Security Modeling of Autonomic-Component Ensembles

Irakli RODONAIA*
Medhat MOUSA**
Vakhtang RODONAIA***

Abstract

Autonomic systems exhibit the ability of self-monitoring, self-repairing, and self-optimizing by constantly sensing themselves and tuning their performance. The notions of autonomic components (ACs) and autonomic-component ensembles (ACEs) are considered in the paper. A language for coordinating ensemble components (SCEL) is used to represent specificity of security issues in autonomic computing environment. To reveal abnormal behavior of ACEs the information theory metric (in particular, Kullback –Leibler divergence) is proposed to use in the approach described in the paper.

Keywords: Cloud computing, autonomic component, autonomic ensemble, formal modeling, security, information theory metrics

Introduction

The notions of autonomic components (ACs) and autonomic-component ensembles (ACEs) (ASCENS,2010), (Rocco De Nicola et al, 2013) have been put forward as a means to structure a system into well understood, independent and distributed building blocks that interact in specified ways. ACs are entities with dedicated knowledge units and resources that can cooperate while playing different roles. Awareness is made possible by providing ACs with information about their own state and behavior that can be stored in their knowledge repositories. These repositories also enable ACs to store and retrieve information about their working environment, and to use it for redirecting and adapting their behavior. Each AC is equipped with an interface, consisting of a collection of attributes, such as provided functionalities, spatial coordinates, group memberships, trust level, response time, etc.

Attributes are used by the ACs to dynamically organize themselves into ACEs. Individual ACs not only can single out communication partners by using their identities, but they can also select partners by exploiting the attributes in the interfaces of the individual ACs. Predicates over such attributes are used to specify the targets of communication actions, thus providing a sort of attribute-based communication. In this way, the formation rule of ACEs is endogenous to ACs: members of an ensemble are connected by the interdependency relations defined through predicates. An ACE is therefore not a rigid fixed network but rather a highly dynamic structure where ACs' linkages are dynamically established.

$$S ::= C \mid S_1 \parallel S_2 \mid (\nu n)S \quad \textbf{Systems}$$

$$C ::= \mathcal{I}[\mathcal{K}, \Pi, P] \quad \textbf{Components}$$

$$P ::= \mathsf{nil} \mid a.P \mid P_1 + P_2 \mid P_1[P_2] \mid X \mid A(\bar{p}) \quad \textbf{Processes}$$

$$a ::= \mathbf{get}(T)@c \mid \mathbf{qry}(T)@c \mid \mathbf{put}(t)@c \mid \mathbf{fresh}(n) \mid \mathbf{new}(\mathcal{I}, \mathcal{K}, \Pi, P) \quad \textbf{Actions}$$

$$c ::= n \mid x \mid \mathsf{self} \mid \mathsf{P} \mid \mathcal{I}.\mathsf{p} \quad \textbf{Targets}$$

Table 1: SCEL syntax

The proposed abstractions are the basis of SCEL (Software Component Ensemble Language), a kernel language for programming autonomic computing systems.

The syntax of SCEL is presented in Table 1. The basic category of the syntax is that relative to PROCESSES that are used to build up COMPONENTS that in turn are used to define SYSTEMS. PROCESSES specify the flow of the ACTIONS that can be performed. ACTIONS can have a TARGET to characterize the other components that are involved in that action.

PROCESSES are the active computational units. Each process is built up from the inert process nil via action prefixing (a.P), nondeterministic choice (P1+P2), controlled composition (P1[P2]), process variable (X), and parameterized process invocation (A($\bar{p}$)). The construct P1[P2] abstracts the various forms of parallel composition commonly used in process calculi. Process variables can support higher-order communication, namely the capability to exchange (the code of) a process, and possibly execute it, by first adding an item containing the process to a knowledge repository and then retrieving/withdrawing this item while binding the process to a process variable. We assume that A ranges over a set of parameterized process identifiers that are used in recursive process definitions. We also assume that each process identifier A has a single definition of the form A($\bar{f}$) $\triangleq$ P where all free variables in P are contained in f and all occurrences of process identifiers in P are within the scope of an action prefixing. and denote lists of actual and formal parameters, respectively.

Processes can perform five different kinds of ACTIONS. Actions **get**(T)@c, **qry**(T)@c and **put**(t)@c are used to manage shared knowledge repositories by with drawing/retrieving/adding information items from/to the knowledge repository c. These actions exploit templates T as patterns to select knowledge items t in the repositories. They rely heavily on the used knowledge repository and are implemented by invoking the handling operations it provides. Action fresh (n) introduces a scope restriction $\mathcal{I}[\mathcal{K}, \Pi, P]$ for the name n so that this name is ensured to be different from any other name previously used.

Action new creates a new component $\mathcal{I}[\mathcal{K}, \Pi, P]$ .An autonomic component $(\mathcal{I}, \mathcal{K}, \Pi, P)$ is graphically depicted in Figure 1 (next page).

* Prof., Faculty of Computer Technologies and Engineering, International Black Sea University, Tbilisi, Georgia. E-mail: irakli.rodonaia@ibsu.edu.ge
** Ph.D. student , Faculty of Computer Technologies and Engineering, International Black Sea University, Tbilisi, Georgia.
E-mail: mmousa@ibsu.edu.ge
*** Lecturer, Faculty of Computer Technologies and Engineering, International Black Sea University, Tbilisi, Georgia. E-mail: vrodonaia@ibsu.edu.ge

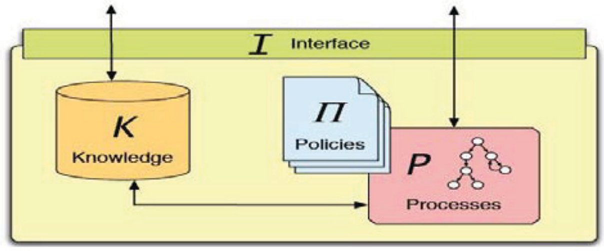

Figure 1: *SCEL components*

• An interface $\mathcal{I}$ publishing and making available structural and behavioral information about the component itself in the form of attributes. Among them, attribute id is mandatory and is bound to the name of the component. Notably, component names are not required to be unique; this would allow us to easily model replicated service components.

• A knowledge repository $\mathcal{K}$ managing both application data and awareness data, together with the specific handling mechanism. The knowledge repository of a component stores also the whole information provided by its interface, which therefore can be dynamically manipulated by means of the operations provided by the knowledge repositories' handling mechanisms

• A set of policies Π regulating the interaction between the different internal parts of the component and the interaction of the component with the others.

• A process P together with a set of process definitions that can be dynamically activated. Some of the processes in P perform local computation, while others may coordinate processes interaction with the knowledge repository and deal with theissues related to adaptation

Finally, SYSTEMS aggregate COMPONENTS through the composition operator $- \parallel -$.

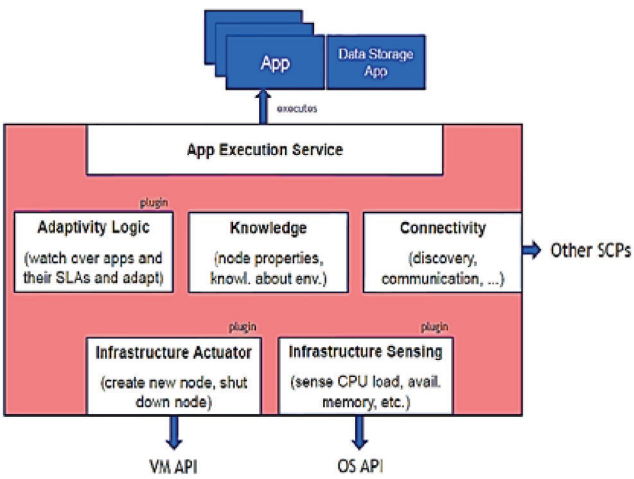

Figure 2: *Functional description of the component*
The operational and system semantics of SCEL is described in detail in (Rocco De Nicola et al., 2013).

Access control is a fundamental mechanism for restricting what operations users can perform on protected resources. Many models of access control have been defined in the literature. One of them is the Policy Based Access Control model (NIST, 2009). In this model, a request to access a protected resource is evaluated with respect to one or more policies that define which requests are authorized. An authorization decision is based on attribute values required to allow access to a resource according to policies stored in system's components. Component attributes are here used to describe the entities that must be considered for authorization purposes. On this basis the SACPL (SCEL Access Control Policy Language), a simple, yet expressive, language for defining access control policies and access requests, is considered (Rocco De Nicola et al, 2013)

Policies are hierarchically structured as trees. A policy is either an atomic policy or a pair of simpler policies combined through one of the decision combining operators p-o (permit override) and d-o (deny override). An atomic policy is a pair made of a decision and a target. The target defines the set of access requests to which the policy applies. The decision, i.e. permit or deny, is the effect returned when the policy is 'applicable', namely the access request belongs to the target. Otherwise, i.e. when a request does not belong to the policy's target, the policy is 'not-applicable', which in this simplified setting has the same effect as deny. A target is either an atomic target or a pair of simpler targets combined using the standard logic operators and and or. An atomic target is a triple denoting the application of a matching function to values from the request and the policy, like e.g. greater-than(subject.skill; threshold – object.dependability). Finally, Expressions are built from values and attributes through various operators. SACPL requests, ranged over by ρ, are functions mapping names to elements and are written as collections of pairs of the form (name; element). A request's element can be a knowledge item, a component's interface, the type of an action, etc. In its turn, an interface provides a set of attributes characterizing the corresponding component, which can be either the subject or the object of the request. A typical example of request is as follows:

$$\rho = \{(\mathsf{subject}, \mathcal{I}), (\mathsf{item}, t), (\mathsf{action}, \mathsf{get}), (\mathsf{object}, \mathcal{J})\}$$

Here, the subject identified by the interface requires the authorization to withdraw the item t from component . For example, the request's subject is obtained by calling ρ (subject), which returns .

Autonomic computingis widely used in spatial-temporal data analysis for online prediction of dengue fever outbreaks (R. Buyya et al.,2009), the science cloud (P. Mayer et al., 2012), real time collection and dissemination of personal health data (ECG: electrocardiograms) to patients and health-care professionals (Suraj Pandey et al., 2012), etc.

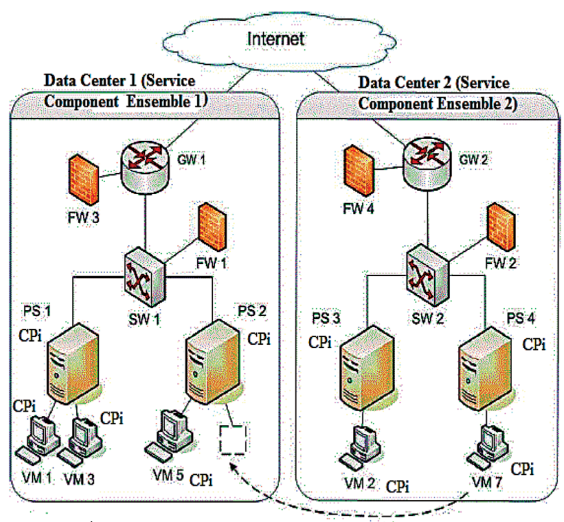

Figure 3: *Cloud Platform*

Cloud computing in these different areas can be thought of as a collection of desktops, servers, notebooks or virtual machines running the Cloud Platform (CP). Each (virtual) machine is running one instance of the Cloud Platform called Cloud Platform instance (CPi).Each CPi is considered to be a service component. Multiple CPs communicate over the Internet (IP ptotocol), thus forming a cloud and within this cloud one or more service component ensembles (*Figure 3*).

Each CPi has knowledge. The knowledge consist of what the CPi knows about itself (properties set by developers), about its infrastructure (CPU load, available memory), and about other CPis (acquired through the network). Since there is no global coordinator, each CPi must build its own world view and act upon the knowledge available. The CPi may acquire knowledge about its infrastructure using an infrastructure sensing plug-in, which provides information about static values such as processor speed, available memory, available disk space, number of cores etc. and dynamic values such as currently used memory, disk space, CPU load, characteristics of current traffic flowing through the CPi,etc).

Each CPi has also a connectivity component which enables it to talk to other CPis over the network. The protocol followed by these communications must enable CPis to find one another and establish links, for example by manually entering a network address or by a discovery mechanism. Furthermore, CPis must be able to query others for knowledge and distribute their own. Finally, the protocol must support exchange of data and applications.

Each CPi is considered to be autonomic in the sense that it may join and leave the cloud at will. The cloud is thus a dynamic cloud and works *without a central coordinator* in a peer-to-peer manner.

1. Statement of the problem

The following scenario is considered. A singleton application currently runs on one of the VMs at Data Center 2 (VM7 in Service Component Ensemble 2. This application runs alone on its node and,since the application is a singleton, no additional instances can be spawned. During the sessionthe application experiences consistently high CPU load. This increase may be caused either by legitimate traffic overload or by coordinated attacks (DDOS) launched against the PaaS provider. The latter might be wrongly assumed to be legitimate requests and resources would be scaled up to handle them. This would result in an increase in the cost of running the application (because provider will be charged by these extra resources) as well as a waste of energy. Hence, it is necessary to distinguish between these two cases, the earlier this distinction is made, the higher is the degree of protection of the application from failure and poor performance.To provide this protection, the following security measures are suggested. The traffic flows through the node (CPi) has to be analyzed using Kolmogorov complexity metrics (see later in the text). During the session the constant monitoring of the metric (by the special probe implemented in the separate module), along with measure of CPU load, is being executed. If the simultaneous increase of these two metrics is registered at least in 3 successive time units, the conclusion about the real treat of the DDOS attack must be drawn. As a result, the application has to migrate from the CPi where it was running to another CPi (which may belong to the same ensemble or other ensemble). A new CPi must be found according to some requirements: complexity level and CPU load must be rather low, integrated hardware index (which includes such indicators as processor speed, available memory, available disk space, number of cores, etc) must correspond to the application resource requirements (they are published in the interface of the CPi where the application is running). If the required CPi is found, the application has to migrate there as soon as possible and stop its running on the "old" CPi.

We assume that, other than id, the interfaces provide the attributes "KLDiv", "CPULoad" and "Hardware" stores a context information, updated by the underlying infrastructure (usually, from the firewalls, gateways or special probes) and are `sensed' by the managed element.

The CPi where the application is running is the SCEL component: $\mathcal{I}[\mathcal{K},\Pi,\mathcal{AM}[\mathcal{ME}]]$

The autonomic manager AM is defined as follows:

AM ≜ $P_{KLDivMonitor}$ [$P_{CPULoad}$]

$\mathbf{P}_{KLDivMonitorr}$ ≜ qry("*KLDivLevel*", "high") @ self.

get("*KLDivHigh*", false) @self.

put(("*KLDivHigh*", true) @self.

qru("*KLDivLevel*","low")@self.

get("*KLDivHigh*",true)@self.

put("*KLDivHigh*",false)@self.PKLDivMonitor

$\mathbf{P}_{CPULoad}$ ≜ qry("*CPUloadLevel*","low")@self.**get** ("*CPULow*", false) @self.

put(("*CPULow*", true) @self. **qru**("CPUloadLevel"," high")@self.

get("*CPULow*",true)@self.

put("*CPULow*",false)@self. $P_{CPULoad}$

$\mathbf{P}_{MigrateCP\,i}$ ≜ **qry**("*required_functionality_id*", ?X)@ self.

/* retrieving from the knowledge repository the process implementing a required functionality id and bounding it to a process variable X */

get("*required_functionality_id_args*", ?y,?z) @self.

qry("*CPiId*", ?c) @ Ω . /* searching an item c among components belonging to the ensemble identified by predicate Ω */

fresh(n). /* fresh name n is used for coordination purposes */

put("*required_functionality_id_params*",n,y,z)@c /* storing actual parameters of theprocess to be executed in the found component c : moving from VM7 to MV5 on fig.2 */

get("*required_functionality_id*", "*terminated*",n) @self. /* removing the process from the knowledge repository of 'old' CPi */

get("required_functionality_id", X) @self.nil

/* eliminating the process in 'old' CPi */

Here the predicate Ω is determined as follows:

Ω ($\mathcal{I}$) = ($\mathcal{I}$. *KLDiv*Level="low") ($\wedge$ $\mathcal{I}$ .CPU-Load <75)

∧ ($\mathcal{I}$ .Hardware>=5))

and is used for group-oriented communication in the action **qry**("CPiId", ?c) @ Ω . This predicate defines the ensemble of components which publish in their interfaces attributes "KL-DivLevel", "CPULoad" and" Hardware"along with relevant values. We assume that these attributes are provided by the interface of each component and obtain dynamically updated values from corresponding probes (sensors) as a result of constant monitoring (sensing) of the computing environment.

We assume also that the attribute "*KLDiv*Level" (*Kullback-Leibler divergence*) gives an indication in the range [0:m] , ***m*** - some positive real number (see explanation below in the text) of data flow through the ensemble, the attribute "CPU-Load" – in the range [0:100], the attribute "Hardware" – in the range [0:10]. In this context the meaning of the predicate is as follows: find a component CPi (or components) where the "ComplexityLevel" is low (i.e. less than 0.8), "CPULoad" is less than 75 and integrated hardware index "Hardware" is more than 5.

The process P_s executed by the managed element ME is:

$\mathbf{P_s} \triangleq$ **get** (*"required_functionality_id_params"*,?id, ?y, ?z)@self.

get("load", ?l) @self.

get("hardware", ?h) @self.

put("load", (l+5))@self.

put("hardware", (h-10))@self.

P_s [X(id, y, z)] /* the new process (additionally to the already running process P_s), having actual parameters id, y, z,starts */

The policy $\prod$ in force at the component results from the composition, by means of the

p-o (*permit override*) and **d-o** (*deny override*) operators, of the following policies:

⟨deny; target:{} *⟩ deny all *

e permit ; target:{equal(subject: id; n) and *permit local **qry***
equal(object: id; n) and
equal(action; **qry**) and
equal(subject: *KLDiv*Level; *level*) and
less-or-equal-than(CPULoad;
treshold)} ⟩

⟨ permit ; target:{equal(subject: id; n) and * permit remote **qry** *
equal(object: id; m) and
equal(action; qry) and
equal(subject: KLDivLevel; level) and
less-or-equal-than(CPULoad;
treshold)} ⟩

⟨ permit;target:{equal(subject:id;n) and * always permit local **put** *
equal(object: id; n) and
equal(action; **put**)} ⟩

⟨ permit;target:{equal(subject:id;n) and *always permit remote **put***
equal(object: id; m) and
equal(action; **put**) } ⟩

⟨ deny;target:{equal(subject: id; n) and *always deny remote **get** *
equal(object: id; m) and
equal(action; **get**) } ⟩

2. Detection of DDoS Attack Using Kullback-Leibler Divergence Metric.

In the approach to autonomous computing security and anomaly detection, developed in the paper, the notions of ***netflows***, their ***informational-theoretical metrics*** and components' autonomic ***manager*** are essentially leveraged.

A network flow can be defined in many ways. In a general sense, a flow is a series of packets with some attribute(s) in common. Each packet that is forwarded within a router or switch is examined for a set of IP packet attributes. These attributes are the IP packet identity or fingerprint of the packet and determine if the packet is unique or similar to other packets. All packets with the same source/destination IP address, source/destination ports, protocol interface, and class of service are grouped into a flow and then packets and bytes are labeled. This methodology of fingerprinting or determining a flow is scalable because a large amount of network information is condensed into a database of netflow information called the netflow ***cache*** (Introduction to Cisco IOS® NetFlow, 2012)

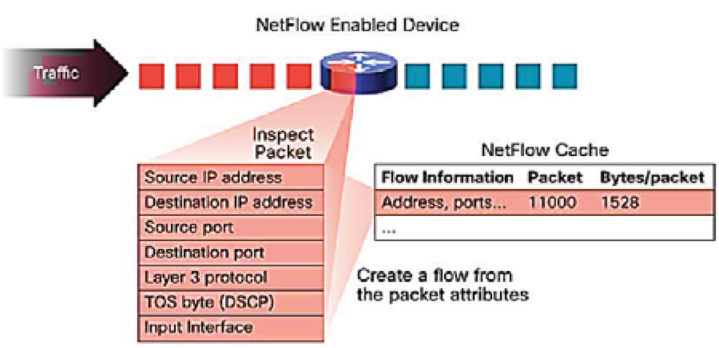

Figure 4: *Netflow schema*

A ***netflow-enabled device (netflow exporter:*** router or switch) (see the fig.4) sends to the ***netflow collector*** single flow as soon as the relative connection expires. This can happen when 1) when TCP connection reaches the end of the byte stream (FIN flag orRST flag) are set; 2) when a flow is idle for a specific timeout; 3) if a connection exceeds long live terms (30 minutes by default). Packets captured by the netflow collector are stored to a flow storage . As stated by (D. Vitali et al., 2012), port and address IP distributions are highly correlated

in network traffic. For this reason, we only considered source and destination IP.

Flows accumulated at the flow storage, are then subdivided into ***component flows***. That is, flows which have the component's IP address as a destination address are grouped and sent to the corresponding component (more exactly, to the autonomic manager of a component - these flows are marked with blue arrows on the *fig.5*). After receiving their destined flows, the component's ***autonomic manager*** can start the processing in order to reveal the abnormal behavior of flows in accordance with the following technique.

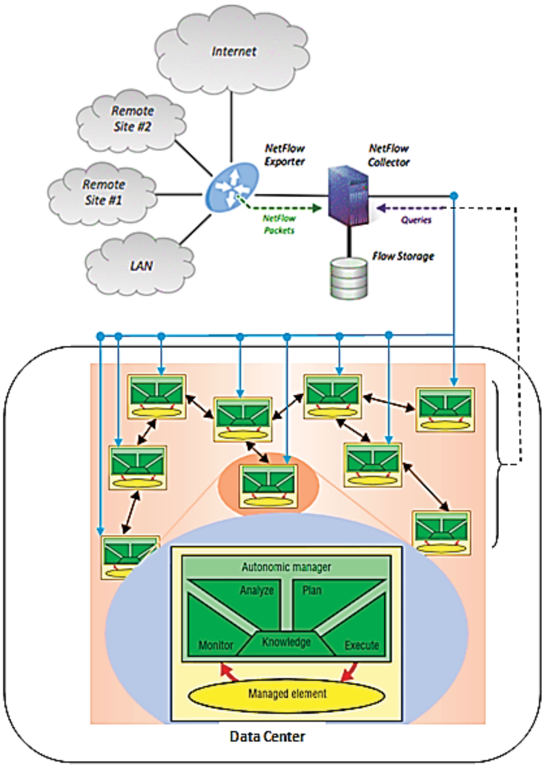

Figure 5: *Interaction between netflow devices and autonomic components*

Information theory based metrics enable sophisticated anomaly detections directly with the whole traffic that are difficult to provide with simpler metrics, like aggregated traffic workload, number of packets or single host traffic. The Kull-

$$D_{KL}(P||Q) = \ {}_{i}\ P(i)log\frac{P(i)}{Q(i)}$$

back-Leibler divergence equation (D. Vitali et al., 2012) is:
where the index i in front of the right part of the equation stands to denote: $\sum_i$

A low D_{KL} value means a high similarity in the two probability distributions, on the other hand, high divergence values correspond to low similarity. Port and address IP distributions are highly correlated in network traffic. For this reason we only considered source and destination IP. Network flows are aggregated into time blocks of a fixed size (1 minute by default). Let f^t be the number of flows that cross the monitored network in a time block. Let f_i^t be the number of flows that have IPi as source (or destination) address. We associate pi to the packet distribution over a time block t and qi to the packet distribution of the previous time block t −1:

$$D_{KL}(t||t+1) = \ {}_{i}\ \frac{f_i^t}{f^t}\log\frac{\frac{f_i^t}{f^t}}{\frac{f_i^{t-1}}{f^{t-1}}} = \ {}_{i}\ \frac{f_i^t}{f^t}\log\frac{f_i^t f^{t-1}}{f^t f_i^{t-1}}$$

The Kullback-Leibler divergence is computed as follows:

So if the D_{KL} during 2 succesive time moments is near to ***zero***, it means that patterns of IP addresses (source or destination) in packets are the same or very close. It can be considered as DDoS (or DoS) attacks (depending on combination of source od destination addresses patterns)

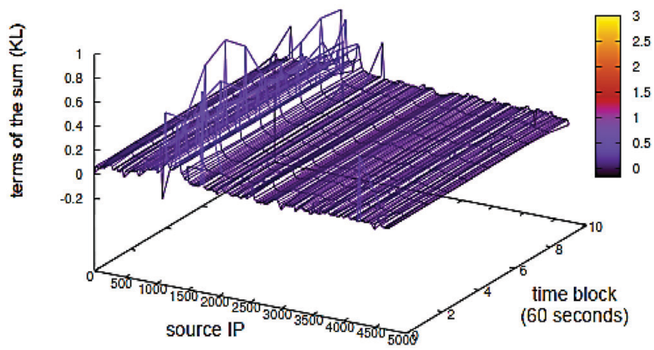

Figure 6: *Kullback Leibler details on source IP address*

On the fig.6 the plot for DDoS attack for some selected time interval is shown. It is characterized by a large number of attack sources. Altough the Kullback-Leiblet metric is predominantly close to zero (it means that the source and destination IP are the same during the specified time interval - this is very suspicious from the anomal behaviourstandpoint), there are numerous peaks in KUllback-Leibler metric's value This fact means that a great number of different IP source hosts are being connected the the attack.

Once the autonomic manager detected the anomaly, it generates the relevant commands and sends them to the managed element of the component (as it is described above in the section "Statement of the problem").

Conclusions

The following issues are considered in the paper:
- The notions of autonomic components (ACs) and autonomic-component ensembles (ACEs)
- Basic constructions of the SCEL (Software Component Ensemble Language) and SACPL (SCEL Access Control Policy Language) - Behaviors, Knowledge and Aggregations, according to specific Policies.
- A scenario of migrating of VM in condition of the DDOS threat
- An approach of detection of DDOS attack using Kullback-Leibler divergence metric

References

[1] ASCENS, P.: http://www.ascens-ist.eu/, (2010)

[2] Rocco De Nicola, Michele Loreti, Rosario Pugliese, Francesco Tiezzi (2013). "SCEL - a Language for Autonomic Computing". ASCENS project , Technical report, pp.10-38

[3] NIST: A survey of access control models, (2009), from http://csrc.nist.gov/news_events/ privilege-management-work-

shop/PvM-Model-Survey-Aug26-2009.pdf

[4] R. Buyya, R.N. Calheiros1, Xiaorong Li. (2009), "Autonomic Cloud Computing: Open Challenges and Architectural Elements", Cloud Computing and Distributed Systems (CLOUDS) Laboratory Department of Computing and Information Systems The University of Melbourne, Australia, pp. 1-20

[5] P. Mayer, C. Kroiss, J.V. (June 2012): Specification: The Science Cloud Case Study. Overview and Scenarios. Technical Report, pp.1-42

[6] Suraj Pandey, William Voorsluys, Sheng Niu, Ahsan Khandoker, Rajkumar Buyya. (2012),: An autonomic cloud environment for hosting ECG data analysis services. Future Generation Computer Systems 28 pp.147-154, Elsevier

[7] Introduction to Cisco IOS® NetFlow, 2012, from http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6555/ps6601/prod_case_study0900aecd80311fc2.pdf

[8] D.Vitali, A.Villani, A.Spognardi, R.Battistoni, L.Mancini, A. Villani, A. Spognardi, R. Battistoni1 and L. V. Mancini1 (2012), "DDoS Detection with Information Theory Metrics and Netflow (A real case), SECRYPT 2012-International Conference on Security and Cryptography", pp.172-181.