Raj, Rohit; Truong, Linh

On Analysis of Security and Elasticity Dependency in IIoT Platform Services

Published in:
2021 IEEE International Conference on Services Computing (SCC)

DOI:
10.1109/SCC53864.2021.00048

Published: 01/11/2021

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

On Analysis of Security and Elasticity Dependency in IIoT Platform Services

Rohit Raj
Department of Computer Science
Aalto University, Finland
rohit.raj@aalto.fi

Hong-Linh Truong
Department of Computer Science
Aalto University, Finland
linh.truong@aalto.fi

***Abstract*—Modern IIoT (Industrial Internet of Things) systems rely on highly decomposed platform services that are distributed across the edge and cloud. Security attacks in these systems can cause indirect and unwanted impacts on the elasticity of platform services. These impacts can often manifest across subsystems and not necessarily remain localized. However, there is a lack of techniques to identify and model cross-layered, cross-system concrete dependencies between specific security attacks and their corresponding elasticity impacts. This paper presents a novel framework SEDAICO that can model the elasticity relationship among IIoT platform services due to a security attack. Secondly, the framework supports the developer to identify concrete dependency between specific security attacks and the corresponding impacts on the elasticity of the platform services. We further illustrate our work with experiments and evaluate the framework through a real-world example.**

***Index Terms*—cloud computing, edge computing, industrial internet of things, cloud elasticity, cloud security analytics**

I. Introduction

IIoT comprises of a wide range of services that constitutes diverse "smart" things and devices integrated with cloud infrastructures [1]. More recently, the edge computing paradigm [2] has emerged to facilitate the low-latency, high-QoS (Quality of Service) requirements by introducing computing and data processing services in between these things and devices and the cloud. The edge and cloud computing paradigms coupled with the advent of containers and microservices have led to a very microservice-oriented approach in the implementation of IIoT architectures.

At the core of a microservice-oriented IIoT architecture, various types of edge and cloud platform services are used. Platform services are composed of heterogeneous and distributed services such as databases and message brokers, and are deployed on a variety of subsystems. As these services are often customized for an organization's needs, they also require tailor-made goals for security, elasticity and resilience. For these goals, it is imperative to have a concrete mapping of the dependency between security, elasticity and resilience. For example, an IIoT infrastructure for monitoring equipment needs to identify a dependency between a security attack on an edge platform service and corresponding elasticity effects on various downstream cloud platform services. Moreover, the behavior of analytics on cloud platform services may vary between an active attack vs a generic failure. An active attack would usually publish/inject a malicious payload (such as a stack-based buffer overflow payload) whereas a software failure may just send harmless data. Generic monitoring systems [3], [4], [5] lack the ability to provide necessary dependencies analytics to differentiate between these two situations. To date, little attention has been paid to identify impacts on the elasticity of different platform services under a specific attack condition. As we discuss in the related work (Section V), there is a lack of comprehensive analysis frameworks that can aid the developer in establishing these dependencies.

This paper presents the SEDAICO (Security Elasticity Dependency Analysis In Computing cOntinuum) framework that takes a holistic approach towards the analysis of security attacks and their impacts on the elasticity of the IIoT platform services. Through monitoring, our framework evaluates and analyzes the elastic changes in these platform services during attack conditions. By mapping these (elasticity) impacts to a specific security attacks, our framework helps developers to establish concrete dependencies between a specific security attack condition and the elasticity of different platform services across the sub-systems. Overall, we present the following contributions of this paper:

1) Techniques to model relationships between the elasticity of different IIoT platform services and a security attack. The developer can then identify and consequently model this into a security-elasticity relationship.
2) Techniques to identify concrete dependencies between a security attack and the corresponding impact on the elasticity of a platform service. This is done by the framework through the analytics of data captured from monitoring and tests profiles.

For example, using our framework, a developer could test how exactly the elasticity of cloud database is dependent on the security attack that is limited to only edge subsystem. Through the use of declarative syntax, the developer can very easily provide security-elasticity analytics specifications to the framework and generate complex deployments, tests and monitoring profiles suitable for their IIoT Under Test (IUT). Finally, through security-elasticity dependency generated by our framework's analytics, the developer can also capture and share the threat intelligence to detect a security attack based on elasticity change patterns for their production IIoT platform

services. In this paper, we demonstrate our framework with a real-world scenario using industrial data. The framework prototype is available at https://github.com/rdsea/SEDAICO.

The rest of the paper is organized as follows: Section II discusses the motivation and presents the research questions. Section III introduces the key models, techniques and implementation of our framework. Section IV presents the evaluation of our framework. In section V we review related literature. Finally, the paper ends with concluding remarks in Section VI.

II. Motivation

A. Motivating scenario

Let us consider a system for monitoring and analysis of an ISP's (Internet Service Provider) equipment. We use the monitoring infrastructure of an ISP network to emulate an IIoT system. Fig. 1 describes a GPON (Gigabit Passive Optical Networks) infrastructure; and the monitoring and analytics services for this infrastructure. At the equipment side (similar to remote factories/industrial sites), sensors are used to monitor multiple types of equipment. GPON equipment such as DSLAM (Digital Subscriber Line Access Multiplexer), PON (Passive Optical Network) must be monitored by these sensors to ensure proper network operations[1]. This motivating example considers the case of monitoring of GPON equipment of a single province-level network with roughly 1 GB of equipment monitoring data per day that supports active monitoring components and serves a total of 2048 ONUs.

The upper edge and cloud subsystems manage and process the health telemetry data originating from the sensors. The subsystems comprise multiple platform services. The platform services on the edge subsystem in the motivating example are less resource-intensive and consist of a message broker, an edge analytics, and a local database such as SQLLite [6]. The cloud platform services are highly scalable: such as Kafka Broker for messaging or Apache Cassandra[2] as a database. These services all together create complex IIoT platform services, which are typical in IIoT centric organizations.

It is important to note that we need to monitor these platform services themselves to ensure proper capturing of anomalies and SLA (Service Level Agreement) guarantees. A typical IIoT system, like in the scenario is geo-distributed. While the security related attacks in this IIoT system can occur at different locations, repercussions (elasticity impacts) of these attacks may not always be localized and may be observed across different subsystems. For example, as shown in Table I, in an unsecured broker configuration attack (such as CVE-2018-12551[3]) `T01-UB`, an attacker can publish on the edge message broker. Since the attacker is publishing on an incorrect topic, it will not cause data poisoning. However, this scenario can still cause a decrease of edge message broker throughput, resulting in a reduced load on the cloud database. This is because the message broker on the edge subsystem will try to allocate resources to the incoming messages on an incorrect topic. Moreover, as the data is not being poisoned, an automated monitoring system on the cloud would not be able to pinpoint the cause to any specific security attack just by looking at reduced database load or the incoming data.

[1]A PON is one of the most common networks used for providing the internet services (such as VoIP (Voice over IP), telephone)

[2]https://cassandra.apache.org

[3]https://nvd.nist.gov/vuln/detail/CVE-2018-12551

B. Research statements and approach

RQ1: How to characterize the relationship between security attacks and elasticity: We must characterize possible relationships between security attacks and the corresponding elasticity impacts in IIoT platform services across different subsystems. Developers need models capturing the relationship between these two aspects. Such a model requires information about the location of impact, duration of impact, etc. Current IIoT monitoring frameworks do not support developers to have a concrete specified model of the relationship between security attacks and elasticity impacts. For example, through a model that captures the relationship between an attack on edge subsystem services such as unsecured edge message broker and its effect on the cloud message broker, the developer can quickly establish a correlation.

RQ2: How to analyze the propagation of security related attacks and its impact on elasticity: We need techniques through which we can attribute a specific security attack on an IIoT platform service with a precise impact on the elasticity of another (or a set of) platform service across the system. For example, in the case of node poisoning, we must be able to identify and differentiate between the anomalies manifested on cloud subsystem's stream-processing service and determine if the poisoning is due to common failures versus an active attack. The present monitoring and elasticity modeling frameworks would tell the developer only about the failure and not identify a concrete security attack related underlying reason.

On the lines of these research questions, we focus on novel techniques to model the security-elasticity relationship and aid the developer in identifying concrete security-elasticity dependencies between the platform services.

III. Methods and Implementation

A. Modelling the dependency between security attacks and elasticity impacts

To capture a relationship between security attack and elasticity impacts from the analytics, we use the *What, Where, When*, and *How* aspects (W3H), which are popular for characterizing the context associated with an entity [7].

1) What: The `What` context of an analytics characterization deals with capturing the anomalous elasticity impacts (such as unnecessary scaling, and slower performance) that can occur in the IIoT infrastructure due to a security attack. The `What` context helps the developer in studying the nature of the attacks on the infrastructure.

For example, the first case of Table I (`T01-UB`) delineates a `What` dependency between elasticity impacts and security attack in an unsecured broker. In an unsecured broker, any

Name	Description	Example	What	Where	When	Developer challenges
T01-UB	Unsecured Broker	An unsecured broker allows rogue nodes to publish data on random topics	Decreased broker throughput and slower database ingestion	On edge subsystem (Broker) and cloud system services	Run-time performance changes, unnecessary scaling operation on edge and cloud subsystem	End-to-end observability including message brokers, data ingestion
T02-PSN	Poisoned Sensor Node	A sensor node publishes wrong data on the *correct* topic	Incorrect analytics on cloud, extra storage consumed in database	On sensor, edge subsystem and cloud subsystem	*Run-time*: unnecessary scaling operations, increased exception production rate; *long-term*: database consumed faster	Correlation of outlier data, batch processing for anomalies in the system

TABLE I: Examples of challenges in studying security attacks and elasticity dependencies

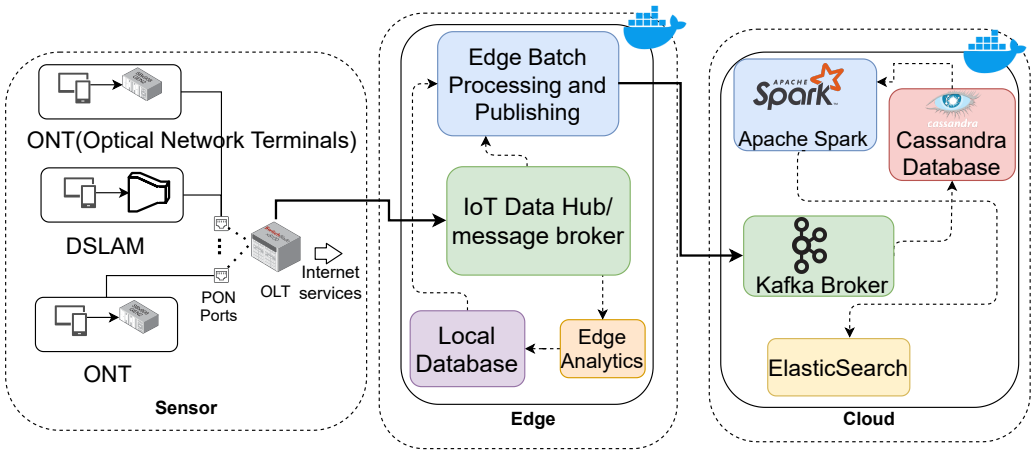


Fig. 1: Platform services in GPON monitoring and analytics system

rouge entity can publish data on a random topic. This can cause decreased broker throughput and result in a slower ingestion rate into the backend cloud database.

2) Where: Indicates the locations where the elasticity effects can manifest as a result of a security attack across the IIoT infrastructure. The location is in the interest of the developer – to know or to be aware of – where the elasticity impacts can be localized. To analyze the `Where`, we need to characterize the location of the security attack based on the result and monitoring data of the elasticity of IIoT infrastructure.

For example, Table I identifies the `Where` dependency between the location of a security attack and the elasticity impacts in the system. In the case of `T01-UB`, the location of the security attack is on the edge message broker. However, its elastic dependency might affect the cloud database as well as the other cloud platform services. Similarly, in the case of `T02-PSN`, the security attack takes place on the sensor subsystem, however, the elasticity of cloud analytics might get affected.

3) When: This aspect of analytics characterization deals with the frequency and timing of anomalies in the elasticity impacts. They can show up immediately during a security attack or might manifest at a later point in time. One common way of information augmentation of such elasticity impacts is to monitor and actively capture the timing data (event and processing). This may have both short-term and long-term `When` context. For example, we can see in example `T01-UB`, the `Where` context relates to the increase in the frequency of the publishing incidents on the edge subsystem. It will cause extra scale-up and scale-down operations dynamically during the runtime. The increase in processing throughput on the edge subsystem can provide us with `When` information. In the case `T02-PSN`, the security attack takes place on the sensor subsystem, however, the elastic dependency of cloud analytics might get affected.

4) How: This analytics context maps a security attack to the impacts on the elasticity of the IIoT infrastructure. The "How" is also logical information integrated into the framework's code. One of the usages of this "How" information is in the selection and recommendation of the tools, analytics and other experiment utilities to the developer. For example, from a database with the catalog of artefacts, the "How" information can be used to select appropriate recommendations.

B. Methods of W3H specification

1) Declarative methodology: To aid the developer in specifying the context characterizing the analysis, we follow the approach of directive specifications. This provides high-level of abstraction required to enable capture of the W3H context. Since the format is declarative, the relevant stakeholders (such

as the developer) can provide information without describing the control flow. We can then extract one or more of the W3H context from the information. Thus, the two relevant aspects are a) the syntax of declaration and b) how the framework extracts possible W3H information from this syntax.

a) Annotation syntax: To capture specifications declaratively, we use an annotation based approach in the configuration documents. Each annotation can be declared as a set of `name` and `value` as a comment in the configuration file. For our current implementation, the developer provides annotations in the YAML configuration files of the IUT.

An example representation of an *annotated* `docker-compose`[4] artefact on edge subsystem for W3H can be seen in Listing 1. Similarly, an example of *annotated* cloud database artefact can be seen in Listing 2.

```
# @framework: SEDAICOFramework
# @test-type: T01-UB
version: "3.7"
services:
  # @subsystem: edge
  # @type: broker
  vmq-main:
    image: vernemq/vernemq
    ports:
      - 1883:1883
 ...
```

Listing 1: Sample annotated edge docker-compose file

```
# @framework: SEDAICOFramework
# @test-type: T02-PSN
version: '2'
services:
  # @subsystem: cloud
  # @type: database
  cassandra-seed:
    container_name: cassandra-seed
    build: build/.
 ...
```

Listing 2: Sample cloud docker-compose file

b) Extracting "How" information from annotations: The second relevant aspect of our declarative methodology is how our framework parses and maps the annotations provided by the developer into W3H specifications. The post-processor component in Fig. 3 first parses annotated documents. For the parsing to begin, the configuration file must begin with annotation `@framework: SEDAICOFramework`. It then extracts all the detected name and value pairs. Each of these pair represents a set of "What" and "Where" information. This information is then used for *a)* selecting the profiles from the information database and *b)* in the final analytics to identify concrete security-elasticity dependency. Fig. 2 demonstrates how this information is used in selection of profiles for Listing 2. The solid boxes represent the filters applied while selecting the artefacts, whereas the dotted boxes are alternate filters.

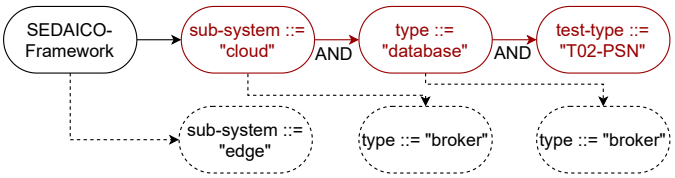


Fig. 2: Using "What" and "Where" information in selection of artefacts

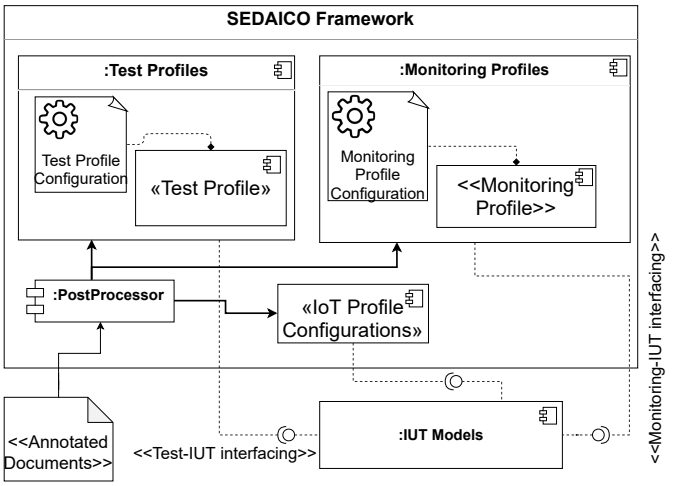


Fig. 3: Components of the SEDAICO framework

While this kind of annotation can be extended for many types of configuration representation, our current implementation supports `docker` and `kubernetes`[5] configuration files.

C. SEDAICO – utilities and artefacts

Based on the methods of capturing W3H from the artefacts, we present SEDAICO framework for analytics of security-elasticity dependency in edge-cloud platform services. The framework consists of multiple profiles and presents a recommendation of profiles to users based on their IUT choice. As mentioned in the previous subsections, these profiles are filtered through the annotation information provided by the developer.

Each profile of our framework is associated with some component. The various components of our framework can be seen in Fig. 3. We first describe the different utilities that constitute our framework's components.

Our framework distinguishes the following types of utilities and artefacts:

- *Public IUT artefacts (Pu_{iut})*: They are artefacts from existing providers and are used by the developer in the IUT. For example, in one of our scenarios, a dockerized configuration of Cassandra database is used as a public IUT artefact.

[4]https://www.docker.com/

[5]https://kubernetes.io

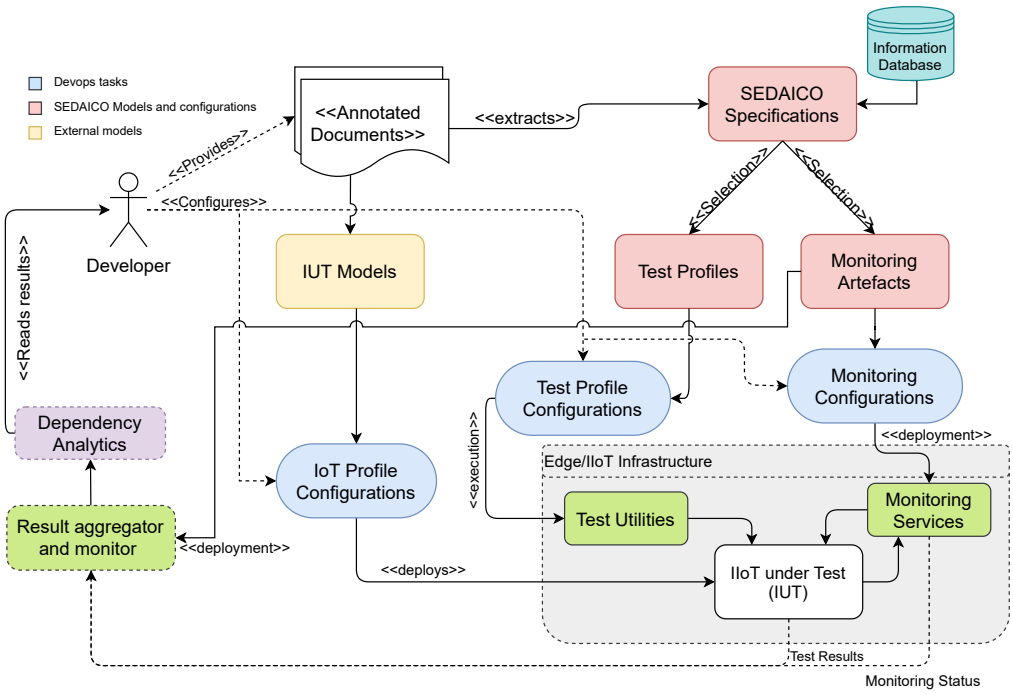


Fig. 4: Generic experiment workflow

- *Private IUT artefacts (Pr_{iut})*: They include custom artefacts available for the developer that have been tailored for the IIoT monitoring and analysis scenario. An example of private IUT artefacts is "mini-batching artefact" that captures and collates data from edge MQTT message broker, adds extra edge related metadata and finally publishes at periodic intervals to a cloud message broker.
- *Test Utilities (T_a)*: They are utilities that are specifically designed for running and deploying the tests. They interact with the above-mentioned IUT artefacts and help simulate the different security attacks. In our framework, they are available as easy to use configurations by the stakeholders.
- *Monitoring artefacts (M_{au})*: These work alongside the testing utility and help in monitoring the different components. The data captured from them are used for the security-elasticity analytics. Prometheus[6] is one such example.
- *Analytics artefacts*: They are responsible for generating the concrete dependency between security and elasticity. An example of analytics artefact is a Jupyer[7] notebook that can evaluate and plot the elasticity of platform services during an attack scenario.

D. Framework design

Fig. 4 shows the different methods involved and used in the framework. In this section, we describe these methods used during the workflow.

[6]https://prometheus.io
[7]https://jupyter.org/

1) Selection of utilities and artefacts: As seen in Fig. 4, the developer first provides the annotated documents. The framework then extracts the annotated data as well as IUT to deploy from these documents. It further validates if this extracted data conforms to SEDAICO specifications. For example, if the `@subsystem` value is either `edge` or `cloud` etc. The next step then iteratively selects the appropriate profiles (and artefacts) from the information database. This relational database holds the information about annotation values and the corresponding relevant utilities and artefacts. The "What" and "Where" information from all the annotation values is used as filters for selection. The framework selects suitable a) test profiles and b) monitoring artefact. For example, if the annotation value is "cloud", the database will select all the monitoring utilities that are suitable for cloud platform services. The next filter will be applied for database monitoring utilities if the `@type` annotation value is a database. They are finally presented as a recommendation to the developer. Since this annotation is filled manually, it provides flexibility to the developer in customizing their tests and IUT.

a) Monitoring artefact selection: The monitoring artefact selection is based on the "What" and "Where" information annotations data provided by the developer. For example, if the `@subsystem` value is "cloud", and the `@type` value is "database", a query will select from this information database monitoring utility that has these values. So Grafana which is suitable for cloud services "AND" database monitoring will be selected. Such a filtered selection ensures that we capture maximum relevant data during the tests.

b) Test case selection: Similar to monitoring artefacts, the test selection is based on the annotation provided by

the developer and is selected from the information database. However, in addition to reading the two previously mentioned annotations, this also requires that the developer provides `test-type` as an annotation. One such example can be seen in Listing 2 and the selection query can be seen in Fig. 2. Test profiles are selected as a set of profile configurations, meaning that the developer needs to fill these configurations based on their requirements.

2) Profile configuration and deployment: The next method in our framework is setting (filling) up of various profile configurations for the test cases and IUT before they can be deployed. Our framework supports three distinct profile configurations (as YAML files), each of them requires DevOps activities from the developer. Our framework needs them to accurately configure, deploy run all the artefacts and tests. These three can be seen in Fig. 4.

a) IoT configuration and deployment: The framework requires IoT configuration to be filled in by the developer. This requires providing the details of the sensor such as the number and duration of sensors that will simulate the sensor subsystem of the IUT. It is required by the framework to kickstart the complete IUT and mimic the real-world scenario that the developer wants to test. It is important to distinguish that the IUT model is separate (and external) from this and these configurations do not deploy the IUT edge-cloud platform services.

b) Test profile configuration: Test profiles emulate the elasticity conditions under security attacks. We represent test profiles by a configuration file. Every test case requires a different type of configuration value to be filled in by the developer. Using test cases as configuration ensures that the developer can specify and test for various types of use-cases on the same IUT.

One example of test profiles is in the first part of Listing 3. This configuration is for case `T01-UB` and describes the information such as the number of sensors, message broker publication topic, type of broker. The framework then reads this configuration to initiate the tests.

c) Test deployment and steering: Our framework does the test deployment based on configuration profiles. This is done by simulating through the test utility profile which is usually a set of malicious IoT subsystem. The developer can steer the tests during runtime by editing them. In some cases, the deployment of testing can also be done reactively by the developer (manually controlled after looking at monitoring data) in order to accurately measure the elasticity of the system during tests. For example, the test steering section in the sample Listing 3 requires the type of sensor, rate of publication, number of parallel sensors and total duration of the test scenario.

Through the use of simple configuration files, our framework makes the dynamic steering of tests easy for developer to configure. By editing these files, they can emulate the elasticity scenario by supplying the rate, number and duration of the poisoned sensors. Based on the configuration, the developer can define the exact test parameters in the test steering section.

d) Monitoring profile configuration: The monitoring profile configurations are available as a set of YAML configuration files of the monitoring artefacts. They contain details that are required to properly interface with tests and IUT artefacts. Hence, the monitoring profile configuration will change depending on the type of artefacts selected for IUT and tests. An example of a monitoring configuration that connects with the `VermeMQ` broker is shown in Listing 4.

E. Artefact coupling and workflow between various configurations

Our framework ensures that the service (test and monitoring) deployment process is mostly automatic. However, the developer needs to provide proper configurations before our framework can deploy and run them. The general artefact deployment order is $IUT \rightarrow Monitoring \rightarrow Test$.

We can see the workflow of the experiment in Fig. 4. It also shows distinct components of the framework as well as the activities that the developer must undertake. The developer first $\langle\langle Creates \rangle\rangle$ IUT (along with integration points) with necessary annotations in the deployment files. The framework then selects these profile configurations and artefacts from the information database based on these annotations. Next, the developer needs to provide suitable configurations for both the monitoring and testing profiles that gets deployed via our framework. The analytic utilities can then be used to perform the security-elasticity dependency analytics on the collected data.

```
# Test Profile Configuration
name: "unsecured_broker"
broker:
  type: "mqtt"
  address: "localhost"
  port: 1883
  topic: "TEST_TOPIC"

# Test Steering Configuration
deployment:
  type: "sensor"
  rate: "10" # per second
  number: "5"
  duration: "1000" # in seconds
```

Listing 3: Example of test utility profile

```
- job_name: 'vernemq'
  scrape_interval: 10s
  static_configs:
    - targets:
    # location of verneMQ IUT broker
      - 195.148.20.12:8888
```

Listing 4: A sample of monitoring utility profile

F. Dependency analytics

To examine the dependency, the developer can use a set of analytics utilities to get the final results of the analytics in the form of a concrete security-elasticity dependency. The analytics utilities interface with the result aggregators as seen in Fig. 4. The input to these analytics constitutes monitoring and test result data. For example, the collected Prometheus metrics are converted in Grafana dashboards and the *csv* files exported from Grafana can be an input to the utilities.

To generate the dependency, an analytics utility performs two different types of operations on the monitoring and test data:

a) Transformations: This step normalizes the data during the normal and test conditions. These transformations are important as they enable the subsequent analytics steps to directly evaluate the elasticity of the platform services. For example, a transformation could average a discreet time-series of 10 seconds interval values to a minute average interval. This would help in evaluating the fluctuation of elasticity of a service that has a scrape interval of 10 seconds with services that have scrape interval of 1minute.

b) Evaluation: The next step of the analytics evaluates elasticity impacts using the approach described in section III-A. This includes a comparative mapping with the attack scenario. It requires three aspects, the "What", "When" and "Where" (such as edge, cloud) of the elastic anomalies that happened during the test conditions. Additionally, the "How" comes from the framework's internal information. Table II presents an example in the real world use-case.

For example, in a simple case of a poisoned sensor attack scenario, an analytics evaluates the rate of change of elasticity every minute of the cloud message broker by varying the sensor load. It then evaluates the elasticity change rate that occurs during the test scenario by varying the poisoned sensors. If this time is higher during test conditions, it is treated as an anomaly. Other cloud platform services (such as a cloud database) that didn't show the deviation are not flagged as anomalous. Additionally, the utility also builds a graph comparing the two operations. We present one concrete example of an analytics in the section IV-C.

Our framework includes analytics as Jupyter python notebooks to ensure proactivity of the developer in the analytics phase. If the developer notices during the analytics that the results are not clear/concrete they can update and re-run the analytics from specific points after steering the tests. Additionally, this also gives flexibility to the developer in selecting different types of analytics such as complex machine learning-based analytics.

IV. Evaluation

We evaluate SEDAICO framework[8] through the study of the propagation of elasticity impacts due to an attack on the message broker service on the edge subsystem. We use the analytics of our framework for finding concrete security-elasticity relationships between different services due to an attack on this broker.

A. Experiment settings

The evaluation testbed comprised of the real-world GPON[9] motivating scenario described in section II-A and presented in Fig. 1. The baseline IUT edge platform services include a VerneMQ MQTT broker "cluster" of two nodes, a mini-batching service for collating the sensor values and a service to push this data to cloud platform services. The cloud platform services had a Cassandra distributed database with 3 nodes and Kafka broker with 3 nodes. Finally, there is a Python service to ingest data from Kafka to the Cassandra database. All these services are containerized as docker containers.

The rate of data production, consumption, and load-distribution of this testbed were configured as follows:

- *Real monitoring dataset*: The dataset for the experimentation was captured from different equipment (ONU devices) of a single province with 104 distinct network equipments sending a total of 478-479 messages every 6 minutes.
- *Edge infrastructure & sensors*: VM's (Virtual Machines) were used to emulate the edge infrastructure. Each VM had platform services running on containers in a dockerized environment. These services included the edge platform services as mentioned in the baseline IUT. Additionally, sensors were simulated through scripts running on the same machine. A simulated sensor produced approx 80 messages per minute onto the edge message broker. Each VM had a varying number of simulated sensor depending on the experiment phase through IoT profile configurations. We created two VMs running same set of platform services at different locations each representing an edge infrastructure.
- *Distributed infrastructures*: To simulate distributed geo-environments, we used GCP[10] (Google Cloud Platform). GCP's compute instance created VM's (along with the platform services running atop) at two different locations. Each VM represents an edge infrastructure as described above. The locations were `europe-west-1` (Belgium) and `europe-west-3` (Frankfurt). CSC's[11] Redhat Openshift was used for hosting cloud platform services. All these cloud platform services were located in Finland.
- *Scalability*: We configured the rate of sensor data publishing and the number of sensors using the IUT Profile configuration. Our cloud platform services had higher horizontal scalability within reasonable resource limitations compared to the constrained simulated edge infrastructure. These services could be scaled manually by providing the resource description into the IUT profile configuration. For example, we can add a node to the cloud message broker by providing an additional service and suitable environment variable.

[8]Current prototype: https://github.com/rdsea/SEDAICO

[9]The implementation of this GPON scenario can be found in the repository https://github.com/rdsea/IoTCloudSamples/tree/master/scenarios/netops/

[10]https://cloud.google.com/

[11]CSC-Tieteen tietotekniikan keskus Oy, https://www.csc.fi/en/csc

Testing and monitoring profiles: We performed experimentation on `T01-UB` use-case (the "Unsecured broker" from Table I) test case and used filtered test profiles to steer the tests. The rate of publishing varied by increment of 10 messages/sec onto the edge message broker. Prometheus and Grafana were filtered by the framework as data gathering tools. The monitoring profile configurations constituted of the IP address and the JMX (Java Management eXtention) ports of these IUT services.

Analytics utilities: The analytics was done on a Jupyter Python notebook. We exported the monitoring data as a *.csv* file for analytics of dependencies. The notebook's Python code performed transformation and evaluation on the elasticity values of the platform services.

B. Running the experiment

We varied the number of concurrent normal and malicious sensors publishing in their respective profiles while keeping the publish rate the same. This was steered using the steering profile configurations. Each of the scenario, i.e. the normal and attack scenario, has multiple phases. The different phases of normal conditions are described below:

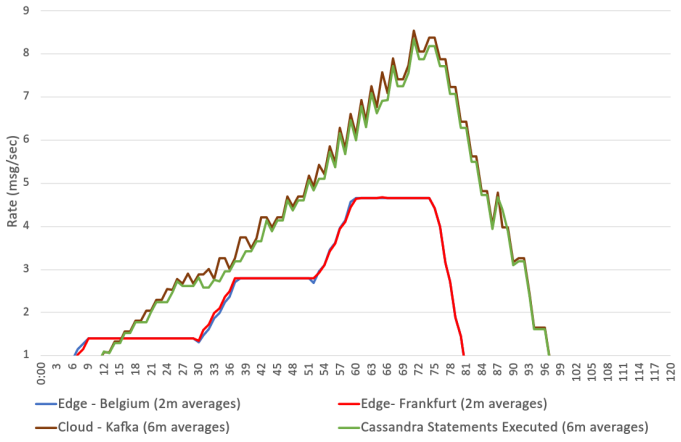
- `t0`: 0 sensors on both the edge locations
- `t1`: 3 sensors (6 in total) each on both edge locations
- `t2`: 6 sensors each on both edge locations
- `t3`: 10 sensors each on both edge locations
- `t4`: 0 sensors. (Removed with a controlled rate of decrease)

On the other hand, the attack scenario had 7 different time phases:

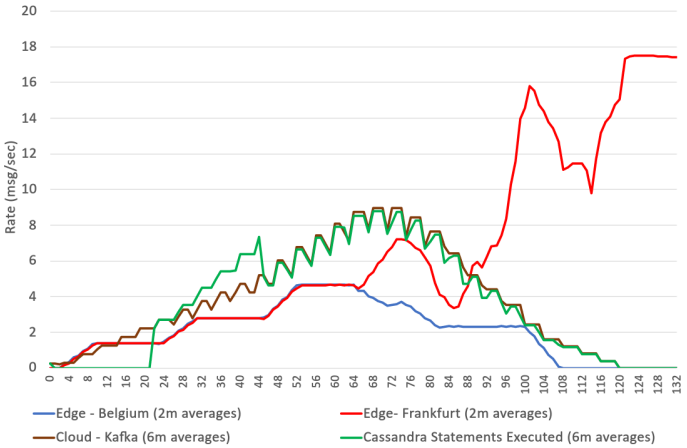
- `t0-t2`: Same as in normal condition
- `t3`: 10 sensors each on both edge locations. 5 poisoned sensors on Frankfurt edge location
- `t4`: 6 sensors each on both edge locations. 15 poisoned sensors on Frankfurt edge location
- `t5`: 3 sensors each on both edge locations. 15 poisoned sensors on Frankfurt edge location
- `t6`: 0 sensors each on both edge locations. 20 poisoned sensors on Frankfurt edge location
- `t7`: 0 sensors each on both edge locations. 23 poisoned sensors on Frankfurt edge location

1) Experiments: During the normal scenario, the message throughput rates in the cloud platform services and edge platform services have similar patterns. This is true for both the scale-up and scale-down operations. Since the values were pushed into the cloud message broker in batches, there are periodic sharp peaks from phase `t1-t3` on the platform services of cloud subsystem (refer to Fig. 5a). During phase `t4`, the drop in cloud message broker throughput follows the same trend as edge message brokers.

In the scenario of attack condition, there is a similar scale-up between phases `t0-t3`. That is, adding 5 poisoned sensors on edge didn't produce any difference on cloud platform services. However, at phase `t4-t6` there is a visible slowdown in the frequency of ingestion on the cloud location. At time phase `t7`, a constant rate can be seen after healthy sensors were decreased to 0.



(a) Normal Operation



(b) Test Operation

Fig. 5: Elasticity of Services under unsecured broker configurations

Characterization	Value
What	Database Elasticity
Where	Cloud
When	6m averages of time-series monitoring data of cloud database
How	• Perform elasticity change detection at each data point • Derive the rate of increase in elasticity of a service from this change detection • Measure the frequency of message ingestion (i.e. how long it takes to ingest x data) and rate of message ingestion (i.e. how many messages are ingested every minute)

TABLE II: Using W3H characterization in analytics

C. Analytics and results

The analytics measured the elastic behaviour of the platform services under the two scenarios. They are plotted in Fig. 5. Fig. 5a shows scalability changes in the infrastructure with respect to different number of sensors. On the other hand, Fig. 5b plots the effect on services under the unsecured broker

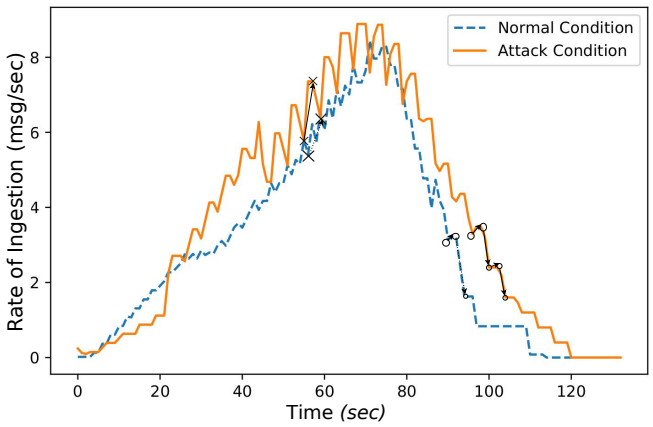


Fig. 6: Comparison of average overall ingestion rate on the cloud database

test condition.

To derive the results, the analytics scripts first normalized, cleaned, and transformed the monitoring data into 2 minute average for edge platform services and 6 minutes average for cloud platform services. This was done to ensure all the data points used in the elasticity evaluation represented accurate batch sizes. The evaluation maps the changes in elasticity points of the services at different intervals and contrasted them with the attack scenario. The notebook used the W3H information to find the dependency between the edge message broker and the cloud database. It looked at the "What", "Where" and "When" and utilized the "How" from Table II. From the "How" information, it performed change point detection as well as the calculation of the duration of elasticity peaks from on the data points in both the normal and attack scenario.

1) Result – security attack on unsecured edge message broker and propagation of elasticity impacts: This analyzes the propagation and impact of elasticity on the various services on the cloud and edge system when the edge message broker service is on an attack. During phase `t4-t6`, the decrease in the elasticity of different cloud platform services is slower than the normal condition. This can be seen in the Fig. 5b as there are flatter peaks at larger intervals compared to normal condition. Another observation about impact is that of the elasticity of edge message broker and its attempts to clear the queue of unsubscribed messages at periodic intervals. It can be observed by the decrease in rate after few minutes. After setting up the IUT, test profiles and monitoring profiles, the analytics utilities in our framework gives an idea about how different services in the IUT would behave (elasticity) during the attack conditions.

2) Results – characterizing dependency of elastic impacts and attack on unsecured broker: To characterize the dependency between the elasticity of the cloud database and the attack on an unsecured broker, the analytics utilities performed an evaluation of the elasticity during the attack and normal scenarios. The comparative plot can be seen in Fig. 6. Based on the analytics (and the graph), the following two characterizations can be made:

- *Cloud platform services scale down slowly during the attack on the unsecured broker on edge:* In the case of attack, the scale-down rate is slightly slower than the normal operation scenario (the circled annotations in the graph). This is partly due to an increased load on the MQTT message broker leading to a larger message queue. A longer message queue at the edge message broker causes an overall latency delay during the scale-down operations.
- *The frequency of message ingestion on the cloud platform services slows down while the rate of ingestion remains the same:* It is important to differentiate here that while the frequency of messages that are pushed onto the cloud slows down (i.e. it took more time to ingest the data), the average rate (per 6 minutes) of ingestion remain the same (See the crossed annotations while scaling up). For example, Kafka still receives ∼1400 messages every minute. But under normal condition, a batch of 700 messages is published from the edge location in 2 seconds whereas it took around 15 seconds when the edge message broker is under attack. Since the average ingestion rate still remained the same, it is very easy for developers to miss the attack on the edge device by just looking at cloud dashboards.

D. Discussion

The above subsection demonstrates the analytics of two concrete security-elasticity dependency during attack on an unsecured edge message broker. While the framework simplifies the selection of appropriate monitoring and test utilities, the developer still needs to be actively involved to provide necessary profile configuration and steer the tests. Also, the framework only aids the developer in security-elasticity dependency by presenting the results. The concrete result still needs to be identified by the developer and is limited by their domain-expertise to infer the outcomes.

Nonetheless, it can be easily seen why such a dependency cannot be generated directly by using monitoring dashboards and how the framework aids in every aspect such as tests, IUT, deployment and analytics etc. Moreover, it is easy for the developer to adapt this framework for their use-case and study the security-elasticity dependency through simple models and configurations.

V. Related work

Security-elasticity dependencies: There are not a lot of researches that analyze the security attacks and elasticity impacts among different platform services. However, many researches aim to solve this problem by increasing the observability of the system. The work presented in [8] and [9] describes a multi-level observability of the cloud orchestration system such as Kubernetes. However, container orchestration systems are not yet used extensively on the edge devices as they are resource heavy. Hence, this approach cannot be directly used in a multi

subsystem distributed IIoT infrastructure that requires resource constrained processing. Similarly, an observability enhancing framework such as [10] discusses the feasibility only over cloud subsystem microservices. Systems such as [11], [12] for observability in IoT and edge networks usually do not integrate with cloud native applications and hence are not scalable onto the cloud subsystems [13]. The BonFIRE framework by Kavoussanakis et al. [14] aims to create a testbed for cloud applications and allows the user to host wide scenarios of services that are not yet deployed to the cloud. This work, however, does not provide its user with the analytics services to help generate security-elasticity dependency. Moreover, it is only able to evaluate cloud platform services. The work by Jiang et al. in [15] analyzes the security of IIoT devices especially against a variation of a popular botnet attack. This work (and the attack) focuses on the impacts on the IIoT devices and not on the platform services. The work in [16] uses a CUmulative SUM (CUSUM) to model state and residual parameters in an IIoT platform service. Unlike our work, the methods used in this framework only focus on anomaly detection for the elasticity impacts of the underlying IIoT network subsystem.

Monitoring utilities: VARYS [3] is a model-driven technology-agnostic monitoring tool that observes the cloud stack at multiple levels. However, this framework is framework-agnostic only at the cloud subsystem and has not been extended to the IoT or edge subsystem. C'Mon [17] defines security related monitoring as a compliance of any service to secSLAs (Security Service Level Agreements). SECUPerf [18] assesses the end-to-end performance and security of a cloud system. Unlike our framework, this work determines security attacks from performance based on certain heuristics and is static unless these heuristics are updated.

We also need to monitor these services to ensure proper capturing of anomalies that creep up in the system. One of the most common techniques to achieve this is through the use of dedicated monitoring frameworks such as SONATA monitoring project [4] or work by Boncea et al. [5]. Both these works discuss monitoring data collection from platform services (through probes), use of push gateways and user alerts through rules via an alert manager. Another important aspect is performing the association of security attacks and its impact on the elasticity of the cloud [19]. The tools used by popular monitoring frameworks such as [20], [4], [5] lack the ability to correlate the monitoring data. Researches such as [21] that attempt to automate testing of elasticity are limited in scope. That is, they are able to only monitor and test the elasticity of infrastructure (such as VM) and not an individual platform service.

Security of IIoT platform services elasticity: While there are multiple literatures that address security in edge and cloud subsystems [22], [23]; there exists little literature that attempt to perform a holistic analysis of the IIoT infrastructure and study the underlying dependence with respect to security attack and elasticity impacts. STRATFram [24] framework allows the users to specify elasticity models and evaluates the elasticity strategies on the cloud. However, this work does not focus on the security (attack) aspect of elasticity. Moreover, the framework has only been evaluated in the context of cloud platform services and not on an edge-cloud continuum like ours. IntelligentSDS [25] employs multiple security agents to perform white-box testing to identify threats on edge-cloud network services. However, this framework does not provide any clarity on the elasticity impacts of these network services. Fadi Al-Turjman and Sinem Alturjman propose an access control framework named "CSIP" [26] that performs analytics and provisions healthcare applications with context-aware features. However, this framework lacks a mechanism to support users' custom analytics. Kingfisher [27] security framework works on intrusion detection using VAE (Variational AutoEncoders) from the incoming data. The scope of this framework only supports the detection of network attacks such as SYN Flood through a pre-trained VAE model. Moreover, we cannot use it to analyze a rich set of elasticity impacts. Many of the other works however are either context-specific ([26], [28]) or application-specific ([29], [30]) and hence, unlike our framework, they do not allow the developer to perform analytics over diverse IIoT infrastructure contexts and applications.

VI. Conclusions and future work

Complex interactions between platform services in an IIoT infrastructure create security-elasticity dependencies that are often inconspicuous and hence are often difficult to identify by the developer of the system. In this paper, we have presented our SEDAICO framework and described the methodologies that it uses to capture and analyze these security-elasticity dependencies. Through the use of these analytics, the developer can a) find a relationship between the elasticity of platform services during a test scenario, and b) establish a concrete dependency between the elasticity impacts on a platform service and a specific security attack.

We evaluated the dependency analytics on different scenarios and utilized the use-case of an unsecured broker for the evaluation of our framework. In future, this framework can be expanded for multiple different use-cases of security attacks and modeling of a data model that can be used to represent the various types of dependency. A key issue before this expansion will require a fine-grained approach for creating the security attack data and test case characterization. Inclusion and integration of domain expertise from the ML and deep learning domains will be useful. For example, an integration of a CRA (Convolutional Recurrent Autoencoder) based anomaly detection engine [31] into the analytics can be done.

Acknowledgments

We thank our collaborators in Vietnam for sharing the real dataset used in this paper. The authors also wish to acknowledge CSC – IT Center for Science, Finland, for cloud resources.

REFERENCES

[1] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial internet of things: Challenges, opportunities, and directions," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 11, pp. 4724–4734, 2018.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[3] A. Tundo, M. Mobilio, M. Orrù, O. Riganelli, M. Guzmàn, and L. Mariani, "Varys: An agnostic model-driven monitoring-as-a-service framework for the cloud," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 10851089.

[4] P. Trakadas, P. Karkazis, H.-C. Leligou, T. Zahariadis, W. Tavernier, T. Soenen, S. Van Rossem, and L. Miguel Contreras Murillo, "Scalable monitoring for multiple virtualized infrastructures for 5g services," in *SoftNetworking 2018, The International Symposium on Advances in Software Defined Networking and Network Functions Virtualization*, 2018, pp. 1–4.

[5] R. Boncea and I. Bacivarov, "A system architecture for monitoring the reliability of iot," in *Proceedings of the 15th International Conference on Quality and Dependability*, 2016, pp. 143–150.

[6] R. D. Hipp, "SQLite," 2020. [Online]. Available: https://www.sqlite.org/index.html

[7] D. R. Morse, S. Armstrong, and A. K. Dey, "The what, who, where, when, why and how of context-awareness," in *CHI 00 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA 00. New York, NY, USA: Association for Computing Machinery, 2000, p. 371.

[8] R. Picoreti, A. Pereira do Carmo, F. Mendona de Queiroz, A. Salles Garcia, R. Frizera Vassallo, and D. Simeonidou, "Multilevel observability in cloud orchestration," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing*, 2018, pp. 776–784.

[9] S. Taherizadeh and V. Stankovski, "Incremental learning from multi-level monitoring data and its application to component based software engineering," in *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, 2017, pp. 378–383.

[10] N. Marie-Magdelaine, T. Ahmed, and G. Astruc-Amato, "Demonstration of an observability framework for cloud native microservices," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2019, pp. 722–724.

[11] A. Johnsson and C. Rohner, "On performance observability in iot systems using active measurements," in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, 2018, pp. 1–5.

[12] N. Jacobs, S. Hossain-McKenzie, A. Summers, C. B. Jones, B. Wright, and A. Chavez, "Cyber-physical observability for the electric grid," in *2020 IEEE Texas Power and Energy Conference (TPEC)*, 2020, pp. 1–6.

[13] S. Niedermaier, F. Koetter, A. Freymann, and S. Wagner, "On observability and monitoring of distributed systems – an industry interview study," in *Service-Oriented Computing*, S. Yangui, I. Bouassida Rodriguez, K. Drira, and Z. Tari, Eds. Cham: Springer International Publishing, 2019, pp. 36–52.

[14] K. Kavoussanakis, A. Hume, J. Martrat, C. Ragusa, M. Gienger, K. Campowsky, G. V. Seghbroeck, C. Vzquez, C. Velayos, F. Gittler, P. Inglesant, G. Carella, V. Engen, M. Giertych, G. Landi, and D. Margery, "Bonfire: The clouds and services testbed," in *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, vol. 2, 2013, pp. 321–326.

[15] X. Jiang, M. Lora, and S. Chattopadhyay, "An experimental analysis of security vulnerabilities in industrial iot devices," *ACM Trans. Internet Technol.*, vol. 20, no. 2, May 2020. [Online]. Available: https://doi.org/10.1145/3379542

[16] H. R. Ghaeini, D. Antonioli, F. Brasser, A.-R. Sadeghi, and N. O. Tippenhauer, "State-aware anomaly detection for industrial control systems," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, ser. SAC '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 16201628. [Online]. Available: https://doi.org/10.1145/3167132.3167305

[17] S. Alboghdady, S. Winter, A. Taha, H. Zhang, and N. Suri, "C'mon: Monitoring the compliance of cloud services to contracted properties," in *Proceedings of the 12th International Conference on Availability, Reliability and Security*, ser. ARES '17. New York, NY, USA: Association for Computing Machinery, 2017.

[18] K. Xiong and M. Makati, "Assessing end-to-end performance and security in cloud computing," in *Proceedings of the Symposium on Applied Computing*, ser. SAC '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 405410.

[19] H. J. Syed, A. Gani, R. W. Ahmad, M. K. Khan, and A. I. A. Ahmed, "Cloud monitoring: A review, taxonomy, and open research issues," *Journal of Network and Computer Applications*, vol. 98, pp. 11–26, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804517302783

[20] M. Yang and M. Huang, "An microservices-based openstack monitoring tool," in *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*. IEEE, 2019, pp. 706–709.

[21] A. Alourani, M. A. N. Bikas, and M. Grechanik, "Search-based stress testing the elastic resource provisioning for cloud-based applications," in *Search-Based Software Engineering*, T. E. Colanzi and P. McMinn, Eds. Cham: Springer International Publishing, 2018, pp. 149–165.

[22] P. Zhang, M. Zhou, and G. Fortino, "Security and trust issues in fog computing: A survey," *Future Generation Computer Systems*, vol. 88, pp. 16–27, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X17329722

[23] A. Sinha, G. Shrivastava, P. Kumar, and D. Gupta, "A community-based hierarchical user authentication scheme for industry 4.0," *Software: Practice and Experience*, 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2832

[24] A. B. Jrad, S. Bhiri, and S. Tata, "Stratfram: A framework for describing and evaluating elasticity strategies for service-based business processes in the cloud," *Future Generation Computer Systems*, vol. 97, pp. 69–89, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X18306125

[25] F. Yang, S. Zhang, S. Song, R. Li, Z. Zhao, and H. Zhang, "A testbed for intelligent software defined security framework," in *Proceedings of the ACM Turing Celebration Conference - China*, ser. ACM TURC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3321408.3321610

[26] F. Al-Turjman and S. Alturjman, "Context-sensitive access in industrial internet of things (iiot) healthcare applications," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 6, pp. 2736–2744, 2018.

[27] G. Bernieri, M. Conti, and F. Turrin, "Kingfisher: An industrial security framework based on variational autoencoders," in *Proceedings of the 1st Workshop on Machine Learning on Edge in Sensor Systems*, ser. SenSys-ML 2019. New York, NY, USA: Association for Computing Machinery, 2019, pp. 7–12. [Online]. Available: https://doi.org/10.1145/3362743.3362961

[28] Z. Khan, A. Anjum, and S. L. Kiani, "Cloud based big data analytics for smart future cities," in *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, 2013, pp. 381–386.

[29] K. Gai, M. Qiu, and S. A. Elnagdy, "A novel secure big data cyber incident analytics framework for cloud-based cybersecurity insurance," in *2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS)*, 2016, pp. 171–176.

[30] K. A. Torkura, M. I. H. Sukmana, F. Cheng, and C. Meinel, "Slingshot - automated threat detection and incident response in multi cloud storage systems," in *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*, 2019, pp. 1–5.

[31] C. Yin, S. Zhang, J. Wang, and N. N. Xiong, "Anomaly detection based on convolutional recurrent autoencoder for iot time series," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pp. 1–11, 2020.