Self-service Cybersecurity Monitoring as Enabler for DevSecOps

Jessica Diaz, Jorge E. Pérez, Miguel A. Lopez-Peña, Gabriel A. Mena, Agustín Yague

Abstract—Current IoT systems are highly distributed systems that integrate cloud, edge and fog computing approaches depending on where intelligence and processing capabilities are allocated. This distribution and heterogeneity make development and deployment pipelines very complex and fragmented with multiple delivery endpoints above hardware. This fact prevents rapid development and makes the operation and monitoring of production systems a difficult and tedious task, including cybersecurity event monitoring. DevSecOps can be defined as a cultural approach to improve and accelerate the delivery of business value by making dev/sec/ops teams’ collaboration effective. This paper focuses on self-service cybersecurity monitoring as an enabler to introduce security practices in a DevOps environment. To that end, we have defined and formalized an activity that supports ‘Fast and Continuous Feedback from Ops to Dev’ by providing a flexible monitoring infrastructure so that teams can configure their monitoring and alerting services according their criteria (you build, you run, and now you monitor) to obtain fast and continuous feedback from operation and thus, better anticipate problems when a production deployment is performed. This activity has been formalized using the Software & Systems Process Engineering Metamodel by OMG and its instantiation is described through a case study that shows the versioned and repeatable configuration of a cybersecurity monitoring infrastructure (Monitoring as Code) through virtualization and containerization technology. This self-service monitoring/alerting allows breaking silos between dev, ops, and sec teams by opening access to key security metrics, which enables a sharing culture and continuous improvement.

Index Terms—Cybersecurity, DevSecOps, Monitoring as Code, Self-service Monitoring.

I. INTRODUCTION

Current societies demand more intelligent, more energy-efficient and more comfortable cities, hospitals, offices, factories, vehicles and homes. The Internet of Things (IoT), understood as those infrastructures of “interconnected entities, people, systems and information resources together with services which processes and reacts to information from the physical world and virtual world” [1], makes possible to provide citizens and businesses with these smart applications and services. The International Telecommunication Union (ITU) defines IoT as a “global infrastructure for the Information Society, enabling advanced services by interconnecting (physical and virtual) things based on, existing and evolving, interoperable information and communication technologies” [2]. These systems are based on a large number of interconnected devices resulting in a global network that interconnects thousands or even millions of “things” [3]. This has generated an important economical and societal impact [4], but also a great challenge to support scalability and performance of IoT systems.

Cloud computing is the natural candidate to support the exponential growth of data generated by IoT systems [5]. Cloud IoT platforms are rapidly extending their use by providing flexible, scalable and high-performance services for data processing, data storing, and data analysis [6][7]. However, last trends also show the advances in implementing ‘edge intelligence’ with different levels of processing capabilities, which is known as Fog computing and Edge computing [8]. Thus, we can say that current IoT systems are highly distributed systems that integrate cloud, edge and fog computing approaches depending on where intelligence and processing capabilities are allocated. This distribution and heterogeneity make development and deployment pipelines very complex and fragmented—devices-side code, server-side code, mobile apps, desktop apps, etc.—with multiple delivery endpoints above hardware. This fact prevents rapid development (aka. continuous software engineering [9]) and makes the operation and monitoring of production systems a difficult and tedious task, and more specifically cybersecurity event monitoring.

Cybersecurity is an area of ICT that is responsible for the protection of information assets, through the treatment of threats that put at risk the information that is processed, stored and transported by interconnected information systems [10]. Cybersecurity can be obtained through systematic development, but developers need to be aware of these risks and security issues [11]. If cybersecurity is done after development is finished, systems are built in an insecure way with bugs that are difficult to fix. However, when security teams share knowledge and provide tools to development and operation teams, these last ones can change the systems and application accordingly [12]. With this purpose, DevSecOps can be defined as a cultural approach to improve and accelerate the delivery of

Jessica Diaz is with Universidad Politécnica de Madrid, c/Alan Turing, Madrid, Spain (e-mail: yesica.diaz@upm.es)
Jorge Pérez is with Universidad Politécnica de Madrid, c/Alan Turing, Madrid, Spain (e-mail: jorgeenrique.perez@upm.es)
Miguel A. Lopez-Peña is with Universidad Politécnica de Madrid & SATEC, Avda. de Europa, 34 A, 28023 Aravaca, Madrid, Spain (e-mail: miguelangel.lopez@satec.es)
Gabriel A. Mena is with Accenture Security & Universidad Politécnica de Madrid (efilami@gmail.com; andres.mena.java@alumnos.upm.es)
Agustín Yague with Universidad Politécnica de Madrid, c/Alan Turing, Madrid, Spain (e-mail: agustinyague@upm.es)

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.
business value by making dev/sec/ops teams’ collaboration effective [13]. DevSecOps is about “breaking the silos of security, giving that knowledge to the different teams, and ensuring that security is implemented at the right level and at right time” [12]. This approach is relatively recent, and thus little is known about good practices associated with DevSecOps in industry in general, and IoT industry in particular. DevOps does not have methodologies as Agile has (e.g., Scrum and extreme programming), and thus companies have developed their DevSecOps practices largely from scratch—by training employees on the fly.

This paper focuses on self-service cybersecurity monitoring as an enabler to introduce security practices in a DevOps environment for IoT-based systems. To that end, we have defined an activity that supports ‘Fast and Continuous Feedback from Ops to Dev’ (aka, F&CF) that enables quick detection of problems and provides information to fix them, ideally long before customers are impacted. This activity provides by a flexible monitoring infrastructure for highly distributed and heterogeneous systems, as IoT systems, so that development and operation teams can configure their own monitoring and alerting services according their criteria (you build, you run, and now, you monitor). The goal is to obtain fast and continuous feedback from operation to development in order to discovering security issues and bugs at earlier stages, and thus, better anticipate security problems—cyberattacks patterns—when systems are deployed into production.

The definition of this activity has been formalized using the Software & Systems Process Engineering Metamodel (SPM) [14] by the Object Management Group (OMG). The instantiation of this activity is described through a case study that shows the automated deployment of a security monitoring infrastructure (and policy configurations) through virtualization and containerization technology, which we have called Monitoring as Code, similarly to the well-known Infrastructure as Code. This code can be versioned and configured, and these configurations are repeatable. This automation allows teams self-service monitoring and alerting, which breaks silos between dev, ops, and sec teams by opening access to key security metrics, enabling a sharing culture and continuous improvement.

This paper is structured as follows: Section 2 briefly introduces the concepts of IoT and DevSecOps. Section 3 describes related work. Sections 4 presents the F&CF activity. The instantiation of this activity is described through a case study in Section 5. Finally, conclusions and further work are described in Section 6.

II. BACKGROUND

A. Internet of Things

IoT is increasingly important due to the huge volume of devices that already are, and that will be connected in the near future, as well as the advanced services that these devices can leverage in smart grids, smart transport, Industry 4.0, digital healthcare, and smart cities, for instance. By the end of 2018 the number of the devices connected to Internet of Things around the world reached 22 billion pieces, and it is predicted that by 2025 and 2030 in the world there will be 38.6 billion and 50 billion IoT-devices respectively. These devices are the hardware infrastructure of IoT systems (set of connected physical devices, sensors, actuators, communication networks, etc.). However, IoT systems are also compound by a software infrastructure, from the software that is embedded on devices (e.g., microcontroller programming with an I/O interface) to the middleware and platforms that are responsible for the ingestion, storage, processing, and analysis of data as well as the advanced and vertical services built on them.

The development and deployment of such systems into production as well as their operation and monitoring are highly complex due to the heterogeneity of delivery endpoints—devices-side code, server-side code, mobile apps, desktop apps, etc.— on a single IoT system. The Cluster of European Projects on Software Engineering for Services and Applications (SE4SA) highlights the importance of ensuring Quality of Service (QoS) and correctness of IoT systems together with the complexity of such purpose as devices and software could change for various reasons such as bugs and failures, changing interfaces and implementations, and changing requirements [15].

Thus, IoT systems, as other systems did, needs to adopt organizational capabilities to develop, release, and learn from software in rapid cycles (rapid software development [9]). Additionally, silos between development, IT operation, and security, which currently exist in most technology companies, make early and frequent releases in production more complex, which means less business innovation and lower capability to compete in the market. Thus, IoT also needs to adopt organizational approaches to make effective the collaboration between development, operation, and security teams, which is crucial to increase QoS and speed up releases into production [15]. To that end, software engineering discipline need to be consolidated with already consolidated engineering—system, control, communication—which are typical of IoT systems. In the last few years, software engineering has shifted to new development paradigms that promote fast speed in releases and quick response time to customer demands. One of them is DevOps, that extended to DevSecOps.

Leading IoT software companies, such as Bosch Software Innovations, are working of these IoT challenges. This work focuses on in this emerging trend: IoT DevSecOps, specifically the continuous feedback from operation to development to learn from systems into production, particularly we focused on monitoring cybersecurity events to learn from security vulnerabilities.

B. IoT Platforms & Cloud & Edge

Currently there are several platforms in the market that facilitate the development of IoT-based systems commonly referred to as IoT Platforms [16][17]. Examples of IoT


\[2\] Forum where European projects funded by the European research programmes collaborate to identify new challenges to be tackled in future initiatives https://eucloudclusters.wordpress.com/software-engineering-for-services-and-applications/
platforms are FIWARE\textsuperscript{3}, SOFIA\textsuperscript{4} and ThingsBoard\textsuperscript{5}, which offer services and APIs (Application Programming Interfaces) for the integration and interoperability of multiple devices as well as the storage, processing and analysis of a huge volume of data (data analytics). These services are fundamental for the construction of intelligent systems and applications in the domains previously mentioned. Together these platforms, a set of IoT services offered by well-known cloud providers are emerging in the last few years. This integration of IoT with Cloud mainly responds to two issues: (i) the processing and storage low capacity of devices together the need of high computational requirements, and (ii) the high initial investment in on-premise infrastructure [18]. Cloud computing offers self-provisioning infrastructure as a service (IaaS) and platform as a service (PaaS), which makes it the natural candidate for providing almost unlimited resources for data ingestion, storage, processing, and analysis tasks as well as the platform to develop, run and manage applications that implement these tasks over cloud infrastructure. Cloud-based services improve productivity and address big data management and high computational requirements [15]. Both research [5][6] and industry—Amazon, Microsoft, Google and IBM—promote the so-called IoTCloud. Consequently, most cloud providers offer IoT-specific PaaS such as Microsoft Azure IoT Suite\textsuperscript{6}, Google Cloud IoT Core\textsuperscript{7}, Amazon Web Service IoT\textsuperscript{8}, and IBM Watson IoT\textsuperscript{9}.

However, last advances in implementing ‘edge intelligence’ aims to deal with open issues in cloud computing [19], such as reduce latencies and save bandwidth. Nowadays most authors accept that fog/edge computing features should also be included in the definition of the new IoT architectures and integrated with cloud-native architecture [20][21][22]. Microsoft Azure IoT Edge, AWS IoT Greengrass, and Kinetics Edge & Fog Processing by Cisco have extended their IoT platforms to the edge computing and/or fog computing paradigms. In previous work [23][24], some authors of this paper have defined new IoT architectural models that integrate cloud, edge and fog models.

C. DevOps & DevSecOps

In the last years software engineering has shifted to new paradigms for rapid and frequent creation and operation of software and services based on the automation of software engineering tasks such as building, testing, deployment, and releasing [25][26]. DevOps can be defined as an organizational approach, culture, and movement that focuses on collaboration and integration of development and operation to produce software products and services more rapidly and more quality [27]. To that end DevOps requires the development team to work closely and efficiently with the operations team. DevOps plays a fundamental role for companies whose business greatly depends on how efficient development and operation are.

DevOps relies on well-known lean and agile principles; therefore, DevOps is also related to the application of lean and agile principles to the whole company—not only development. In fact, Gartner defines DevOps as a change in IT culture, focusing in rapid IT service delivery through the adoption of agile and lean practices. Many of these principles and practices can be applied to security by building security in from the start (shifting security left [12]), automating security test and monitoring, and making collaboration between developers and security engineers effective [13], aka. DevSecOps.

However, a complicated task when adopting DevOps is to define the set of practices and processes that must guide the development lifecycle. DevOps embodies a vast and diverse set of practices from which some patterns can be generically applied under certain conditions, but there is no set of standardized practices. In fact, the edition of a new standard on DevOps is expected for December 2020 (IEEE P2675 DevOps Standard for Building Reliable and Secure Systems Including Application Build, Package and Deployment). In this paper we follow the classification and practices collection described in [28] that summarizes the practices in three ways:

- 1st way: fast flow of work from development into operation
- 2nd way: fast and continuous feedback
- 3rd way: continuous improvement

This paper focuses in the second way, which means creating and analyzing telemetry to detect problems, facilitating better informed decisions making to fix them. Specifically, this paper focuses on self-service cybersecurity monitoring as an enabler to introduce security practices in a DevOps environment for IoT-based systems.

III. RELATED WORK

The European cluster SE4SA emphasizes that “Managing the development complexity and risks in both design and runtime phases is considered crucial, in order to increase QoS, reduce the time needed to move new releases in the operation environment and enable the constitution of new and more effective cooperation processes between the development and the operation team” [6]. This assertion was made in 2016 in a context defined to identify new challenges of software engineering for smart systems and applications, specifically in domains such as IoT, cloud, and big data. During the same year, some work already addressed this challenge: The work [29] describes a system for automated lifecycle management of IoT applications requiring cellular network access. This system supports DevOps by automating the deployment pipeline of IoT applications, i.e. the system automates allocation and deallocation of network and cloud resources based on the information provided by a monitoring infrastructure—network, CPU and memory status. This monitoring infrastructure is implemented using the open source technology named Elastic Stack [30]. Similarly, [31] describes a system for data operation visibility based on technologies such as Apache Kafka, Spark, HDFS (Hadoop distributed file system), MongoDB NoSQL database, and Node.js-based web servers. The same authors in [32] focus on automated continuous integration and deployment of IoT cloud services using containers. The work [33] describes a cloud ecosystem to support both IoT—fog computing—and

\textsuperscript{3} https://www.fiware.org
\textsuperscript{4} http://sofia2.com
\textsuperscript{5} ThingsBoard Open-source IoT Platform, https://thingsboard.io/
\textsuperscript{6} https://azure.microsoft.com/iot/Suite
\textsuperscript{7} https://cloud.google.com/iot-core/
\textsuperscript{8} https://aws.amazon.com/iot
\textsuperscript{9} https://www.ibm.com/internet-of-things/
DevOps—automated management of infrastructure—but not explicitly together. Finally, in [34] the authors identify the main challenges to adopt DevOps to the embedded systems domain in front of the usual practice in the web domain.

Therefore, only [29] and [31] address monitoring infrastructures for IoT systems to convey feedback from operation to development. Our contribution differs from previous ones in two aspects: (i) we focus on advanced cybersecurity detection by analyzing devices’ log traces (i.e. application log), not only network, CPU and memory usages (i.e. operating system log); and (ii) the monitoring infrastructure is created and configured automatically, it is versioned and repeatable, easily adaptable to changes into production environments following DevOps good practices, such as of Infrastructure as Code, which we have recalled as Monitoring as Code (MaC).

IV. MODELING FAST & CONTINUOUS FEEDBACK

This section defines the activity that supports DevOps principle “fast and continuous feedback from operation to development” (aka, F&C&F). This activity monitors key metrics and enables quick detection of problems and provides developers with information to fix them, ideally long before customers are impacted. This activity provides a flexible monitoring infrastructure for IoT systems, so that developers and operators can configure their own monitoring and alerting services according their criteria (you build, you run, and now, you monitor). The goal is to obtain fast and continuous feedback from operation to development in order to discovering security issues and bugs at earlier stages, and thus, better anticipate problems—such as, cyberattacks patterns—when systems are deployed into production.

A. Definition of F&C&F Activity using SPEM

We have formalized the activity decomposition of the DevOps principle “fast and continuous feedback from operation to development” using the standard SPEM [14]. A key feature of SPEM is the separation of the ‘method content definitions’ from their application in the development process. A method content element represents libraries of reusable content such as definition of tasks, roles, tools or work products, whereas a process element represents a specific way of performing a project using a specific technology [14] (see Appendix A for graphical notations).

Fig. 1 shows the activity F&C&F that consists of three TaskUse, each of which is described in its corresponding TaskDefinition through the relation <<content trace>>. Each TaskDefinition, as a method content element, provides a step-by-step explanation, describing how specific development goals are achieved independent of the placement of these steps within a development lifecycle. Each TaskUse, as a process element, takes its method content element and relate them to partially ordered sequences that are customized to specific types of projects. The TaskDefinition we have defined are (see Fig. 1):

1. Create Telemetry: This task is responsible for reading the values of those elements to be monitored—that are indicated in the WorkProductDefinition ‘target telemetry’—and generating a new WorkProductDefinition ‘telemetry’ with the collected data. Table 1 shows the formal description of this task conforms to SPEM 2.0.

2. Analyze telemetry, This task is responsible for detecting anomalies from collected data in the WorkProductDefinition ‘telemetry’ and gathering these anomalies in a WorkProductDefinition called ‘anomalies’.

3. Supervision, This task is responsible for diagnosing problems the WorkProductDefinition ‘anomalies’ and execute an action to prevent the problem (e.g., to deploy a new version of the software on some of the monitored elements). This task generates the appropriate reports, alarms, and corrective actions that are described in the WorkProductDefinition ‘activity log’.

Table 1. SPEM description of Create Telemetry TaskDefinition

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name (from UML 2 Infrastructure)</td>
<td>create telemetry</td>
</tr>
<tr>
<td>presentationName (from Describable Element.ContentDescription)</td>
<td>Create Telemetry</td>
</tr>
<tr>
<td>Kind (from Extensible Element.Kind)</td>
<td>Discipline: feedback Ops2Dev</td>
</tr>
<tr>
<td>briefDescription (from Describable Element.ContentDescription)</td>
<td>This task defines how to obtain telemetry and how this telemetry is stored.</td>
</tr>
<tr>
<td>mainDescription (from Describable Element.ContentDescription)</td>
<td>This task is responsible for reading the values of those elements to be monitored—artifact ‘target telemetry’—and generating a new artifact with the collected data.</td>
</tr>
<tr>
<td>purpose (from Describable Element.ContentDescription)</td>
<td>Create the telemetry to be analyzed.</td>
</tr>
<tr>
<td>ownedTaskDefinitionParameter: Default_TaskDefinitionParameter</td>
<td>It is an association (composition) that related two WorkProductDefinition: target telemetry (mandatory, input) and telemetry (mandatory, output)</td>
</tr>
<tr>
<td>usedTool: ToolDefinition</td>
<td>Event &amp; data collection tool</td>
</tr>
<tr>
<td>/Step: Step</td>
<td></td>
</tr>
<tr>
<td>Precondition (from Work Definition)</td>
<td>The artifact ‘define telemetry’ exists</td>
</tr>
<tr>
<td>Postcondition (from Work Definition)</td>
<td>Generates the artifact ‘telemetry’</td>
</tr>
</tbody>
</table>
Regarding the WorkProductDefinition called ‘target telemetry’, it is necessary to highlight that this input to the activity provides a complete description of the elements of the IoT system that must be monitored and how to do it. The artifact ‘target telemetry’ is an aggregation of several metrics, described in the SPEM model (Fig. 1) through Guidelines. The taxonomy of metrics we defined is as follows:

a. User behavior metrics come from real people's actions lead to a deeper understanding of the experience users are having with the IoT-based system.

b. Business metrics related to vertical applications deployed on the IoT platform.

c. Application metrics, such as transaction times, user response times, application faults, etc.

d. Environment metrics that refer to the IoT platform and the physical infrastructure on which the system is deployed, such as server traffic, CPU load, and communications networks.

e. Pipeline metrics (integration and deployment pipelines, such as, code build/test/deployment history, lead times, deployment frequencies, test coverage, etc. These metrics can be collected form CI/CD (continuous integration/continuous deployment) servers and/or application-release automation tools, and software orchestrators (e.g., Kubernetes) commonly used for deploying IoT-based systems.

f. Security metrics related to the logical security of the information systems, i.e. cybersecurity, and specifically IoT-based systems. Example are: Declared Vulnerabilities or CVEs (Common Vulnerabilities and Exposures) that are the measures taken from the so-CVSSs (Common Vulnerabilities Scoring Systems) that indicate levels of vulnerability of the elements that make up a system (HW and SW). System Availability measurements that indicate whether the system, from the point of view of users and external applications, is responding to their requests—they can provide information for the detection of denial of service attacks. Finally, intrusion, such as abnormal behavior of devices, detection of signatures, DNS-based intrusion detection. Commonly these measures are taken from SIEMs (Security Information and Event Management) that collect data from system logs (e.g., antivirus, servers, network appliances, software components, etc.), analyze and correlate data from these logs, and detect and report security events.

Finally, SPEM model (Fig. 1) shows a set of tools that support a TaskDefinition. These tools are part of a monitoring infrastructure (ToolDefinition) that is flexible enough so that developers and operator can configure their own monitoring and alerting services according to their criteria (self-service capabilities). To that end we have defined a set of configurable scripts that automate the deployment of a monitoring infrastructure through virtualization and containerization technology, which we called Monitoring as Code, similarly to the well-known Infrastructure as Code. This code can be versioned and configured, and these configurations are repeatable. This automation allows teams self-service monitoring and alerting, which breaks silos between dev, ops, and sec teams by opening access to key metrics, enabling a sharing culture and continuous improvement. This automation is described in detail in the instantiation of the F&CF activity for cybersecurity monitoring of an IoT-based system.

B. Discussion

Some of the advantages of using SPEM 2.0 for this formal specification are as follows:

1. Systematic creation of processes based on the reusable method content. It separates reusable method content from its application in processes. Taking into account that the field of software engineering associated with the development,
deployment and operation of IoT systems is considered as “green”, it is fundamental to be able to isolate the description of good practices (method content) from the use of them in different modalities of software development (processes).

2. Flexible process variability and extensibility plug-in mechanism. This is a fundamental property given the huge variety of devices that appear in IoT systems, both in its nature and in its form of communication.

3. Reusable process patterns of best practices for rapid process assembly. Since the different applications will have different supervision objectives, it seems essential to have patterns that guide the diagnosis and treatment processes, although finally there is a TaskUse that adapts this pattern to the specific problem that concerns us.

4. Replaceable and reusable Process Components realizing the principles of encapsulation. It is very interesting that the actions associated with the taking of values (create telemetry) as well as the analysis of them (analyze telemetry) can be supported as the analysis of them (analyze telemetry) can be supported as components that can be exchanged (without changing the interface) depending on the application in question. This property together with the plug-in mechanism allows to automate the dynamic reconfiguration of the application because of the supervision process.

V. CASE STUDY: SELF-SERVICE CYBERSECURITY MONITORING — INSTANTIATION OF THE F&CF ACTIVITY

This section describes the instantiation of the F&CF activity for a self-service cybersecurity monitoring as an enabler to introduce security practices in a DevOps environment for IoT-based systems. Specifically, we focused on threats detection through the log traces files from an IoT system into production.

A. Case study description

Case study research is a technique that consists of the investigation of contemporary phenomena in their natural context [35][36][37]. The case study was conducted and deployed at the Universidad Politécnica de Madrid as part of a demonstrator for a smart campus. The research group SYST (System and Software Technology) is working on an IoT based demonstrator for a smart campus. The research group SYST (System and Software Technology) is working on an IoT based demonstrator for a smart campus. The research group SYST (System and Software Technology) is working on an IoT based System for controlling the Campus Sur Library to improve the energy-efficiency plans: The Library Energy-Efficiency System. This system consists of multiple connected devices—currently, sensor motes connected to Raspberries Pi 3—, sending data to an infrastructural soft that consists of a data aggregation entity, which collects data from all devices, analyze them, and stores them to a database. The data analysis is sent to a mobile application to support building operators on making better informed decisions.

This case study illustrates how rapid and simple its deployment was, in accordance with the DevOps principles, and therefore focusing on how self-service monitoring infrastructure for threats detection provided engineers—both developers and IT operators—fast and continuous feedback of the Library Energy-Efficiency System deployed into production. In addition, the case study tested the feasibility of DevOps practices during the construction of an IoT system.

The Library Energy-Efficiency System has been implemented using two different IoT Platforms: Microsoft Azure IoT Suite and FIWARE (see Fig. 2, left side). The first one is a cloud based IoT Platform, whereas the second one is an open source initiative that aims to promote the creation of necessary standards for the development of smart applications in different domains. Both generate log traces as a result of the interaction between the devices and the components that implement the data ingestion, i.e. Azure IoT Hub and Orion Context Brokers, respectively. Fig. 3 also shows the main functions and components of a monitoring infrastructure: Shipper for Logs centralizes logs; Process centralizes the data from logs and transforms these data; Search & Analytics creates indexes over data to optimize searches; Alert management detects threats and anomalies over the indexed information in the previous component based on a set of rules, generates alerts, and creates new indexes with these alerts; and finally, Visualization shows queries on index and alerts (see Fig. 3 right side).

B. The instantiation of F&CF activity

This case study illustrates the instantiation of the F&CF activity that was described in Section IV. Fig. 1 also shows the instantiation of the content method elements. These are called process elements and represent a specific way of performing a project using a specific technology. As mentioned before, the F&CF activity provides developers and IT operators with a monitoring infrastructure that we have particularized for cybersecurity events. The process elements that have been instantiated are the following:

1. The WorkProductUse ‘log trace’ are the logs from Orion Context Brokers. Code 1 shows an illustrative example of log traces that are printed by Orion Context Broker. These traces show a ‘device’ sending an update of context information, that is pressure=1023 and temperature=20ºC. The format of traces is described as follows: time | level | correlator id | transaction id | source IP | service associated to the transaction | subservice | component | op (function of source code that generated the log message) | message.

Code 1. Orion Context Broker Log traces

```json
    query: { _id.id: "device1", _id.servicePath: { Sin: [ null, */$/ ] } }
}
}
    update: { _id.id: "device1", _id.type: "Dispositivo", _id.servicePath: { Sin: [ null, */$/ ] } }, { Sset: attr: { pressure: { type: "int", creDate: 150663259, modDate: 150663259, value: "1023", mdNames: [ ] }, temperature: { type: "int", creDate: 150663259, modDate: 150663259, value: "20", mdNames: [ ] }, { attrNames: [ "pressure", "temperature"], modDate: 150663259, lastCorrelator: "ddf3df4-a491-11e7-b8c6-0242ac110004" })
}
}
```
2. The TaskUse ‘read and filter log trace’ is responsible for reading the ‘log trace’, extract relevant data, and generate the WorkProductUse ‘filtered log trace’. In this case study, this TaskUse transforms the ‘log trace’ described in Code 1 in new traces with the fields “source_ip” and “uuid” (see Code 2). This task is implemented by the components “Shipper for Logs” and “Process” of the monitoring infrastructure (Fig. 2).

Code 2. Filter log trace TaskUse

```plaintext
[...]
filter {
  if [type] == 'filebeat-docker-logs' {
    kv {
      source => 'message'
      value_split => '='
      field_split => '|'
      target => 'kv'
    }
    mutate {
      add_field => { 'source_ip' => '%{kv}[from]' } 
      update => { 'message' => '%{kv}[msg]' } 
    }
    grok {
      match => { 'message' => '%{DATA:preMsg}update: < \{ _id.id: "%{NOTSPACE:uuid}"%{GREEDYDATA:rest_of_message}' 
      add_tag => ["uuid"] 
    }
  }
[...]
```

3. The TaskUse ‘search for threat patterns’ analyses the ‘filtered log trace’ to search possible threats. If detected, this task generates an artifact for ‘threats detected’. This task is implemented by the component ‘Alert Management’ of the monitoring infrastructure. Code 3 shows the implementation of searches for spoofing patterns to provide alerts.

Code 3. Search for threat patterns TaskUse

```plaintext
# Rule name, must be unique
ame: UUID_cardinality_rule
# Type of alert. The frequency rule type alerts when num_events events occur
# with timeframe time
type: cardinality

# (Required, cardinality specific)
# Count the number of unique values for this field
cardinality_field: "source_ip"

# (Required, frequency specific)
# Alert when there more than 1 unique source_ip
max_cardinality: 1

# For the same uuid
query_key: uuid

# (Required, frequency specific)
# The cardinality is defined as the number of unique values for the most recent
# timeframe: hours: 24

 [...] 
```

The cybersecurity infrastructure is implemented using Elastic Stack and Splunk. Elastic Stack (ELK) is a distributed search and analytics engine. In this case, ELK was configured to analyse log file traces from Orion Context Brokers that collect data from devices connected to the Library Energy-Efficiency System (i.e. application logs). Splunk is a SIEM that collects security events from devices with web-server capabilities. The self-service cybersecurity monitoring infrastructure has been designed to detect, among others, path transversal attacks, suspicious IP’s, and spoofing (also known as identity theft) of devices connected to the IoT platforms. The installation and deployment of all the components stack is automated through containerization and virtualization technology (Docker10 and Vagrant11, respectively), as shown in Fig. 3.

10 https://www.docker.com/
11 https://www.vagrantup.com/ Vagrant is a tool for building and managing virtual machine environments in a single workflow
Fig. 3. Deployment view of the case study

Fig. 3 shows the deployment of two virtual machines. The virtual machine 1 runs a set of Docker containers: a haproxy container that provides high availability, load balancing, and proxying for TCP and HTTP-based applications; two Orion Context Broker containers—i.e., Orion 1 and Orion 2—for data ingestion from devices; and a set of containers for the Elastic Stack and Splunk. Specifically, these containers are FileBeat that centralizes logs and files (it is the implementation of Shipper for Logs in Fig. 3); Logstash that centralizes and transforms (filters) data (it is the implementation of Process in Fig. 3); Elasticsearch that searches and analyses data (it is the implementation of Search & Analytics in Fig. 3); Elastalert agent that pushes notifications based on anomalies, patterns, or changes in data over time (it is the implementation of Alert Management in Fig. 3); and finally Kibana that visualizes elasticsearch data (it is the implementation of Visualization in Fig. 3). Appendix B shows an illustrative pseudocode of the vagrantfile for configuring ‘virtual machine 1’ and running the abovementioned containers.

The virtual machine 2 runs a web server application as well as an application that simulates mass data sending to the Orion Context Broker components to stress the system (see Fig. 4). Finally, this application also allows us to simulate spoofing, i.e., an identity theft of a device and kibana allows visualizing this alert (see Fig. 5).

The SIEM Splunk detect cybersecurity attacks, such as the following cases: multiple connections from the same IP to the same port of the web server application until blocking it, with which the service is the victim of a denial of service attack; multiple failed login from the same IP, with which the service is being victim of a brute force attack; connections from sources or IP's not allowed, therefore the service is being attacked from a device that suffers a possible identity spoofing; improper access to directories or resources that are not allowed, which means that the service does not have or has adequate access control mechanisms and a transversal path or elevation of privileges is taking place, among many other cases.

Fig. 4. Kibana – normal work stress

In the case of the multiple failed login from the same IP, it is possible to generate an alert in the SIEM in such a way that when detecting a number of login attempts higher than a determined value, the traffic is blocked from that IP origin to prevent the attack. Fig. 6 shows the number of attacks received, the number of unique attacks received, the unique URL requests received, the HTTP queries executed, the number of attacks produced every hour, and finally a map with the source of the IPs from which the attacks have taken place. This feedback allows developers to take the appropriate actions on web development, such as an automatic login blocking mechanism that is activated if the number of login attempts allowed is exceeded.

Fig. 6. Splunk dashboard configuration
Appendix B shows some configuration files and scripts that development and operation teams can execute to create their own monitoring services. It is possible to instantiate the activity, as many times as teams need it. If the monitoring needs vary, it is possible to design another activity that extends the current activity (SPEM extension, localContribution or localReplacement).

VI. RESULTS

The contributions of this paper can be summarized as follows:

(i) Formalization of the DevOps activity ‘fast and continuous feedback from operation to development’ using SPEM. The separation of good practices (method content) from their use in different modalities of software development (processes) provides reusability and flexibility.

(ii) The instantiation, configuration and deployment of the F&CF tasks is automated through configuration files and scripts. We called to this process as Monitoring as Code (MaC) as a monitoring infrastructure can be automatically created by code. It is replicable, reconfigurable, and adaptable instead of manually configuration.

(iii) The result is a cybersecurity infrastructure so that teams can configure their monitoring and alerting services according to their security criteria—self-service, i.e. you build, you run, and now you monitor. In this way, teams share the responsibility of releasing secure software and have the necessary infrastructure and information to do it, which is key to enable DevSecOps culture.

VII. CONCLUSION

This paper presents the formalization of the activity for supporting the feedback from Operations to Development of IoT systems. This activity allows development and IT operation teams to convey feedback—telemetry—from systems in production. Security teams can use the F&CF activity to instantiate a self-service cybersecurity monitoring (as we illustrated in the case study) to enables the introduction of security practices in a DevOps environment. In this way, it is possible to generate insight for different teams with different needs—building collaboration and transparency. The monitoring we instantiated is based on the detection of threats and anomalies from the devices’ log traces (i.e. application log). The feedback that developers obtain from the detection of operation threats facilitate to make more informed decisions to fix problems or support software evolution.

The case study provides evidence of how this cybersecurity monitoring infrastructure enabled to detect threats, such as denial attacks, and helped to better anticipate spoofing problems. This infrastructure has been implemented following DevOps good practices: it is automated through scripts and configuration files (Monitoring as Code, versioned (GitHub repository) and its deployment has been automated through virtualization and containerization technology.

The validity of the approach has been performed with a case study. All case studies are qualitative in nature. That is, they are, in general terms, very difficult to be objectively judged [36]. However, they are often used in software engineering given the difficulty to produce multiple experiments, or experiments with large populations. In fact, the major limitation in case study research concerns external validity: “the generality of the results with respect to a specific population’’ [38] that is one case is studied. In our case, though only one case was studied, from the humble opinion of the authors, it was sufficient to validate the contributions claimed. In fact, this case study allowed the research team to evaluate the activity designed, and the process practices in a real setting. This is something important in software engineering in which a multitude of external factors may affect to the validation results [39].

In the future we will address more case studies in industrial contexts to strengthen and enlarge the conclusions of the contribution. We also plan to extend the instantiation of the F&CF activity to Supervision as Code, introduce some advanced configuration of the monitoring infrastructure using approaches based on software product lines, and use machine learning algorithms for advanced monitoring and supervision.

ACKNOWLEDGMENT

This work is supported by CROWD$AVING (TIN2016-79726-C2-1-R) and Horizon 2020 No. 732667 (RECAP). Thanks to Esperanza Rama for her support in the case study.

REFERENCES


12 https://github.com/jdiazfernandez/DevOps4IoTProject.git


Rafiuallah Khan SUK, Rifaqat Zaheer, Shahid Khan, Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges, 10th International Conference on Frontiers of Information Technology, 2012.


P. Östberg et al., Reliable capacity provisioning for distributed cloud/edge/fog computing applications, 2017 European Conference on Networks and Communications (EuCNC), Oulu, 2017, pp. 1-6. doi: 10.1109/EuCNC.2017.7980667


P. Östberg et al., Reliable capacity provisioning for distributed cloud/edge/fog computing applications, 2017 European Conference on Networks and Communications (EuCNC), Oulu, 2017, pp. 1-6. doi: 10.1109/EuCNC.2017.7980667


P. Östberg et al., Reliable capacity provisioning for distributed cloud/edge/fog computing applications, 2017 European Conference on Networks and Communications (EuCNC), Oulu, 2017, pp. 1-6. doi: 10.1109/EuCNC.2017.7980667


## APPENDIX A

<table>
<thead>
<tr>
<th>Icon</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Activity Icon]</td>
<td>Activity</td>
<td>It is a Work Breakdown Element and Work Definition that defines basic units of work within a Process as well as a Process itself.</td>
</tr>
<tr>
<td>![Composite Role Icon]</td>
<td>Composite Role</td>
<td>It is a grouping of Role Definitions that can be used in an Activity or Process to reduce the number of Roles defined in Method Content.</td>
</tr>
<tr>
<td>![Metric Icon]</td>
<td>Metric</td>
<td>It is a special Describable Element that contains one or more constraints that provide measurements for any Describable Element</td>
</tr>
<tr>
<td>![Step Icon]</td>
<td>Step</td>
<td>It is a Section and Work Definition that is used to organize a Task Definition’s Content Description into parts or subunits of work.</td>
</tr>
<tr>
<td>![Task Definition Icon]</td>
<td>Task Definition</td>
<td>It is a Method Content Element and a Work Definition that defines work being performed by Roles Definition instances.</td>
</tr>
<tr>
<td>![Task Use Icon]</td>
<td>Task Use</td>
<td>It is a Method Content Use and Work Breakdown Element that represents a proxy for a Task Definition in the context of one specific Activity.</td>
</tr>
<tr>
<td>![Tool Definition Icon]</td>
<td>Tool Definition</td>
<td>It is a special Method Content Element that can be used to specify a tool’s participation in a Task Definition.</td>
</tr>
<tr>
<td>![Work Product Definition Icon]</td>
<td>Work Product Definition</td>
<td>It is Method Content Element that is used, modified, and produced by Task Definitions.</td>
</tr>
<tr>
<td>![Work Product Use Icon]</td>
<td>Work Product Use</td>
<td>It is a special Breakdown Element that either represents an input and/or output type for an Activity or represents a general participant of the Activity.</td>
</tr>
</tbody>
</table>
APPENDIX B. VAGRANTFILE – CASE STUDY
Below is an illustrative pseudocode of the vagrantfile for configuring ‘virtual machine 1’ and running the containers shown in Fig. 4:

```vagrant
config.vm.box = "ubuntu/xenial64"
config.vm.box = "ubuntu/xenial64"
config.vm.provider :virtualbox do |vb|
config.vm.provision "docker" do [d]
  d.run "mongodb",
  d.run "orion1",
  d.run "orion2",
  d.run "haproxy",
  d.run "elasticsearch",
  d.run "logstash",
  d.run "logstash",
  d.run "filebeat",
  d.run "kibana",
  d.run "splunk",
end
```

Below is an illustrative pseudocode of the vagrantfile for configuring ‘gateways’ and running the Apache containers shown in Fig. 4:

```vagrant
config.vm.box = "ubuntu/xenial64"
config.vm.box = "ubuntu/xenial64"
config.vm.provision :docker do [d]
  d.run "httpd",
end
```

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.