Gerard Frankowski
Marcin Jerzak
Maciej Miłostan
Tomasz Nowak
Marek Pawłowski

APPLICATION OF THE COMPLEX EVENT PROCESSING SYSTEM FOR ANOMALY DETECTION AND NETWORK MONITORING

Abstract *Protection of infrastructures for e-science, including grid environments and NREN facilities, requires the use of novel techniques for anomaly detection and network monitoring. The aim is to raise situational awareness and provide early warning capabilities. The main operational problem that most network operators face is integrating and processing data from multiple sensors and systems placed at critical points of the infrastructure. From a scientific point of view, there is a need for the efficient analysis of large data volumes and automatic reasoning while minimizing detection errors. In this article, we describe two approaches to Complex Event Processing used for network monitoring and anomaly detection and introduce the ongoing SECOR project (Sensor Data Correlation Engine for Attack Detection and Support of Decision Process), supported by examples and test results. The aim is to develop methodology that allows for the construction of next-generation IDS systems with artificial intelligence, capable of performing signature-less intrusion detection.*

Keywords network monitoring, intrusion detection, anomaly detection, complex event processing

Citation Computer Science 16 (4) 2015: 351–371

1. Introduction

1.1. IT security landscape and trends

The rapid development of the Internet, the numerous technologies and services, and ubiquitous access to easy-to-use penetration testing tools (combined with abusing anonymity opportunities on the Internet) create room for different types of vulnerabilities. It is only a matter of time before each system exposed to the Internet is attacked [9]. According to the [20] report, 6,787 web software vulnerabilities were disclosed in 2013, while in 2011, there were only 4,989, and in 2012 5,291.

An example of threats of special concern are zero-day attacks. Attacks of this type exploit vulnerabilities that have yet to be been publicly disclosed. Thus, defense becomes extremely difficult, because when the vulnerability is unknown, the software vendor cannot issue the appropriate security update, and the antivirus vendor is unable to provide the relevant signature of attack symptoms or exploits[4]. These vulnerabilities are often used in highly targeted attacks, especially concerning Advanced Persistent Threats scenarios. The [20] report states that 2013 was the year with the greatest number of detected zero-day vulnerabilities in use (23, compared to only 14 in 2012). There are still an unknown number of critical flaws of which we are unaware. The famous Heartbleed bug was discovered after no less than 2 years, while the Shellshock bug (allowing an arbitrary code to run on the affected servers) remained in the source code of Unix/Linux shells for more than 22 years [14].

The partial (and currently only applicable) response to these types of threats may be signature-less detection, based on advanced anomaly detection techniques. Responding to the zero-day threat requires the ability to detect anomalies, correlate across disparate systems, identify the threat, and mitigate accordingly [10].

The goal of the SECOR project is to improve the protection of highly heterogeneous and dynamically changing infrastructures, including those for e-science (like grid environments and NREN facilities). This requires using novel techniques for anomaly detection and network monitoring in order to raise overall situational awareness and provide early warning capabilities.

1.2. Complex Event Processing

The concept of Complex Event Processing (CEP) is about listening to events originating from multiple sources and detecting complex patterns in real-time or nearly real-time without the need to store individual event information for a long term. Within the CEP engine, the events are usually filtered into separate data streams and correlated using the idea of time-dependent sliding windows along with queries written in a specially crafted domain specific language; e.g., Event Processing Language (EPL). EPL allows for the processing of multiple streams within a single query in order to identify particular relationships between events. The most important point in complex event processing is to analyze as many events as possible and report the data regarding the spotted complex events to the operator as quickly as possible.

CEP has a broad range of applications; for example, in:

- funds (trade algorithms, risk management, fraud detection);
- the management and automation of business processes (relationship management (CRM), workflow applications, process tracking, operational intelligence);
- sensor network applications (analysis and management of public transport, production lines);
- network and application monitoring (SLA monitoring);
- security policy risk management.

The following chapter of this article highlights previous research and related work in the relevant area and describes the PSNC experience within an Intrusion Detection field, and it also reveals our motivation for this work and the SECOR project itself. The next chapter – CEP – anomaly detection approach – contains a step-by-step guide of preparing and configuring the necessary environment in order to detect anomalies. This introduction has been prepared to help understand the WSO2 environment as well as the groundwork leading up to a description of our approach undertaken in the SECOR project. The following chapter – CEP in the SECOR project – starts with a brief introduction to the project and then concentrates on the architecture and elements necessary to describe an example attack discovery scenario based on graph-clustering algorithms. The paper finishes with conclusions from the undertaken research and summarizes plans for the future development of ideas and solutions present in the SECOR project.

2. Related work

2.1. IDS development experience in PSNC

PSNC (Poznań Supercomputing and Networking Center) [17], affiliated with the Institute of Bioorganic Chemistry, Polish Academy of Sciences, is a research center that is the operator of the PIONIER network (Polish National Research and Education Network [15]). PSNC is an HPC Center, a Systems and Network Security Center, as well as an R&D Center of New Generation Networks, Grids, and Portals. PSNC has participated (and continues to participate) in numerous R&D projects, both for the EU and Poland. As security has always been an important topic for PSNC, we had the opportunity to perform different research security tasks in large European projects (like GANT [7] and EGEE [5]) as well as a number of domestic projects.

As the PIONIER operator, PSNC has broad experience with the development and deployment of network monitoring and intrusion detection/prevention systems (IDS/IPS). For instance, we have designed and developed a distributed IDS system, MetaIDS, within one of the R&D projects led by PSNC within the confines of the Polish Platform for Homeland Security initiative [16]. MetaIDS has been deployed for the Polish National Police [9, 6]. MetaIDS is a signature-based IDS that detects threats based on information from multiple sources (internal sensors and external

IDS/IPS systems). The core of the system is a correlation engine that is able to combine several individual events and perform signature-based attack sequence discovery. The drawback is that system relies heavily on expert knowledge. Overcoming this limitation is a challenge that PSNC confronted during the SECOR project.

2.2. Applying CEP for Grid monitoring

Applying Complex Event Processing in the grid computational model has been addressed in [1, 2]. The authors address a more-general use of CEP for grid monitoring, mainly (but not only) for optimizing the utilization of grid resources. However, the evaluated generic approach may also be applied to the collection of information related only to security, especially as the availability of data (or more generally: resources) is one of the three main pillars of security (along with confidentiality and integrity).

The authors used the Esper CEP engine. No sophisticated correlation methodologies were applied, as the built-in facilities are absolutely enough for the purposes of the described application. The authors successfully proved the applicability of the CEP engine to security data monitoring in real time, and that it is possible to apply complex queries over monitoring data streams. Both features may be useful for detecting security threats, although the used metrics or queries would have to be significantly different (additionally, in SECOR, we have to apply expert knowledge to security threats upon the events collected from the CEP data streams).

The authors also proved that an overhead related with CEP is acceptable using the properly adjusted configuration, although CPU utilization for handling events coming in from large numbers of sensors may be significant and, thus, require installation of the main monitor on a dedicated system with more-robust computational capabilities [1, 2].

2.3. CEP Engine in Detecting Security Threats

The application of CEP purely to security reasons is described in [12], where the authors propose building a wide collaborative platform across different organizations facing the same cyber threats. Such an approach apparently requires coping with enormously large sets of data sent by particular bodies, especially for centralized CEP systems (like Esper).

There, the authors propose establishing an SR (Semantic Room) abstraction layer that defines the so-called objectives that may be generally compared to the particular type of cyber threat being monitored. Particular instrumentation of system components (like gateways and the central processing engine) allows us to decrease the amount of processed events to an acceptable level. On the other hand, this approach is conceptually similar to the instrumentation of different-block analysis in SECOR that is optimized to detect certain type(s) of security vulnerabilities.

In the context of our work, algorithms utilizing expert knowledge on cyber attacks are especially interesting. The authors described two use cases; one concerning TCP SYN scan detection, and the other – fraud monitoring.

In the case of the former, the attack relies upon unfinished TCP connections. The attacker sends a SYN network packet as is typical during a common three-way TCP handshake. But after the probed host responds with a SYN-ACK packet, either an RST (reset) packet is sent by the attacker or nothing further happens, leaving the connection in a "half-open" state. The authors proposed an R-SYN rank-based scan-detection algorithm that combines different port-scan-detection techniques: half-open connections, horizontal and vertical port scans, and entropy-based failed connections. Additionally, the authors have applied the Storm [19] open-source distributed CEP engine implemented in Java and Clojure in order to improve performance and fault tolerance.

3. CEP – anomaly detection approach

This section presents an exemplary implementation of a complex-event-processing approach to correlating homogeneous event information (i.e., netflow records containing a summary of connection data in IP networks). The main aims are to present how the CEP analysis can be deployed for rule-based anomaly detection and to introduce the reader to our implementation mechanisms. The WSO2 software stack[22] with a built-in complex-event-processing engine was chosen as the basis for our implementation, due to the fact that it is built in a very modular way and simplifies software development in several ways (e.g., provides a convenient API for custom plugins and features, file based configuration, automatic handling of component updates, visualization components, etc.).

A web server was chosen as a monitored service for two reasons. Firstly, web servers are sometimes the only publicly available resource, and thus, their security is crucial. Secondly, it is considered a promising potential entry point for attackers trying to get into corporate networks, because web applications are prone to a number of known attack vectors, some of which require only a basic skillset to be performed yet have a significant impact.

The exemplary attack described in this section uses access logs from a web server to show the power of complex-event processing in discovering hijacked sessions in web traffic that are covered up with decoy visits. It is a common practice for attackers to send large amounts of typical data to the server to decrease the probability of discovering their activities, hoping that administrators will not notice the real threat. Hijacked sessions are detected in real-time by correlating entries from the web servers access log: entries originating from the same web application user with alternating-source IP addresses.

The described scenario consists of the following steps:

1. Data source preparation – an Apache web server is configured to send information about visits on the web site.
2. Event stream definition – a series of events with common attributes is defined. Rules defined in this step describe the dependencies between several types of events originating from different streams.

3. Event source definitions – a source of data is defined along with a schema. Sources of data may include CSV files, e-mails, etc. CSV files are used in the described scenario, so it is necessary to define a schema with a relevant description of fields.
4. Execution plan and result storage – execution plans describe patterns to find in analyzed-event streams and actions to undertake in case they are discovered. The plan is provided in an SQL-like Siddhi Query Language [23].

3.1. Data source preparation

Before the Complex Event Processing engine can be started, it is required to configure the data source. In the described scenario, a web server logging mechanism is used. Two commonly distinguished types of such log messages are access logs and error logs. The former contains various details about each HTTP request/response cycle and is used in this scenario.

To log visits in a simple CSV format, the Apache web server was configured with a CustomLog directive. By using a pipe character in the first parameter, execution of an external program was requested, with the contents of each log entry provided on the standard input. In this example, netcat utility was used to send the entries via UDP protocol to a remote server (here: cep.man.poznan.pl, UDP port 7777):

```
CustomLog "\|nc -u cep.man.poznan.pl 7777" "%{%Y-%m-%d
%H:%M:%S}t,www,%h,%u,\"%m\",\"%U\",\"%q\",%>s,%b,\"
%{Referer}i\",\"%{aUser-agent}i\""
```

The second parameter defines the contents of the log entries using specific codes, which can be verified in the web servers documentation[21]. To differentiate between sessions, user names provided with HTTP Basic Authentication were used; but it is also possible to use the session identifier instead.

The resulting messages appear as follows:

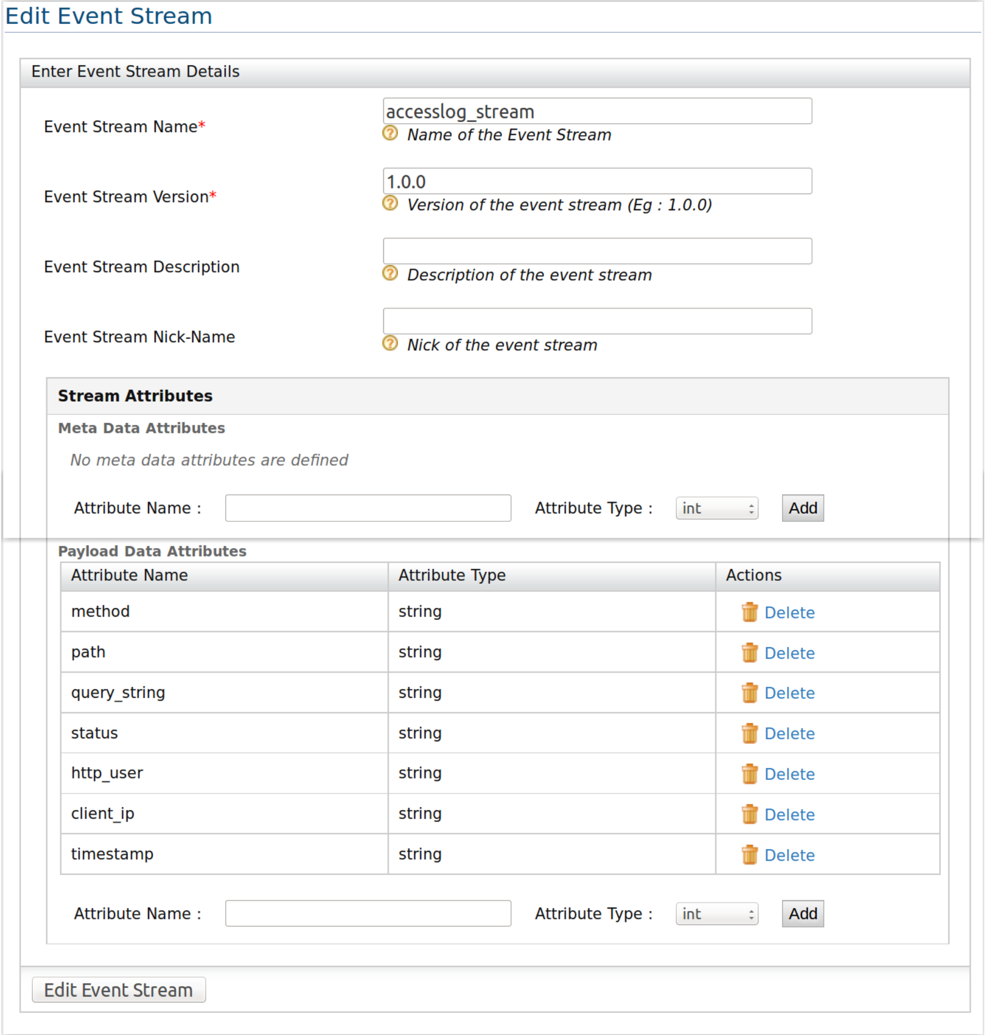
```
2014-11-07 12:35:44,www,192.168.150.3,pla,
"GET","/img/calendar.png","",304,-,"http://www.example.com/","-"
```

3.2. Event stream definitions

The next step was to define an event stream by using the WSO2 CEP GUI within a web browser. This is an important step because it allows us to aggregate the events and format them into a standardized form (for example, a relational database table). Events are structures of key-value attributes. Figure 1 shows an example of an event stream definition for the described use case. The important elements are the stream name and version that will be used later as a reference. Within the streams, events can be easily managed and filtered using patterns in order to get the specific information.

3.3. Event source definitions

As the next step, there is a need to define the source of events that should be put into the event stream. The available mechanisms include (among others) XPath expressions, to pick event details from XML documents and regular expressions to parse



Edit Event Stream

Enter Event Stream Details

Event Stream Name*: accesslog_stream
Name of the Event Stream

Event Stream Version*: 1.0.0
Version of the event stream (Eg : 1.0.0)

Event Stream Description:
Description of the event stream

Event Stream Nick-Name:
Nick of the event stream

Stream Attributes

Meta Data Attributes

No meta data attributes are defined

Attribute Name : Attribute Type : int

Payload Data Attributes

Attribute Name	Attribute Type	Actions
method	string	Delete
path	string	Delete
query_string	string	Delete
status	string	Delete
http_user	string	Delete
client_ip	string	Delete
timestamp	string	Delete

Attribute Name : Attribute Type : int

Figure 1. Stream edit panel.

text files. In this scenario, CEP is configured to use a file as an event source. In the example scenario, a host running CEP writes incoming UDP packets to a file by using the netcat tool:

```
 nc -v -u -l 7777  >> /tmp/customlog
```

Event values are extracted from the CSV records using regular expressions saved in the WSO2 Event Builder definition. The definition can be entered using the CEP Admin GUI with a web browser or entered in a file which will be automatically read and deployed. In the described case, the path of the file is as follows:

```
wso2cep-3.1.0/repository/deployment/server/eventbuilders/accesslog_1.0.0_builder.xml
```

The order of the mapped attributes should be identical to those in the stream definition.

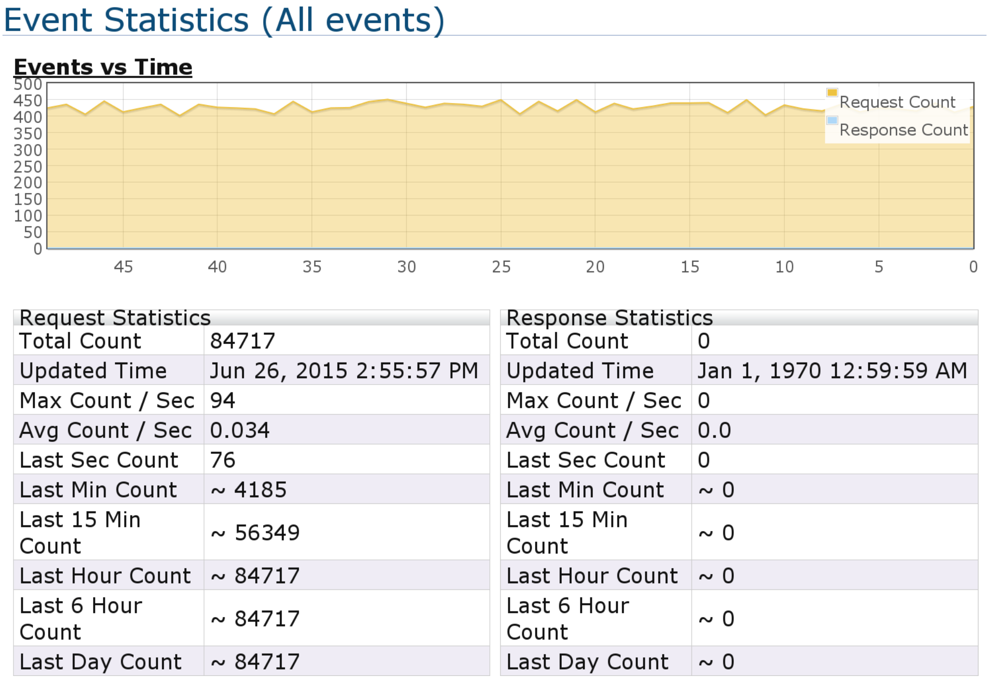
```
<?xml version="1.0" encoding="UTF-8"?>
<eventBuilder name="accesslog_builder" statistics="enable"
trace="enable" xmlns="http://wso2.org/carbon/eventbuilder">
<from eventAdaptorName="accesslog" eventAdaptorType="file">
<property name="filepath">/tmp/customlog</property>
</from>
<mapping customMapping="enable" type="text">
<property>
        <from regex="^[^,]+,[^,]+,[^,]+,[^,]+,&quot;([^,]+)&quot;,"/>
        <to name="method" type="string"/>
</property>
<property>
        <from regex="^[^,]+,[^,]+,[^,]+,[^,]+,[^,]+,([^,]+),"/>
        <to name="path" type="string"/>
</property>
<property>
        <from regex="^[^,]+,[^,]+,[^,]+,[^,]+,[^,]+,[^,]+,([^,]+),"/>
        <to name="query_string" type="string"/>
</property>
<property>
        <from regex="^[^,]+,[^,]+,[^,]+,[^,]+,[^,]+,[^,]+,[^,]+,([0-9]+),"/>
        <to name="status" type="int"/>
</property>
<property>
        <from regex="^[^,]+,[^,]+,[^,]+,([^,]+),"/>
        <to name="http_user" type="string"/>
</property>
<property>
        <from regex="^[^,]+,[^,]+,([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+),"/>
        <to name="client_ip" type="string"/>
</property>
<property>
<from regex="^([^,]+),"/>
        <to name="timestamp" type="string"/>
</property>
</mapping>
<to streamName="accesslog_stream" version="1.0.0"/>
</eventBuilder>
```

Any errors in the file syntax will be reported in the log file:
`wso2cep-3.1.0/repository/logs/wso2carbon.log`

If tracing is enabled for input adapters, event streams, execution plans, or output adapters, real-time monitoring is available through a log file; in this case:
`wso2cep-3.1.0/repository/logs/wso2-cep-trace.log`

The process monitor in Figure 2 is accessible from the GUI and displays statistics of the number of tasks delivered, processed, and sent.

Figure 2 presents statistics from the event monitor on the whole WSO2 CEP system level. Events are information which is sent to the CEP system, processed by CEP, sent from CEP to the output source, and many others. In the event statistic monitor, particular details may be seen about; e.g., streams, the event stream builder, or the input adapter.



Event Statistics (All events)

Request Statistics	
Total Count	84717
Updated Time	Jun 26, 2015 2:55:57 PM
Max Count / Sec	94
Avg Count / Sec	0.034
Last Sec Count	76
Last Min Count	~ 4185
Last 15 Min Count	~ 56349
Last Hour Count	~ 84717
Last 6 Hour Count	~ 84717
Last Day Count	~ 84717

Response Statistics	
Total Count	0
Updated Time	Jan 1, 1970 12:59:59 AM
Max Count / Sec	0
Avg Count / Sec	0.0
Last Sec Count	0
Last Min Count	~ 0
Last 15 Min Count	~ 0
Last Hour Count	~ 0
Last 6 Hour Count	~ 0
Last Day Count	~ 0

Figure 2. CEP Events Statistics – all streams.

3.4. Execution plan and result storage

The final preparation step is creating an execution plan. This is a tool built into the WSO2 system in order to group, correlate, and process events from multiple streams. Correlation of events is performed by the Siddhi CEP engine using the Siddhi Event Query Language[23]. The output can be stored into a separate event stream or sent directly to the database for later visualization. Currently, only relational databases are supported.

An exemplary rule that discovers users accessing the same account from distinct IP addresses can be provided either by CEP GUI or by putting it in a file. The path in this case is:

```
wso2cep-3.1.0/repository/deployment/server/executionplans/5f.xml
```

The query relates to the database named SECOR_DB, defined earlier in the WSO2 Data sources page.

```
<?xml version="1.0" encoding="UTF-8"?>
<executionPlan name="5f\_ExecutionPlan2" statistics="enable"
  trace="enable" xmlns="http://wso2.org/carbon/eventprocessor"\>
  <description/>
  <siddhiConfiguration>
    <property name="siddhi.persistence.snapshot.time.interval.minutes">0
</property>
    <property name="siddhi.enable.distributed.processing">false</property>
  </siddhiConfiguration>
  <importedStreams>
    <stream as="ac" name="accesslog\_stream" version="1.0.0"/>
  </importedStreams>
  <queryExpressions><![CDATA[
 define table accesslog\_{}stream (IPa1 string, IPa2 string,
UserName string, timestamp  string)
  from ( 'datasource.name'='SECOR\_{}DB',
'database.name'='SECOR\_{}DB',
  'table.name'='accesslog\_{}stream',
  'create.query'='CREATE TABLE accesslog\_{}stream ( IPa1 VARCHAR(40),
IPa2 VARCHAR(40), UserName VARCHAR(40), timestamp VARCHAR(24))'});
     from every} a1 = ac
    -> a2 = ac[ http\_{}user== a1. http\_{}user
and  client\_{}ip != a1. client\_{}ip]
    -> a3 = ac[ {\color{blue} \verb client_ip == a1. client\_{}ip
and  http\_{}user == a1. http\_{}user
and  http\_{}user == a1. http\_{}user ]
         within 1 min
     select|a1.client\_{}ip as IPa1,
       a2.client\_{}ip  as IPa2,
       a3.http\_{}user  as UserName,
    a2.timestamp  as timestamp
    insert into accesslog_{}stream;
]]></queryExpressions>
  <exportedStreams>
    <stream name="accesslog_duplicate_stream" valueOf="acsd" version="1.0.0"/>
  </exportedStreams>
</executionPlan>
```

The CEP GUI provides the Event Flow diagram (cf. Fig. 3), which visualises links between system elements like event adapters, event builders, event streams, and execution plans. The direction of data flow is represented by arrows.

After the data correlation with Siddhi CEP in the WSO2 system, results showing a successful attack discovery were sent to the defined MySQL database, structured as follows:

IP address 1 IP address 2 Username Time 192.168.150.2 192.168.150.10 john 2014-11-07 12:35:44

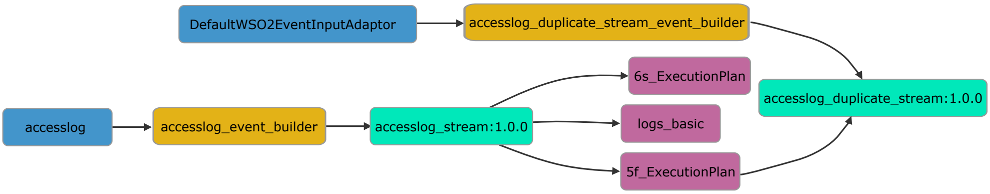

DefaultWSO2EventInputAdaptor
accesslog_duplicate_stream_event_builder
6s_ExecutionPlan
accesslog_duplicate_stream:1.0.0
accesslog
accesslog_event_builder
accesslog_stream:1.0.0
logs_basic
5f_ExecutionPlan

Figure 3. CEP Event Flow.

The source IP address has changed from 192.168.150.2 to 192.168.150.10. While this may be legitimate (IP addresses may change, especially when using mobile devices), this may also indicate that the session of the involved user has been hijacked. Another factor that may be taken into consideration under these conditions is to verify the user-agent HTTP header, which should be constant for legitimate users.

3.5. Results

This chapter has been prepared to show how easy it is to deploy and configure the Complex Event Processing framework (WSO2) and detect application layer misuse in a simple scenario. The rules are quite simple to define and apply. This example is based on a real web application vulnerability that may also be detected by manually reviewing web server logs. The real strength of this solution, however, lies in the fact that this type of attack may be also discovered when it has been camouflaged with misleading background traffic (which is a common practice used by intruders for decreasing the probability of detection).

4. CEP in the SECOR project

This chapter describes an exemplary use case originating from the SECOR project based on the WSO2 and netflow records.

The chapter consists of the following sections:

1. The SECOR project – describing the motivation behind the project and a brief introduction to its goals and foundation.
2. Component description – this paragraph shows several building components necessary for this scenario to succeed – from NetFlows and FlowCollectors to Cypher queries and the Neo4j database.
3. Netflows – example – the specification of the used testbed, attack scenario description, proposed execution query, and test results.

4.1. The SECOR project

SECOR is an acronym for the national research project ”Sensor Data Correlation Engine for Attack Detection and Support of Decision Process” funded by Polish National

Centre for Research and Development (NCBiR) within the confines of the Applied Research Programme. The project consortium consists of the Military Communication Institute (the coordinator), PSNC, and an SME company – ITTI. SECOR finishes in May 2015. The main goal of the project is to build different models of anomaly detection and to correlate their results.

In SECOR, we utilize several novel approaches to achieve the desired functionality of the next-generation IDS/IPS. The portfolio of investigated methods includes Graph Clustering algorithms, Petri Nets, Neural Networks, and advanced statistical methods [3] in conjunction with the Complex Event Processing engine.

The CEP component is the main module of the SECOR system. Its mission is to aggregate and correlate information from the various components of the system. Correlation primarily allows for combining data from different Blocks of Analysis, sensors, and other systems like IPS/IDS. CEP collects and analyzes data about different events reported by the anomaly detectors as well as other unauthorized activities and assesses the probability of an attack. In the SECOR project, the CEP module allows for establishing a connection between other modules such as sensors, Blocks of Analysis, the user management application, and the database.

The SECOR project is mainly focused on developing methodology that allows for the construction of next-generation IDS/IPS systems with built-in artificial intelligence that are capable of performing signature-less intrusion and anomaly detection. The architecture of SECOR is presented in Figure 4.

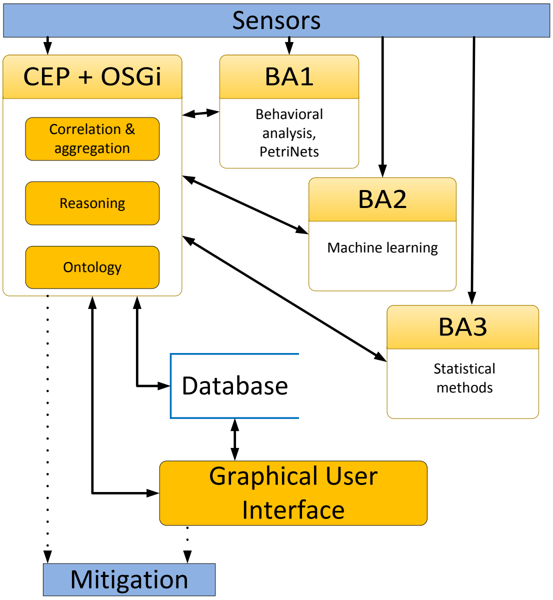


Figure 4. Architecture of the SECOR system.

4.2. Component description

A network flow is a unidirectional or bidirectional sequence of packets transferred between the source and destination. A sequence is described by certain common attributes. The most important key fields include: first and last time of flow received, source/destination IP address, source/destination port number, OSI Layer 3 protocol type, types of services, bytes transferred, and input logical interface [11]. The list of attributes can be further extended in most scenarios, and its contents depend completely on the device, exporter, and importer capabilities.

In our analysis, we focus on monitoring network flows in the form of Cisco's NetFlow or IPFIX records. We assume that data is collected from a large number of devices (flow probes) placed in several distant locations.

The sources of data could be firewalls, network switches or routers, and (in some cases) software daemons monitoring particular hosts or a virtualized infrastructure.

NetFlows are collected by FlowCollectors periodically (e.g., every 5 minutes), stored, and further processed (cf. Fig. 5).

In the next section, a novel representation of flows is proposed in the form of a graph. The graph is stored in a dedicated graphical, NoSQL like, database – Neo4j [18, 13].

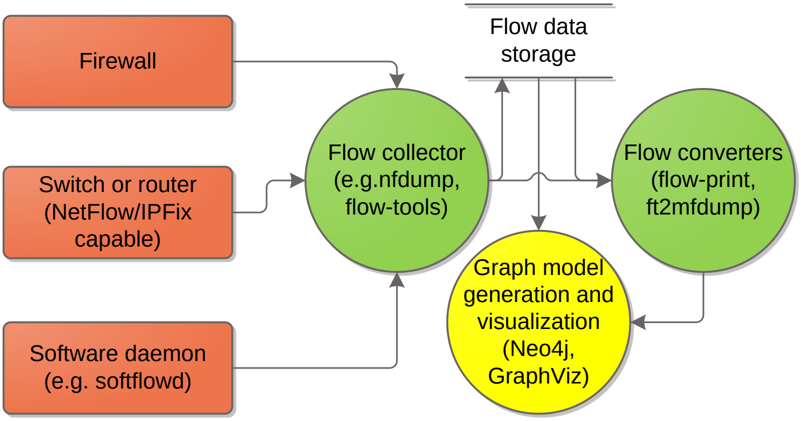

Firewall
Switch or router (NetFlow/IPFix capable)
Software daemon (e.g. softflowd)
Flow data storage
Flow collector (e.g.nfdump, flow-tools)
Flow converters (flow-print, ft2mfdump)
Graph model generation and visualization (Neo4j, GraphViz)

Figure 5. NetFlow data processing in one of the SECOR components.

NetFlows and graphs The idea of a graph model for NetFlows is to build a social network of communicating hosts (clients and services), and observe how the topological and quantitative properties of the graph (representing the interactions between them) change over time.

The proposed graph model (see Figure 6) is a trade-off between simplicity and the technical constraints of graph databases; in particular, Neo4j (where it is implemented in our case).

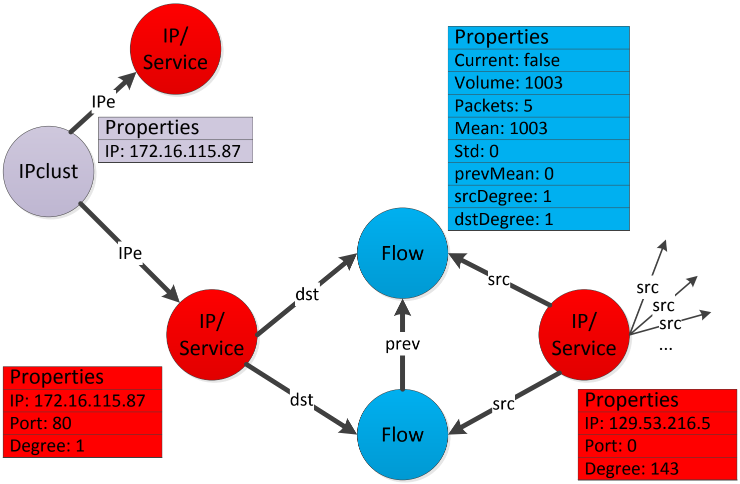


Figure 6. A graph-based model for the representation of NetFlow/IPFIX data.

The analysis within the proposed model is mainly conducted using Cypher queries. Cypher is a specialized query language [8] designed by the Neo4j development team. This language allows for relatively easy analysis of the graph – example queries for our model are shown in Chapter 6.

Within the model, several types of vertices and edges can be distinguished. Clients and services are represented by *IP/Service* vertices: the client has the port number set to zero in the applied convention, and the service: a number other than zero. On a host, there could be multiple services, or a host may simultaneously be client and service provider. In such cases*IPclust*, nodes are used to represent this fact. To be precise, the *IPclust* vertex is an artificial vertex that is used to represent clusters of services on the same host. The *IP/Service* vertices represent particular services or client machines. The *Flow* vertices represent individual flows along with aggregated data from preceding flows. The *src* and *dst* directed edges are used to link particular *Flows* with their source and destination machine or service. The *IPe* edges are used to show that a particular service belongs to a particular cluster (e.g., a particular host). The *prev* edge denotes a relationship between flow records gathered in consecutive periods of time. Additionally, three kinds of *TO* edges connecting clients with services are introduced (tcp, udp, and other).

The Neo4j database can easily be used to search for particular kind of traffic patterns; e.g., the existence of extensively used services running on high ports (tcp/udp port number greater than 1024), to analyze historical NetFlow data taking into account standard deviation, and to identify scan targets. However, the main advantage of a graph database over a traditional relational database is the built-in support for operations typical for graphs; for example, path traversal, identification of shortest-paths, etc. Thus, it is easier to retrieve data about possible attack paths; for example, in case of forensic investigation. In Neo4j, it could be achieved out-of-the-box by one simple query; in MySQL, it is impossible without external software or utilizing stored procedures.

In SECOR, the query results are gathered using JDBC, packed into a STIX-compliant format, and sent to the CEP engine as an event. The queries can be customized and are executed in regular periods of time (every 5 minutes by default) whenever a new dump of flow records is received from flow probes. The flows are analyzed in sliding windows (2 hours long), and a new graph is generated each time new flow records are available (every 5 minutes). The statistical information about each analyzed graph is stored in the relational database. Thus, in addition to Cypher queries, it is also possible to analyze statistical profiles of graphs in various time periods. It is worth noting that batch processing of netflows and regeneration of the whole Neo4j graph database using low-level API is over two orders of magnitude (over 100x times) faster than updating the graph online using the transactional API and Cypher merge command (in our experiments, the difference was 30s vs. 4h for 373,000 flow records).

4.3. Netflows – example

A Netflow-based approach is being tested in a virtualized testbed consisting of:

- a dozen Linux boxes with fprobe software (flow probe) installed
- server with nfcapd daemons (NetFlow collector) – one instance of a daemon on the server for one monitored host
- nfdump for reading, aggregating, and converting flow records from the binary format to a properly formated text
- Perl scripts importing flow records from nfdump text format into specially designed .csv files storing graph model in the format used by batch-importer
- batch-importer written in Java and downloaded from Neo4j repositories to import data to Neo4j database
- dedicated query engine written in Java that sends a predefined set of Cypher queries to the database in order to predict malicious activities
- bash scripts to combine all pieces together
- the attackers machine (IP address 192.168.101.100) with Kali Linux

The testbed may be used to simulate various scenarios; in this example, the attention is focused on only one.

Frequently (one of the earliest indicators of an impending network attack is the presence of network reconnaissance[1]), certain attacks are preceded by a reconnaissance phase. Based on this information, we focused on detecting specific symptoms indicating that a ground of attack is being prepared and the initial phase of the attack is underway. One of these symptoms is a network scan. Network scanners (network mappers) use a number of methods to discover hosts and enumerate running services (open ports). In order to gather as much data about the target as possible, they analyze server responses, response times, headers, etc. Traces of such activities are hidden in the flow records. With the help of the elements described above and using data gathered from our testbed, we were able to successfully discover and qualify similar types of malicious activity.

Below, the query used in the detection process is shown along with the obtained results. As it is easy to note, the Source IP address is identical to the attackers IP address in the SECOR testbed. The query returns TOP 5 (LIMIT 5) connection initiators (n:IPnode) that established unique connections to over 10,000 (WHERE clause) services. In this case, only one host meets these criteria.

```
>>Executing query:
START n=node(*)  MATCH (n:IPnode)-[:TO]->(c:IPnode)  WITH
  n as IPnode,
  n.ip_port as Source,
  count(*) as Connections,
  "Potential Scan Source" as Tag,
  "0.9" as Confidence
WHERE Connections >10000 RETURN IPnode, Source,Connections,Tag,Confidence
ORDER BY Connections DESC LIMIT 5

>>Results:
IPnode: {
  "ip":"192.168.101.100",
  "port":0.0,
  "ip_port":"192.168.101.100:0",
  "type":"IS"
},
Source: 192.168.101.100:0,
Connections: 26981,
Tag: Potential Scan Source,
Confidence: 0.9
```

In our test for 300k flows, it takes less than 30s to generate a graph and send a set of six queries returning information about the most-active hosts. The restart of the database server takes 8s on average; the execution of queries, 12s; import from the text file with flows exported using nfdump to the new Neo4j database, 10s. For 1k flows, the overall time is 19s. The tests were performed on a four-processor machine with Intel(R) Xeon(R) CPU E5-2603 v2 @ 1.80GHz processor and 16GB of RAM. For 300k flows, around 98k edges and 49k nodes were generated.

[1] Detecting Network Reconnaissance Guide – Cisco

The representation of NetFlows in the form of a graph database along with a set of queries can be successfully used as the source of information for CEP.

4.4. Results

Using the Complex Event Processing approach allows us to identify meaningful patterns by analyzing multiple event streams, fulfilling carefully predefined conditions during the defined time windows. A dedicated query language is used to represent the rules in programmer-familiar SQL-like grammar. Its support for temporal windows performs well in detecting anomalies. By combining several modules that operate on different levels, a very important piece of information (i.e., attack confirmation) is gained. SECOR contains a number of Blocks of Analysis, and each of them acts on a different level and with a different dataset. In order to compromise the particular network, intruders have to touch several of these levels: SECOR takes advantage of this fact by monitoring seemingly separate elements. However, when the anomalies are put together, the indication of intrusion is confirmed (which significantly reduces the number of false positives [understood as unnecessarily raised alarms]). This type of correlation has great potential in terms of detection capabilities, but further research in this area is required to obtain a reliable and efficient detection rate.

This approach has also a drawback when the number of sensors is limited – not only may the separated events be correlated improperly, but the attack may become unnoticed. This proves once again that applying a security-in-depth strategy is crucial to keep the security level high.

5. Conclusions and future work

Modern infrastructure is becoming more complex, and the consumed network throughput plus the potential volume of monitoring data are constantly increasing. This situation creates an opportunity for Complex Event Processing to be applied.

The two described use cases certainly do not exhaust the number of possible approaches to network monitoring and anomaly detection that may be successfully used in connection with CEP.

The SECOR project is an example of such a use case; and due to its modular architecture, its functionality may be further enhanced in order to employ different scenarios and to cover more attack patterns. Because the correlation engine may accept data from arbitrary blocks of analysis, further developments to the system are planned in the future (either by using the same methodologies [accordingly adjusted] to detect new types of cyber threats, or by applying different decision-support methods).

Another aspect, however purely technical it may be, is the appropriate configuration and instrumentation of the prepared environment in order to achieve the best performance indicators. The environment built contains a lot of configuration parameters, where modifying one or a certain combination of parameters may modify the processed events rate even by 1-2 orders of magnitude. Additionally, optimal configuration varies through different scenarios. Work on deriving a general optimal configuration and automatically adjusting it to particular scenarios would, therefore, be useful.

Using anomaly detection in conjunction with complex-event processing for advanced network monitoring is a great tool, and its capabilities may still be further developed and enhanced. We envision, however, that it may be significantly more useful for static environments where changes are rarely made and routine actions are common; thus, rendering anomalies noticeable.

The integration with Intrusion Detection Prevention systems, firewalls, and other protection systems should be taken into account to enable quick reactions to the detected threats.

The strength of CEP lies in the numbers: analyzing data acquired from numerous heterogeneous systems and information sources located all over the network allows us to draw more-certain conclusions on the overall state of the infrastructure and increases the protection level of the protected environment.

Concluding this study, we find the application of CEP for detecting particular types of cyber threats as promising in general, but still requiring thorough research and instrumentation per each application scenario in order to achieve the desired results.

6. Examples of Cypher queries

IP/Service vertices are tagged with the *IPnode* label for simplicity purposes.

Example 1. Identification of services listening on high ports and their clients.
Cypher query:

```
MATCH (ip:IPclust)-->(s:IPnode)--> (f:Flow {current:true})<--(d:IPnode)
 WHERE d.port >1024
RETURN DISTINCT s.ip,d.ip;
```

Results of the query:

s.ip	d.ip
172.16.114.168	194.27.251.21
172.16.114.168	197.182.91.233
172.16.114.168	195.115.218.108
172.16.114.50	194.27.251.21
172.16.114.50	197.218.177.69
172.16.114.50	195.115.218.108

Example 2. Analysis of historical NetFlow data to spot probable anomalies in windows of 10 records by the analysis of the distance from the mean flow volumes, taking into account standard deviation.

Cypher query:

```
MATCH (src:IPnode)-[:src]->(c:Flow {current:true})-[:prev*1..10]->(p:Flow)
                                       <-[:dst]-(dst:IPnode)
 WHERE c.nf=p.nf+10 and p.nf>5 and c.mean>p.mean
RETURN src.ipport,dst.ipport,c.nf,c.mean,p.mean,
                max(c.mean-(p.mean+2*sqrt(p.std/(p.nf-1))))
                as delta_mean
ORDER BY delta_mean DESC;
```

Results of the query:

src.ipport	dst.ipport	c.nf	c.mean	p.mean	stdev	Δ_{mean}
172.16.113.105:0	197.218.177.69:20	19	729	185	97.588	348.82
172.16.114.169:0	199.123.32.60:80	18	876	592	116.6	50.80
172.16.113.105:0	204.146.18.33:80	24	517	500	1.4936	14.17
172.16.115.5:0	207.90.155.39:80	30	540	531	0.3974	9
172.16.114.168:0.0	207.25.71.142:80	17	466	446	5.9301	8.17
172.16.117.103:0	205.180.59.51:80	17	471	440	12.517	6.02
172.16.117.103:0	206.132.25.41:80	17	484	470	4.1433	5.75
		...				

Acknowledgements

This work was partially supported by the Applied Research Programme (PBS) of the National Centre for the Research and Development (NCBiR) funds allocated for the Research Project number PBS1/A3/14/2012 (SECOR).

References

[1] Balis B., Kowalewski B., Bubak M.: Leveraging Complex Event Processing for Grid Monitoring. In: *Parallel Processing and Applied Mathematics*, R. Wyrzykowski, J. Dongarra, K. Karczewski, J. Wasniewski, eds, *Lecture Notes in Computer Science*, vol. 6068, pp. 224–233. Springer, Berlin-Heidelberg, 2010. `http://dx.doi.org/10.1007/978-3-642-14403-5_24`.

[2] Balis B., Kowalewski B., Bubak M.: Real-time Grid monitoring based on complex event processing. *Future Generation Computer Systems*, vol. 27(8), pp. 1103–1112, 2011. `http://www.sciencedirect.com/science/article/pii/S0167739X11000562`.

[3] Bereziński P., Pawelec J., Małowidzki M., Piotrowski R.: Entropy-Based Internet Traffic Anomaly Detection: A Case Study. In: *Proceedings of the Ninth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX. June 30 – July 4, 2014, Brunów, Poland*, *Advances in Intelligent Systems and Computing*, W. Zamojski, J. Mazurkiewicz, J. Sugier, T. Walkowiak, J. Kacprzyk, eds, vol. 286, pp. 47–58. Springer International Publishing, 2014. `http://dx.doi.org/10.1007/978-3-319-07013-1_5`.

[4] Bilge L., Dumitras T.: Before We Knew It: An Empirical Study of Zero-Day Attacks In The Real World. *Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 833–844, 2012. `http://users.ece.cmu.edu/~tdumitra/public_documents/bilge12_zero_day.pdf`.

[5] EGEE – Enabling Grids for E-sciencE, 2010. `http://eu-egee.org`.

[6] Frankowski G., Jerzak M.: Advanced Architecture of the Integrated IT Platform with High Security Level. In: *Multimedia Communications, Services and Security, Communications in Computer and Information Science*, A. Dziech, A. Czyżewski, eds, vol. 287, pp. 107–117. Springer, Berlin-Heidelberg, 2012. `http://dx.doi.org/10.1007/978-3-642-30721-8_11`.

[7] GÉANT: the pan-European research and education network, 2014. `http://www.geant.net`.

[8] Holzschuher F., Peinl R.: Performance of Graph Query Languages: Comparison of Cypher, Gremlin and Native Access in Neo4J. In: *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, EDBT '13, pp. 195–204. ACM, New York, NY, USA, 2013. `http://doi.acm.org/10.1145/2457317.2457351`.

[9] Jerzak M., Wojtysiak M.: Distributed Intrusion Detection Systems – MetalDS case study. *Computational Methods in Science and Technology*, Special Issue (1), pp. 135–145, 2010.

[10] Kliarsky A., Atlasis A.A.: Responding to Zero Day Threats, 2011. `http://www.sans.org/reading-room/whitepapers/incident/responding-zero-day-threats-33709`.

[11] Li B., Springer J., Bebis G., Gunes M.H.: A survey of network flow applications. *Journal of Network and Computer Applications*, vol. 36(2), pp. 567–581, 2013. `http://www.sciencedirect.com/science/article/pii/S1084804512002676`.

[12] Lodi G., Aniello L., Luna G.A.D., Baldoni R.: An event-based platform for collaborative threats detection and monitoring. *Inf. Syst.*, vol. 39, pp. 175–195, 2014. `http://dblp.uni-trier.de/db/journals/is/is39.html#LodiALB14`.

[13] Neo4j: Neo4j – The World's Leading Graph Database, 2012. `http://neo4j.org/`.

[14] Security Experts Expect 'Shellshock' Software Bug in Bash to Be Significant, 2014. `http://www.nytimes.com/2014/09/26/technology/security-experts-expect-shellshock-software-bug-to-be-significant.html`.

[15] PIONIER, 2014. `http://www.pionier.net.pl`.

[16] Polish Platform for Homeland Security, 2014. `http://www.ppbw.pl/en`.

[17] Poznań Supercomputing and Networking Center, 2014. `http://www.psnc.pl`.

[18] Robinson I., Webber J., Eifrem E.: *Graph Databases.* O'Reilly Media, Inc., 2013.

[19] Storm, Distributed and fault-tolerant realtime computation, 2014. `http://storm.apache.org`.

[20] Symantec Corporation: Internet Security Threat Report 2014, 2014. `http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_v19_21291018.en-us.pdf`.

[21] The Apache Software Foundation: mod_log_config: CustomLog Directive, 2014. `https://httpd.apache.org/docs/2.4/mod/mod_log_config.html#customlog`.

[22] WSO2 Carbon System, 2005. `http://wso2.com/products/carbon/`.

[23] WSO2 Siddhi CEP engine, 2005. `http://siddhi.sourceforge.net/`.

Affiliations

Gerard Frankowski
Poznań Supercomputing and Networking Center, Institute of Bioorganic Chemistry, Noskowskiego 10, 61-704 Poznań, Poland, `Gerard.Frankowski@man.poznan.pl`

Marcin Jerzak
Poznań Supercomputing and Networking Center, Institute of Bioorganic Chemistry, Noskowskiego 10, 61-704 Poznań, Poland, `Marcin.Jerzak@man.poznan.pl`

Maciej Miłostan
Poznań Supercomputing and Networking Center, Institute of Bioorganic Chemistry, Noskowskiego 10, 61-704 Poznań, Poland; Institute of Computing Science, Poznań University of Technology, Piotrowo 2, 60-965 Poznań, Poland, `Maciej.Milostan@man.poznan.pl`

Tomasz Nowak
Poznań Supercomputing and Networking Center, Institute of Bioorganic Chemistry, Noskowskiego 10, 61-704 Poznań, Poland, `Tomasz.Nowak@man.poznan.pl`

Marek Pawłowski
Poznań Supercomputing and Networking Center, Institute of Bioorganic Chemistry, Noskowskiego 10, 61-704 Poznań, Poland, `Marek.Pawlowski@man.poznan.pl`

Received: 30.11.2014
Revised: 10.01.2015
Accepted: 18.01.2015