Anomaly Detection in the Cloud: Detecting Security Incidents via Machine Learning*

Matthias Gander[1], Michael Felderer[1], Basel Katt[1], Adrian Tolbaru[1], Ruth Breu[1], and Alessandro Moschitti[2]

[1] Institute of Computer Science, University of Innsbruck, Austria
[2] Information Engineering and Computer Science Department, University of Trento, Italy

Abstract. Cloud computing is now on the verge of being embraced as a serious usage-model. However, while outsourcing services and workflows into the cloud provides indisputable benefits in terms of flexibility of costs and scalability, there is little advance in security (which can influence reliability), transparency and incident handling. The problem of applying the existing security tools in the cloud is twofold. First, these tools do not consider the specific attacks and challenges of cloud environments, e.g., cross-VM side-channel attacks. Second, these tools focus on attacks and threats at only one layer of abstraction, e.g., the network, the service, or the workflow layers. Thus, the semantic gap between events and alerts at different layers is still an open issue. The aim of this paper is to present ongoing work towards a Monitoring-as-a-Service anomaly detection framework in a hybrid or public cloud. The goal of our framework is twofold. First it closes the gap between incidents at different layers of cloud-sourced workflows, namely we focus both on the workflow and the infrastracture layers. Second, our framework tackles challenges stemming from cloud usage, like multi-tenancy. Our framework uses complex event processing rules and machine learning, to detect populate user-specified metrics that can be used to assess the security status of the monitored system.

Keywords: Monitoring, Behaviour, Anomaly Detection, Clustering, Fingerprints.

1 Introduction

Building your own monolithic IT infrastructure is slowly rendered obsolete by cost efficient cloud solutions that promise on-demand scalability with leased

* This work is supported by QE LaB-Living Models for Open Systems (FFG 822740), and SECTISSIMO (FWF 20388) and has been partially supported by the European Community's Seventh Framework Programme (FP7/2007-2013) under the grants #247758: EternalS – Trustworthy Eternal Systems via Evolving Software, Data and Knowledge, and #288024: LiMoSINe – Linguistically Motivated Semantic aggregation engiNes.

hardware, i.e. by contracting Infrastructure as a Service (IaaS) provider such as Amazon's "elastic compute cloud" EC2 cloud) [1, 2]. Therefore it is not surprising that corporations opt to outsource IT related computing units, such as hosts or services, to such clouds (*cloud-sourcing*) to become cloud *tenants*. Leading analysts forecast a dramatic increase of cloud services revenue, i.e. Gartner, Inc. forecast Software as a Service (SaaS) to increase 17.9% from the 2011 revenue of $12.3 billion.[1] Cloud tenants though, often have to pay a price. Increased scalability of resources demands dynamical compositions of computing machinery resulting in design inherent weaknesses, for instance, tenants share the same cloud and are potentially allowed to interact by design [3].

This results in potentially hostile machines residing within the corporate network that has to be secured. Hostile machines on the network tear security holes in multiple layers of computation. Infrastructure items, such as hosts, can be broken into by a competing company to attain confidential information about its users and other data that is stored on the machine. This in turn allows workflows to be changed, i.e. by breaking in a system and patching the code-base or the platform itself [4, 5], or simply by reverse engineering workflows and creating rogue clients. A thusly changed workflow has semantical consequences on its logic, for instance, bypassed checks for sufficient funds in a credit card application, a compromised XACML (or Kerberos) infrastructure that grants authorizational access to restricted entities.

Another problem is that attacks themselves have become sneakier. Attackers tend to use more advanced techniques, and more persistence to eventually mask an attack as inside job[2]. For example, if credentials of legitimate service users are stolen and information is leaked gradually and persistently over a longer time period. Such attacks usually manifest in a change of behavior of entities involved in any given activity (e.g. behavioural changes observed in off-key working hours, spiking access over document data etc.).

To decrease the chance of successful attacks, security monitoring was introduced to analyse events committed by sensors in the corporate network. The analysis of events usually involves *signature-based* methods. Features, extracted from logged event data, are compared to features in attack signatures which in turn are provided by experts [6, 7]. Other approaches, e.g. *anomaly detection*, often make use of machine learning-based algorithms [8]. Anomalies are an unexpected event (or a series of unexpected events) that exhibit a significant change in behaviour of an entity, for example, a user. If anomalous behavior can be distinguished from normal behavior by hard bounds that are known beforehand, then signature-based approaches can be used to classify attacks immediately. However, when it is hard to specify all entities and their normal behaviour completely beforehand, then statistical measures have to be used to classify deviations in oder to detect possible attacks.

[1] `http://www.gartner.com/it/page.jsp?id=1963815`, Accessed: July 30, 2012.

[2] `http://www.schneier.com/blog/archives/2011/11/advanced_persis.html`, Accessed: July 30, 2012.

Unfortunately, probabilities and patterns of unwanted behaviour are very hard to procure and labeled training data for a new system is sparse [8, 9]. But it is reasonable to assume that most activity in a network is not triggered by compromised machines and attacks are represented by only a tiny fraction of the overall behaviour. Therefore, methods provided by unsupervised learning yield outliers, which in turn may represent attacks [9–11]. Unsupervised learning can roughly be classified in, nearest-neighbour, rule-mining, statistical, and clustering techniques. Each of which have advantages and disadvantages, depending on how they are used, see Chandola et al. [11]. For our purpose of grouping anomalous instances, clustering seems best suited. The disadvantages of clustering, i.e. the complexity of clustering algorithms and possible misclassifications, can be reduced by leveraging optimized algorithms, assumptions, and false-positive reductions [9, 12].

Both methods, signature-based and anomaly-based, have strengths and weaknesses. The main drawback of signature-based methods is the inherent limitation that they always have to consult the signature database to match detected features with the information therein [9]. If a new attack is out, it is probable that the signature database does not contain the latest attack pattern. Anomaly-based detection techniques, on other hand, have their true potential in detecting previously unseen patterns [8]. A common limitation both detection techniques share is a lack of "context". This context needs to provide information about inherent relations among users, services they use, the hosts from which they operate, and for which workflow they are assigned to. For instance, it is not sufficient to know that a service has longer than average response time, the correlation of response time and measurable changes of user and network host behaviour offers more valuable clues.

In order to get benefits from signature- and anomaly-based monitoring we propose to combine them into a context-based anomaly detection framework. This framework consists of three main tiers:

i The specification of a DSL which allows to model the cloud-sourced IT landscape in detail such that workflows can be specified, monitoring rules can be generated, and computing entities can be put into relation.
ii The detection of workflow aberrations, or semantic gaps, caused by attacks via Complex Event Processing (CEP) based on monitoring rules generated by the model. CEP is a signature-based method to analyze event streams in a midtoupper size IT infrastructure [13]. The purpose of CEP is to derive more meaningful events (in this case alerts).
iii The detection of abnormal entities, i.e. users, services, network hosts, and workflows, by leveraging unsupervised machine learning, to detect unforeseen changes in the behavior.

The application of our framework in a cloud-sourced health-care environment provides the means necessary to unravel the following incidents:

– *Semantic Gaps.* A document retrieval workflow doctor accessing the database without proof of first having received a permission token, replay attacks, workflow aberrations through patched code.

– *Anomalies.* An increase of service activity, service calls at unusual hours, abnormal users, detectable by a gradually increasing number of document requests, suspiciously active hosts, but also a change in flow behavior of service calls and network hosts (i.e. payload analysis of web-service parameters). The entities, services, users, hosts, workflows, constituting the unusual behaviour are labeled as anomaly.

The paper will continue with a description of the framework in Section 2, including the DSL 2.1, the usage of CEP 2.2, profiling entities 2.3 and anomaly detection via fingerprints 2.4. Section 3 depicts the used architecture and Sections 5 and 4 discuss future work and related work respectively.

2 Framework Overview

In this section we discuss the framework in more detail. We begin with the DSL to specify the IT infrastructure consisting of workflows, services, hosts, users, and their relations. This in turn leads to the discussion of how CEP is included in the framework. Afterwards our discussion will continue with details about the profiling of entities for anomaly detection purposes, i.e. discuss the different profiles, the features for fingerprints, the clustering method and distance measure, and round it up with a description of the architecture.

Every monitoring system needs events to determine the actual state of the system. Our framework expects events from the infrastructure, in form of TCP and UDP packets sent from the machines in the network, and in form of service calls. TCP and UDP packets are aggregated as flows that have multiple characteristics, such as, source, destination, ports, time, among others, duration. Service events are used to derive the current state of the services, show user behaviour (i.e. access requests), and give general information on the state of workflows. Information that should be present is, the duration of a call, the time, the user, and the object id that was requested.

2.1 A DSL for IT Landscapes

The use of metamodels or domain specific languages (DSL) is not uncommon [14, 15], their main use is to provide the vocabulary for experts to let them express their knowledge to represent the system in a textual[3] (or graphical) model. These models can later be accessed for look-ups, reasoning, and/or code generation.

Our DSL, therefore, allows the creation of a model that in turn allows harvesting information of entities (i.e. traceability of deployed entities to model information) and monitoring rule-generation. The model in Figure 1 reuses concepts from Breu et al. [14, 15], for example the introduction of multiple conceptual layers. The event-driven process chain paradigm [16] that is used in the model facilitates the modeling process, since it allows to represent services through

[3] xText: `http://www.eclipse.org/Xtext`, Accessed: July 20, 2012.

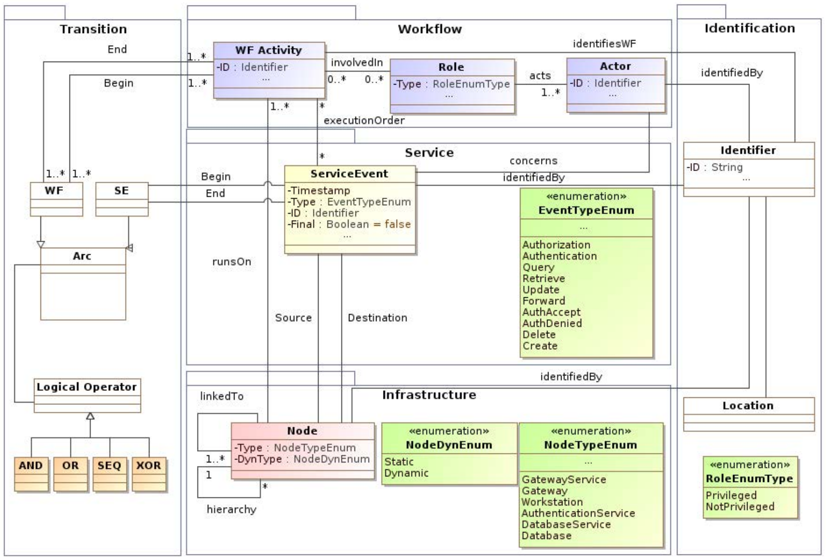

Fig. 1. A language to describe an IT landscape

their behaviour in form of events. A workflow activity, therefore, is not modeled via services and their call-sequence but rather as a series of events.

A model derived from the DSL contains three layers, *Workflow, Service* and *Infrastructure*. The workflow layer contains three classes, these are *WF Activity, Role*, and *Actor*. Activities and service events are related by arcs (*Arc*) which describe the way a workflow is executed. These arcs can have different types, i.e. *AND, OR, XOR, SEQ*. SEQ denotes that if said arc lies between two workflow activities A and B, then A is followed by B. *AND, OR* and *XOR* relate events in a boolean fashion. For instance A AND B,C denotes that after A, B and C is executed. Roles, *role* is a set of responsibilities and obligations for a stakeholder, that can influence heuristics during the analysis of events. As discussed above, services are not modeled directly, but are modelled as *ServiceEvent* of various types (*EventTypeEnum*). Event emitters are services, on top of hosts. Hence, among other features provided by the service event, i.e. variable ones such as timestamps and session ids (to identify the *Actor*), we assume a source and a destination pointing to the hosts that were responsible for the event. This allows us to connect the service layer to the infrastructure layer. Hosts (Node in the model) can be of various types (*NodeType*), this makes it easier to map events to their corresponding workflow activity during runtime.

Identifier defines the set of identifiers, i.e. all elements are connected to it via *identifiedBy*, such as hosts, service events, and actors are identified by it (via UUID and a location). The elements doing the execution are hosts from the infrastructure, hence the (*runsOn*) class.

2.2 Complex Event Processing

To monitor proper execution of systems, rule-based approaches tend to be used, i.e. in form of CEP. For CEP much research has been invested in query languages to handle the stream of events in query-based languages similar to SQL[4], ESPER[5], Oracle CEP[6], Coral8[7] and Aleri[8]. In our case we need to listen for events that are modelled beforehand, i.e. we need to listen for sequences that represent a workflow. These sequences give all the information necessary to infer who is responsible for certain actions. Part of our work focuses on the creation of CEP rules automatically based on the model created by the expert. For CEP rules the Esper Query Language (EQL)[9] in combination with the Esper CEP engine was chosen, since it is open source (GPL GNU Public License v2.0), has an active community and has shown potential in several benchmarks [17]. The translation from workflow models to query rules is straight forward, since EQL provides the same boolean logical connectives as our model and also provides the possibility to model sequences $\xrightarrow[seq]{}$. For instance, the formula $Ev_0 \xrightarrow[seq]{} Ev_1$ is only satisfied if and only if Ev_0 is emmitted before Ev_1. In summary a workflow model, as used for compliance detection, is nothing more than a series of CEP rules that are verified by the CEP engine.

2.3 Profiling of Entities

To determine anomalies in the activity of a corporate network, the accounting information of banks, or more general in usage behaviours, it is common to first create a profile that describes a normal behaviour of key entities [18, 19]. The profile types, *service, user, host,* and *workflow*, that we consider reflect the key entities that are involved in an on-line data processing. Gartner, Inc. [20] states, for instance, that there is the need for *user profiling* to monitor user behaviour to prevent data theft. *Service profiles* are needed to determine, among others, a gradual decrease of performance compared to itself or an overall different behaviour from other services. Communication patterns among hosts also need to be considered in form of a *host profile*. Outliers in each of these types of entities have an impact on the performance/security of *workflows* and their activity profile.

Assume, for instance, a compromised machine that gradually increases the number of requests for classified object information in the name of an existing user U over service S by using machines $M_{0..n}$. Normally, this is not easy to trace, especially if U has permissions to query restricted information (no CEP alerts will be generated). A time-based analysis, though, yields detectable changes

[4] `http://www.w3schools.com/sql/default.asp`, Accessed: July 20, 2012.
[5] `http://esper.codehaus.org/`, Accessed: July 20, 2012.
[6] `http://tinyurl.com/OracleCEP`, Accessed: July 20, 2012.
[7] `http://tinyurl.com/Coral8CEP`, Accessed: July 20, 2012.
[8] `http://tinyurl.com/AleriStreaming`, Accessed: July 20, 201.2.
[9] `http://esper.codehaus.org/`, Accessed: July 20, 2012.

in the behaviour of U, S, and $M_{0..n}$. These are, more queries in U's name, more queries spread to machines $M_{0..n}$, more queries at unusual hours for S by U, and at the end, a detectable change of the workflow behaviour itself. The profiles are further refined into, an *immediate, hourly*, and *monthly* track.

i To perform an on-line analysis of individual service events, CEP is used. CEP alerts have an immediate impact on the immediate track as well as statistical information gathered from the event itself, i.e. z-scores from parameters, duration, and the payload.
ii An hourly track allows to aggregate some more information about hourly deviances, for instance, the average number of calls for a service, the number of its users, average call duration, extreme values such as maximum duration and minimum duration, the number of alerts produced by the immediate track during selected hours, and more.
iii To assess more subtle patterns of deviance, a longer time-period is needed. To give an example consider the following scenario of a persistent attack. A competing company or government managed to break into the system and hides its activities of espionage, e.g., by leaking of sensitive documents, in form of an insider attack. For this, the real attackers stole the credentials of some user U to gradually query more and more documents, for instance creating 2-3% more queries per day (hour) than was normal. The immediate and hourly track are not built to detect such subtle aberrations and, hence, fail to detect them. The comparison of absolute access numbers over, for instance a monthly basis, shows a huge increase of query activity.

Information from the hourly (h) and monthly (m) track of an entity is represented by fingerprints (F_h^e, F_m^e) and represent, hence, a measure of the overall behaviour of the selected entity (e). Fingerprints are basically feature vectors $\mathbf{v}_i = (v_{i0}, \ldots, v_{in-1})$, containing continuous data. Fingerprints contain for instance, the number of CEP alerts in an hour, the number of alerts raised from immediate profiles, or z-score outliers. Our framework uses these fingerprints to compare its behaviour to other entities' behaviours but also to measure potential deviances of its own behaviour over time.

2.4 Clustering Fingerprints for Anomaly Detection

To determine abnormal entities in relation to other entities of the same type it is necessary to compare individual features of a profile and attain a sense of distance. Since individual characteristics of a profile might not change sufficiently to determine that an entity is an anomaly, we take into consideration all of the individual features that were collected. To take all features into consideration clustering can be used [21]. Clustering makes use of the inherent structure of data and groups data instances (clustering) by common attributes and a similarity measure. After the outliers have been found, the model can then be used to further link entities and detect correlations among outlying users, and, for instance, services. Figure 2 summarizes how the layers are related.

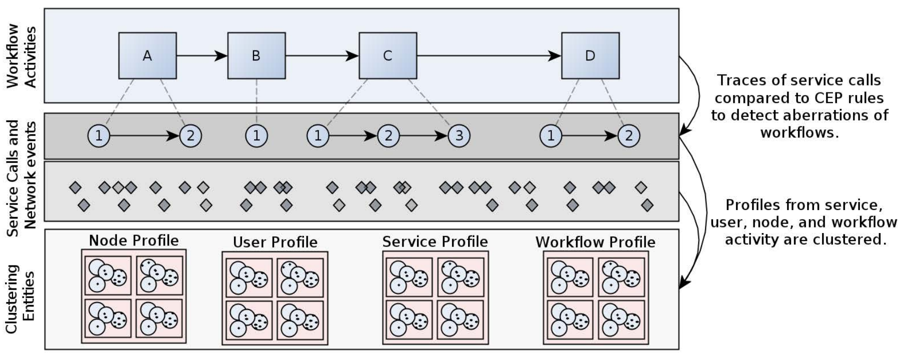
Workflow Activities
A
B
C
D
Service Calls and Network events
Clustering Entities
Node Profile
User Profile
Service Profile
Workflow Profile
Traces of service calls compared to CEP rules to detect aberrations of workflows.
Profiles from service, user, node, and workflow activity are clustered.

Fig. 2. Overview of connecting the layers

Although other distance measures exist, e.g., Jaccard, Dice, and Russell/Rao [21, 22], which have their use when comparing dichotomous data, the measure of distance which we use is the Euclidean metric, see Formula 1. It gives us the opportunity to measure distances of continuous multi-dimensional variables, i.e., $\mathbf{v}_i \in R^d$.

$$d(\mathbf{v}_i, \mathbf{v'}_i) = \left(\sum_{k=0}^{n-1} (\mathbf{v}_{ik} - \mathbf{v'}_{ik})^2 \right)^{\frac{1}{2}} \qquad (1)$$

Various clustering algorithms have been proposed, e.g. DBSCAN [23]. DBSCAN finds clusters based on a density measure, i.e., it finds clusters in which data instances have only a maximal distance to each other. Hence, points near to each other are grouped in the same cluster. This may lead to arbitrary shaped clusters, including spherical cluster shapes. On the one hand, arbitrary shaped clusters do not lead to any clear results, and on the other hand, clusters in our case might have varying density values ϵ, which is problematic for DBSCAN. The algorithm of our choice is fixed-width clustering [9, 24]. The algorithm, described in Figure 3, has the benefit of a better runtime complexity, compared to other clustering algorithms, e.g., standard k-means, since it computes clusters with just a single passage through the data instances (fingerprints). In fixed-width clustering, clusters have a maximal width and a cluster center, called centroid. Data instances that are clustered based on their feature vector either surpass the maximal width (based on the distance measure) and create a new cluster or have a smaller distance and become part of the cluster and have a certain distance to its centroid. The fewer data instances are inside a cluster the more probable it is that those data instances are in fact outliers. This is basically the assumption discussed before: *normal behaviour represents the majority of data instances whereas abnormal behaviour is represented by only few data instances* (which represent potential attacks). Hence, clusters containing fewer instances than a user-configured threshold, represent anomalous data points. For instance if less than 1% of data instances are within a cluster it is labeled as anomalous. We leverage the distance notation from Formula 1 to $d(\mathbf{v}_i, C)$ to denote the distance from a feature vector to a cluster (represented by its centroid). The algorithm to cluster the fingerprints, as described in [24, 9], consists of 3 steps:

1. The set S of clusters is first initialized to the empty set.
2. A fingerprint $\mathbf{v}_i = (\mathbf{v}_i, \ldots, \mathbf{v}_{in-1})$ is taken from the set of fingerprints (unlabeled set of fingerprints).
   IF The set S is still empty then the fingerprint will create a new cluster C and $\mathbf{v}_i$ will be the centroid.
   ELSE The cluster C with the smallest distance is selected $\arg\min_{C \in S}(d(\mathbf{v}_i, C))$ such that the fingerprint does not surpass the maximal width. If such a cluster is found, the fingerprint is inserted, otherwise a new cluster is generated and $\mathbf{v}_i$ will be the centroid.
3. The second step is repeated for all remaining fingerprints.

Fig. 3. Clustering as described in [24, 9]

Detecting Abnormal Entities and False-Positives. Clusters containing less fingerprints than the user-specified threshold are automatically labeled as outliers. The fingerprints within, and their entities they represent, are then also labeled as anomalous. For each entity there are two possibilities for creating an anomaly alert, (i) either through a change of behaviour from itself, or (ii) by being substantially different from other entities of the same type. The idea behind (i) is that the system collects fingerprints for a single entity over an amount of time, i.e., hours or months, and clusters them. If an entity did not change its behaviour, its fingerprints are in the same dense cluster c. The more changes an entity undergoes (stored in the behavioural profile) the more the fingerprints change. Eventually the generated fingerprint surpasses the distance to the centroid of c and results in an anomaly alert. In case of (ii) fingerprints are used to compare entities among each other. A user, who exhibits a significant different usage pattern, creates his own cluster and is labeled as anomalous. In case a new user, service, or host is introduced to the system, it can be determined automatically if said entity is abnormal or not, simply by comparing its fingerprint.

Through the use of the domain model, entities are put in relation to each other, i.e., users to hosts, or services to workflows. Anomalies are, thus, put into context and alerts propagated upwards. For instance, abnormal services, hosts, and users, determine the security status of the assigned workflows. Vice-versa, drilling down on an abnormal workflow (e.g., too much network traffic or too many document queries), exposes abnormal entities, e.g., anomalous services, users, hosts, and speeds-up root-cause analysis.

It is possible, even likely, that some clusters that are detected anomalous are actually not anomalous. Groups of fileservers will, for instance, have different fingerprints than mail servers or timeservers. It is therefore important to consider various degrees of optimization to prevent false-positives. There are a couple of options, since the clustering algorithm is parameterized by two variables,

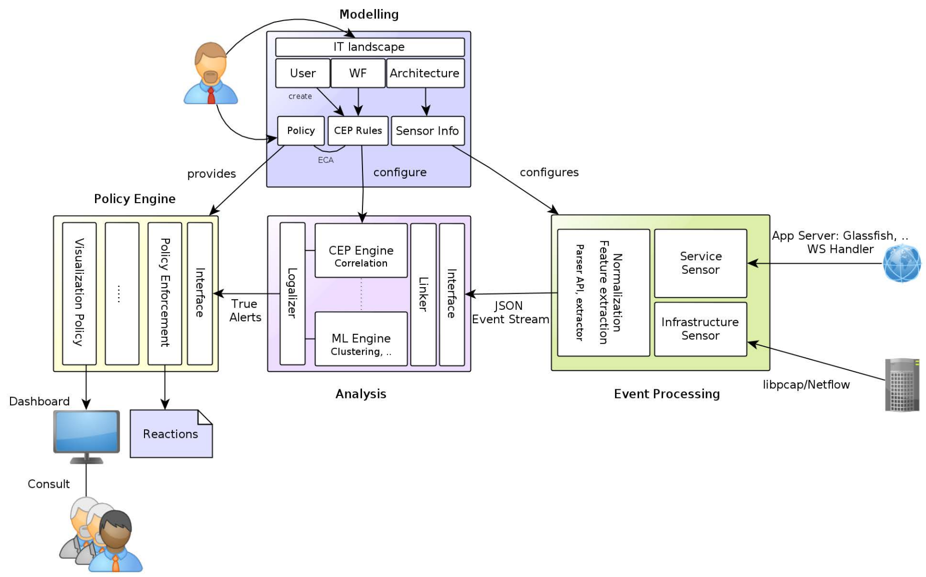

Fig. 4. Overview of the monitoring architecture

the width and the threshold for anomalies, tweaking either of them will reduce false-positives. An increased cluster-width allows sparse clusters, exhibiting a significant higher variance, to be normal. Rising the threshold allows to have clusters with few instances to be normal as well. Another option is the creation of tests to determine the true state of an entity, but that is left for future work. If a cluster, and the ensuing entities within, are still labeled as anomalies the framework provides to relabel them as normal.

3 Architecture

Figure 4 indicates the different components of our monitoring architecture, which can be offered by a cloud provider as a *Monitoring as a Service* solution. A tenant uses the DSL provided by the *modeling component* to provide a model which describes his IT landscape. This model is aligned to the three layers we discussed above. Based on the model, rules to detect workflow non-compliance are created to configure the CEP engine. To customize the monitoring service, the tenant supplies the *policy engine* with policies (which are rules or metrics) to enable the cloud provider to react on alerts. Policies specify (i) the gravity of alerts and (ii) what should happen in case they happen. By providing a policy, a tenant bids the cloud provider to cut off a virtual host from the network, if said host is classified as an information leaking host. This state can be mantained until the host is classified as normal. The *event processing* component consists of service and network sensors as well as a normalization feature extraction element. The sensors act as event sinks for multiple service and network event emitter sources. The service sensor receives JSON[10] encoded service call data, whereas the network monitor is built as a netflow-collector. Analysis of workflow compliance (i.e. via CEP) and outlier detection (i.e., via Clustering) is done in the *analysis component*. Statistical methods, i.e., z-scores are computed by "The Apache Commons Mathematics Library".[11] The CEP engine of choice is ESPER[12]. Based on the outcome of the analysis and the severity of alerts, the policy engine populates the *dashboard* and determines reactive measures for the cloud provider (policies provided by the tenant). The dashboard displays integral information about a tenant's infrastructure, i.e., the infrastructure in tabular form, important alerts, and anomalous entities.

4 Related Work

In this section we discuss related work in the areas of cloud security monitoring, anomaly detection, and CEP. In the area of cloud security monitoring several related papers have been published, yet among those [25] seems the most related. Vieira et al. [25] focus on distributed architectures in grid and cloud computing and perform behavioural analysis via neural networks. [25] leverage neural networks for behavioural analysis we use clustering. Moreover the anomalies we find are disjoint from theirs. There has been plenty of research for anomaly detection via clustering, a survey on this topic is provided by [8]. Clustering is quite versatile as the approaches in [9, 24, 10, 12] point out. Portnoy et al. [9] detect attacks, e.g., denial of service, in the KDD 1999 data via clustering of network activity set.[13] Gu et al. [12] use clustering for the detection of botnets by a framework called "Botminer". The Authors in [10] improve clustering for NIDS by using a density-based clustering algorithm and a grid-based metric and evaluate their efforts on the KDD 1999 data set. To measure hosts we create profiles of their network behaviour by sampling their TCP/UDP flows based on [26, 12]. To our knowledge, the clustering algorithm itself was first presented in [9]. Instead of clustering individual multi-dimensional features form the KDD training set we cluster fingerprints of various entities. The main difference from the proposed work of Gu et al. [12] is that the former only profiles hosts for the specific detection of botnets, whereas we only try to find outliers and assemble outliers in a holistic profile of the infrastructure. The approach presented in [26] is more similar to ours since it also profiles machines in the network. But we're not restricted to machines only, but also services, users, and workflows.

The multi-tier DSL proposed in this paper allows the definition of node hierarchies, roles, actors, and distinguishes three layers. These design decisions are in its core similar to [14, 15]. Breu and Innerhofer et al. provide a model-based approach with concepts for security management. There is related work for DSLs

[10] `http://www.json.org/`, Accessed: July 30, 2012.

[11] `http://commons.apache.org/math/`, Accessed: July 30, 2012.

[12] `http://esper.codehaus.org/`, Accessed: July 20, 2012.

[13] `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`, Accessed: July 20, 2012.

to create service infrastructures Berre et al. [27] present the *Service Oriented Architecture Modeling Language (SoaML)* and Popescu et al. provide the *Service Markup Language* SML. SoaML was not desireable for our scenario, since our interest was more cloud oriented than SOA-centric. Our DSL allows the definition of event-sequences, which in turn allow to detect deviances to rules generated by the workflow model. The paradigm of modeling services as events is similar to event-driven process chains (EPC), discussed in-depth in [28]. Workflow compliance in SOA via CEP has been discussed by Mulo et al. [16]. A service invocation is regarded as an event and business process activities as event-trails. These event-trails guide the creation of queries which a CEP engine uses to identify and monitor business activities. Anomaly detection itself has been done frequently in many domains, though to the best of our knowledge, there is no cloud monitoring approach that allows CEP and anomaly detection to monitor (a) the execution of workflows for semantic gaps and (b) detect infrastructure anomalies relative to said workflows. Due to the formal representation of "behaviour" of entities we're able to pinpoint suspicious services, users, hosts, and workflows.

5 Conclusion and Future Work

We have sketched a context-based anomaly detection framework to facilitate real-time monitoring of cloud-sourced workflows and infrastructures. Our research differs from existing monitoring work as we aim to mitigate cloud threat-scenarios with web services and infrastructure anomaly detection, and CEP. The framework aims to keep multiple profiles of entities on various layers and to link detected anomalies and semantic gaps to workflows. Future work will consist of,

– An implementation and an evaluation based on a real-world scenario. The planned evaluation will consist of a real-life healthcare scenario where services, data, and hosts, are outsourced to an IaaS cloud. The architecture consists of all services necessary to allow a regulated flow of action in a hospital, e.g., image retrieval services, diagnose services, and an XACML-Kerberos like access control infrastructure. Based on the runtime behaviour of the system we train our machine-learning component and measure deviations of user- and network-activity. To measure the effectiveness of our approach the healthcare architecture will be subject to various use cases/attacks, i.e., a failed XACML architecture, leak attacks from insiders, fuzzy security-testing of web-services from other tenants, or TCP/UDP malware propagation across the cloud. The evaluation will show if the anomaly detection can provide information about these attacks.
– Carefully evaluating other clustering methods, e.g., Entropy Maximization, to reduce false-positives and attain a better clustering result.
– A CEP rule repository to further allow the reduction of false-positives with domain knowledge, detect additional signature-based events to augment the profiles for entities in general. Along the way goes the inclusion of other

monitoring tools such as Snort[14] and Ossec[15] to get a more elaborate profile for hosts.
– Finding anomalies is a good first step, but it serves a wider purpose, i.e., the semi-automatic labeling of clusters via supervised learning. First, normal and anomalous clusters are labeled, then based on the fingerprints in these clusters training data for supervised learning, e.g., Naive-Bayes, Random Forests, is easily generated. New fingerprints can then be readily classified as a specific form of behaviour.

References

1. Amazon, EC: Amazon elastic compute cloud (amazon ec2). Amazon Elastic Compute Cloud, Amazon EC2 (2010)
2. Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al.: A view of cloud computing. Communications of the ACM 53(4), 50–58 (2010)
3. Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, pp. 199–212. ACM (2009)
4. Walker-Morgan, D.: Vsftpd backdoor discovered in source code. Website (2011), `http://h-online.com/-1272310` (visited: July 4, 2011)
5. Hoglund, G., Butler, J.: Rootkits: subverting the Windows kernel. Addison-Wesley Professional (2006)
6. Koziol, J.: Intrusion Detection with Snort, 1st edn. Sams, Indianapolis (2003)
7. Trend Micro, Inc.: Ossec documentation, `http://www.ossec.net/` (accessed: December 14, 2010)
8. Garcia-Teodoro, P., Diaz-Verdejo, J., Macia-Fernandez, G., Vazquez, E.: Anomaly-based Network Intrusion Detection: Techniques, Systems and Challenges. Computers & Security 28(1-2), 18–28 (2009)
9. Portnoy, L., Eskin, E., Stolfo, S.: Intrusion detection with unlabeled data using clustering. In: Proceedings of ACM CSS Workshop on Data Mining Applied to Security (2001)
10. Leung, K., Leckie, C.: Unsupervised anomaly detection in network intrusion detection using clusters. In: Proceedings of the Twenty-eighth Australasian Conference on Computer Science, vol. 38, pp. 333–342 (2005)
11. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM Computing Surveys (CSUR) 41(3), 15 (2009)
12. Gu, G., Perdisci, R., Zhang, J., Lee, W.: Botminer: clustering analysis of network traffic for protocol-and structure-independent botnet detection. In: Proceedings of the 17th Conference on Security Symposium, pp. 139–154 (2008)
13. Eckert, M., Bry, F.: Complex Event Processing, CEP (2009)
14. Breu, R., Innerhofer-Oberperfler, F., Yautsiukhin, A.: Quantitative assessment of enterprise security system. In: The Third International Conference on Availability, Reliability and Security, pp. 921–928. IEEE (2008)

[14] `http://www.snort.org/`, Accessed: July 30, 2012.
[15] `http://www.ossec.net/`, Accessed: July 30, 2012.

15. Innerhofer-Oberperfler, F., Breu, R., Hafner, M.: Living security collaborative security management in a changing world. In: Parallel and Distributed Computing and Networks/720: Software Engineering. ACTA Press (2011)
16. Mulo, E., Zdun, U., Dustdar, S.: Monitoring web service event trails for business compliance. In: 2009 IEEE International Conference on Service-Oriented Computing and Applications (SOCA), pp. 1–8. IEEE (2009)
17. Grohe, S., Schlameu, C., Sommer, R.: Performancevergleich von cep-engines. Technical report, Hochschulschriftenserver der Universitt Stuttgart, Germany (2010), `http://elib.uni-stuttgart.de/opus/oai2/oai2.php`
18. Denning, D.: An intrusion-detection model. IEEE Transactions on Software Engineering (2), 222–232 (1987)
19. Durgin, N.A., Zhang, P.: Profile-based adaptive anomaly detection for network security (2005)
20. Nicolett, M., Kelly, K.: 2012 Gartner Critical Capabilities and Magic Quadrant for SIEM (2012)
21. Tan, P., Steinbach, M., Kumar, V.: Cluster Analysis: basic concepts and algorithms. In: Introduction to Data Mining. Addison-Wensley (2006)
22. Finch, H.: Comparison of distance measures in cluster analysis with dichotomous data. Journal of Data Science 3(1), 85–100 (2005)
23. Ester, M., Kriegel, H., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, vol. 1996, pp. 226–231. AAAI Press (1996)
24. Oldmeadow, J., Ravinutala, S., Leckie, C.: Adaptive clustering for network intrusion detection. In: Dai, H., Srikant, R., Zhang, C. (eds.) PAKDD 2004. LNCS (LNAI), vol. 3056, pp. 255–259. Springer, Heidelberg (2004)
25. Vieira, K., Schulter, A., Westphall, C., Westphall, C.: Intrusion detection for grid and cloud computing. IT Professional 12(4), 38–43 (2010)
26. Hernandez-Campos, F., Nobel, A., Smith, F., Jeffay, K.: Understanding patterns of tcp connection usage with statistical clustering. In: 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, pp. 35–44. IEEE (2005)
27. Berre, A.: Service oriented architecture modeling language (soaml)-specification for the uml profile and metamodel for services, upms (2008)
28. van der Aalst, W.: Formalization and verification of event-driven process chains. Information and Software Technology 41(10), 639–650 (1999)