQuery. Furthermore, the *AlertSourceController* provides a memory based storage called *In-Memory Alert Storage*. This integrates the different technologies and enables the usage of multiple approaches at once. The column-oriented database and memory-based storage are useful for the analytical operations, such as clustering and correlation. The row-based approach can be useful for IDS management at runtime, as fast inserting of alerts and persistence are major requirements. Additionally, the *AlertSourceController* provides information to the *Correlation Modules* that can be used to establish a dedicated direct connection to the database, which is useful for a distributed architecture.

B. IDS Correlation Platform for Distributed and Parallel Computing

Considering multi-core systems, the proposed architecture can easily be distributed as the major components are independent. There are two major possibilities to distribute the architecture over a multi-core system: sharing processing power or sharing available memory. Figure 2 shows the architecture of these two approaches.

By sharing processing power, the system can perform the correlation very fast as each component has its own processor. Thus, the system would have to use one single data source

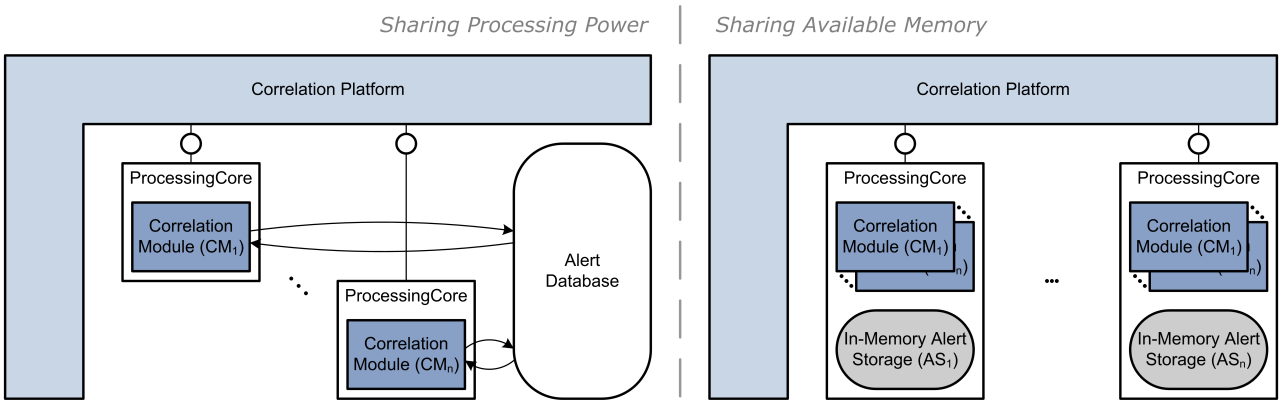

Fig. 2. IDS Correlation Platform for Distributed and Parallel Computing

to do processing, which can limit the approach by available memory space (e.g. in case of using a In-Memory approach). The main controllers (i.e. *AlertSourceController, AlertUpdate-Controller, RequestController*) can work on separated cores. The plugins (i.e. *Alert Receivers* and *Source Adapters*) can also be distributed to multiple cores for processing the alerts. The *Correlation Modules* can easily be distributed to multiple cores as they work independently on a single data source. This approach can be useful in connection with column-oriented databases, which can perform the analytical operations very fast. Figure 2 shows the distribution of the modules CM_1-CM_n to the *ProcessingCores*. The modules are working on a single data source (i.e. an *Alert Database*).

By sharing available memory space, the *In-Memory Alert Storage* can be distributed to the dedicated memory of the processors, which increases the amount of available memory. In this case, the *Correlation Modules* need to run on each core processor as only a part of the data is accessible. The processing based on the *In-Memory Alert Storage* is supposed to be very fast. In this case, an aggregation of the correlation results needs to be performed additionally. Figure 2 shows the distribution of the *In-Memory Alert Storage* AS_1-AS_n to the *ProcessingCores*. The modules CM_1-CM_n are working on each *ProcessingCore* and an integration of the correlation results is needed.

V. Performance Analysis and Discussion

To prove the applicability of our approach, we analyzed the performance of the system and compared the introduced approaches (i.e. In-Memory data storage and column-oriented database) to the row-oriented database approach for clustering and correlation.

A. Performance Analysis

We conducted practical experiments to perform a performance analysis on the different approaches. The analysis has been performed on a system with two *Intel Core(TM)2 Duo* CPUs running on $1.4GHz$ with a cache size of $3,072kB$ each. The system possesses $2GB$ RAM and Solid-State-based hard drive with a size of $128GB$. The running operating system (OS) was a Gentoo Linux. The time and memory consumption are measured by the Eclipse Test & Performance Tools Platform Project (Eclipse TPTP) [24]. The following additional software packages were used for the experiments:

- MySQL version *5.0.70*
- MonetDB Release *Aug2009-SP2*
- Sun Java Development Kit (JDK) version *1.6.0.15*
- Snort version *2.8.3*

We used an alert data set collected by running a Snort[22] IDS sensor connected to the backbone of the university network. The sensor generated $1,391,520$ real alerts in six month of runtime. Based on this data set, we generated three databases: one with $43,485$ alerts (called DB_1), one with $695,760$ alerts (called DB_2), and one with $1,391,520$ alerts (called DB_3). DB_1 and DB_2 are a part of the basic data set and have a chosen size (i.e. exactly $1/16$ and $1/32$ of the original data set). We created these databases based on *MySQL* and *MonetDB* to conduct the experiments. We measured the inserts within the creation process, the clustering, and the correlation based on the improved simple algorithms using a row-oriented database (MySQL), a column-oriented database, and the *In-Memory Alert Storage*. The index tables are used in connection with the *In-Memory Alert Storage* to clearly separate memory-oriented and database-oriented approaches. Table I shows the conducted local experiments.

	Alerts	Insert	Simple Clustering	Aggregated Clustering	Simple Correlation
Row-based DB	43485	x	x	-	x
	695760	x	x	-	x
	1391520	x	x	-	x
Column-based DB	43485	x	x	-	x
	695760	x	x	-	x
	1391520	-	-	-	-
In-Memory DB	43485	x	x	x	x
	695760	x	x	x	x
	1391520	x	x	x	x

TABLE I
Experiments Overview

Table II shows the processing time per alert in milli seconds (ms). A parallel plot visualization of the results is shown in Figure 3. By comparing the results, we conclude that a row-oriented database shows poor performance for clustering and correlation. It handles between $2,034$ and $2,784$ alerts

Results - Correlation Platform

Processing time per alert in ms				
	Alerts	Simple Clustering	Aggregated Clustering	Simple Correlation
Row-based DB	43485	0,3752	-	0,1983
	695760	0,3592	-	0,1939
	1391520	0,4917	-	0,3314
Column-based DB	43485	0,0582	-	0,0204
	695760	0,2121	-	0,0097
	1391520	-	-	-
In-Memory DB	43485	0,0016	0,0002	0,0038
	695760	0,0013	0,0013	0,0014
	1391520	0,0065	0,0065	0,0018

Results - Distributed Correlation Platform

Processing time per alert in ms (1391520 alerts)				
	Processing Units	Simple Clustering	Aggregated Clustering	Simple Correlation
In-Memory DB	1	0,004830	0,000074	0,000899
	8	0,000651	0,000011	0,000151
	16	0,000376	0,000010	0,000102

TABLE II
EXPERIMENT RESULTS

per second for the simple clustering, and between $3,018$ and $5,156$ alerts per second for the simple correlation. However, with approximately $16,000$ alerts per second, the creation of the database is as fast as the creation of an *In-Memory Alert Storage*, which can be important for an IDS management system that needs to insert many alerts in a short time frequently. The column-oriented database shows better performance for correlation and clustering. It handles between $4,714$ and $17,177$ alerts per second for the simple clustering, and between $49,018$ and $102,792$ alerts per second for the simple correlation. With approximately 63 alerts per second, an important problem is the poor performance for database creation, which makes it difficult to use as main alert database for IDS management. The best performance is shown by the *In-Memory Alert Storage*. It handles between $153,188$ and $779,672$ alerts per second for the simple clustering, and between $261,367$ and $725,600$ alerts per second for the simple correlation. By using index hash tables, the aggregated clustering can handle between $153,188$ and $5,288,716$ alerts per second. A major issue is the memory consumption of this approach. It uses 144 MB, 884 MB, and 1.6 GB of memory for an *In-Memory Alert Storage* with the databases DB_1, DB_2, and DB_3.

As the *In-Memory Alert Storage* shows the best results, we investigated the performance of the distributed correlation platform using the *In-Memory Alert Storage*. The analysis has been performed on a cluster of systems with two *Intel Pentium 4* CPUs running on $3.2GHz$ with a cache size of $1,024kB$ each. The system possesses $2GB$ RAM and hard drive (i.e. SATA, $7200rpm$) with a size of $150GB$. The running operating system (OS) was an Ubuntu Linux. The following additional software packages were used for this set of experiments:

- MySQL version *5.0.67*
- Sun Java Development Kit (JDK) version *1.6.0.0*

The results of the experiments with a distributed correlation platform are shown in Table II. The measured time considers the whole data set DB_3 with $1,391,520$ alerts. A single processing unit was working on a similar part of the data set to perform clustering and correlation. i.e., each processing unit has $173,940$ alerts in case of 8 processing units and $86,970$ alerts in case of 16 processing units. In this way, the same amount of memory is consumed in each processing unit and the correlation and clustering can be done much faster. Additional time is necessary to distribute the alerts to the processing units and to merge the results of the processors together. The distribution time is represented by the value *Inserts* in Table II. The merge of the results can be done quite fast by including the result sets of each processing unit with the same result identifier (e.g. clustered value) into a new result set. The merging of the results is included in the measurements shown in Table II. By using this cluster, we can show that the clustering and correlation is up to 12 times faster. The distributed platform can handle up to $2,659,574$ alerts for simple clustering and $9,803,922$ alerts for simple correlation. By using hash tables, we can handle up to $103,092,784$ alerts for aggregated clustering.

B. Discussion

As shown in the experiments (Table II), the proposed platform can provide high performance correlation and clustering of IDS alerts. Compared to the row-oriented database approach used by many existing IDS management systems, a column-oriented storage can improve the performance of analytical operations, such as clustering and correlation. Using the distributed platform in connection with the column-oriented database provides the advantages of a reliable data storage and fast processing of the computations. The memory-based approach improves the performance further and leads to clustering and correlation in real-time. The distributed platform decreases the memory usage of each processing unit and shares the computation tasks fairly. It assigns tasks dynamically and perform complicated calculations in a small amount of time. The number of processing units is flexible and can be adjusted according to the requirements. By using hash tables to store the results of the correlation and clustering, sophisticated algorithms can make use of former results and perform extraordinary fast, even for a large number of alerts that need to be processed (e.g. *Aggregated Clustering* in Table II).

As shown in Table III, the proposed column-oriented database approach is very slow for insert operations (approx-

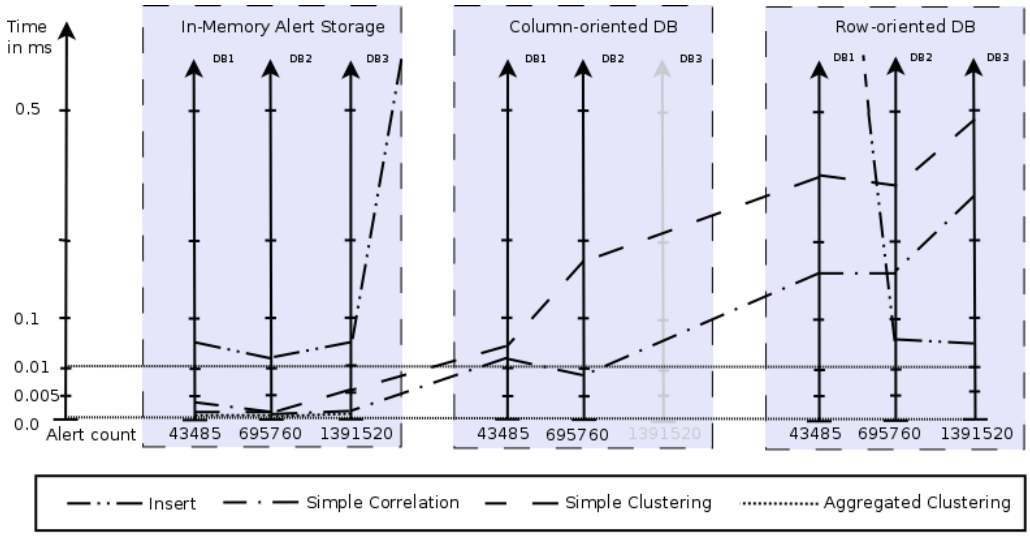

Fig. 3. Experiment Results - Parallel Plot

Results - Correlation Platform

Processing time per alert in ms		
	Alerts	Insert
Row-based DB	43485	1,0800
	695760	0,0606
	1391520	0,0553
Column-based DB	43485	12,4800
	695760	21,6616
	1391520	-
In-Memory DB	43485	0,0520
	695760	0,0320
	1391520	0,0556

Results - Distributed Correlation Platform

Processing time per alert in ms (1391520 alerts)		
	Processing Units	Insert
In-Memory DB	1	0,020715
	8	0,003506
	16	0,001982

TABLE III
EXPERIMENT RESULTS - INSERTS

imately 63 alerts per second), which renders it unfeasible for general IDS management, as frequent inserts of alerts into the database are a standard requirement. To overcome this issue, the column-oriented database can be used for analytical operations only while insert operations are performed on a row-oriented database. Furthermore, the column-oriented database seems to occupy more memory on hard disk than the row-oriented database, e.g., the DB_1 occupies $62,460kB$ using a row-oriented database (MySQL) and $132,416kB$ using a column-oriented database (MonetDB). The implementation of the chosen column-oriented database seems to be unstable at the moment. Unfortunately, it was impossible to create DB_3 based on MonetDB, as the creation always crashed and left an inconsistent database. We are sure this issues can be fixed in future releases of MonetDB. Although the memory-based approach offers very good performance, it comes with a very high memory consumption, i.e., 144 MB, 884 MB, and 1.6 GB of memory for an *In-Memory Alert Storage* with the databases DB_1, DB_2, and DB_3. Thus, the platform can only handle a small number of alerts compared to the database approaches, where a huge number of alerts can easily be handled. The number of alerts can be decreased by defining the time frame of alerts that need to be correlated, e.g., correlation and clustering should only be performed on alert data that is not older than two weeks. Another solution is the usage of the *In-Memory Alert Storage* for recent alert data (e.g. not older than n days) and to use a column-based database for old alert data (e.g. older than n days). There should be a strict time-frame configured to avoid *Out-Of-Memory* problems. Another drawback is the missing persistence of the memory-based storage, as a crash of the correlation platform would lead to missing alerts. The persistence can be improved by storing the alert data redundantly in a database.

VI. FUTURE WORK

The implemented correlation platform was tested with very simple algorithms to show the benefits and reduce the huge impact an algorithm has to the performance of correlation and clustering. As an important next step, we will implement and test sophisticated algorithms for correlation and clustering on our the platform and further verify this approach. The correlation techniques and query optimization methods described in [26] will be used as correlation modules for the platform. The distributed platform was tested on a cluster of 16 processing units. To gain more insight in the efficiency on multi-core systems, we will conduct experiments on large clusters. Based on the multi-core architecture, we will develop and evaluate parallelized algorithms to perform clustering and correlation on huge data sets. In addition to the existing data sets, the system will be tested with large data sets generated in a real network under attack. We expect to work with data sets with more than 50 million alerts. Handling such data

sets does not only require a high-performance approach, but also a sound architecture and implementation. Furthermore, a detailed investigation of the query fingerprint of existing correlation systems and algorithms is considered as one of the next steps. An adjusted algorithm using database queries that exploit the benefits of a column-oriented database is supposed to provide promising results. Finally, the deployment the correlation platform in a practical network to perform correlation and clustering with real world data is planed, which will improve the reliability of the system and proves the applicability of the implemented platform.

VII. Conclusion

To fulfill the challenging task of fast correlation and clustering, storage and processing algorithms of the data set are identified as most important influential factors for the performance. To improve those factors, we propose to use memory-supported algorithms, a column-oriented database and an *In-Memory Alert Storage* for correlation and clustering. To meet the requirements of synchronizing, unifying, and analyzing the high amount of security related events produced by the integrated sensors, an extensible IDS correlation platform is proposed in this paper. Different types of correlation modules can easily be integrated and compared on this platform. The contributions can be summarized as follows:

1) Design and implement memory-supported alert correlation and clustering algorithms by using hash-based index tables
2) Propose to apply advanced database techniques, i.e., column-oriented DB and In-Memory DB, for storing, correlating, and clustering IDS alerts
3) Design and implement an extensible IDS alert correlation platform
4) Conduct several experiments based on our proposed platform for evaluating and comparing the performance of different algorithms and storages

References

[1] R. Sadoddin, A. Ghorbani: *Alert Correlation Survey: Framework and Techniques*, In: Proceedings of the International Conference on Privacy, Security and Trust (PST'06), ACM Press, Markham, Ontario, Canada, pp. 1-10 (2006).

[2] Debar, H., Curry, D., Feinstein, B.: *The Intrusion Detection Message Exchange Format, Internet Draft*, Technical Report, IETF Intrusion Detection Exchange Format Working Group (July 2004).

[3] Mitre Corporation: *Common vulnerabilities and exposures*, CVE Website: *http://cve.mitre.org/*, (Accessed March 2009).

[4] K. Julisch: *Clustering intrusion detection alarms to support root cause analysis*, In: ACM Transactions on Information and System Security, vol. 6, Issue 4, pp. 443-471 (2003).

[5] F. Cuppens: *Managing alerts in a multi-intrusion detection environment*, In: Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC'01), IEEE Press, New-Orleans, USA, pp. 0022 (December 2001).

[6] A. Valdes and K. Skinner: *Probabilistic alert correlation*, In: Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection (RAID'00), London, UK, Springer LNCS 2212, pp.54-68 (2001).

[7] H. Debar and A. Wespi: *Aggregation and correlation of intrusion-detection alerts*, In: Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection (RAID'01), London, UK, Springer LNCS 2212, pp. 85-103 (2001).

[8] P. Ning, Y. Cui, and D. Reeves: *Constructing attack scenarios through correlation of intrusion alerts*, In: Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS'02) ACM Press, Washington, DC, USA, pp. 245-254 (2002).

[9] X. Qin: *A Probabilistic-Based Framework for INFOSEC Alert Correlation*, PhD thesis, Georgia Institute of Technology, 2005.

[10] W. L. Xinzhou Qin: *Statistical causality analysis of infosec alert data*, In: Proceedings of the 6th International Symposium on Recent Advances in Intrusion Detection (RAID'03), London, UK, Springer LNCS 2820, pp. 73-93 (2003).

[11] S. Manganaris, M. Christensen, D. Zerkle, and K. Hermiz: *A data mining analysis of rtid alarms*, In: Computer Networks, vol. 34, Issue 4, pp. 571-577 (2000).

[12] A. Siraj and R. B. Vaughn: *A cognitive model for alert correlation in a distributed environment*, In: Proceedings of IEEE International Conference on Intelligence and Security Informatics (ISI'05), IEEE Press, Atlanta, GA, USA, pp. 218-230 (2005).

[13] P. Ning, D. Xu, C. G. Healey, and R. S. Amant: *Building attack scenarios through integration of complementary alert correlation method*, In: Proceedings of the Network and Distributed System Security Symposium (NDSS'04), The Internet Society, San Diego, California, USA, 2004.

[14] P. A. Porras, M. W. Fong, and A. Valdes: *A mission-impact-based approach to infosec alarm correlation*, In: Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID'02), London, UK, Springer LNCS, pp. 95-114 (2002).

[15] H. Plattner: *A Common Database Approach for OLTP and OLAP Using an In-Memory Column Database*, In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'09), ACM Press, Providence, Rhode Island, USA, pp. 1-2 (2009).

[16] D. J. Abadi, S. Madden, and M. Ferreira: *Integrating Compression and Execution in Column-Oriented Database Systems*, In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'06), ACM Press, Chicago, Illinois, USA, pp. 671-682 (2006).

[17] P. A. Boncz, S. Manegold, and M. L. Kersten: *Database Architecture Optimized for the New Bottleneck: Memory Access*, In: Proceedings of 25th International Conference on Very Large Data Bases (VLDB'99), Edinburgh, Scotland, UK, pp. 54-65 (1999).

[18] MonetDB: WEBSITE: *http://monetdb.cwi.nl/* (accessed Nov 2009).

[19] P. Boncz: *Monet: A Next-Generation DBMS Kernel for Query-Intensive Applications*, PhD Thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands, 2002.

[20] MySQL: WEBSITE: *http://www.mysql.com/* (accessed Nov 2009).

[21] PostgreSQL: WEBSITE: *http://www.postgresql.org/* (accessed Nov 2009).

[22] Snort IDS: WEBSITE: *http://www.snort.org/* (accessed Nov 2009).

[23] Roschke, S., Cheng, F., Meinel, Ch.: *An Extensible and Virtualization-Compatible IDS Management Architecture*, In: Proceedings of 5th International Conference on Information Assurance and Security (IAS'09), IEEE Press, vol. 2, Xi'an, China, pp. 130-134 (August 2009).

[24] Eclipse Test & Performance Tools Platform Project, WEBSITE: *http://www.eclipse.org/tptp/* (accessed Nov 2009).

[25] Tedesco, G. and Aickelin, U.: *Real-Time Alert Correlation with Type Graphs*, In: Proceedings of the 4th international Conference on Information Systems Security (ISS'09), Springer LNCS 5352, Hyderabad, India, pp. 173-187 (2008).

[26] Ning, P. and Xu, D.: *Adapting Query Optimization Techniques for Efficient Intrusion Alert Correlation*, Technical Report, North Carolina State University at Raleigh, 2002.

[27] Northcutt, S., Novak, J.: *Network Intrusion Detection: An Analyst's Handbook*, New Riders Publishing, Thousand Oaks, CA, USA (2002).