A Reference Architecture for Management of Security Operations in Digital Service Chains

Alessandro Carrega[1], Giovanni Grieco[2], Domenico Striccoli[2], Manos Papoutsakis[3], Tomas Lima[4], José Ignacio Carretero[5], and Matteo Repetto[6](✉)

[1] S2N National Lab, CNIT, Genoa, Italy
alessandro.carrega@cnit.it
[2] Politecnico di Bari, Bari, Italy
giovanni.grieco@poliba.it
[3] Foundation for Research and Technology – Hellas, Heraklion, Greece
paputsak@ics.forth.gr
[4] Mind & Sparks GmbH, Vienna, Austria
tomas.lima@mindsandsparks.org
[5] Fiware Foundation, Berlin, Germany
joseignacio.carretero@fiware.org
[6] IMATI, CNR, Genoa, Italy
matteo.repetto@ge.imati.cnr.it

Abstract. Modern computing paradigms (i.e., cloud, edge, Internet of Things) and ubiquitous connectivity have brought the notion of pervasive computing to an unforeseeable level, which boosts service-oriented architectures and microservices patterns to create digital services with data-centric models. However, the resulting agility in service creation and management has not been followed by a similar evolution in cybersecurity patterns, which still largely rest on more conventional device- and infrastructure-centric models.

In this Chapter, we describe the implementation of the GUARD Platform, which represents the core element of a modern cybersecurity framework for building detection and analytics services for complex digital service chains. We briefly review the logical components and how they address scientific and technological challenges behind the limitations of existing cybersecurity tools. We also provide validation and performance analysis that show the feasibility and efficiency of our implementation.

Keywords: Cybersecurity architecture · Digital service chain · Detection · Analytics pipeline

1 Introduction

Around thirty years after its original definition, pervasive computing has become a reality and probably has gone far beyond what could have been expected at that

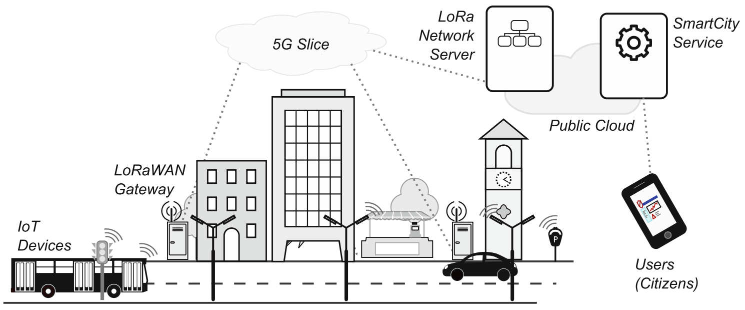


Fig. 1. An example of service chain for Smart City services.

time [29]. As a matter of fact, the integration of (invisible) computing capabilities into nearly all possible objects has only represented a part, yet relevant, of the implementation of this concept, while virtualization and ubiquitous connectivity have fully triggered the transition from device-centric to data-centric models. It therefore comes as no surprise that the more remunerative business in the digital economy is the creation of value added services for data retrieval, transformation, and sharing, no matter where and how processing is actually performed. This has been largely supported by a progressive (and still on-going) evolution from monolithic applications to service-oriented architectures, web services, and, more recently, service meshes [24].

Modern digital applications and services are increasingly designed and implemented according to microservice patterns, by composing (also denoted as *chaining*) digital resources (data, networks, cloud services, applications, and things) from multiple vendors on a growing scale; this allows to create, process, share, and consume data and content in a digital continuum, blurring the frontiers between application domains and breaking the current closed silos of information. Examples of digital resources include industrial and financial data sets, cloud infrastructures, lambda functions, storage services, smart devices in the Internet of Things, 5G network slices, LoRaWAN networks, and so on. An illustrative example is shown in Fig. 1. It depicts a possible chains of digital resources to create Smart City services. It includes Internet of Things (IoT) devices around the city (public transportation, traffic lights, public lightning, parking meters, etc.) connected to LoRaWAN gateways. The backbone is a leased 5G network slice, which interconnects the gateways to the LoRaWAN server deployed in the public cloud. In the same or a different cloud infrastructure, one or more applications implement the desired services (e.g., fleet and lightning management, free parking notification, traffic routing), maybe by interfacing with external datasets and users.

The loosely-coupled nature of micro-services allows to replace, duplicate, or remove part of them without affecting the operation of the overall application. As main result, digital services can be quickly provisioned in a matter of minutes or hours instead of days or weeks, can grow or shrink dynamically according to the evolving workload, can be easily deployed, replicated and migrated in multiple locations and even over heterogeneous infrastructures. This perfectly fits the dynamic and agile nature of modern business models, where digital services and business chains are expected to emerge, evolve, and dissolve much faster than traditional value-creating networks.

Unfortunately, cybersecurity mechanisms have largely been unable to keep the pace of this evolution towards data-oriented models. Indeed, they are still largely conceived to protect individual devices and specific infrastructures, either physical or virtual, but are largely ineffective to cope with complex dynamics, dispersion of data among the multitude of digital objects and infrastructures, non-deterministic, opaque and partially inscrutable service topologies [24]. Most issues come from privacy and integrity concerns. As a matter of fact, although the location and confidentiality of private and sensitive data are rather straightforward to manage inside devices and infrastructures of a single provider, this is more difficult when resources from multiple providers are involved (e.g., cloud, storage, things, networks) [2]. Similarly, detection of cyberattacks in these conditions is extremely challenging, due to the lack of proper visibility over third party's infrastructures and services.

The GUARD framework was conceived as an evolutionary step in the design and operation of cybersecurity frameworks, towards data-centric models already implemented by modern computing paradigms [23]. It advocates the augmentation of service management interfaces with security capabilities, according to on-going efforts in IETF [13,19] and OASIS [1]. Further, it provides a way to leverage such extended interfaces for programmatically collecting data and feeding detection algorithms and security analytics, while providing the necessary agility to chase the dynamics of complex digital service chains. In this Chapter, we follow up on our preliminary architectural design [24] with a more concrete implementation of its core platform. We briefly describe the different components and the main benefits of the GUARD approach; we also provide validation and performance evaluation for the most critical parts of our platform.

The rest of the Chapter is organized as follows. We initially discuss related work in Sect. 2. We briefly review the overall GUARD framework in Sect. 3, while in Sect. 4 we describe the last version of the core platform architecture and its main components, namely the Context Broker Manager, the Security Controller, and the Dashboard. Then, we describe how the platform interacts with digital services and its internal workflows in Sect. 5. We report numerical results from our validation and performance analysis in Sect. 6. Finally, we give our conclusion and plan for future work in Sect. 7.

2 Related Work

The interconnection of digital resources from multiple heterogeneous domains (cloud, data sets, networks, IoT, etc.) creates complex systems of systems, which introduce more functions, management aspects, and security issues than the plain sum of the constituent components [25,26]. They represent a fertile ground for new forms of attacks, often leveraging advanced persistent threats (APT). Research efforts have largely focused on building complex analytics for spotting anomalies in large bulks of data, often leveraging big data techniques. Specific topics in this context include the heterogeneity of log data and events (e.g., structured, semi-structured, and unstructured samples) [11,31], scalability and multidimentional anomaly detection [17,21], the automatic generation of configurations from high-level policies [7,30], data handling architectures for Security Operation Centers [3,9], data collection strategies [18]. Besides, the concept of Security Monitoring-as-a-Service (SMaaS) was introduced, but it still remains bound to rigid architectures [10]. Notably, while many frameworks allow to create multiple analytics processes (e.g., Elastic Stack, Apache Metron), to the best of our knowledge there is no way to automate this configuration on dynamic systems-of-systems as digital service chains.

A meaningful step in this direction is the creation of standard interfaces to security functions. The I2NSF framework [13,19] defines YANG [5] data models for describing security capabilities and changing the configuration of network security functions with the RESTCONF/NETCONF protocols [4,6]; the framework also defines monitoring and reaction interfaces. Unfortunately, there is only a detailed data model for stateless packet filters, whereas other kinds of supported appliances (antivirus, IDS, web and Voice-over-IP firewall) have a much barer interface. The OpenC2 initiative [1] takes a different approach, focusing more on a standard set of commands rather than data models. Indeed, currently there is only an operation profile for the stateless packet filter [22], but the usage of this interface for other cybersecurity appliances requires far less extensions than I2NSF.

Identity management and access control always represent a big challenge in decentralized environments. Most of the emerging solutions exploit a decoupled mechanism, which aims at separating authentication and authorization functionalities in a harmonized fashion [16]. Many interesting solutions have been recently formulated in the scientific literature for identity management in multi-domain environments, like OpenID Connect and OAuth 2.0 [20,27,28]. They introduce the possibility to authenticate users within a federated ecosystem by means of a trusted Identity Provider. This is necessary to delegate security management to external providers, without sharing full access to the management interface.

3 The GUARD Framework

Recent advances in cybersecurity have largely focused on improved detection and correlation capability, in order to recognize advanced persistent threats and complex multi-vector attacks. In this respect, machine learning and other forms of artificial intelligence are often used for spotting anomalies in large bulks of data, which may be generated by zero-day attacks. Despite of the large effort in improving the detection and automating response and mitigation actions, the installation and configuration of protection systems is still largely a manual process, which takes time and needs continuous tuning every time the system changes (e.g., devices are added/removed/replaced, software is installed/updated/purged, new configurations are applied). Even if this approach has always been accepted for long-lived installations that are seldom modified or upgraded, it does not fit the agile nature of digital services that are quickly provisioned and changed.

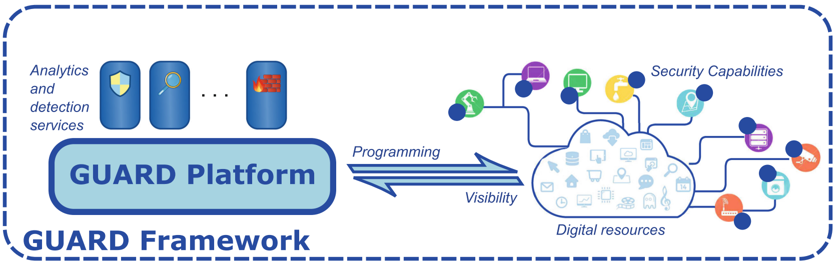


Fig. 2. Overview of the GUARD framework.

The GUARD framework is conceived as a new paradigm to implement detection and analytics processes for digital service chains. Its objective is to facilitate the creation and adaptation of such processes in fast-changing environments, by automating as much as possible the setup of security analytics pipelines. We define as *security analytics pipeline* the processing chain from security agents to detection algorithms, which includes the necessary tasks for data collection, transformation, filtering, indexing, analysis, correlation, and storage. This is implemented by the GUARD Platform, which acts as a sort of middleware between security agents and detection/analytics services, as depicted in Fig. 2.

The most innovative aspect of the GUARD framework is the multi-domain scope. Indeed, even if a digital service chain is created and managed by a single entity (denoted as *Service Provider*), it is usually composed by resources owned and operated by different *Resource Providers* [24]. Additionally, security aspects are often delegated to external parties, i.e., *Security Operators*, which have the skills, expertise, and technological assets to face a large number of diverse threats and attacks. The GUARD Platform is the tool that allows these operators to easily connect their analytics engines with their customers' services scattered across the Internet.

Monitoring a digital service chain is therefore extremely challenging, because security agents cannot freely be installed by Security Operators. A shift in mindset is necessary with respect to legacy paradigms, because security processes must build on embedded security capabilities rather than more conventional discrete security agents (even if a concrete agent will run anyway in most cases behind the abstraction provided by a security capability). Indeed, this transformation has already started, because many cloud providers already integrate security services in their offers, like firewalls, intrusion detection and prevention systems, antivirus software; these appliances work at the infrastructure layer, hence this approach ensures better visibility and safety than running them inside virtual resources (which is anyway unfeasible in case of containers and serverless computing).

An additional issue is represented by the delegation of security processes to external entities. Indeed, these operators should be granted access to the management interface of the resources involved in the chain, but this approach might not be recommended for confidentiality, safety, and privacy reasons.

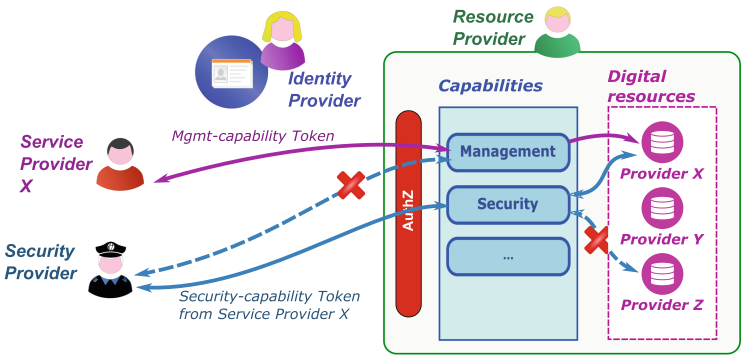


Fig. 3. Access control model for the GUARD Framework.

Fine-grained access control should instead be implemented in each digital resource to restrict their access to security-related capabilities only, preventing any other management action that could affect the continuity of the service. This requires scalable authentication and authorization schemes that give Security Operators limited access to digital resources (i.e., security capabilities) on behalf of the resource owner (see Fig. 3). Token-based mechanisms like OAuth2 [12] provide an effective solution to this problem, by avoiding the need to share secrets and to grant full control over the resources.

Although standard firewalling, antivirus, and intrusion prevention systems implemented by Resource Providers may be enough for some users, they are not probably enough for detecting and mitigating complex attacks. In this respect, we argue that security capabilities should include a "programmability" dimension, as shown in Fig. 2, that allows more flexibility in retrieving relevant security context (events, measures, data). Programmability may consist, for instance, in the possibility to select and chain multiple Logstash modules for data transformation; more interestingly, it may also entail the execution of custom code in a safe way (which means no vulnerability is introduced for the service and the whole infrastructure) [8].

4 GUARD Platform

The GUARD Platform is designed to implement a typical closed control loop for security enforcement, as shown in Fig. 4. In the upper part of the loop, a data handling pipeline is implemented, from data collection to detection, up to visualization; in the bottom part, control response strategies are applied for enforcement and mitigation actions. Detection and analytics can be performed both on real-time data, according to a streaming pattern, or on historical data, by offline analysis. As we already pointed out in the previous Section, the scope of the GUARD Platform does not include monitoring and inspection tasks, which are performed by local security agents embedded in digital services.

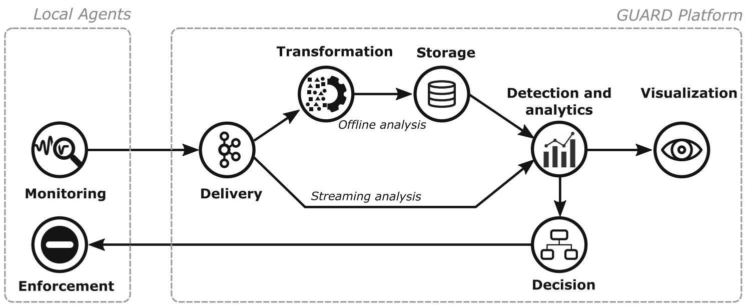


Fig. 4. The typical structure of a data handling pipeline for security analytics.

Figure 5 shows the software architecture of the GUARD Platform. The dark components are taken from typical Elastic Stack setup (though not all components are part of it), and represent a common solution for building a data handling pipeline that delivers data to one or more processing algorithms. The light boxes are GUARD components that implement the necessary feature to make this system flexible and adaptable to highly-dynamic computing environments. The Context Broker Manager provides an abstraction of security capabilities provided by Resource Providers. It uses data models to abstract the concrete implementation of security agents and the communication protocols; therefore, it offers to the rest of the system technology-agnostic access to monitoring and enforcement capability in local environments. The Security Controller is the smart component that takes decision based on control policies. Its main purpose is to trigger response and mitigation actions based on the evolving context (events, conditions) generated by the detection algorithms or directly by remote agents. It is expected to provide the necessary logic to resolve conflicts and to generate additional outputs by combining the set of provided rules. Additionally, the Security Controller is responsible for the setup of multiple security analytics pipelines within the Platform. Finally, the Dashboard is the user interface to visualize the service topology, security features, and data generated by agents and detection algorithms. The Dashboard also features an editor for creating analytics pipelines, starting from the list of available detection and analytics algorithms, as well as security capabilities of each digital resource.

One major contribution from GUARD is the definition of an API for exposing security capabilities. This goes in the direction of providing a uniform and common interface towards security agents, beyond the current set of heterogeneous services offered by different providers. Since the definition of such kind of interface is still an open issue and no implementations are available for on-going standardization initiatives [1,13], we built on existing management interfaces in the FIWARE domain and extended them to include security capabilities, as discussed in Sect. 5.1.

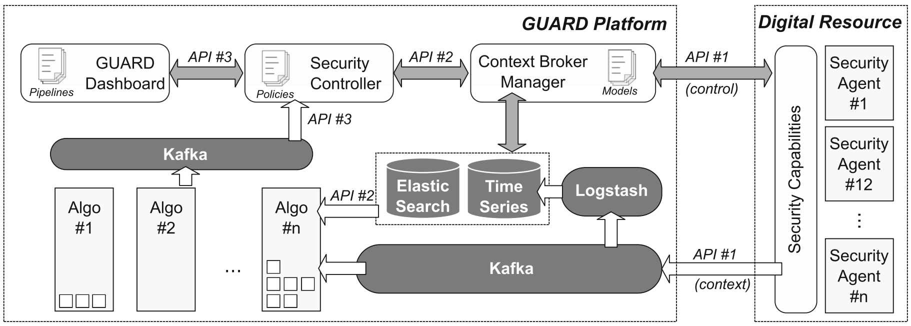


Fig. 5. Software architecture of the GUARD Platform.

4.1 Security Analytics Pipelines

One common issue in setting up detection services is the time needed to configure all the necessary elements. Although graphical user interfaces are often available for this purpose, manual operation for deployment and initial configuration are still required. We leverage the components of our architecture, as well as the interface to security capabilities, to bring more automation in this process, in order to better follow the dynamics of digital services. We introduce therefore the notion of *security analytics pipeline* (SAP), which includes the logical stages shown in Fig. 6:

- monitoring and inspection functions that are available in digital resources (Local Agent);
- aggregation, transformation, and enrichment tasks that are necessary to adapt the format of data generated by sources (agents) to what expected by consumers (algorithms);
- indexing and normalization operations before storing data;
- analytics and detection algorithms that are used to provide notifications and alerts;
- storage of data end events;
- analysis of events and conditions to undertake response and mitigation actions (Security Controller).

Figure 6 also shows typical configuration aspects for a SAP. They partially require input from Security Operators and partially can be automatically derived by the Security Controller. The definition of a SAP includes at least the following steps:

- The selection of one or more logical agents exposed by digital resources and their configuration. This provides raw data and measurements for further processing, or refined events if at least part of the detection is already implemented locally.

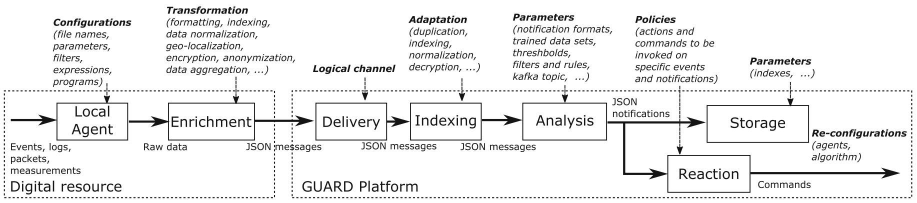

Configurations (file names, parameters, filters, expressions, programs)
Transformation (formatting, indexing, data normalization, geo-localization, encryption, anonymization, data aggregation, ...)
Logical channel
Adaptation (duplication, indexing, normalization, decryption, ...)
Parameters (notification formats, trained data sets, threshbolds, filters and rules, kafka topic, ...)
Policies (actions and commands to be invoked on specific events and notifications)
Parameters (indexes, ...)
Local Agent
Enrichment
Delivery
Indexing
Analysis
JSON notifications
Storage
Re-configurations (agents, algorithm)
Reaction
Commands
Events, logs, packets, measurements
Raw data
JSON messages
JSON messages
JSON messages
Digital resource
GUARD Platform

Fig. 6. The logical stages of a Security Analytics Pipeline.

- The selection of one detection/analytics service. This is an optional step, that correlates data and measurements for detecting known attack patterns or anomalies. In any case, all data will be dumped to a database.
- The definition of control policies for the Security Controller. These policies are kind of rules that define actions to be performed in response to given events, and after considering a set of conditions (e.g., current status).

Other stages of the pipeline (e.g., for data adaptation, transformation and indexing, delivery topics) should be automatically configured by the SC. However, this logic is only partially implemented and the intervention of the SC is currently mostly limited to the configuration of a dedicated Kafka topic for each SAP.

4.2 Context-Broker Manager

The Context-Broker Manager (CB-Manager) provides uniform access to heterogeneous security capabilities of different digital resources. There are two main aspects behind this abstraction:

- hiding the technological details and the heterogeneity of security capabilities in different domains;
- depicting a logical view of the whole service chain, including capabilities and the relationships between the different resources.

Each digital resource that exposes a management interface is considered an independent Execution Environment (ExecEnv). The main security aspects of each ExecEnv concern what kind of monitoring, inspection and enforcement processes are available to its user (collectively indicated as "security capabilities"), and what kind of relationship exists with other resources (e.g., a cloud service that keeps data in an external storage server; an application that uses an external data store). This information is retrieved from the interface towards digital services (we describe our implementation in Sect. 5.1, and then organized within the CB-Manager is a tree-like fashion, which root is represented by the service entry point (usually the service directly used by the user).

Service topology and security capabilities are abstracted in a data model that captures both the relevant parameters and the current configuration. The model is exposed through a REST interface, where POST/GET verbs on the endpoint

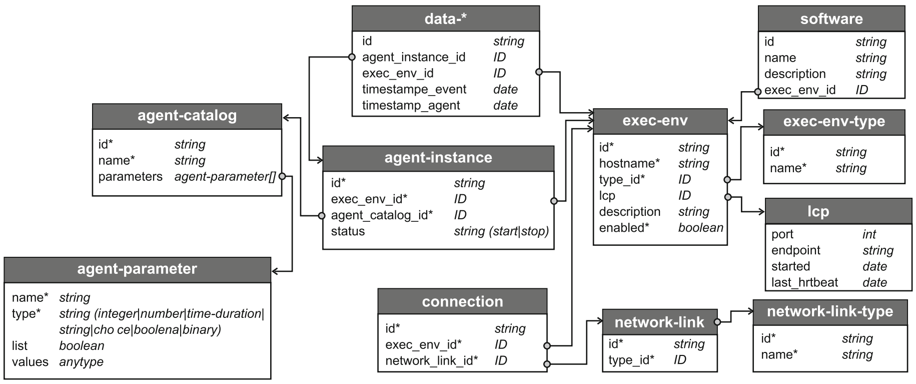


Fig. 7. Data model of the CB-Manager abstraction.

are translated by the CB-Manager in the corresponding operations on concrete elements.

Data Model. Figure 7 shows the Data Model diagram implemented by the CB-Manager abstraction. It captures both the service topology, security capabilities of each ExecEnv, and current configuration of local security agents. The service topology is represented in terms of Nodes and Edges. Nodes correspond to the ExecEnvs of each digital resource, while edges represent relationships (communication links, management or data interfaces, etc.).

The *exec-env* object contains the name of the digital resource, whereas the type_id denotes the type of ExecEnv. Examples of ExecEnv types include bare metal servers, virtual machines, containers, Platform-as-a-Service, IoT device. The *software* object contains the installed software in each ExecEnv, also including relevant fields (vendor, version, security patches, installation e.g., plain, chrooted, containerized, virtual machine, configuration options, etc.). This could be useful for risk assessment, to spot known vulnerabilities. Additionally, an endpoint for the security capability API is present, which is indicated as *LCP*. This name is taken from the component that implements this interface in our framework (see Sect. 5.1).

The definition of *network-link* includes an identifier and a type. All the possible network link types are defined in the *network-link-type* object. This can correspond to a physical link (e.g., Point to Point (Point 2 Point, LAN), a virtual link (IPSec, VxLAN, GRE, etc.), a network slice (as for a 5G network), a communication interface (protocol). Network links are connected to their ExecEnv by the *connection*; this approach is necessary to deal with multipoint communications, as in case of LANs. This object contains all the information regarding the configuration of the network link in the ExecEnv as, for example, the IP address (version 4 and/or 6), if the link is encrypted and how (which method, etc.).

The *data* object contains all the data exposed by the security capability of each ExecEnvs, in terms of security agent models. The different types of security agents and the corresponding configurable properties are registered in the *agent-catalog*. The model includes the list of parameters that can be configured, as *agent-parameter*. This nested object describes the parameter in terms of name and relative type. The *agent-catalog* does not need to hold a separate model for each different implementation; instead, the main purpose is to hide as many implementations as possible behind the same abstraction. For example, a "file log" monitoring agent is a quite general component, which parameters include the name of the files to monitor, the frequency of updates, the retrieval paradigm (differences or whole file); it may be implemented as Elastic FileBeat or by the native Kubernetes log retrieval command, but this is totally hidden by the CB-Manager abstraction. Clearly, the more specific the agent, the less parameters will be available for configuration.

The agents available in each ExecEnv are listed as *agent-instance* objects. Their properties can be read and changed, and this operation is reflected on their concrete implementation; in addition, they can be started and stopped. While the *agent-catalog* index is provided as part of the CB-Manager implementation according to the list of supported interfaces/agents, the *agent-instance* catalog is automatically populated by the discovery mechanism described in Sect. 5.1.

Control/Management API. Access to the Data Model is possible through a RESTful interface. This interface can be used both for management (i.e., including new agent models in the *agent-catalog*) and control (i.e., reading/setting parameters in the *agent-instances*.

The semantics of the CB-Manager API reflects the structure of the Data model. It is a REST server built with swagger that enables the standard CRUD operations. The HTTP syntax for requests includes therefore the following elements:

– HTTP method, which identifies the requested operation; supported values are: GET, POST, PUT, DELETE.
– Path, which identified the logical element to act on.
– Body, which contains a JSON object for passing arguments and options.

Responses are given with standard HTTP messages and status codes; they contain requested information in the body formatted as a JSON object.

The RESTful API is organized in three main path roots:

– *config*, which gives access to information about current instances and their configuration (Topology data);
– *catalog*, which contains the types of security agents that can be used in each ExecEnv (Catalog data);
– *data*, which gives access to the set of information collected by security agents (measures, events, etc. for historical data).

The first 2 elements support every HTTP method (subject to authorization and access control), since they refer to the overall service topology and its configuration; the last element only supports the GET method, as the injection of data with other methods than Kafka is not supported.

The *config* path is used to store the service topology and available agents. It is also used both to set and to retrieve the current configuration of available agents. When invoked with a PUT or POST method, the CB-Manager stores the required value and invokes the necessary interface to apply the new configuration to the remote agent. When invoked with the GET method, the CB-Manager only supplies the stored value, but does not query the remote agent, to improve performance. The CB-Manager does not solve configuration conflicts; this must be accomplished by the Security Controller, which implementation is more suitable for this function.

The *catalog* is a sort of driver repository for known agent types, since it defines the necessary commands to translate the capabilities into concrete configurations. Objects in the catalog are inserted by the system administrator, and then they are retrieved by the CB-Manager every time a configuration change is requested. The definition of the commands for each agent depends on the security capabilities exposed by digital service. The current implementation can i) load configuration files; ii) execute shell commands; and iii) forward HTTP requests to local agents.

The *data* path is used to retrieve data stored internally to the CB-Manager. It is conceived for offline analysis, since real-time data are collected more efficiently through Kafka. It is worth pointing out that the content of the data path is deliberately unspecified at this stage, because security operators are responsible to select the agents they need and the content and format of produced data, as well as any conversion and transformation that may be required to adapt such format to the input of relevant processing algorithms.

There are two ways to access elements from this REST API. If a single element is required and its identifier is known, it can be specified at the end of the path. This is the only allowed semantics to update elements (i.e., when using the PUT method). Otherwise, a query can be specified in the body of the request. Queries are performed by a JSON request that follows the SQL syntax[1].

4.3 Security Controller

The Security Controller (SC) automates management and control actions for SAPs. This is achieved by implementing behavioral guidelines that define the actions to be undertaken in response to specific events. In practice, this means that the SC configures and launches detection algorithms when it is necessary, listen for relevant data and events, and dispatches notifications to users.

Regarding its implementation, the SC is based on a rule engine that is able to express the logic behind the decision making due to the security policies as

[1] Elasticsearch guide – SQL language. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/sql-spec.html.

production rules. Drools [18] is the chosen rule engine applying and extending the Phreak algorithm. Drools is a Business Rules Management System (BRMS) solution. It provides a core Business Rules Engine (BRE) among others. It is open-source software, released under the Apache License 2.0. It is written in pure Java^{TM}, runs on any JVM and is available in the Maven Central repository too. Finally, SC integrates different sub-components required by the rule engine such as the knowledge base, the core engine, and the compiler.

Drools Engine in the SC is running continuously. Whenever a request reaches one of the SC's endpoints, an instance of the corresponding Java class is created and then inserted into the Drools working memory as a Fact. Facts are triggering Drools Rules that perform actions. For example, when a PUT request, with a JSON representation of a SAP in its body, arrives at the /startSecurityPipeline endpoint of SC (see Sect. 5.3), the following actions take place:

- An instance of the Pipeline Java class is created and some attributes that are meant for internal use are filled in, such as the uniqueID attribute of the said instance.
- The created instance is checked against a pipeline array list located locally in the file system. If there is no other instance with the same information, the newly created one is added in the array list. Otherwise, the pre-existed one is updated.
- The newly created/updated pipeline instance is then added into the Drools working memory.
- Based on the information carried by the pipeline instance, a number of instances of the AgentInstance Java class are created along with their configuration and are also inserted into the Drools working memory.
- A subset of the Drools rules is triggered due to the insertion of the new Drools Facts, invoking Java methods that send a request for each AgentInstance Fact to the appropriate Context Broker Manager endpoint.
- The endpoints' responses are then inserted as Drools Facts into the working memory triggering, this time, a different subset of Rules.
- The Java instances in the pipeline array list and the Smart Controller's log file are updated based on the Context Broker's responses.
- Drools Facts that are not necessary are retracted from the working memory.

The reason for storing Facts (pipeline array list) outside of the Drools Engine is two-fold: debugging purposes and pipeline status reporting.

Drools Rule Example. We have already mentioned that Drools rules perform actions. What follows is an example of such rule, which updates the status of incoming SAP based on the CB-Manager responses to the update-agent requests. Listing 1 depicts the *when* and *then* parts of the rule. The conditional elements in this case are all the agent instances of a pipeline and their corresponding responses received from the CB-Manager's update agent instance endpoint. The response of every one agent instance of the pipeline should report code 200 for the rule to be triggered. This is checked using the *forall* statement. One again we make sure that we check only the agent instances of a specific pipeline using the *from $agents* part of the *forall* statement.

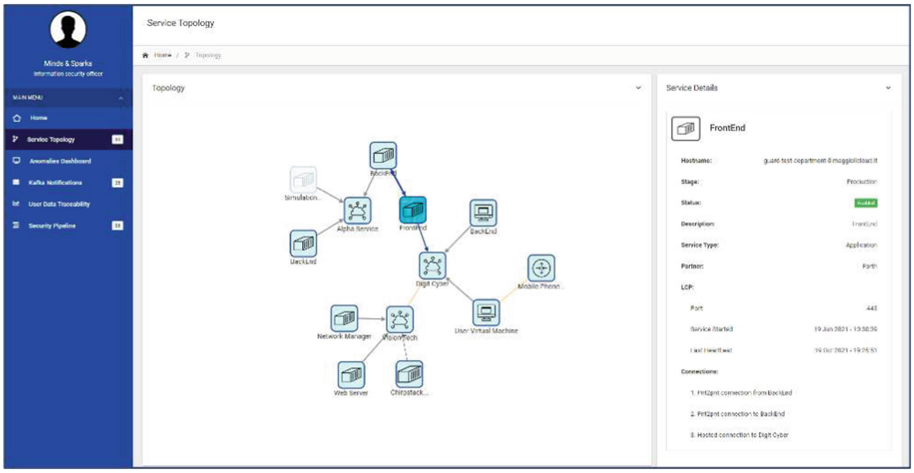
```
rule "StartPipeline"
when
  $pipeline: Pipeline(status=="start", $agents:=agents)
  forall( $agent:Agent($agentId:=agent_catalog_id) from
    $agents
     ContextBrokerUpdateAgentResponse($agentId:=subiectUuid,
      code==200)
  )
then
  KafkaProducerController.logger.info ("All agent instances
     of pipeline " + $pipeline.getId() + " - " + $pipeline.
    getUuid() + " have been successfully updated.");
  $pipeline.storePipelineToContextBroker();
  retract($pipeline);
  KafkaProducerController.logger.info("Pipeline " +
    $pipeline.getId() + " - "  + $pipeline.getUuid() + " is
     retracted");
  springBootkafkaAppApplication.pipelineArray.
    updatePipelineStatus($pipeline.getId(), "started");
end
```

Listing 1. StartPipeline Drools rule - The SAP status is updated and the pipeline is retracted from the working memory.

If the conditional elements are present in the working memory, the actions in the *when* part of the rule take place. The first action of this rule is to report in the log file that all agent instances of the specific pipeline have received a response with code 200. After that, a method of the pipeline Java class is called to store the pipeline with the new status in the Context Broker. The pipeline instance is then retracted from the working memory of the Drools Engine with the *retract* statement. This action is also reported in the log file. Finally, another Java method is called, part of the PipelineArray Java class that changes the status of the pipeline to “started”. This attribute is retrieved when the SC is asked for the status of a pipeline.

4.4 Dashboard

The Dashboard is the Graphical User Interface (GUI) that simplifies the interaction between the end users (e.g., Security Operators) and the GUARD framework, by giving the user an easy way to apply configurations and interpret data and results. The Dashboard includes general features such as a secure authentication system with roles and permissions, profiles and account settings, as commonly required for this kind of tools. More interestingly, the Dashboard provides a schematic representation of the service topology, which helps the user to discover what difference resources are involved in the service, their relationships, and their security capabilities (see Fig. 8).



FrontEnd

Hostname:	guard-test-department-0.maggioli.cloud.it
Stage:	Production
Status:	[illegible]
Description:	FrontEnd
Service Type:	Application
Partner:	Forth
LCP:	
Port	443
Service Started	19 Jun 2021 - 10:30:39
Last Heartbeat	19 Oct 2021 - 19:25:51

Connections:

1. Pnt2pnt connection from BackEnd
2. Pnt2pnt connection to BackEnd
3. Hosted connection to Digit Cyber

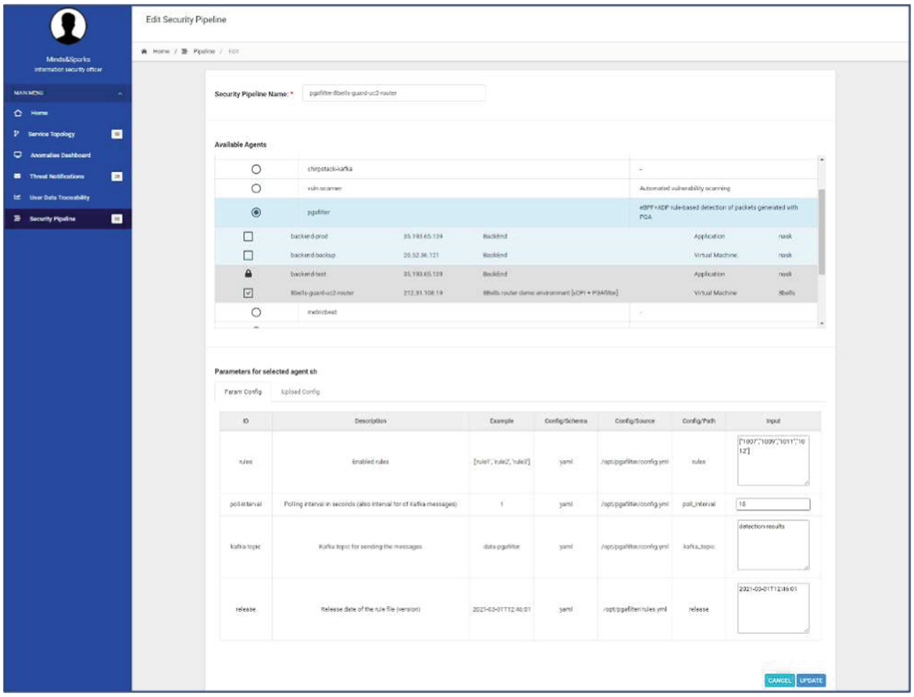
Fig. 8. Service topology visualization in the Dashboard.

The graphical visualization of service topology effectively supports the preliminary security analysis and risk assessment processes, which culminate in the design and instantiation of SAPs. These represent the main goal for the GUARD framework, therefore the Dashboard provides all functions to view, create, update, manage and remove them. The creation of a new pipeline requires to choose a name, select one or more agent instances among those reported by the CB-Manager, and (optionally) an analysis/detection algorithm that will be fed with monitoring data. The process includes the configuration of each agent (also including the possibility to upload configuration files) and the detection algorithm, as shown in Fig. 9; moreover, a set of rules can be inserted in the policy editor that automate response actions for events generated by the pipeline. Management operations include start, stop, and reload of the pipeline, which are concretely translated into management actions by the Security Controller (see the activation workflow in Sect. 5.3).

The Dashboard also provides situational awareness, by visualizing in the notification and anomaly view all relevant messages. They can be representative of relevant monitoring events from security agents or threat alerts from detection algorithms. Each event reports source, severity, description, data and date/time. Filtering allows to restrict the scope and not being overwhelmed by verbose agents. This notification feature allows to supervise the operation of the Security Controller and to undertake corrective actions that are not implemented by automation policies.

5 GUARD Operation

The functional components described in Sect. 4 interacts with additional elements to implement the functionality described for the whole framework. In



Edit Security Pipeline

Security Pipeline Name: * pgafilter-8bells-guard-uc2-router

Available Agents

○	chirpstack-kafka		-		
○	vuln-scanner		Automated vulnerability scanning		
◉	pgafilter		eBPF+XDP rule-based detection of packets generated with PGA		
☐	backend-prod	35.193.65.139	BackEnd	Application	nask
☐	backend-backup	20.52.86.121	BackEnd	Virtual Machine	nask
🔒	backend-test	35.193.65.139	BackEnd	Application	nask
☑	8bells-guard-uc2-router	212.31.108.19	8Bells router demo environment [vCPI + PGAfilter]	Virtual Machine	8bells
○	metricbeat		-		

Parameters for selected agent sh

Param Config | Upload Config

ID	Description	Example	Config/Schema	Config/Source	Config/Path	Input
rules	Enabled rules	['rule1', 'rule2', 'rule3']	yaml	/opt/pgafilter/config.yml	rules	["1007","1009","1011","1012"]
poll-interval	Polling interval in seconds (also interval for of Kafka messages)	1	yaml	/opt/pgafilter/config.yml	poll_interval	10
kafka-topic	Kafka topic for sending the messages	data-pgafilter	yaml	/opt/pgafilter/config.yml	kafka_topic	detection-results
release	Release date of the rule file (version)	2021-03-31T12:40:01	yaml	/opt/pgafilter/rules.yml	release	2021-03-01T12:46:01

Fig. 9. Configuration of a pipeline.

the following, we describe the interaction with digital services and the identity management and access control subsytem.

5.1 Discovery and Management of Security Capabilities

One of the most challenging part of our framework is dynamic management of local monitoring and enforcement capabilities available in digital resources. To this aim, we implemented the Local Control Plane (LCP), which exposes a REST API that describes the main characteristics of an ExecEnv and its embedded security capabilities. This includes the relevant security context that is already part of the CB-Manager's data model, namely the type of ExecEnv, the installed software, links to external resources, etc. The LCP also represents the local control hook to configure and run security agents.

Table 1 shows the REST API exposed by the LCP, which is mainly consumed by the CB-Manager. The most relevant part is the information about child LCPs, namely LCPs of other digital resources that are chained. This is the case, for instance of a virtual machine that uses storage provided by a different provider, or an IoT agent that pushes data through a dedicated 5G network slice. By starting at the service entry point and recursively querying each LCP and its children, the CB-Manager can therefore discover the whole service topology. The concept

Table 1. The LCP API to expose security capabilities for a digital resource.

Path	Action
/self	Returns a simple identification of the resource, including the URL of the LCP, its identifier and name, the type of ExecEnv, and a human-readable description
/self/deployment	Returns the description of the ExecEnv, including the type of hardware (baremetal server, virtual machine, LXC or Docker Container), operating system, number of (virtual) CPUs, amount of RAM, disk devices and partition size, network interfaces and IP addresses
/lcp_parent	Returns the parent LCP(s) of this resource
/lcp_son/id	Returns the list of LCPs of subsidiary resources used by this ExecEnv. If an *id* is specified, only the corresponding LCP information is returned
/self/software	Returns the software run in this ExecEnv. The description includes the software name, vendor, connections to other software, and open network ports
/self/container	Returns the container name and the description of the hosted software (same information as previous object)
/agent/instance	Returns the list of security agents available in the ExecEnv. The description includes name, version, vendor, status (running, stopped, etc.), endpoint URL, identifier and description
/interactions	Returns the list of external resources used by this ExecEnv but without an LCP. This includes any kind of digital resource (cloud installations, applications, storage servers, network slices). It may also include a specific endpoint to retrieve additional details
/poll	Returns all the objects listed before, but formatted in a way that better fits the CB-Manager's data model

is schematically shown in Fig. 10, with reference to the example introduced in Sect. 1. Here, we see that VM1 hosts the "Smart City App", which is taken as the "entry point" of the service because it is the final service offered to citizens. VM2 hosts the "LORA Server", which collects data from IoT devices through a dedicated 5G slice. VM1 and VM2 run on resources provisioned by Cloud1 and Cloud2, respectively, which are not necessarily operated by the same provider not use the same technology. Additionally, the SmartCity App stores data on an external storage service. The different relationships between digital resources (e.g., application *hosted* in VM, VM *provisioned* on cloud, data *pushed* to server, etc.) are reflected in the parent-child relation between LCP instances, which can be discovered by recursive queries (not only at the root). This configuration must be provide by each Resource Provider. We argue that some providers might not expose security capabilities; this creates "blind spots" in the service which cannot be monitored (see, for example, the storage service in Fig. 10). Based on the

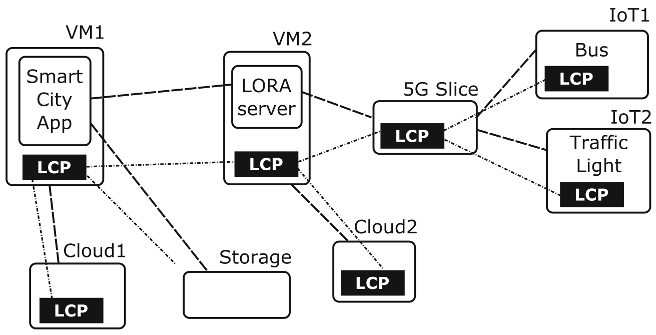


Fig. 10. Schematic representation of a service chain topology and logical tree built by the LCPs.

result of security and risk assessment, the Security Operator may suggest the Service Provider to look for alternative resources, which implement the required interface or provide stronger security guarantees.

Beside discovery, the LCP provides the list of available security capabilities, in terms of security agents that can be configured. Additionally, the description of the ExecEnv provides useful context information to point out vulnerabilities and potential threats, by performing queries to Common Vulnerabilities databases about the hosted software.

5.2 Identity Management and Access Control

The Identity Management and Access Control functionalities are provided by means of the Authentication & Authorization (AA) module. It defines secure authorization procedures for protecting the access to distributed resources in digital service chains. In particular, the module provides authentication services for system components and external agents. Furthermore, authorization services are configured to provide access control through the use of JSON Web Tokens (JWTs) and Access Control Lists (ACLs). JWT is an open standard that defines a methodology to transmit information between parties as a JSON object [15]. This information can be verified and trusted because it is digitally signed. JWT enables a security mechanism to rely on security policies based on Attribute-Based Access Control (ABAC) paradigm, which is based on granting or denying user requests based on both attributes of users and attributes of the object to be accessed to [14].

On the bases of the main architectural components, the AA module distinguishes two security zones: an internal GUARD domain, and an external zone in which there are local agents that offer security services.

The entities inside the GUARD Platform are protected through a centralized authentication mechanism, which ensures that components are recognized through a trusted internal Identity and Access Manager, so that operations

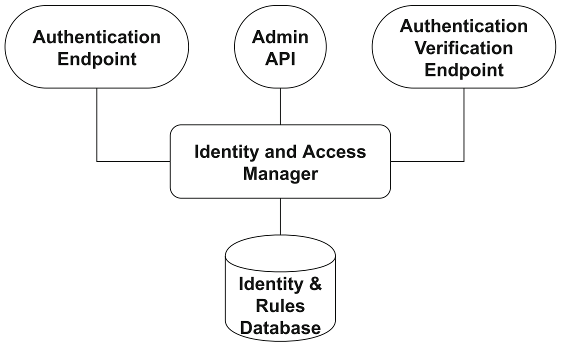

Authentication Endpoint
Admin API
Authentication Verification Endpoint
Identity and Access Manager
Identity & Rules Database

Fig. 11. AA Security Mechanism internal to the GUARD Platform to protect communications between GUARD internal components and security services.

can be confined according to an ABAC-based strategy. The authentication and authorization module of the GUARD Platform can be summarized in Fig. 11.

Protection of GUARD Internal Components. Regarding the protection of GUARD internal components and security services, the main components of the modules are presented hereby.

- ***Identity & Rules Database.*** It stores information relative to the users managed by GUARD and their attributes. This database is accessible by the Identity and Access Manager exclusively.
- ***Identity and Access Manager.*** It employs security routines and services to manage identities and the release of information in the form of a security token.
- ***Admin API.*** It is a REST and SOAP-based interface used for maintenance purposes. A web-based interface is also available that facilitates the administration of security policies by system administrators.
- ***Authentication Endpoint.*** It allows user identification through the exchange of security questions and challenges to prove the identity of the interacting service. If the service is correctly identified, the Identity and Access Manager releases an ABAC-based security token encoded in JWT.
- ***Authentication Verification Endpoint.*** It enables the verification of the security token signature if a resource protected by GUARD is serving an authenticated service. If the signature verification is correct, the protected resource can proceed to service authorization according to attributes present in the security token.

The entire service workflow of the AA Security Mechanism for the protection of internal services is presented in Fig. 12.

Protection for Entities External to the GUARD Platform. The entities external to the GUARD Platform need a decentralized protection, so a different

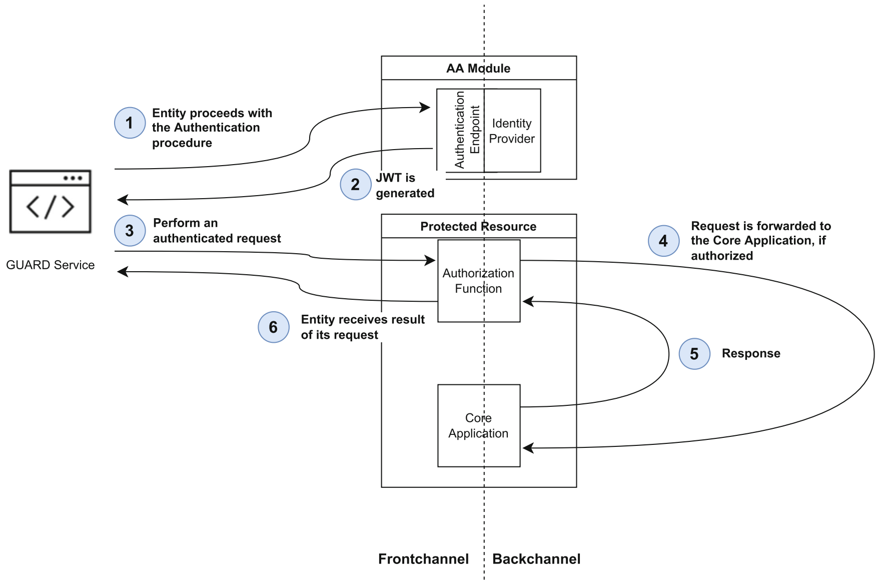

AA Module
Authentication Endpoint
Identity Provider
1 Entity proceeds with the Authentication procedure
2 JWT is generated
GUARD Service
3 Perform an authenticated request
Protected Resource
Authorization Function
4 Request is forwarded to the Core Application, if authorized
5 Response
6 Entity receives result of its request
Core Application
Frontchannel
Backchannel

Fig. 12. Workflow of the AA Security Mechanism internal to the GUARD Platform.

approach is taken to ensure that mutual authentication is established and strong security policies can be applied between digital resources and the GUARD Platform. The security architecture and mechanism for users, LCP and Local Agents are illustrated in Fig. 13, and the main components of the AA module for the protection of the zone external to the GUARD Platform are described in what follows.

– ***Identity Provider.*** The Identity Provider (IdP) handles authentication procedures between services and GUARD users. It also defines the authorizations each actor has in the entire architecture by encoding the information in JWT. Before client authentication can be performed, an Identity Server must be configured in order to register the users that belong to the GUARD domain. This operation is permitted only by the system administrator and is performed through a Management Console with the administrator credentials. The configuration of the Identity Server includes also the assignment of the user roles, to allow or deny access to some applications or privileged operations, according to the ABAC mechanism.
  Once the user is created and registered by the Identity Server, the client can perform the authentication process. It consists of two phases: in the first, the user authenticates to the IdP with username and password; in the second, the IdP returns to the user a JWT, that is appended to the future requests made by the user, to demonstrate that it possesses certain attributes and is correctly authenticated to the system.

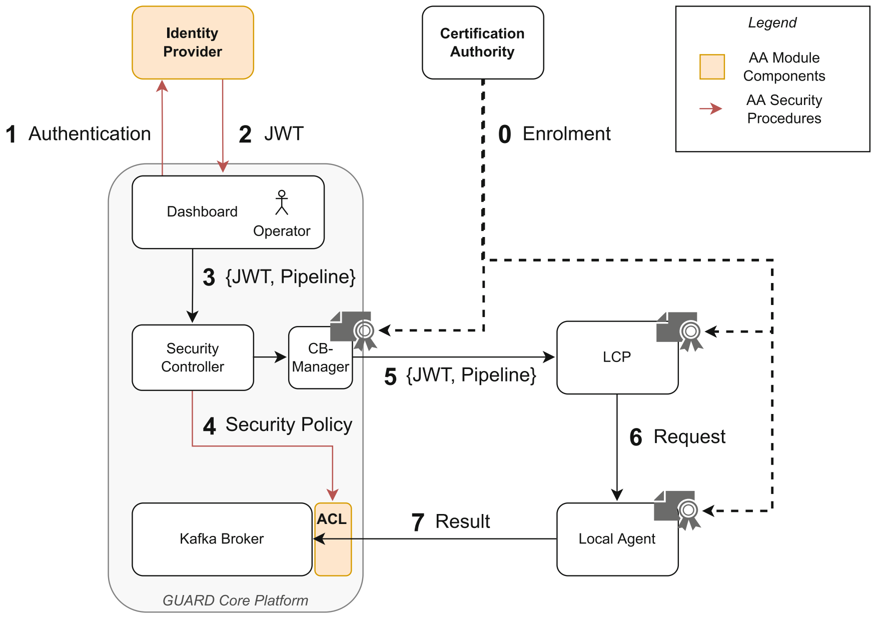


Fig. 13. AA Security Mechanism to protect communications between GUARD and LCP/Local Agents. Key actors are highlighted with bold characters.

- ***Certification Authority.*** In order to guarantee the authentication of LCP/Local Agents and their communications between CB-Manager and Kafka, a GUARD Public Key Infrastructure (PKI) with a Certification Authority (CA) is used to issue certificates for the mentioned actors. In this way it is possible to perform TLS mutual authentication procedures between these services.
- ***Kafka Broker.*** Kafka Broker has been configured to handle authentication with Kafka-enabled clients through TLS mutual authentication. X.509 certificates are used during the TLS handshake process to perform mutual authentication. So, the broker and clients certificates need to be enrolled and trusted. Kafka clients are represented by Local Agents that produce data to be sent to GUARD. Each entity involved in the secure communication must have keystore and truststore files to ensure that the mutual authentication is correctly performed. The keystore contains each machine's own identity. It is an archive which stores the private key and the public certificate signed by a CA. This certificate is also saved in the truststore and distributed among communicating parties, which contains the list of trusted certificates used to determine which entity or CA to trust. In this way, any certificate that was signed by the CA whose public key is in the truststore will in turn be trusted. This property is called the chain of trust, and it is particularly useful when deploying TLS on a large Kafka cluster. It is noteworthy that both the keystore and the truststore are protected by a password. Kafka listens on a

dedicated port with TLS enabled and mutual authentication between broker and clients is performed through specific commands. During this phase, the SSL truststore and keystore are defined, together with the respective passwords.
Finally, once the broker is configured, Kafka clients have to be configured in order to communicate in a secure way with the broker.
- ***Kafka Access Control List.*** The Kafka Broker is configured to check client authorizations by means of an ACL. Kafka provides an ACL for each topic, allowing or denying Agents to perform operations on it. ACLs are handled by an "Authorizer", which is a server plugin used by Kafka to authorize operations. More specifically, the Authorizer controls whether to authorize an operation based on the identity and the resource being accessed. This ACL can be controlled by the Security Controller in order to express security policies at Kafka, thus confining the allowed operations for each Agent.
The designed ACL works as a table of rules. Each rule is composed by the client identity written in its TLS certificate. Furthermore, the rule has a validity in time, for which, once it expires, the ACL denies any further communication with the Kafka client. Each client is authorized to perform a certain operation, i.e., read or write, on a determined topic of interest.

The Security Mechanism for the Protection of the External Components

Figure 13 is of great help to have a precise idea of the sequence of steps needed to describe the proposed security mechanism. More specifically, the procedure consists of the following phases:

1. ***Enrollment Procedure.*** The Enrollment Procedure is illustrated as *Step 0* in Fig. 13. The GUARD's trusted CA enrolls both the CB-Manager and LCPs by providing them a unique public identity encoded in X.509 format. This grants the possibility to mutually authenticate the exchange of the information between the CB-Manager and each LCP at *Step 4* and secure the communications between Agents and Kafka at *Step 6* of Fig. 13.
2. ***Client Authentication.*** Client Authentication is represented in *Step 1* and *Step 2* of Fig. 13. The entities involved in this phase are the GUARD client and the IdP. The GUARD clients can be of different kinds: they could be GUARD components (like the Dashboard depicted in Fig. 13) or human operators of the GUARD Platform. The client initiates the authentication procedure by contacting the IdP through its Authentication Endpoint. Once authenticated, the IdP returns a JWT with attributes characterizing that user for ABAC-based service authorization.
3. ***Service Request and Response Mechanism.*** Once the client is authenticated, each subsequent request to GUARD services and resources includes the JWT. Steps 3–7 of Fig. 13 show the sequence of operations executed to request and obtain a service. More specifically, in *Step 3* the client generates the security pipeline and appends the JWT obtained in the authentication process. This request is sent to the Security Controller, which ensures that the correct ACL policies (*Step 4*) are setup for Kafka messaging, and then forwards

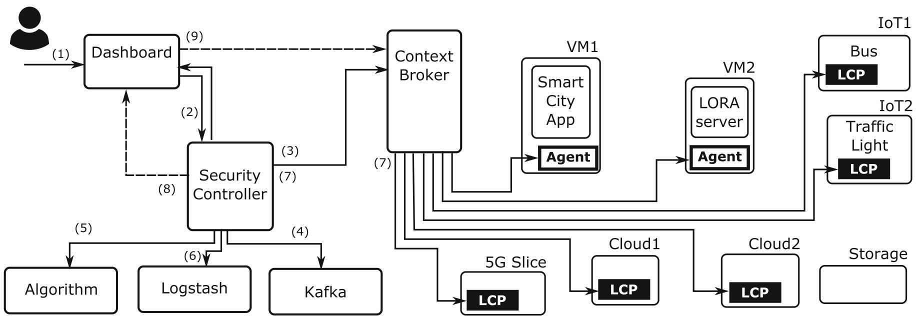


Fig. 14. Workflow for activation of a new security analytics pipeline.

the request to the CB-Manager. In *Step 5*, the CB-Manager is connected to the LCP through mutually-authenticated TLS connection. By leveraging the TLS session and its security services, the CB-Manager forwards the request to the LCP. The LCP has an authorization module that verifies the JWT validity and signature with the public keys of the IdP. If the token signature is correct, the authorization module then proceeds to decode user attributes and use them to detect if the request is valid and authorized for such service operations. These authorization procedures are performed by the LCP. If the request is legit and the user has a valid JWT with the right permissions, the LCP forwards the request to the appropriate Agent (*Step 5*). Finally, in *Step 6* the Agent produces the response and sends it to the Kafka Broker using mutual-authenticated TLS connection.

5.3 Workflows

There are two major workflows implemented by the GUARD Platform, one for creating new SAPs and one for response and mitigation.

Figure 14 shows the workflow for creating a new security analytics pipeline, with reference to the Smart City service illustrated in Sect. 1. The process is initiated by the Dashboard and then carried out by the SC, and consists of the following steps:

1. The user designs a security processing pipeline, by connecting data sources (security agents) with producers (algorithms). The process includes the configuration of both sources and algorithms, as well as the definition of control and reaction policies, as described in Sect. 4.4. The SAP is assigned a unique identifier and the Dashboard stores the SAP information centrally in the CB-Manager.
2. The SAP is encoded in a JSON message and sent to the SC using the REST interface available between the SC and the Dashboard.
3. The SC retrieves from the CB-Manager any necessary information that is referenced in the SAP (e.g., location and parameters of the algorithms, credentials).

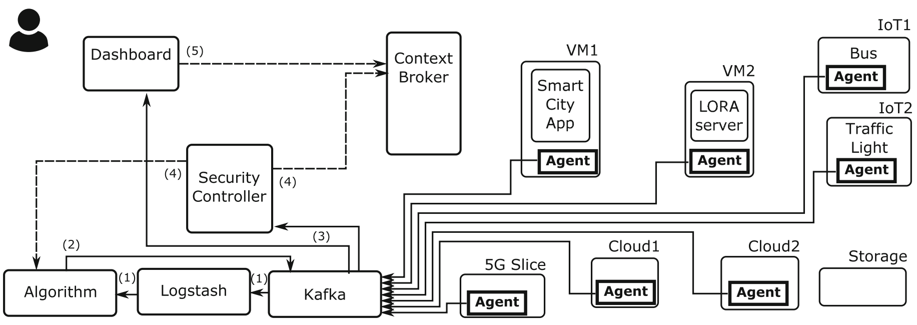


Fig. 15. Workflow for mitigation and reaction.

4. The SC creates a Kafka topic which will be dedicated to deliver data from sources to algorithms.
5. The SC configures and starts the requested algorithm, according to the message from the dashboard. In our implementation, we use a simple REST interface with a few commands to load one or more configuration files and start/stop the algorithm. This should cover a broad range of existing algorithms.
6. The SC downloads, installs, and configures any Logstash plugin which is required to run in the centralized platform. Currently, this step is not implemented yet.
7. The SC configures remote agents through the CB-Manager. The configuration received by the Dashboard is extended to include any necessary transformation and delivery task (currently, this is limited to the automatic generation and configuration of a dedicated Kafka topic). At this point, data are expected to start flowing through the pipeline. The SC also install all control policies included in the pipeline description, which describes what actions should be undertaken under the occurrence of specific events in given conditions. Finally, once the SAP has started correctly, the SC informs the success of the action to the Dashboard using the REST interface. The Dashboard stores both the descriptive template and the configuration of the actual instance of the security pipeline in the Context Broker, with the unique identifier assigned by the Dashboard.
8. Any error that may occur during the configuration process is reported to the dashboard, to inform the security operator that the process did not start.
9. After the security processing pipeline is operational, the security operator can retrieve its description from the CB-Manager, visualize it, and modify some parameters (e.g., frequency of collection, Logstash plugins, etc.). In this case, the dashboard retrieves the configuration of the active SAPs from the Context Broker.

Following its activation, the SAP becomes operational. A closed-loop control workflow is therefore implemented that takes decision based on the evolving context, according to the following steps (see Fig. 15):

1. Data starts flowing from configured agents to the intended algorithm.
2. When the algorithm detects something (attack, anomaly), it reports the information on a specific notification topic on the Kafka bus.
3. All messages published on Kafka (both data from agents and events from algorithms) are available to the SC and the Dashboard; in addition, they are stored in Elasticsearch for offline analytics.
4. Messages published in Kafka topics subscribed by the SC during SAP initialization (step 7 of the activation workflow) triggers the execution of the corresponding security policies. The Security Controller re-evaluate all the rules and tries to satisfy them. The result is usually an action of this kind:
   (a) re-configuration of one or more agents (to apply enforcement actions or to change some detection tasks);
   (b) re-configuration of the detection algorithm, in case for instance an early alarm would trigger more specific investigation.
5. (Optional) Similar reaction operations can be undertaken by the Security Operator through the dashboard, in case there are no suitable policy to handle the event.

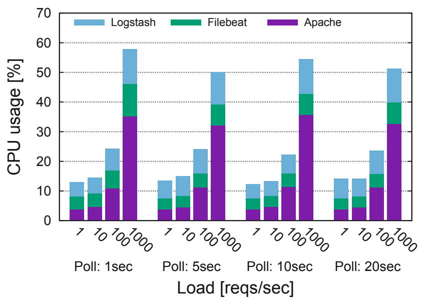
6 Validation and Performance Analysis

We conducted validation and performance analysis to investigate the efficiency of both the data handling pipeline and control plane of the proposed framework. We mostly focused on virtualized environments, which are largely used for the implementation of digital services.

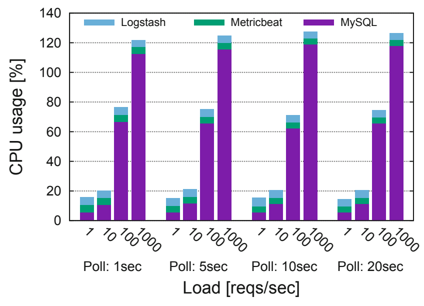
6.1 Data Handling Pipeline

When digital services are monitored remotely, the most critical part of the data handling pipeline is represented by the segment upstream the Kafka bus, involving the security agents and transmission over the Internet. We set up an experimental testbed with considered two pipelines: an Apache web server (monitored by Filebeat) and a MySQL database server (monitored by Metricbeat). Additionally, we consider a Logstash instance for data enrichment and transformation. Both the main service and the corresponding agents are standalone containers that run in the same pod. In case of Filebeat, the agent periodically scans the logs generated by Apache and checks for new records to be sent to the CB-Manager. In our testbed, Logstash adds a timestamp to each log records, and this implies more processing in case of larger workload.

All pods are deployed in a local testbed, made of 3 Kubernetes nodes equipped with 2x Intel Xeon CPU E5-2660 v4 @ 2.00 GHz with 14 cores and hyperthreading enabled, 128 GB RAM, 64 GB SSD storage. The local connection is a plain 1 Gbps Ethernet. We used the default configuration for all containers (1 vCPU, 250 MB RAM).



(a) Apache/Filebeat



(b) MySQL/Metricbeat

Fig. 16. Cumulative CPU usage by all containers in the same pod.

The evaluation was conducted by varying the following parameters:

– the *period of collection*, which affects the latency to access the context and in some cases the volume of traffic generated over the network (for Metricbeat, because it reports the current status at each request), from 1 to 20 seconds;
– the *rate of requests* to the Apache and MySQL servers, which in some cases increase the volume of logs generated (Apache function, which records every access), from 1 to 1000 requests/s. We used jmeter[2] and mysqlslap[3] to generate a variable amount of requests for Apache and MySQL, respectively.

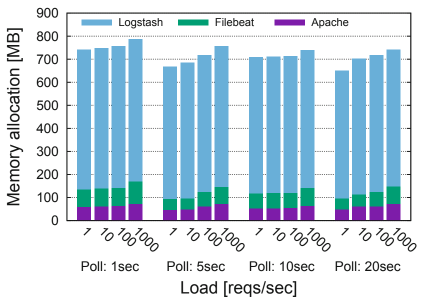
We initially consider the impact of security agents on the main business logic (namely Apache/MySQL). Figure 16 shows that the overhead of the agents is rather limited (below 10% of the available CPU) in all conditions but for the largest number of requests. Indeed, the relative impact on MySQL is much lower, especially at high-load because this application uses more CPU cycles. We conclude that the impact of agents remains quite limited (below 10%), which is acceptable in most practical cases. By comparing Filebeat and Logstash, we conclude that the latter has a higher overhead, even in case of simple operations (i.e. timestamping).

Memory allocation of the security components has a large impact on the main business logic, especially due to Logstash, as shown in Fig. 17. Our understanding is that the current implementation of Logstash is not suitable for lightweight operation in cloud-native applications, because its memory footprint is often bigger than the main application. In general, it would be preferable to directly write to the Kafka bus with Filebeat, if additional transformation operations are not strictly necessary.

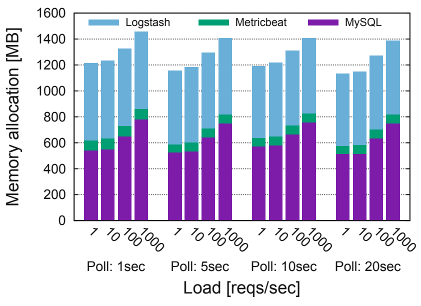
Finally, we consider the delay introduced along the pipeline, up to the Logstash instance deployed in the GUARD Platform.

Figure 18(a) shows the latency to gather data from the Apache log file by Filebeat, to move data from the Filebeat to the local Logstash instance, and to

[2] https://jmeter.apache.org/.
[3] https://mariadb.com/kb/en/mysqlslap/.

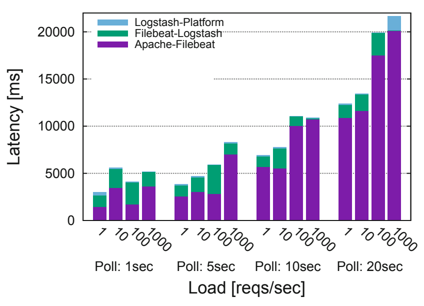


(a) Apache/Filebeat

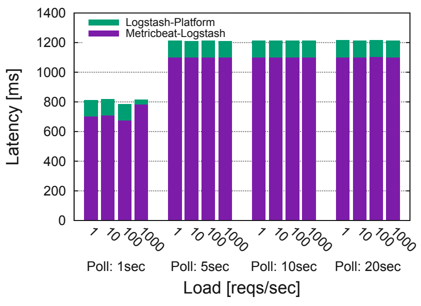


(b) MySQL/Metricbeat

Fig. 17. Cumulative memory allocation for the pod.



(a) Apache/Filebeat

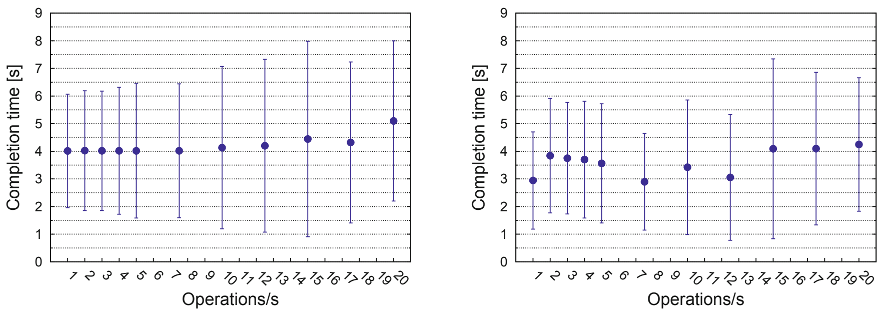


(b) MySQL/Metricbeat

Fig. 18. Cumulative latency for the pipeline up to Logstash instance in the GUARD Platform.

transfer data to the GUARD Platform. The latency is generally shorter than a few seconds, and it is mostly affected by longer polling intervals. Indeed, there are two main consequences when varying the parameters under investigation. On the one hand, higher workloads result in more lines in the log file, hence bigger messages to be sent. On the other hand, longer sample times also result in more data lines and, most of all, in larger delay in the delivery of older lines. The second effect is predominant, since the transmission of larger packets does not affect significantly the latency with high-speed links.

Similarly, Fig. 18(b) shows the latency to move data from Metricbeat to Logstash and to transfer data to the GUARD Platform. The latency is generally around one second, and this time does not increase with the workload or the polling interval. The reason is that the same amount of information is collected in this situation (metrics), independently of the workload and polling interval. In any case, we note that a greater latency in data collection is not a specific performance limitation of the agents, but it may have side effects on the timely of the detection. The polling interval must therefore be selected case-by-case depending on the specific needs of the detection process.



(a) Topology retrieval (b) Agent configuration

Fig. 19. Latency of common operations on the CB-Manager. The picture shows mean value and standard deviation.

6.2 Control Abstraction

The CB-Manager is queried by other components to retrieve the current topology and change the configuration of agents. Therefore, it represents a bottleneck for the operation of both the Dashboard and SC. We therefore investigated the delay in using the CB-Manager REST API, also involving operation of the LCP. Several scenarios were analyzed, but for the sake of brevity we only show the results for two of them. We therefore consider *i*) the latency to get the service topology (this operation does not involve the LCP, because the discovery workflow is performed with lower frequency); *ii*) the latency to configure and start an agent (which is the common part to the activation and reaction workflows). Both operations are made of multiple HTTP requests (12 for getting the topology and 10 for configuring the agent) and corresponding responses from the CB-Manager.

The evaluation was done using Apache JMeter, to generate a variable load of operations per seconds, from 1 to 20. Figure 19 shows that it takes on average a few seconds to complete an operation with the CB-Manager. In case of topology retrieval, there is slight increment of the latency with more concurrent operations per second; the trend is less clear for the other scenario. Overall, the latency is quite good for control and management purposes.

7 Conclusion

In this Chapter we have described the GUARD architecture to create detection and analytics pipelines for multi-domain distributed digital service chains. Our implementation demonstrate the feasibility of the overall concept, by providing the necessary interfaces and authentication and authorization mechanisms to perform monitoring and inspection tasks in Third Party's digital resources. We have also demonstrated that common agents entail a low overhead in the execution of the main business logic, so their implementation does not lead to major costs for resource providers.

Future work will be addressed to investigate monitoring and inspection mechanisms that fit modern computing paradigms. Indeed, we argue that the implementation and proper isolation of local security agents remains a responsibility of Resource Provider, which must guarantee this does not turn into a security vulnerability. Additionally, PaaS and serverless environments are challenging to be monitored by legacy agents and tools. Our work will focus on the possibility to create monitoring and inspection tasks at run-time, by leverage the concept of code augmentation in a safe way.

Acknowledgment. This work was supported in part by the European Commission under Grant Agreement no. 833456 (GUARD).

References

1. Open command and control (OpenC2). OASIS standard, language Specification Version 1.0, Committee Specification 02, November 2019
2. Alt, F., Zezschwitz, E.: Emerging trends in usable security and privacy. J. Interact. Media **18**(3), 189–195 (2019)
3. Bienias, P., Kołaczek, G., Warzyński, A.: Architecture of anomaly detection module for the security operations center. In: IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), Napoli, Italy, 12th–14th Jun 2019 (2019)
4. Bierman, A., Bjorklund, M., Watsen, K.: RESTCONF protocol. RFC 8040, January 2017. https://tools.ietf.org/html/rfc8040
5. Bjorklund, M.: YANG - a data modeling language for the network configuration protocol (NETCONF). RFC 6020, October 2010
6. Bjorklund, M., Schoenwaelder, J., Bierman, A.: Network configuration protocol (NETCONF). RFC 6241, June 2011. https://tools.ietf.org/html/rfc6241
7. Bringhenti, D., Marchetto, G., Sisto, R., Valenza, F., Yusupov, J.: Towards a fully automated and optimized network security functions orchestration. In: 4th International Conference on Computing, Communications and Security (ICCCS), Rome, Italy, 10th–12th October 2019 (2019)
8. Caviglione, L., Mazurczyk, W., Repetto, M., Schaffhauser, A., Zuppelli, M.: Kernel-level tracing for detecting stegomalware and covert channels in linux environments. Comput. Netw. **191**, 108010 (2021)
9. Chamiekara, G.W.P., Cooray, M.I.M., Wickramasinghe, L.S.A.M., Koshila, Y.M.S., Abeywardhana, K.Y., Senarathna, A.N.: AutoSOC: a low budget flexible security operations platform for enterprises and organizations. In: National Information Technology Conference (NITC), Colombo, Sri Lanka, 14th–15th September 2017, pp. 100–105 (2017)
10. Elsayed, M.A., Zulkernine, M.: Towards security monitoring for cloud analytic applications. In: IEEE 4th International Conference on Big Data Security on Cloud (BigDataSecurity), Omaha, NE, USA, 3rd–5th May 2018, pp. 69–78 (2018)
11. Elsayed, M.A., Zulkernine, M.: PredictDeep: security analytics as a service for anomaly detection and prediction. IEEE Access **8**, 45184–45197 (2020)
12. Hardt, D.: The OAuth 2.0 authorization framework. RFC 6749, October 2012. https://www.rfc-editor.org/rfc/rfc6749.txt

13. Hares, S., Lopez, D., Zarny, M., Jacquenet, C., Kumar, R., Jeong, J.: Interface to network security functions (I2NSF): problem statement and use cases. IETF RFC 8192, July 2017. https://www.rfc-editor.org/rfc/pdfrfc/rfc8192.txt.pdf
14. Hu, V., et al.: Guide to Attribute Based Access Control (ABAC) Definition and Considerations. Special Publication (NIST SP) 800-162, National Institute of Standards and Technology (NIST) (2019)
15. Jones, M., Bradley, J., Sakimura, N.: JSON Web Token (JWT). RFC 7519, May 2015
16. Lang, B., Wang, J., Liu, Y.: Achieving flexible and self-contained data protection in cloud computing. IEEE Access **5**, 1510–1523 (2017)
17. Laue, T., Kleiner, C., Detken, K.O., Klecker, T.: A SIEM architecture for multidimensional anomaly detection. In: 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, Cracow, Poland, 22nd–25th September 2021, pp. 136–142 (2021)
18. Lin, H., Yan, Z., Chen, Y., Zhang, L.: A survey on network security-related data collection technologies. IEEE Access **6**, 18345–18365 (2018)
19. Lopez, D., Lopez, E., Dunbar, L., Strassner, J., Kumar, R.: Framework for interface to network security functions. IETF RFC 8329, February 2018. https://tools.ietf.org/pdf/rfc8329
20. Lynch, L.: Inside the identity management game. IEEE Internet Comput. **15**(5), 78–82 (2011)
21. Nespoli, P., Papamartzivanos, D., Marmol, F.G., Kambourakis, G.: Optimal countermeasures selection against cyber attacks: a comprehensive survey on reaction frameworks. IEEE Commun. Surv. Tutor. **20**(2), 1361–1396 (2018)
22. Open command and control (OpenC2) profile for stateless packet filtering, version 1.0, Committee Specification 01, July 2019
23. Repetto, M., Carrega, A., Duzha, A.: A novel cyber-security framework leveraging programmable capabilities in digital services. In: Proceedings of the Fourth Italian Conference on CyberSecurity (ITASEC20), Ancona, Italy, 4th–7th February 2020, pp. 201–211 (2020)
24. Repetto, M., Carrega, A., Rapuzzi, R.: An architecture to manage security operations for digital service chains. Futur. Gener. Comput. Syst. **115**, 251–266 (2021)
25. Repetto, M., Striccoli, D., Piro, G., Carrega, A., Boggia, G., Bolla, R.: An autonomous cybersecurity framework for next-generation digital service chains. J. Netw. Syst. Manag. **29**(37) (2021)
26. Sage, A., Cuppan, C.: On the systems engineering and management of systems of systems and federations of systems. Inf. Knowl. Syst. Manag. **2**(4), 325–345 (2001)
27. Shehab, M., Marouf, S.: Recommendation models for open authorization. IEEE Trans. Dependable Secure Comput. **9**(4), 583–596 (2012)
28. Vapen, A., Carlsson, N., Mahanti, A., Shahmehri, N.: A look at the third-party identity management landscape. IEEE Internet Comput. **20**(2), 18–25 (2016)
29. Weiser, M.: The computer for the 21st century. Sci. Am. **265**(3), 94–105 (1991)
30. Yang, J., Jeong, J.: An automata-based security policy translation for network security functions. In: International Conference on Information and Communication Technology Convergence (ICTC), Jeju, South Korea, 17th–19th October 2018, pp. 268–272 (2018)
31. Zuech, R., Khoshgoftaar, T.M., Wald, R.: Intrusion detection and big heterogeneous data: a survey. J. Big Data **2**, 3 (2015)