The 8th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications
24-26 September 2015, Warsaw, Poland

Intelligent monitoring with background knowledge

Prof. Dr. Kai-Oliver Detken[1] , Prof. Dr. Stefan Edelkamp[3] , Dr. Carsten Elfers[2] ,
Malte Humann[3] , Thomas Rix[1]
[1] DECOIT, Fahrenheitstraße 9, 28359 Bremen {detken,trix}@decoit.de
[2] neusta GmbH, Konsul-Smidt-Str. 24, 28217 Bremen, c.elfers@neusta.de
[3] University of Bremen, Am Fallturm 1, 28205 Bremen, {edelkamp,mhumann}@tzi.de

***Abstract* – This paper describes the design and implementation of an intelligent monitoring system, that runs advanced inference mechanisms to correlate events from various sensors. Different to existing monitoring approaches, it exploits taxonomic background knowledge in form of ontological information to draw refined inferences. The monitoring system provides abstract knowledge exchange capabilities between different monitoring clients to support users during the setup and maintenance process. The system supports a variety of sensors and collectors, also including new sensors that can be mapped to the system conveniently.**

***Keywords* – IT security, monitoring, artificial intelligence (AI), security incident and event management (SIEM), Icinga (Nagios), event correlation, anomaly detection**

I. Introduction

Monitoring a computer network is an important but tedious task for a system or security administrator to avoid severe consequences like network downtime, or data theft and loss. Even if there are some good sensors available that might be present in the network, given growing infrastructures in companies to monitor, it becomes more and more labor-intensive to judge and analyze, how dangerous reported incidents are. Intelligent correlation of detected events and tool support that suggests various kinds of counter-measures are required.

In this paper we consider concept and implementation of an intelligent monitoring system, which serves as backbone for security incident and event management (SIEM). Several sensors and data collectors have already been attached to the system, including automated anomaly detection based on time series analysis. Tool support for quick integration of new sensors is offered.

First, design options and architecture of the intelligent monitoring system are described. Next, algorithmic details of the advanced correlation are considered by taking hierarchical information into account to allow for its tolerant pattern matching. An outlook is given on how computer infrastructure scanning tools can be integrated into the system providing background information for further improved inference. Finally, possible forms of interactivity between the integrated web-based graphical user interface and the administrator are highlighted.

II. Architecture of the System

IT-asset monitoring has become an essential part in all enterprise IT infrastructures. Icinga [1] is widely used in order to fulfill this task. It allows usage of a wide spectrum of sensors such as Snort, nmap or OpenVAS and alarms the administrators in case of an incident. However, this tool has its limitations when it comes to the complexity of incident analysis. This problem can be well addressed by the use of an ontology based correlation as suggested in this paper. Thus additional advantages arise, which include detecting incident variations or exchanging incident detection patterns between different clients while considering the protection of privacy. That way the wheel does not need to be reinvented every time IT assets are monitored but the administrator is up-to-date with knowledge of detecting and handling incidents simply by downloading shared correlation rules from a server that can be used ad-hoc in the local system.

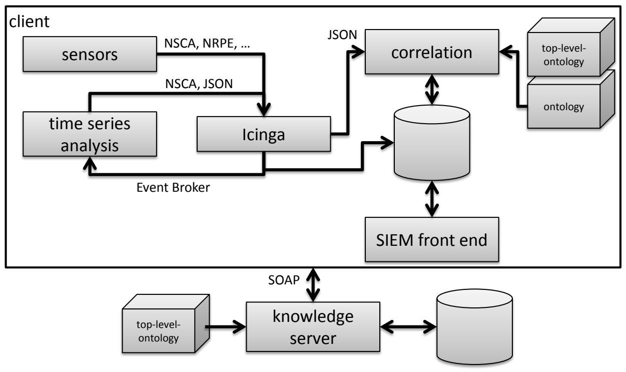

Figure 1. iMonitor architecture overview

Fig. 1 shows an overview of the iMonitor system. Since the system is based on Icinga, all the Icinga check plugins can be used and are gathered under the term *sensors*. Additionally, the input for Icinga is complemented by a time series analysis which detects anomalies from different data sources or directly from Icinga data. An event handler reports the events received by Icinga to the correlation using the JSON data exchange format. The event handler approach was chosen because it allows to keep Icinga 'as is' without the need for changes of its core implementation. The communication over JSON is easy to configure in Icinga and allows a loose complement to an existing Icinga system. The correlation requires a database for fetching rules and an ontology for getting background knowledge which An additional front-end has been implemented to make maximum use of the Icinga data in conjunction with the advanced correlation results. The front-end adds some missing SIEM features such as specifying correlation rules and visualizing incidents. The client setup has access to an external knowledge server using SOAP requests to exchange knowledge in form of rules. Beyond this setup exist several tools, e.g. for semi-automatically integrating new sensors, importing Icinga asset config files to the ontology or the automatic discovery of IT-infrastructure [2].

III. Knowledge representation

One advantage of using ontology based correlation is the flexibility of background-knowledge that can be used to detect incidents. The background-knowledge is well structured by the T-Box part of an ontology defining a common language for all objects that are needed for the correlation. Such objects are called concepts. They may include the possible events fired by the sensors, types of IT assets like servers, workstations or mobile devices or information about different software products and their vulnerabilities. In contrast, enterprise or network specific information can be stored in the A-Box part of an ontology which naturally supports to distinguish which knowledge can be exchanged and which should not be exchanged since it contains individual or private data.

Exchangeable data comprises of correlation conditions for detecting incidents, explanations of incidents and recommendations how to resolve them. In combination this information is called a rule because it defines the conditions for its own trigger and the following action by issuing explanations and recommendations. Since ontologies are used to represent the background-knowledge, SPARQL [3] query language for ontologies can be used to specify rules. To allow a correlation with the events fired by the sensors, specific placeholders should be inserted into the SPARQL query. These placeholders start with a dollar sign followed by the name of a variable from the event. For example, Icinga may transmit an event to the correlation with the *host state DOWN* and the *host address 192.168.178.4*. The correlation looks these values up in the ontology and replaces the placeholders *$hoststate* and *$hostaddress* in the SPARQL query with the appropriate concepts or individuals. This allows, for example, to check if an event indicates that a host is down and if some processes depend on that host. Furthermore, it may be checked if these processes are critical for some customers or other business processes. Beside such conditional requests, the correlation allows SPARQL SELECT queries to gather useful information from the ontology, which is used for generating explanations and recommendations. For example, the list of affected customers can be used in the recommended action, e.g. by suggesting that the affected customers need to be informed pro-actively.

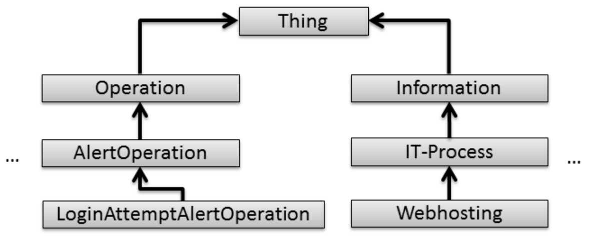

Figure 2. Excerpt from the ontology structure

To allow time based correlation, e.g. to check if specific events such as a failed login occurred several times within the last ten minutes, the ontology is structured according to the approach of Granadillo et al. [4] on the top level of the ontology. The approach suggests to separate the ontology into two parts, the (static) information and the (dynamic) operations as shown in Fig. 2. While the information part of the ontology contains background-knowledge such as asset information, the operations part holds information that may be included and used by rules. This has the advantage to easily access both kinds of information (dynamic/static) with the expressiveness of SPARQL and to use this information for the rule trigger conditions, as well as the generation of explanations and recommendations.

IV. Knowledge exchange process

iMonitor uses a centralized approach for exchanging knowledge. Therefore, the data first needs to be committed to a central server (called *knowledge server*) prior to being downloaded by any client. Before the rule is committed to the server, the committing client checks that the rule only uses concepts from the top-level ontology, i.e. the T-Box part that is used by all clients. This avoids that private data is accidentally committed to the server. The *knowledge server* checks again if no individual knowledge has been conveyed to guarantee that the rule can be integrated into each client using the same top-level ontology. This process ensures protection of sensitive data but lacks a check for nonsense rules. In order to avoid that nonsense rules are made available for download and integration, there is a quality assurance process in the *knowledge server*. When a new rule arrives at the server it is marked as *new*. These new rules need to be verified by a neutral moderator who accepts the given rule. Only after acceptance by the moderator, the rule can be downloaded by the clients. The following items sketch a typical sequence for exchanging knowledge:

- The user selects a rule to commit on the client side.
- The client automatically validates that no individual knowledge is shared.
- The client transmits the knowledge to the knowledge server by the use of a web service.
- The knowledge server validates that no individual knowledge is in the given rule.
- The new rule is sent to an internal pool of new rules.
- A moderator accepts the rule, which then becomes public.
- Another client can access the rule and integrate it into its local rule repository.

This scenario describes a global knowledge server; however, if an enterprise wants to exchange knowledge only within the enterprise, e.g. among different locations, the enterprise may host its own knowledge server. Beyond this, the global knowledge server may be used with appropriately configured groups and rights. An hierarchical group structure should be set up in which each specific group can download and integrate all rules from abstract groups. Enterprise A should be able to download all global rules; however, not all other enterprises should be able to download rules from enterprise A without its authorization. Therefore, the client needs to be able to decide to which group the rule is committed.

V. Tolerant pattern matching

Basically, the user of the monitoring software is responsible for managing the rules or the patterns that the system uses for detecting incidents. It is a difficult task for the user to consider all possible situations and to find appropriate rules for them. Most systems handle missing rules by simply suppressing the generated events. However, this impedes to find missing patterns. Therefore, a different approach is suggested: The iMonitor system helps the user by checking variations of known rules. A soft or tolerant pattern matching approach is used to detect these variations in case no rule matches but an event has occurred. This approach is based on the idea of abstracting known patterns as suggested in [5] and [6]. In contrast to the mentioned approaches, the rule definitions are specifically annotated by SPARQL functions to allow abstraction and specify how the abstraction is performed. The main function is the *abstract* function with three parameters. The first parameter is the original concept for the condition in the ontology, e.g. *Webhosting* from Fig. 2. The second parameter is the maximum abstraction of the concept, e.g. *IT-Process*. The third parameter is a custom name to reference the abstraction, for example, to later use it in a recommendation. In this example, there is only one rule which requires that a *Webhosting* process is affected. Furthermore, the event received does not affect a *Webhosting* process, however, it affects other IT processs, e.g. an e-mail process. Given that an event occurred, something must have happened but the correlation does not know how to handle the event. The correlation automatically replaces the *Webhosting* conditionwith the next more general concept, in this case the *IT-Process*, which leads to a matching condition. This abstraction is made successively to ensure that the abstracted pattern or rule is as specific as possible. At that time the user knows that an event has occurred and is not directly handled by the specified rules. Additionally, he is informed about which most similar rules match (by abstracting them). This helps the user to assess the situation by looking for similar situations in explanations and recommendations, and to adapt correlation rules if necessary. The other function is *useAbstraction* which references the name parameter of the *abstract* function to replace concepts in the explanation or recommendation by the abstracted concept. This is required to adapt explanation and recommendation according to changed/abstracted conditions.

VI. Integration of sensors

To integrate new sensors, each correlation requires mapping of vendor specific event information to a common language used by the correlation. This mapping can be modeled manually into the ontology or may already be given by the top-level-ontology; however, it limits the user in integrating its own individual sensors. A tool called sensor mapper was developed to support the user. It reads the possible outcomes of a sensor from a file and compares it to known elements from the ontology. The user can optionally specify a top concept for the comparison to avoid that the tool tries to match all elements in the ontology. For example, it does not make sense to match a Snort event outcome with an IT-Asset like a web server as it is done with Trojan activity. Furthermore, the user needs to specify which variable of the event he wants to map, e.g. event text, host name or host state. The tool compares the elements by matching their names using the Jaro-Winkler [7] distance and generates a CSV output file with the best matches. Firstly, the user can revise the mapping and secondly, import it in the correlation which allows the correlation to 'understand' the new sensor.

VII. Anomaly Detection

Sensors may report all sorts of data to Icinga, including general host and service checks via SNMP for example, which do not necessarily assess if certain information implies an incident. A time series based anomaly detection was developed to further add to the correlation. It is capable of finding anomalies in generic time series which consist of numerical data for different points in time, e.g. information included in performance data provided by Icinga plugins.

One objective of the approach was to keep the user's configuration requirements to a minimum. A similar approach, Brutlag's Aberrant Behavior Detection [8], relies on various smoothing parameters as well as on the length of a season to be used in the underlying exponential smoothing process. In this paper's approach only the connection with Icinga has to be set up. Any further information about possible anomalies is learned from the incoming data itself. Based on those data the anomaly detection estimates a repeatable pattern for each time series, if present, and uses this pattern to predict the latest measurement. The prediction is then compared to the actual value and rated anomalous or normal.

A slightly modified version of the auto period approach by Vlachos et al. [9] is used to estimate patterns with an additional preprocessing step. Since the original measurements may already include anomalous entries, statistical outliers are removed by bounding them to an appropriate level. Furthermore, the trend is removed from the time series by subtracting the trend determined via STL STL [10]. Following Vlachos et al. the periodogram and circular autocorrelation of the time series are calculated to select potential pattern length candidates, validate and refine them. Based on this estimated pattern length all available data from the time series is combined into a single pattern. The pattern entry is taken. It corresponds to the time stamp of the latest measurement while allowing for a short lag in time. The trend that was removed beforehand is added again to create a prediction. Given that the predicted value rarely matches the exact measurement, an interval is defined, in which the measured value is not considered an anomaly. Since a single fluctuation might be caused by noise, we apply the strategy already used by Brutlag [8] and demand multiple anomalous entries in a short period of time to report an incident to Icinga. Single anomalies can still be reported to Icinga as warnings to support plugins that gain advantage of those warnings.

VIII. Graphical User Interface

A web-based framework was selected for the intelligent monitoring system to support wide and remote applicability of our system. The architecture of the SIEM-GUI consists of back-end kernel and front-end mainly. The back-end manages communication to ticket system, user management, security incidents, and event database. These are external systems for the SIEM-GUI software. The web-interface is responsible for the visualization of events, incidents, and tasks. Additionally, the SIEM-GUI has to handle user input. As connection between both modules a web socket is used. This socket is able to manage bi-directional communication between both components. Fig. 3 shows the architecture of the SIEM-GUI as overview.

The back-end has been written in Java with the framework Spring. It works as endpoint for the provision of web-socket endpoints and uses STOMP as message format for the communication via web-sockets. Along with the security module it can be used for the implementation of authentication and authorization within the application.

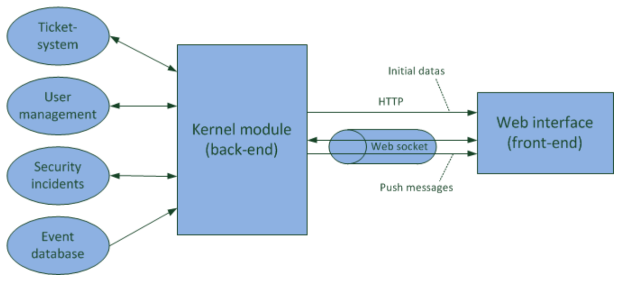

Figure 3. Architecture of the SIEM-GUI

The main issue of the back-end is to establish different connections to the external services for read and change data of the front-end. Furthermore, these data have to be edited in a common format. External services are systems, which are very important for the functionality of a SIEM system but can be used as independent systems, too. They deliver data and provide the SIEM-GUI with their functionality. The used services for iMonitor are:

- user management
- event database
- correlation
- ticket-system

The interfaces of these services are well defined, which is why an efficient adaptation to the SIEM-GUI is possible. The effort to integrate these services is only determined by implementing specific classes like DAO, converter, or filter. The remaining part of the application works independently from specific services.

The front-end was implemented as a single page app in JavaScript with the framework AngularJS. Because of using single page app the website does not have to be reloaded after first selection. If another view has to be loaded, the content of the current view is rejected by the help of JavaScript and the new content is inserted. The framework AngularJS is prepared to handle this by using modules, which provide this functionality. In addition, a basic layout was developed and is provided via the web server. This layout initializes the AngularJS application and includes an area in which the template of a view may be inserted. This template is dynamically loaded from the server as soon as needed. The basic layout is not loaded again.

IX. Experimental Results

In first experiments the functionality of the correlation was tested in use-cases. One use-case showed that a host down event, which affects critical processes, is detected by correlation and that additional queries are correctly evaluated for recommendations and explanations, e.g. when determining the affected customers. With respect to correlation performance, correlation could handle 406 events per second with one rule and 187 events per second with five rules. The requested time by a rule to be processed, of course, depends on the complexity of rule and background knowledge. So far, the performance is sufficient since Icinga can be used for pre-filtering the events from the sensors. However, with an increasing complexity or in huge infrastructures, the correlation may reach its limit. Therefore, the correlation was parallelized and it was achieved to process 535 events per second with five rules on an AMD Phenom II X6 1055T.

Additionally, several different SIEM systems were tested in order to be compared to the solution developed in the iMonitor project:

- OSSIM
- LogApp
- ArcSight
- LogRhythm

All SIEM systems are mainly based on pattern-detection and do not analyze all available data of the correlated content. That means, if a malware pattern does not exist, the attacking process cannot be recognized. Additionally, even if all log-data are available in a database, the correlation will not work with all information. Only the SIEM system LogRhythm collects a complete set of data from across the entire IT environment of an enterprise and processes as well as analyzes most relevant information from multiple dimensions. The Advanced Intelligence (AI) Engine by LogRhythm performs correlation and behavioral analytics on machine data throughout an enterprise's IT environment. It identifies and alerts on devices, hosts, applications and users, which have been targeted and/or successfully impacted, so that administrators can take immediate action. By utilizing contextual information such as vulnerability data along with other disparate machine data, this SIEM system is used to help correlate and alert on security events and incidents that have not yet happened but have the potential to occur. Therefore, LogRhythm's SIEM system works similar to the iMonitor approach and thus can be compared to it.

The next experiment tested the number of events the iMonitor system is capable of processing and its according reactions. The test environment included 80 clients, server, phones etcetera and the system was tested for four days (Friday - Monday). For this experiment only the snort sensor was activated because this sensor had generated the most events during previous tests. The experiment started at 10 A.M. on Friday and ended at 4 P.M. on Monday. This period was chosen to gain representative statistics for both work days and the weekend. The system had collected a total number of 2.495.041 events during these four days. Figure 4 tells us that SNMP and ICMP were the event types that occurred most. This can be explained by the fact that the active monitoring system uses SNMP and ICMP to receive the status of the monitored systems. The amount of the events can be reduced by adjusting the snort configuration. Nevertheless, the iMonitor system was able to handle the number of data without failure.

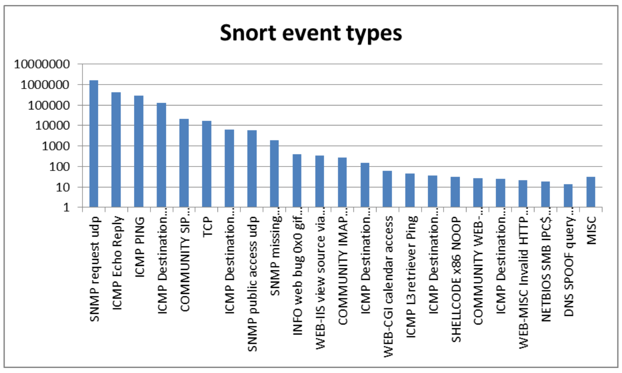

Figure 4. Distribution of Snort events among event types

X. Conclusion and Outlook

This novel papers' approach improves monitoring to reduce the work load of system and security administrators generated by the maintenance of their computer infrastructure. To assess the health state of the system, the approach is able to correlate and condense events that are triggered by different sensor sources as well as to learn and generalize the knowledge that comes from particular observations. Expert knowledge is included in form of tolerant inference pattern matching rules, from which most-likely hypotheses are generated. Beyond this, a machine learning anomaly detection processes the data for detecting unknown incidents and handling them during the correlation process.

For the future, we plan to integrate an automated approach for the tight integration of an intelligent network scanner that exceeds the functionality of known tools like nmap. The output of the scanner should be provided in form of an ontology to our reasoning system.

Acknowledgment

The authors give thanks to the BMWi-ZIM[1] for the financial support as well as all other partners involved in the research project iMonitor for their great collaboration. The project consists of the industrial partners DECOIT GmbH, neusta software development, and the research partner University of Bremen represented by the institute TZI – Center for Computing and Communication Technologies.

[1]Federal Ministry for Economic Affairs and Energy, "Central Innovation Program SME: http://www.zim-bmwi.de"

References

[1] Icinga, “Icinga - open source monitoring,” Feb. 2015, http://www.icinga.org/.

[2] H. Birkholz, I. Sieverdingbeck, K. Sohr, and C. Bormann, “IO: an interconnected asset ontology in support of risk management processes,” in *Seventh International Conference on Availability, Reliability and Security, Prague, ARES 2012, Czech Republic, August 20-24, 2012*. IEEE Computer Society, 2012, pp. 534–541. [Online]. Available: http://dx.doi.org/10.1109/ARES.2012.73

[3] A. S. Steve Harris and E. Prud’hommeaux, “Sparql 1.1 query language,” Feb. 2015, http://www.w3.org/TR/sparql11-query/.

[4] G. G. Granadillo, Y. B. Mustapha, N. Hachem, and H. Debar, “An ontology-driven approach to model siem information and operations using the swrl formalism,” *Int. J. Electron. Secur. Digit. Forensic*, vol. 4, no. 2/3, pp. 104–123, Aug. 2012.

[5] C. Elfers, S. Edelkamp, and O. Herzog, “Efficient tolerant pattern matching with constraint abstractions in description logic,” in *International Conference on Agents and Artificial Intelligence (ICAART)*, 2012, pp. 256–261.

[6] Y. He, W. Chen, M. Yang, and W. Peng, “Ontology based cooperative intrusion detection system,” in *Network and Parallel Computing*, ser. Lecture Notes in Computer Science, H. Jin, G. Gao, Z. Xu, and H. Chen, Eds. Springer Berlin Heidelberg, 2004, vol. 3222, pp. 419–426.

[7] W. E. Winkler, “The state of record linkage and current research problems,” in *Statistical Research Division, US Census Bureau*, 1999.

[8] J. D. Brutlag, “Aberrant behavior detection in time series for network monitoring,” in *Proceedings of the 14th USENIX Conference on System Administration*, ser. LISA ’00. Berkeley, CA, USA: USENIX Association, 2000, pp. 139–146.

[9] M. Vlachos, P. Yu, and V. Castelli, “On periodicity detection and structural periodic similarity,” in *Proceedings of the 2005 SIAM International Conference on Data Mining*, 2005, pp. 449–460.

[10] R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning, “STL: A seasonal-trend decomposition procedure based on loess,” *Journal of Official Statistics*, vol. 6, no. 1, pp. 3–73, 1990.