Normalizing Security Events with a Hierarchical Knowledge Base

David Jaeger, Amir Azodi, Feng Cheng, Christoph Meinel

Normalizing Security Events with a Hierarchical Knowledge Base

David Jaeger, Amir Azodi, Feng Cheng, and Christoph Meinel

Hasso Plattner Institute, University of Potsdam, Germany
{firstname.lastname}@hpi.de

Abstract. An important technique for attack detection in complex company networks is the analysis of log data from various network components. As networks are growing, the number of produced log events increases dramatically, sometimes even to multiple billion events per day. The analysis of such big data highly relies on a full normalization of the log data in realtime. Until now, the important issue of full normalization of a large number of log events is only insufficiently handled by many software solutions and not well covered in existing research work. In this paper, we propose and evaluate multiple approaches for handling the normalization of a large number of typical logs better and more efficient. The main idea is to organize the normalization in multiple levels by using a hierarchical knowledge base (KB) of normalization rules. In the end, we achieve a performance gain of about 1000x with our presented approaches, in comparison to a naive approach typically used in existing normalization solutions. Considering this improvement, big log data can now be handled much faster and can be used to find and mitigate attacks in realtime.

Keywords: network security, event logs, normalization, knowledge base

1 Introduction

During the last years, the number and complexity of cyber-attacks has dramatically increased[1, 2]. An important instrument for the monitoring and mitigation of attacks is the logging of suspicious activities in networks and on hosts. This logging is usually performed by operating systems, applications or security software[3] and creates huge amounts of log events for even relatively small networks. Handling these amounts of events manually or automatically by software with typical normalization approaches is almost impossible. Besides that, the incoming information cannot be handled in realtime, which is extremely important to detect ongoing attacks.

A difficult challenge that has to be mastered in software are the variety of formats for event logs that do not follow a common standard. Additionally, the formats usually represent a major part of the activity in an unstructured textual format, which is easy to read for humans, but not for machines[3].

At the moment, most solutions that gather and interpret event logs, so called Security Information and Event Management (SIEM) systems, only focus on structured event information and do not sufficiently interpret the unstructured information. Only a few SIEMs and log interpreters, such as HP's ArcSight[1] or Flowerfire's SawMill[2], actually interpret the contents of unstructured data but lack in performance when it comes to huge amounts of events or when data of different formats is mixed.

Although the use of structured log-formats is encouraged, they can also bring up new challenges for normalization, i.e. the structure often introduces a hierarchy of sub-formats. At the moment, there is only limited research on how such hierarchically structured formats can be efficiently normalized.

This paper provides a solution for effective normalization of events in general and for hierarchical log-formats in particular and is structured as follows. Section 2 gives an overview of related work to event normalization. Section 3 shows how a single event can be normalized. The following Section 4 then introduces the concept of a knowledge base for event normalization and shows two possible designs for it. Then, in Section 5, the hierarchical design is further detailed and features are described, that can improve normalization performance. In order to show the applicability of the mentioned concept, Section 6 shows performance values for two concrete scenarios. In the end, Section 7 gives a conclusion and an outlook to further research.

2 Related Work

2.1 Log-Formats

Several efforts have been made to normalize log-formats and simplify the interpretation by machines. Examples for such efforts are the general purpose formats *Syslog*[4], *Common Event Expression* (CEE)[5] or the *Common Event Format* (CEF)[6]. Another set of formats have been introduced for specific domain applications, such as *Cyber Observable eXpression* (CybOX)[7] for network and host-based detection systems.

However, all of the formats allow event normalization on very different levels of detail. Whereas the popular Syslog-format only provides rudimentary semi-structured normalization, the rarely used CybOX provides normalization for almost every imaginable information. Due to this situation, log information is mostly available in semi-structured formats and requires further processing to obtain relevant information.

In this paper, we focus on the Object Log Format (OLF)[8], which combines extensibility and object orientation of CEE and the variety of attributes of CEF.

[1] ArcSight Enterprise Security Management - `http://www8.hp.com/us/en/software-solutions/arcsight-esm-enterprise-security-management/`

[2] SawMill - `http://sawmill.co.uk`

2.2 Event Normalization and Analysis

The normalization and analysis of log events has already been implemented in a variety of methods and has been integrated into many existing software solutions.

Normalization Methods There are mainly four normalization methods that can be observed on the market and the research community.

Rule Matching (e.g., Regular Expressions) The normalization of each event log type is described in a rule that specifies how important information can be extracted from a concrete event. A popular approach in this category are regular expressions, especially Named-Group Regular Expressions (NGRE)[9]. This method associates information in the event to concrete event fields, which can be very useful for normalization. However, choosing the right regular expression for a random event type is processing intensive.

Tokenization A concrete event log is split up into tokens. These tokens could be the words of the human readable part of the log event or even phrases or certain notations in the log. The most common approach for tokenization is by word, which allows to group event logs containing the same words. However, this method heavily relies on static words in logs. A concrete implementation for tokenization is Apache Lucene[3].

Natural Language Processing (NLP) A human readable log line is decomposed by its language structure into subject, object, verbs and more. Once the information is extracted, it can be used to understand the meaning behind the phrase, as a human reader would see it. However, the method relies on the human readability of the log. Examples for NLP implementations are Stanford's CoreNLP library or SAP HANA's text analysis capabilities[10]. A concrete usage of this technique for log analysis has been proposed by Kobayashi et al.[11].

Custom Normalization The most effective but also most complex method is to use custom code for the normalization of each log format. As an example, one format is read with a CSV parser, while another one is parsed with a special Syslog parser and yet another one is handled with a combination of multiple regular expressions being applied in order. This type of normalization can be partially observed in the log analysis tools Logstash[4] and Sawmill.

Normalization Software There are a variety of software solutions that perform normalization based on event logs. Log analysis tools like the OSSEC IDS[5] monitors important event log files and extracts potential threats from these using regular expressions. On the other hand, there are complex SIEM systems

[3] Apache Lucene - http://lucene.apache.org/
[4] Elasticsearch Logstash - `https://www.elastic.co/products/logstash`
[5] OSSEC - `http://www.ossec.net`

like HP's ArcSight and RSA Security Analytics[6]. Both solutions have limited capabilities when it comes to normalization and hence also for complex attack detection. Firstly, event streams have to be associated with the right log-format to be used for normalization. Secondly, log events are mostly normalized by their structured parts and only a few log events are normalized completely.

3 Basic Normalization of Log Events

Different log sources produce events in different formats and encodings. Whereas almost all log sources produce logs in a textual way, the formats are different. The information in event log-formats can usually be categorized into dynamic, static and semantic parts.

Listing 1.1. Log Event in the Syslog-Format with categorized information parts (static, dynamic, semantic)

```
Jan 31 15:08:43 combo sshd[29819]: Failed password for root from 70.84.72.68 port 35933 ssh2
```

Listing 1.1 applies the concept of different information categories on a Syslog event.

The *static* information defines the structure of the log, including white spacing and glue words, such as *for* or *from*, for the dynamic parts. Because of the structure-defining property of the static information, it perfectly fits the purpose of identifying the type of log that is at hand.

The *dynamic* information bears the explicit information for a concrete instance of an event type. This information can change for each log event, but still has a common format, such as a time or a number. For example the port number and IP address in the listing are dynamic, but always have a certain format.

Besides the explicit information, *semantic* information provides a classification of the event through tagging. In the example, the marked text *Failed password* indicates that the event refers to a *failure* for a *login* activity on an *account*. So, the semantic tags for the given event could be *failure*, *login* and *account*.

Azodi et al.[12, 13] describe the process of extraction into the OLF-format in detail.

4 Knowledge Base Approaches

An event can theoretically be normalized with the help of a knowledge base that consists of entries of regular expressions and corresponding static fields and tags. However, as soon as multiple log-formats have to be supported for normalization, choosing the right entry becomes an issue. Assuming a KB of n normalization rules would mean that on average $\frac{n}{2}$ entries have to be checked, i.e. applying the entry's NGRE on the event, before the right matching entry is found.

[6] RSA Security Analytics - `http://www.emc.com/security/security-analytics/`

To overcome this processing overhead, NGREs have to be applied more efficiently to a given event. The following subsections show different approaches for applying regular expressions more efficiently by changing the organization of the KB.

For a better demonstration of the concepts, the two example log events in Figure 1 are chosen. Both are Syslog events typically created by Snort and have a similar structure, but they are of different event types.

Listing 1.2. Event #1 for a Snort event of type 648, i.e. an attempt to execute shellcode

```
Mar  1 16:02:40 bastion snort: [1:648:7]
     SHELLCODE x86 NOOP [Classification:
     Executable code was detected] [
     Priority: 1]: {TCP}
     4.152.207.238:3521 -> 11.11.79.83:80
```

Listing 1.3. Event #2 for a Snort event of type 1807, i.e. an attempt to inject commands over HTTP

```
Mar  1 16:03:01 bastion snort: [1:1807:10]
      WEB-MISC Chunked-Encoding transfer
     attempt [Classification: Web
     Application Attack] [Priority: 1]: {
     TCP} 4.152.207.238:3718 ->
     11.11.79.84:80
```

Fig. 1. Two log events produced by Snort with a similar structure

4.1 Flat Knowledge Base

The simplest approach to organize a KB is to have one normalization rule per possible event type. We call it flat KB, because there is no multi-level organization of the rules. Rather, each rule describes an event type in its entirety, including an NGRE matching the entire event text and all static fields. Figure 2 demonstrates the processing of the two log events from Figure 1.

There are two raw events to be normalized. In a flat KB, there is a separate normalization rule for each event. Rule #1 handles a Snort event of type 648, which is embedded into a Syslog event. Rule #2 handles a Snort-event of type 1807. Looking closer at the two rules, it can be seen that the rule parts for Syslog and Snort are identical. The only difference in the rules is the inner part being specific to the concrete Snort rule. It is obvious that flat normalization rules have many redundancies that can be eliminated with a more sophisticated KB approach.

4.2 Hierarchical Knowledge Base

The structure of typical logs is that more specific log parts are wrapped in parent formats, such as Syslog. In the case of Snort log events, there is an intermediary Snort wrapping format, as already indicated in the processing of the flat KB in Figure 2. An approach to overcome the redundancies in normalization rules of a flat KB is to organize normalization rules in multiple levels, i.e. hierarchically.

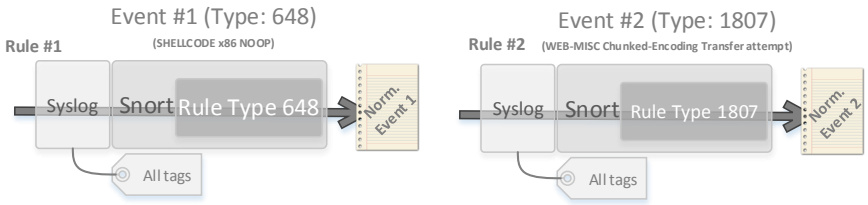

Fig. 2. Normalization with a flat KB

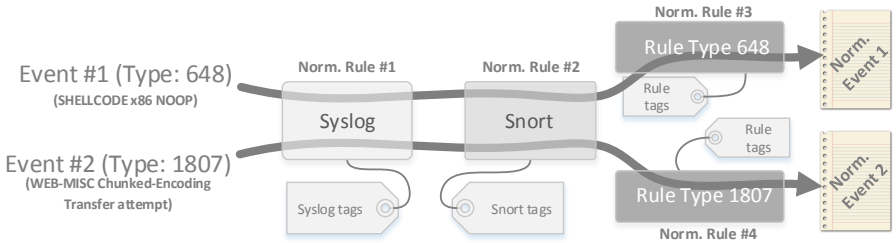

Fig. 3. Normalization with a hierarchical KB

Our hierarchical KB has one rule per identified parent format and one rule per event type. All these rules are loosely coupled by specifying possible parents for each rule. A concrete example for the events from Figure 1 is illustrated in Figure 3.

In the example, there could be four normalization rules. There are the two parent formats, for Syslog and Snort, being specified in rule #1 and #2. #2 has a link to #1 as possible parent. Rule #3 and #4 specify the concrete parts of the two given event types and both link to rule #2 as parent.

In order to match one of the given log events, we look for rules that are applicable in the current matching context. We call these rules *candidate rules*. This means, at the beginning we first check all rules that can stand alone, which in this case is only Syslog, because it has no parents. When the Syslog part is matched, its message part is used to match the next-level rule, which has the Syslog-rule as its parent. In our concrete example, this would be the Snort-rule. Once Snort is matched, the rule's message part is extracted and then checked against the rules that have the Snort-rule as parent. This could be rule #3 and #4. According to the concrete content, the right rule would be matched.

4.3 Comparison of approaches

Structuring Taking the effort of defining rules, the hierarchical approach is much more organized, if many event types of similar structure have to be described. For example, taking the thousands of existing Snort rules, the hierarchical approach only defines Syslog and the basic Snort event once and then specifies

Snort's event type specific parts in separate simple normalization rules. In the flat approach, all parts, especially Syslog and Snort, have to be repeated for every Snort-rule.

Another benefit with the hierarchical approach is that static fields and tags can be defined at parent-level rules. As an example, to tag an event as a network-related event (expressed with `tag.domain = net`), a corresponding tag only has to be attached to the Snort-rule, because Snort only deals with network data.

Performance When it comes to performance, the structuring can help to improve the performance of normalization significantly, too. Let us assume there are logs and normalization rules for two different applications, namely `Snort` and `sshd` being wrapped in Syslog.

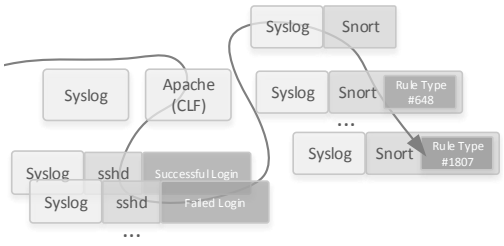

Fig. 4. Matching with a flat KB

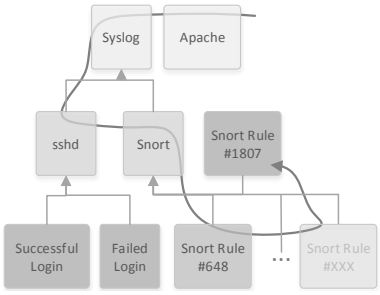

Fig. 5. Effective matching with a hierarchical KB

For the flat approach, as seen in Figure 4, in the worst case a Snort event has to be checked against all sshd-related normalization rules until the Snort normalization rule is found as a match.

For the hierarchical approach, as seen in Figure 5, in the worst case only a single Syslog- and sshd-rule is checked until the Snort normalization rules are found as a match. So taking the fact that log events of many applications have to be supported by the KB, the hierarchical approach can have major performance benefits.

5 Improving Knowledge Base Matching Performance

While the main idea, to organize the KB hierarchically, can already bring major performance benefits, further improvements can speed up the normalization even further. Following subsections give an overview of the different approaches we have evaluated. Additionally, we point out which technologies and libraries have been used for our implementation in the Java programming language. The evaluation results of the approaches are presented in Section 6.

5.1 Rule Indexing

When working with hierarchical rules, it is important to choose the right candidate rules as fast as possible. These candidate rules are selected by a number of criteria that effectively filter the number of rules to be applied on a given log fragment. We came up with following criteria:

- **Standalone:** This rule matches an event that can stand on its own. These rules do not require a parent.
- **Level:** Logs are constructed in multiple levels. There are global wrapper formats, such as Syslog, application wrappers, such as Snort, and event parts that are specific to the event type. Based on the current level, rules of upper levels do not have to be checked again.
- **Parent:** Some event parts can only appear as a child of another format. As an example, a specific Snort rule can only appear in the general Snort wrapper. This property can rule out most of the available rules.

Based on these criteria, we tried three different strategies to search applicable rules more efficiently.

Iteration
: In the easiest case, the program would iterate through all available rules and check the criteria for each rule. However, iteration over large amounts of rules creates major performance impacts.

CQEngine
: A straight-forward approach is to index all available rules by the presented criteria. For our evaluation, we used a library called CQEngine[7], a NoSQL indexing and query engine.

Lucene
: Another approach is to create a similarity measure between a given event and the events a rule is able to match. A detailed description of this approach for a flat knowledge base can be found in a paper[13] by Azodi et. al.

5.2 Rule Selectors

Log-formats with many different event types as a child give an indication of the child's type. For example, Syslog has an application name that indicates what kind of application logs are in the message part. Such indicators can be used to directly select the right normalization rules without searching through all possible candidate rules for the parent. An example of this rule selection can be seen in Figure 6.

5.3 Result Caching

The searching for candidate rules can be seen as one of the most processing intensive tasks in the normalization process. We propose to cache the search results for a given set of criteria. Caching can then make candidate rules directly accessible and improve normalization of logs without format indicators.

[7] CQEngine - `https://code.google.com/p/cqengine/`

Fig. 6. Rule selector approach on log event from Listing 1.2

5.4 Priority Lists

A feature that can be added in conjunction with result caching is the ordering of all candidate rules by their frequency they have been used for the normalization of previous events. Creating a priority list with every log event is unfeasible, because the sorting would result in major performance drops. Rather, ordering by frequency should be performed in a time interval that takes the required processing overhead and the actuality of the frequencies into account.

6 Evaluation

We have implemented all of the above mentioned knowledge bases, indexing strategies and other proposed improvements into our Real-time Event Analysis and Monitoring System (REAMS)[14] and ran performance tests for various combinations of these on two data sets. Each combination was run 10 times per data set, so that a mean runtime could be calculated. As our implementation is highly parallelized and can run multiple normalization threads simultaneously, we decided to run the performance tests on eight parallel normalization threads on our server[8].

6.1 Experiment 1: Normalizing Hierarchical Logs

In the first experiment, we have normalized all Snort event logs, i.e., *69039 log events*, from Honeynet Challenge #34[15]. We have generated normalization rules for all Snort rules from the rule snapshot[9] for registered users and the emerging threats open rules[10]. These rules, altogether 55547, cover all Snort events that were present in the challenge's logs. The performance results of normalization are visualized in the box and whisker diagram in Figure 7.

We see that the flat KB performs at least a magnitude worse than the hierarchical KB. This is because the handling and checking of more and longer rules as used with the flat KB is more time consuming than with a hierarchical one. Both factors have a major impact on hierarchically structured formats as used by Snort.

We can also deduce that direct indexing with CQ is faster for the hierarchical KB than normal iteration. The better indexing with CQ is not relevant for

[8] Virtual Machine (Debian 7.8, 32GB RAM(dedicated), 8 cores(dedicated)) on VMware ESXi host with 256GB RAM and 8x Intel Xeon X7560 CPUs @ 2.27GHz
[9] Snort Ruleset (Registered User) - `https://www.snort.org/downloads`
[10] ETOpen Ruleset - `https://portal.emergingthreats.net/etpro/open`

the flat KB, because there is almost no data to index. The Lucene indexing generally performs better than iteration and CQ, but because of its mechanisms the rules cannot be further cached or prioritized. Unfortunately, Lucene cannot outperform the benefits of caching and prioritization.

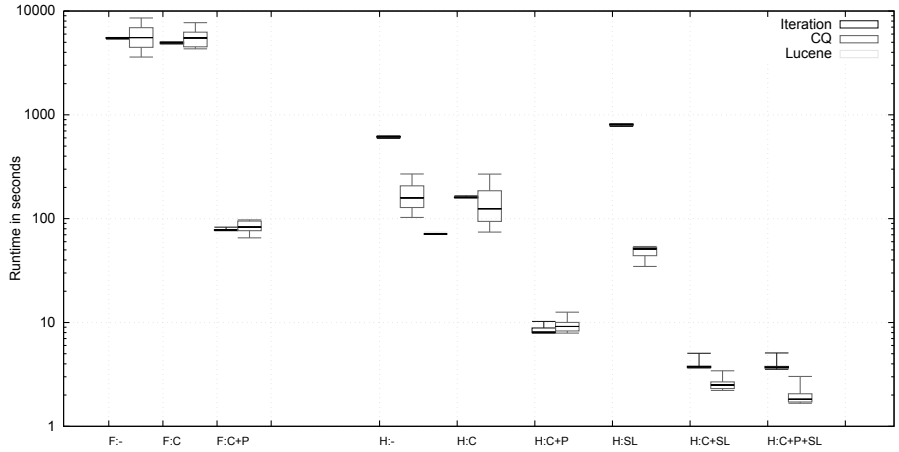

Fig. 7. Evaluation results for normalizing 69039 Snort log events, **left (F)** - flat knowledge base, **right (H)** - hierarchical knowledge base (**C** - Cache, **P** - Prioritization, **SL** - Rule Selector)

Within the hierarchical performance results, prioritization and rule selectors achieve the biggest performance benefits. Both concepts can reduce the number of rules to be checked per log event dramatically. In comparison to the plain KB performance, both concepts also bring the most stable runtime values. The other concepts all heavily rely on the ordering of the rules, because they match rules in the order they are stored in memory. Prioritization and rule selectors mostly choose the right rule directly. It should be mentioned that we intentionally did not fix the rule order, because in reality the correct order is not known and is highly dependent on the concrete distribution and order of log events.

The caching of results can bring benefits, if a large number of varying data lookups would have to be performed. As an example, it is not so relevant for caching of big data sets (e.g. H:- $\leftrightarrow$ H:C), but for caching of smaller data sets that apply to only a few log events (e.g.: H:SL $\leftrightarrow$ H:C+SL).

6.2 Experiment 2: Normalizing Mixed Logs

In the second experiment, we applied our algorithm to mixed logs consisting of Snort and Apache events, i.e., *76659 log events*, from Honeynet Challenge #34[15]. In comparison to experiment 1, the performance test results are almost

unchanged, because there are only roughly 11% more events now, which are even easier to normalize.

6.3 Summary

Altogether, a combination of a hierarchical KB with CQ indexing, caching, prioritization and rule selectors seems to have the best normalization performance. In our experiments, we could reach a normalization speed of 37,000 events/s for a highly hierarchical log-format on eight threads.

7 Conclusion and Future Work

In this paper, we have shown how the speed of event log normalization can be drastically improved by using normalization algorithms that consider the typical hierarchical structure of log events. We have performed normalization with Lucene on a flat KB before, which did not perform well enough for big data. However, by using hierarchical normalization rules in combination with CQ indexing, caching and prioritization of rules we could speed up normalization by approximately 3 orders of magnitude. Altogether, we were able to normalize around 37,000 events/s, which should already be enough for large network environments. This fast speed can even be achieved, if logs with different parent formats are intermixed, like in real world log environments.

A starting point for further research could be the improvement of the normalization speed in a way that even event logs of companies with multiple billion events per day can be handled. One direction could be the parallelization of the processing on multiple machines.

Additionally, normalized logs can be used for further analysis. For example, inventory information can be extracted from logs to create an overview of machines, software and users in a network. Furthermore, the log information can be used to detect or prevent intrusions in a network environment.

Acknowledgment

We would like to thank HPI FutureSoC lab for providing us with the latest and powerful computing resources, which make the testing and experiments specified in the paper possible.

References

[1] United States Computer Emergency Readiness Team (US-CERT). *US-CERT Year in Review CY 2012*. Tech. rep. US Department of Homeland Security, 2012.

[2] US Office of Management and Budget. *Fiscal Year 2012 Report to Congress on the Implementation of The Federal Information Security Management Act of 2002*. Mar. 2013.

[3] Karen Kent and Murugiah Souppaya. “Guide to Computer Security Log Management”. In: *NIST special publication* (Sept. 2006). URL: http://212.200.39.245:81/CrnaRupa/2009-2010/FIM/ZIS/Literatura/GuidetoComputerSecurityLogManagementSP800-92.pdf.

[4] Rainer Gerhards. *The Syslog Protocol*. RFC 5424 (Proposed Standard). Internet Engineering Task Force, Mar. 2009. URL: http://www.ietf.org/rfc/rfc5424.txt.

[5] Anton Chuvakin, Raffael Marty, et al. *Common Event Expression*. White Paper. MITRE, June 2008.

[6] Hewlett-Packard. *Implementing ArcSight CEF*. 20. Hewlett-Packard. June 2013.

[7] Sean Barnum, Robert Martin, et al. *The CybOX Language Specification*. Draft 1. The MITRE Corporation, Apr. 2012.

[8] Andrey Sapegin, David Jaeger, et al. “Hierarchical Object Log Format for Normalisation of Security Events”. In: *Proceedings of the 9th International Conference on Information Assurance and Security (IAS2013)*. Yassmine Hammamet, Tunisia, Dec. 2013, pp. 25–30.

[9] Jeffrey E. F. Friedl. *Mastering Regular Expressions*. Ed. by Andy Oram. 3rd. O’Reilly Media, Aug. 2006.

[10] Lucas Sparvieri. *SAP HANA Text Analysis*. SAP. Jan. 2014. URL: http://scn.sap.com/community/developer-center/hana/blog/2013/01/03/sap-hana-text-analysis.

[11] Satoru Kobayashi, Kensuke Fukuda, and Hiroshi Esaki. “Towards an NLP-based log template generation algorithm for system log analysis”. In: *Proceedings of The Ninth International Conference on Future Internet Technologies*. 2014, p. 11.

[12] Amir Azodi, David Jaeger, et al. “Pushing the Limits in Event Normalisation to Improve Attack Detection in IDS/SIEM Systems”. In: *Proceedings of the First Internation Conference on Advanced Cloud and Big Data (CBD2013)*. Nanjing, China, Dec. 2013.

[13] Amir Azodi, David Jaeger, et al. “A New Approach to Building a Multi-Tier Direct Access Knowledge Base For IDS/SIEM Systems”. In: *Proceedings of the 11th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC2013)*. Chengdu, China, Dec. 2013.

[14] *Real-time Event Analysis and Monitoring System (REAMS)*. URL: http://hpi.de/en/meinel/security-tech/network-security/security-analytics/reams.html (visited on 05/11/2015).

[15] The Honeynet Project. *Honeynet Challenges: Scan of the Month 34*. Web Site. 2005. URL: http://old.honeynet.org/scans/scan34/ (visited on 04/05/2013).