A Comparative Study of Correlation Engines for Security Event Management

Conference Paper · April 2015

CITATIONS
8

READS
7,313

5 authors, including:

Luis Rosa
University of Coimbra
15 PUBLICATIONS **133** CITATIONS
SEE PROFILE

Pedro G. Alves
1 PUBLICATION **8** CITATIONS
SEE PROFILE

Tiago J. Cruz
University of Coimbra
89 PUBLICATIONS **511** CITATIONS
SEE PROFILE

Paulo Simoes
University of Coimbra
177 PUBLICATIONS **895** CITATIONS
SEE PROFILE

A Comparative Study of Correlation Engines for Security Event Management

L. Rosa, P. Alves, T. Cruz, P. Simões, E. Monteiro
University of Coimbra, Portugal
{lmrosa, pgalves, tjcruz, psimoes, edmundo}@dei.uc.pt,

Abstract: SIEM (Software Information and Event Management) systems are becoming increasingly commonplace in scenarios as diverse as ICT environments or Critical infrastructures, providing the means to process and analyse multiple distributed sources of information and events, for auditing or security purposes. The main component of its architecture is the correlation engine, which is used to normalize, reduce, filter and aggregate events from a set of heterogeneous inputs. Other modules of SIEM systems include agents for data acquisition; reporting modules for event notification, and storage components for log and auditing purposes.

From a cyber-security standpoint, correlators play a vital role in SIEM architectures, providing the means to infer security information from existing event sources such as security agents or services and device logs. In this perspective, correlator performance is a very relevant matter, as it needs to process large amounts of inputs, while having to provide fast results (i.e. security event notifications). Despite the existence of several correlation engines, there is a scarcity of published work comparing their characteristics and performance, a gap this paper addresses.

This paper presents the concept of SIEM systems and correlation engines, providing a description of their architecture and functional characteristics, with a focus on some of the most popular open source rule-based correlation engines, such as the Simple Event Correlator (SEC), Esper, Nodebrain and Drools. It also provides a comparative performance evaluation of these correlation engines, based on experimental results.

Keywords: SIEM, Event Correlation, Complex Event Processing

1. Introduction

SIEM (Software Information and Event Management) systems are becoming increasingly commonplace both in ICT and Critical Infrastructure Protection (CIP), providing real-time (or soft real-time) analysis of security events collected from an array of sources and probes spread throughout the infrastructure. From an operator point of view, it is not feasible to manually analyse thousands of events and messages coming on a real-time basis and match them against a set of policies and rules. Therefore, the correlation engine, which is the main component of a SIEM system, is used to process those large amounts of inputs. Correlators are also useful to normalize all the sources and reduce the number of events (through deduplication and filtering), from a set of heterogeneous inputs. Other complex role may include a root cause analysis by comparing several events and understand the root of a given problem like a security issue. Using a correlator makes all the process of implementing rules and policies an automated task where only a small number of relevant messages are reported. To the operator, this allows to better understand a given issue and timely take an appropriate action.

In fact, when it comes to security management of infrastructures and services, a SIEM system is not only be a sensible choice, but rather a mandatory component, as demonstrated by several examples such as: the Sarbanes-Oxley Act of 2002, Sec 103 (*Auditing, Quality Control, and Independence Standards and Rules*) which regulates the use of log collection, processing and retention for any traded company in the US (Sarbanes-Oxley Act 2002); the Payment Card Industry (PCI) council requisites for Data Security Standards (as stated by Requirement 10 “Track and monitor all access to network resources and cardholder data”) (Payment Card Industry Data Security Standard version 2.0 2010); the North American Electric Reliability Corporation (NERC) CIP-009-2 Critical Infrastructure Protection (CIP-009-2 2009); or the Information Technology Infrastructure Library (ITIL) framework (Steinberg et al 2011), which encompasses components for incident response in line with the role of the SIEM concept.

However, most SIEM frameworks are provided in the form of generic appliances, software or managed services, requiring customization for particular purposes. Specifically, correlation – the capability of relating multiple events together to detect anomalies, which is the heart of SIEM –

requires fine-tuning to be able to successfully extract meaningful conclusions from several event sources and even from different infrastructure domains. Each log entry or event contains information that the correlator combines together with other instances for joint analysis, as pieces of a puzzle.

The need to diversify event sources to improve security context information creates serious problems for SIEMs, as scalability issues start to hamper performance. The sheer number of events flowing at increasing rates, together with complex correlation rules (such as second-order rules that correlate the output of other rules), increasingly larger event context timeframes and (in certain scenarios) the need to process rules within a limited time window pose a significant burden for the correlator engine, which generally means either a performance degradation over time (implying event loss or high processing latencies) or the need for expensive upgrades.

Once the role of the correlator engine as a critical component in terms of the global SIEM performance is clearly understood, its efficiency and performance characteristics become a concern for end-users and developers alike. However, while various correlators are available in the public domain, with many being incorporated in commercial products, the authors are not aware of any studies comparing them. This paper fills this gap by presenting a comparison of popular correlator engines such as Simple Event Correlator (SEC), Esper (Esper 2014), Nodebrain (Nodebrain 2014) and Drools (Drools 2014). This comparison covers their architectures and feature sets, as well as a comparative performance analysis based on a baseline set of events and correlation rules with different complexity levels.

This paper also provides a comprehensive answer to what correlation engine best suits for a multi-level correlation approach where the correlation boxes should be capable of handling thousands of events from multiple input sources. This document is structured as follows: Section 2, presents a review of several correlation engines, also providing a state of the art analysis on correlation techniques, particularly focused on rule-based techniques. Section 3, provides a comparative study of several correlation engines, describing the testing methodology and test scenarios, also presenting and discussing obtained results. Finally, Section 4 summarizes and concludes the paper.

2. A Survey of Rule-Based Event Correlation Engines

Event correlators are used to process and filter input data from an environment. That data can represent almost any kind of information that can be collected within a system or infrastructure, such as log messages or network trace data. Rule based correlators use a set of rules to represent all the logic and policies of an environment. Each input event, which might undergo a previous normalization stage, is matched against the rules. If there is a match, the event can be reported, suppressed or trigger any pre-defined action.

This section provides an overview of several event correlation tools, focusing on their architectures, operational behaviour, configuration and management, modules and rule format. This constitutes a qualitative analysis that may help distinguishing the engines that are better suited for a specific purpose by understanding their differences, strengths and limitations – going beyond the indicators for rule processing performance, frequently used for such comparisons. The set of correlator engines selected for this comparison is based on a selection of the most representative components for the rule-based correlator category, according to available literature.

2.1 Esper

Esper is a correlation framework capable of analysing streams of events, constituting a complex event processing (CEP) engine (Esper 2014). There is an open source version with a limited number of components and two full-featured commercial versions: Esper Enterprise Edition and EsperHA. In the open source category, two distinct distributions are available: a Java version, evaluated in the scope of this study, and a .NET based version designated by Nesper. The Java based variant is available as a set of Java packages that can be integrated into a full solution using their Java API.

In the literature there are references to the usage of Esper to detect inter-domain stealthy port scans (Aniello et al. 2011), analysing the establishment of TCP connections and applying a rank based

algorithm to classify the scans. Another usage of Esper (Dunkel et al. 2011) refers to event processing of road traffic information produced by a group of sensors such as the average speed, occupancy or number of vehicles to support decisions on a traffic management system.

Esper uses a SQL-like approach for rule description, designated by Event Processing Language (EPL), providing pattern-matching mechanisms via state machines. Esper loads the EPL query set and performs matching within the incoming stream of events, implicitly incorporating a time-based correlation logic that accounts for ordering and sequencing, within a time-window.

Events can be represented in Java structures like Plain Old Java Objects (POJO) or in a more abstract way using XML. The insertion and handling operations need to be developed on top of the Esper library and input /output adapters API's.

2.2 SEC

The Simple Event Correlator, or SEC, it is a lightweight but powerful event correlation engine (SEC 2014). It is written in Perl and distributed under an open source license, with ports for multiple operating system platforms, also being easy to integrate as a component within a full-featured correlation or SIEM framework.

SEC has been used for event reduction within distributed scenarios, helping reduce the amount of information transmitted between log-producing generating agents and the central servers (Myers et al 2011), also implementing different operation modes to define the level and type of information that it passed between the logger systems. SEC has also been integrated within a generic intrusion detection architecture (Ficco and Romano 2011), where it was used to diagnose whether a previously detected attack was successful or did not lead to an intrusion.

SEC rules follow a text-based approach, with matching conditions being defined as string occurrences, regular expressions or Perl subroutines. Input data can be read from text files, named pipes or using the standard output descriptor. Output actions can be defined as the execution of a shell command or the creation of a context to use as matching condition in other rules (therefore allowing second order correlation). SEC sequentially tests all the rules found inside a text configuration file but can perform parallel analysis if the rules spread over multiple files.

2.3 Drools

Drools is constituted by a suite of tools classified as a Business Rules Management System (BRMS) (Drools 2014). The correlation rule engine referred in this survey is part of the complete suite, being designated by Drools Fusion (simply referred as Drools in this paper). It is deployed as a set of Java libraries that can run on multiple operating system environments.

There is a documented use case explaining how to integrate Drools within an architecture to filter and aggregate RFID information in a "smart" hospital environment (Yao et al. 2011), using RFID sensors distributed all over the hospital to collect information about doctors, patients and objects, which is correlated to detect critical time emergencies, based on a set of policies. Another literature reference to Drools describes the use of the complex event processing capabilities to track and trace an object on a large-scale environment, once again using RFID information of a large number of customers spread along different physical locations to trace and follow logistics operations (Wu et al. 2012).

Drools has two operation modes: cloud mode without a time notion; stream modes where events are processed using time notion as they are inserted in the correlation engine. Events are defined as Java classes. Rules are defined in a custom expression-based format that may contain Java code to express the actions to perform. For rule processing, Drools uses an adapted version of the Rete (Liu et al 2010) algorithm, which was designed to sacrifice memory for increased speed.

2.4 Nodebrain

Nodebrain is described by its authors as a lightweight event monitoring agent (Nodebrain 2014). It has the capability to apply a set of rules to a stream of events in order to correlate them. Nodebrain is available under an open source license, also including a set of scripts and a C-based API designed to extend its features.

An example of documented use case describes the use of Nodebrain to control the policy rules of a military *ad hoc* network in a lightweight management system (Jormakka et al. 2008). To configure a remote network element, each element can trigger an action/command in another one, based on a set of policies (Nodebrain rules). The communications between elements are done using TCP connections and either IPv4 or IPv6 protocol.

Nodebrain events are based on text inputs, while the rules use a custom declarative format. It supports several operation modes, including: a daemon mode to continuously execute in the background, a batch mode to process and terminate execution and an interactive mode that can help to debug its behaviour. The input and output options may vary from simple named pipes, secure TCP/IP communications between multiple nodes or custom and specific modules like remote syslog communication. Custom shells commands can be defined as actions.

2.5 Prelude

The Prelude tool is classified as a SIEM system as it already provides additional features, besides the rule engine component (Prelude 2014). Two versions are available: a full featured commercial version and a more limited open source version, based on Python and with substantially lower performance and less features.

There are several use cases for Prelude, mostly related to its use to provide event normalization and correlation for intrusion detection. For instance, (Ficco et al. 2013) propose a Prelude-based distributed architecture for Cloud Computing with multiple instances of correlation engines collecting information at several levels, together with the use of Bayesian networks to detect sequences of unauthorized activities. Another scenario describes the use of Prelude for reduction, normalization and aggregation of events (Petri et al. 2013) produced by different network intrusion detection systems (NIDS) deployed in a university computer network.

The design of Prelude is based on a modular approach, encompassing components such as the correlation engine (Prelude-Correlator) or the log analysis (Prelude-LML) tool. Prelude provides native support to handle and create Intrusion Detection Message Exchange Format (IDMEF) (Debar et al 2007) events. The rules are defined using Python code, also being able to handle text based logs using regular expressions. The Prelude-Manager module provides input via Unix sockets or TCP/IP communications and output via different adaptors like database or text based ones.

2.6 Feature Comparison

When it comes to designate or refer to a correlation tool or system, multiple distinct nomenclatures are found in the literature. In this research the term *correlation engine* refers to a software component capable of processing and correlate events, based on set of rules for both standalone and libraries tools. Correlators are frequently incorporated within larger solutions or frameworks, such as SIEM systems – therefore a standalone tool may not necessarily be the most adequate form for a correlator engine that is designed with integration in mind.

Esper and Drools are both deployed as Java-based libraries, which are platform independent, already providing several internal components and APIs such as input/output adaptors. On the other hand, Nodebrain and SEC are standalone applications developed for GNU/Linux environments, which rely on a scripting-oriented approach to execute actions, handle inter-process communication and integration with other components. The simplistic approach of SEC may be adequate for small environments, or in situations where the correlation overhead must be minimal (for instance, in embedded systems). The Prelude tool provides native support to handle the IDMEF messages. In a

heterogeneous environment, this represents one significant advantage as the events can be represented in a unified format. Nevertheless, it was not part of the remaining study due to deployment issues previously experienced by the authors.

While some tools may be better suited than others for complex scenarios, the fact is that all of them will likely require an additional integration effort to create a capable SIEM. All the described tools provide an open source version, are well documented and are undergoing active development and maintenance, with at least one new version release during the last year.

3. Comparative Assessment Study

This section provides a comparative performance evaluation of four rule-based correlation engines: Esper, SEC, Nodebrain and Drools. Its structure is divided in two parts: the first one describes the tools and components that were used, the use cases for testing and the adopted methodology while the second is dedicated to the presentation and discussion of test results, with special relevance for the performance impact for the number of events and rules, for each correlator engine.

3.1 Testbed, Testing Scenarios and Methodology

The complete assessment scenario includes several components, namely: a set of scripts for pseudo random generation of events (providing the means to generate data in a reproducible way), start / stop test handlers, the correlator engines under evaluation and modules for output filtering and automated report generation. Figure 1 depicts the assessment scenario, also including information about the test stages, tools and interaction.

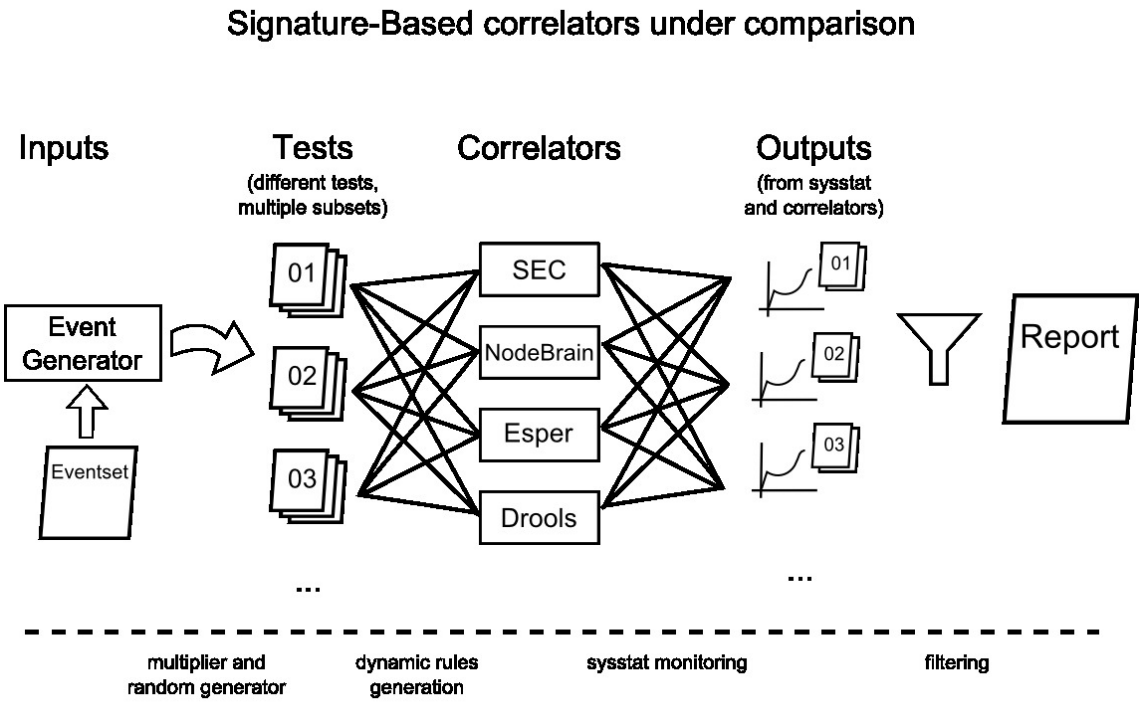

Figure 1: Assessment scenario

Each test starts with an event generation stage, whose purpose is to generate a number of events (published in the syslog format) that are positive (i.e., that trigger a matching rule) or negative (i.e., do not match any rule), following predefined percentage distributions for each test case. After all events are generated, the correlation engine reads the events from storage, processes them and logs each matching event, accordingly to each test. By establishing a predefined number of event sets, it becomes possible to ensure that all correlation engines are accurately (and predictably) evaluated

under the same circumstances. Moreover, all correlator engines were fine tuned in order to streamline input/output operations, minimizing the probability of bottlenecks.

The analysis methodology that was devised encompasses different types of tests, including multiple runs with different combinations of events and rules. Also, the resource consumption overhead for each correlator engine was measured during the event processing performance tests using the sysstat tool (Sysstat 2014), for each test input dataset.

The assessment was conducted in a virtualized environment using an Intel Xeon CPU X5660 processor as host. All correlators were deployed in the same virtual guest machine, assigned with a single core (using core affinity) and 4GB of allocated memory (with a RAM reservation of the same size). The tests were performed in a GNU/Linux system based on the CentOS 6.4 distribution, in series of multiple tests and runs. The report generation module used the gnuplot tool (Gnuplot 2014) to provide graphic outputs for each test. Between each test, a time interval was inserted to avoid interferences in the resource usage and management. The correlation engines assessed were Esper 4.9, Drools 5.5.0, SEC 2.74 and NodeBrain 0.8.15. The commercial versions of Drools and Esper are not covered by this analysis.

The set of tests performed include several combinations of input events and number of rules, described in further detail in the next section. In addition, two categories of rules were assessed: independent rules - where each event matches one rule without any relation with the remaining ones; dependent rules - using a nested approach scheme where one matching is true if the previous rules were verified; other assessments include counting the number of events, keeping some context information in memory.

3.2 Results

The results presented refer to an average of values of three runs per each test. Due to considerable differences in performance, all graphs use a logarithmic scale. Presented values refer to correlation performance in terms of execution time and throughput (i.e., number of events processed per second). In this scope, two major groups of results are represented, namely: for a fixed number of rules (500), with a variable number of input events; for a fixed number of input events (1 million) and a variable number of rules. The adopted test methodology intends to isolate performance measurements, enabling the possibility for separately evaluating the performance penalty for complex rulesets and for an increased number of events.

Table 1, Figure 2 and Figure 3 show the results for execution time and throughput for the event scalability rests, performed with a fixed ruleset of 500 rules and datasets of 1,000, 10,000, 100,000 and 1,000,000 events.

Table 1: Execution time and throughput for the 500 rule tests

	1,000 events		**10,000 events**		**100,000 events**		**1,000,000 events**	
	Throughput (events/sec)	Duratio (sec)	Throughput (events/sec)	Duration (sec)	Throughput (evts/sec)	Duration (sec)	Throughput (evts/sec)	Duration (sec)
Esper	200	5	1,000	10	2,041	49	2,320	431
Drools	125	8	1,111	9	6,250	16	14,286	70
NodeBrain	500	2	3,333	3	3,571	28	3,623	276
SEC	333	3	345	29	346	289	338	2957

The values for test duration (i.e., execution time for each test) were rounded to seconds, while throughput values (i.e. number of events processed per second) are rounded to whole numbers.

CPU and memory overhead was measured during the tests. While these overhead numbers are not detailed in this paper, it was observed that all the correlation engines can effectively use the available CPU processing power. As for memory usage, it was observed that Java-based correlation engines, such as Drools and Esper, use more memory, comparatively to the other alternatives.

Figure 2: Execution times for the 500 rule tests

All correlation engines were able to handle 500 independent rules with ease, for all test cases. For a small number of events, all correlation engines finished under 10 seconds (this value, which includes initialization times and some load operations does not express anything significant about their relative processing capabilities). There is a expected increase in terms of the test execution times which is directly related with the number of events being processed.

Also, the performance differences between different correlator engines begin to grow apart with the increase in the number of processed events. For instance, while Nodebrain initially shows a significant improvement over SEC, its performance suffers an inflection after a certain point, showing a stabilization trend that hints at the existence of scalability issues. Similarly, Esper seems to suffer from the same symptoms, starting to slow down after the test with 100,000 events.

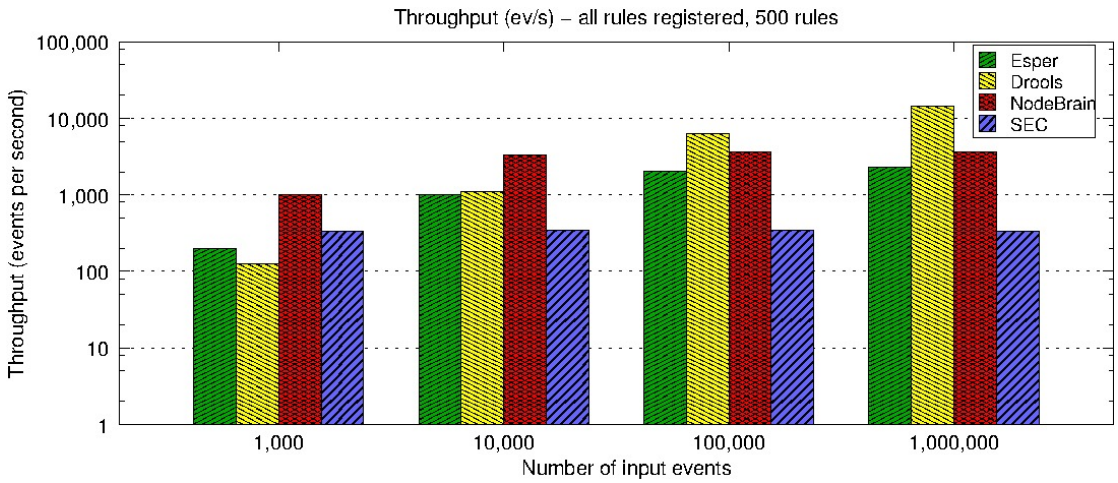

Figure 3: Number of events per second for the 500 rule tests

The SEC throughput remains constant in the remaining tests even for a larger number of input events, indicating that, for 500 rules, it reached the peak of its processing capabilities. Only Drools

stands out in terms of scalability for this test, providing a throughput of 14,000 events per second, for a 1,000,000 event dataset. Also, it is interesting to note that, apart from SEC, Java-based engines were consistently the slowest among the group.

Next, Table 2, Figure 4 and Figure 5 show the results for execution time and throughput for the rule scalability tests, performed with a fixed number of 1,000,000 input events, matched against rulesets of 20, 200 and 500 rules.

Table 2: Execution time and throughput for the tests with 1,000,000 events

	20 rules		**200 rules**		**500 rules**	
	Throughput (events/sec)	Duration (sec)	Throughput (events/sec)	Duration (sec)	Throughput (events/sec)	Duration (sec)
Esper	38,461	26	5,780	173	2,320	431
Drools	21,272	47	14,925	67	14,286	70
NodeBrain	6,369	157	4,329	231	3,623	276
SEC	4,405	227	812	1232	338	2,957

As expected, all tools suffer a degradation of performance with a larger number of rules, albeit in a different manner. Also, and similarly to previous tests, there is not a single correlator engine that is consistently superior in all tests.

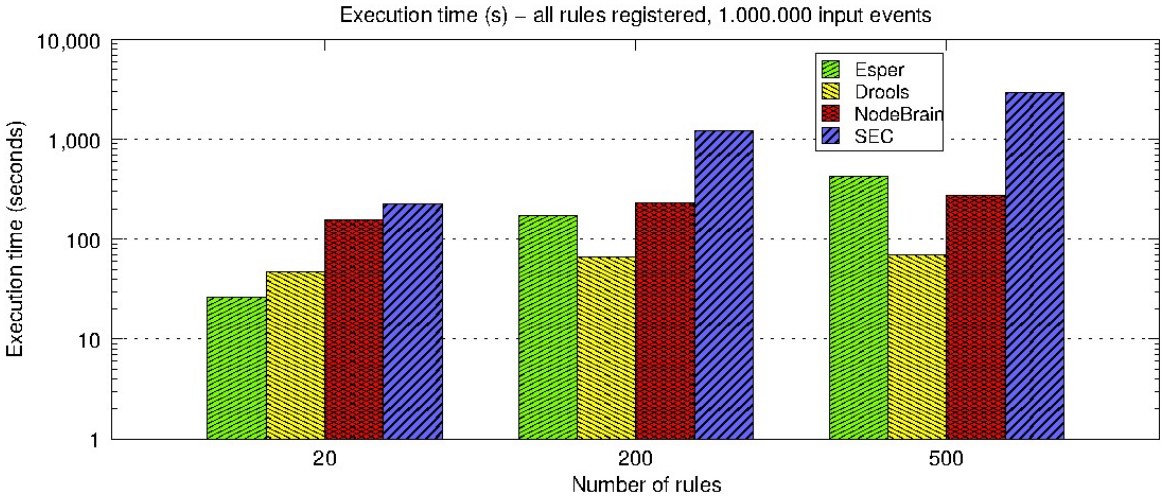

Figure 4: Execution time for the tests with 1,000.000 events

While Esper provides the best performance for a small ruleset, it loses its advantage over the remaining engines as the number of rules increase, up to the point of finishing in the penultimate position, for the 500 rules test case. Nodebrain shows a less steep penalty (almost linear) as the rule size increases, with SEC showing the worst results across the test range. Drools once again shows the best (and more consistent) results for the worst case scenario of 500 rules, with a throughput of 14,286 rules per second.

It is interesting to note that Drools and Nodebrain presented a lower variation rate across the test set, possibly hinting at an inflection point over the 500 rule threshold – present results are not conclusive about this hypothesis, therefore requiring further testing.

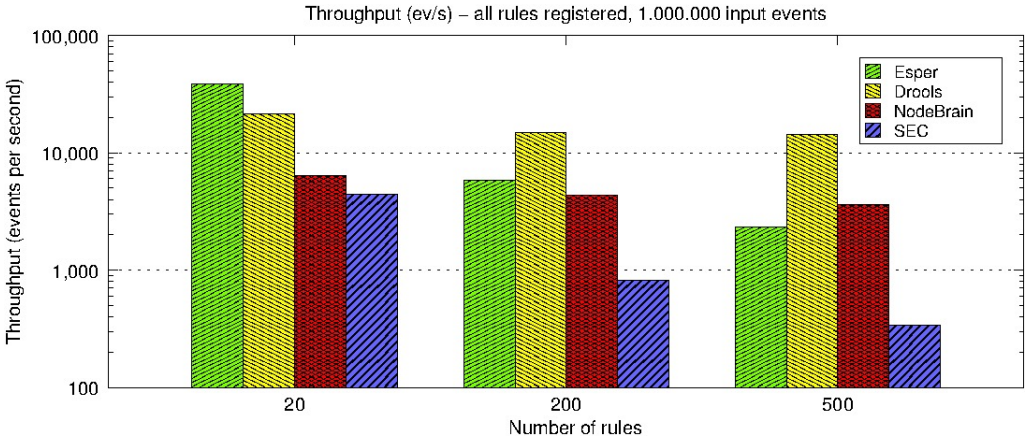

Figure 5: Number of events per second using 1,000,000 events

4. Conclusion

The correlation engines compared in the scope of this analysis are generic tools, designed to perform event correlation, filtering and/or aggregation of multiple events, not constituting complete SIEM solutions for event management. To achieve a complete SIEM solution build around a correlation engine requires some additional effort, such as the addition of input and output components and storage modules, the introduction of support for dynamic configuration or the development of APIs to ease adaptation and integration.

For the scope of this research some of the most popular open source, well supported and documented correlation engines were chosen. The results hereby presented are based on simple correlation scenarios devised to provide performance baseline indicators for comparison, based on rule matching and reporting without additional or parallel processing. Despite the effort to reproduce equivalent scenarios in each tool, there are differences in terms of rule syntax and semantics or advanced fine tuning parameters that may prove determinant, leading to different results.

As for the comparison study, it must be said that if the sole criteria was raw performance Drools would be considered the best correlation engine, for several reasons: its consistent behaviour and superior performance in the most demanding test cases. Similarly, SEC would be considered the worst correlator, for the opposite reasons, suffering from serious scalability issues in complex scenarios. Nodebrain obtained some positive results, specifically for complex scenarios with a large number of events and/or rules. Esper also obtained some positive results, specifically for a range of 200 rules that could represent a medium/large scenario.

Nevertheless, there are other aspects besides performance that may prove determinant for a correlator. For instance, despite its humble performance, SEC might suit some simpler scenarios, like local correlation or filtering at host level using a small number of rules, thanks to is simplistic architecture and configuration – also, the lightweight nature of the SEC makes it adequate for embedded systems or low-power platforms. Moreover, Drools was not able to finish some experiments, suffering from memory management issues. As for Nodebrain, and despite its good documentation, the authors consider its rule syntax to be somewhat cryptic and unintuitive, likely demanding a longer learning curve. Finally, Esper was considered as a robust and adequate compromise between performance, configuration flexibility and easiness of setup – for instance, in the CockpitCI project (CockpitCI 2014), a multi-layered intrusion detection solution was implemented using multiple Esper instances.

5. Acknowledgements

This work was partially funded by project CockpitCI – Cybersecurity on SCADA: risk prediction, analysis and reaction tools for Critical Infrastructures (FP7-SEC-2011-1 Project 285647) and project iCIS (Intelligent Computing in the Internet of Services, CENTRO-07-0224-FEDER-002003).

References

Aniello, Leonardo and Lodi, Giorgia and Baldoni, Roberto. (2011) “Inter-domain stealthy port scan detection through complex event processing”, Proceedings of the 13th European Workshop on Dependable Computing, pp 67-72.

CIP-009-2. Available at: http://www.nerc.com/files/CIP-009-2.pdf [Accessed 21 Oct. 2014].

CockpitCI. Available at: http://www.cockpitci.eu/ [Accessed 21 Oct. 2014].

Debar, Herve and David A. Curry and Benjamin S. Feinstein. (2007) "The intrusion detection message exchange format (IDMEF)”.

Drools, Business Rules Management System (BRMS) solution. Available from: <http://www.drools.org/>. [21 October 2014].

Dunkel, Jürgen and Fernández, Alberto and Ortiz, Rubén and Ossowski, Sascha. (2011) “Event-driven architecture for decision support in traffic management systems”, Expert Systems with Applications, Elsevier, Vol 38, No. 6, pp 6530-6539.Esper - Complex Event Processing. Available from: <http://esper.codehaus.org/>. [21 October 2014].

Esper, Event Processing with Esper and NEsper. Available from: <http://esper.codehaus.org/>. [21 October 2014]

Ficco, M. and Romano, L. (2011) “A generic intrusion detection and diagnoser system based on complex event processing”, First Int. Conf. on Data Compression, Communications and Processing (CCP), pp 275—284.

Ficco, M. and Tasquier, L. and Aversa, R. (2013) “Intrusion detection in cloud computing”, Eighth Int. Conf. on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), IEEE, pp 276-283.

Gnuplot. Available at: http://www.gnuplot.info/ [Accessed 21 Oct. 2014].

Jormakka, J. and Jormakka, H. and Väre, J. (2008) ”A lightweight management system for a military ad hoc network”, Information Networking. Towards Ubiquitous Networking and Services, Springer, pp 533-543.

Liu, Di and Gu, Tau, and Jiang-Ping Xue. (2010) "Rule Engine based on improvement Rete algorithm." Apperceiving Computing and Intelligence Analysis (ICACIA), International Conference, IEEE, pp 346-349.

Myers, Justin and Grimaila, Michael R. and Mills, Robert F. (2011) “Log-based distributed security event detection using simple event correlator”, 44th Hawaii International Conference on System Sciences (HICSS), pp 1-7.

NodeBrain, Open Source Project. Available from: <http://nodebrain.sourceforge.net/>. [21 Oct. 2014].

Payment Card Industry Data Security Standard version 2.0. Available from: <https://www.pcisecuritystandards.org/documents/PCI%20SSC%20Quick%20Reference%20Guide.pdf> [21 October. 2014].

Petri, Giani and Nunes, Raul C and Lopez, Victor LO and Junior, Tarcisio C and dos Santos, Osmar M. (2013) “KBAM: Data Model of a Knowledge Base for Monitoring Attacks”, LADC.

Prelude, Prelude OSS Edition. Available from: <https://www.prelude-ids.org/>. [21 October 2014].

Sarbanes-Oxley Act. Available at: https://www.sec.gov/about/laws/soa2002.pdf [21 October 2014].

SEC, Simple Event Correlator. Available from: <http://simple-evcorr.sourceforge.net/>. [Oct. 2014].

Steinberg, R., Rudd, C., Lacy, S. and Hanna, A. (2011). “ITIL service operation. London: TSO”. Sysstat. Available at: http://sebastien.godard.pagesperso-orange.fr/ [Accessed 21 Oct. 2014].

Wu, Yanbo and Sheng, Quan Z and Ranasinghe, Damith and Yao, Lina. (2012) “PeerTrack: a platform for tracking and tracing objects in large-scale traceability networks”, Proceedings of the 15th International Conference on Extending Database Technology, pp 586-589.

Yao, Wen and Chu, Chao-Hsien and Li, Zang. (2011) “Leveraging complex event processing for smart hospitals using RFID”, Journal of Network and Computer Applications, Elsevier, Vol. 34, No. 3, pp 799-810.