Automatic analysis of containerized application deployment models based on ontologies and knowledge graphs

Andrei Brazhuk

***Abstract*— Modern network applications commonly work in cloud environments, based on orchestration of hardware virtualization or containers. Design of such applications meets new security challenges related to their distributed architectures, the use of third-party components, deployment flexibility, and short life-cycle stages. These challenges require application security analysis to be continuous and automated.**

To implement the secure by design principle, automatic threat modeling based on threat/security patterns can be added to design secure applications. However, for a long time both threat modeling and threat/security patterns operate manual procedures and time consuming methods. Currently, automation in this field is still in the low maturity layer, caused by lack of experimental research and machine-readable data.

Toward overcoming these challenges, the work researches an approach based on ontologies and knowledge graphs to automatically determine application architecture (functions, structure) from software deployment models for farther adding right patterns. In particular, an ontology-driven framework has been adopted for automatic semantic representation of multi-container applications (Docker Compose) and learning their architectures. To prove the effectiveness of semantic patterns, used to automatically detect application structures and functions, an open dataset of 200 semantic diagram has been created.

***Keywords*— Threat/security patterns, Threat modeling automation, Ontology, Knowledge graph, Container application, OWL, RDF, SPARQL, Docker Compose.**

I. INTRODUCTION

Modern network applications are characterized by complexity and frequent modifications. Nowadays, most of applications work in cloud environments, based on orchestration of hardware virtualization or containers. The use of cloud technologies causes new security challenges. It is common for cloud applications to have distributed structures and include several remote components (services), what may cause security breaches via weak design of a particular service. Some of remote components might be uncontrolled by application's owner; such a third-part service may violate security via poor service level agreement. Also, an application might use local infrastructure at one stage of life-cycle (e.g. development), and it may move to cloud at another stage (e.g. operations), what changes threat landscape and requires its re-evaluating.

Manuscript received August 12, 2025.

Andrei Brazhuk (email: brazhuk@grsu.by) is with the Yanka Kupala State University of Grodno, Republic of Belarus (st. Ozheshko, 22, Grodno, 230023, Belarus)

In cloud environments automatic software deployment techniques have been spread, in which a *software deployment model* in form of declarative configuration file(s) can be used to automatically run an application. Such an approach makes application life-cycle stages extremely short, what requires security analysis to be continuous and automated.

As a way of implementing the secure by design principle [1], *threat modeling* can be added to tackle security challenges, related to application design, via identifying potential threats and devising appropriate security solutions [2, 3]. For a long time, this discipline has operated manual procedures and time consuming methods. Currently, its automation is still in the low maturity layer, caused by lack of experimental research and machine-readable data ([4, 5]. However, the expansion of agile methodologies, CI/CD (Continuous Integration/Continuous Deployment), and DevOps (Development plus Operations) requires new approaches to the threat modeling automation [6, 7].

Another keystone of the secure by design implementation is collecting and leveraging the knowledge regarding application threats and safe architectural solutions. *Threat/security patterns* can be used as a source of structured security knowledge [8]. Either of them aims to formulate criteria, which help to determine relevance of a pattern for a particular system design. This makes the use of threat/security patterns a promising strategy for the threat modeling automation. However, their automation seems to be an underrated research area [9]. Currently, adding patterns to a design has to be done by humans via an advanced methodology [10].

This work is moving toward automation of threat modeling based on threat patterns by filling the gap in the *threat pattern contextualization*. At early stages the automatic security analysis of an application supposes determining its architecture (functions, structure) for farther adding right patterns [11]. Applications can serve different functions: web applications, data processing applications, financial services, machine learning services, etc, and they can utilize various design features. For example, a web application may have a simple structure (e.g., comprising only a web server and a background database) or a sophisticated structure (e.g., comprising several web servers with a database behind a load balancer). For the simple web application, the ’web server - database’ flow could be affected by a set of threats, such as SQL injections [12], unauthorized access through publicly available network port of the database, and exploiting default administrator

credentials used by the database. For the sophisticated web application, the flow 'load balancer - web server' may produce extra threats, such as failure of the load balancer and HTTP Request Smuggling [13]. Flows like 'web server - database', 'load balancer - web server' can be considered as *design primitives*, bringing relevant sets of potential threats to an application.

Considering automatic threat/security patterns contextualization for the use case of cloud application architectures, the primary Research Question (RQ) of the work is:

RQ: How to enable the automatic analyse of application structures and functions from software deployment models?

To address the research question, ontologies and knowledge graphs can be used in order to automatically learn functions and structures of real cloud applications [14]. Ontologies follow the object-oriented paradigm, in which a domain is thought as classes, objects and relations between objects. An ontology-driven framework enables a semantic representation of an application configuration both as an ontology in the OWL (Web Ontology Language) format and as a knowledge graph in the RDF (Resource Description Framework) format [11]. The ontological representation is used to perform intermediate automatic reasoning and infer extra facts about the configuration. The extra (implicit) knowledge allows to learn structures of applications and their security aspects from software deployment models.

Docker Compose is a container platform tool, in particular, it enables description of applications in a single configuration file [15, 16]. Such software deployment models as raw data have been collected for this research. Software processing has been added to create ontologies and knowledge graphs from the Docker Compose configurations. To analyse application structure and functions, semantic templates have been used in form of SPARQL requests under the RDF representation of the configurations. A *semantic template* indicates the presence of a design primitive and therefore related set(s) of threats in the configuration.

In summary, the work contributions are as follows:

1) *The ontology-driven framework has been adopted for automatic semantic representation of container applications and learning their structures and functions.*

2) *An open dataset of 200 semantic diagrams based on real container applications has been generated to mitigate the lack of research data in this field.*

3) *An experimental estimation based on the generated dataset has been conducted that proves the effectiveness of semantic patterns for automatic detection application structures and functions.*

II. Literature Review

Currently, threat modeling is considered as a semi-automatic process, added primarily at the requirements and design stages of a system lifecycle [17, 18]. Well-known manual methodologies tend to add more formalization to this process [19]. The efforts to solve the challenge of threat modeling automation are primary based on graph theory, Domain Specific Languages (DSL) and rule based languages [20], First Order Logic (FOL), Prolog, and Modal Logic [21]. Adding Machine Learning (ML), Neural Networks (NNs), and Large Language Models (LLMs) to the threat modeling is a great open challenge [22].

Another direction of the threat modeling automation is the use of ontologies and automatic reasoning. It is simpler than FOL and Prolog, follows the object-oriented paradigm, and can be easily extended via knowledge management means (OWL, RDF, SPARQL, SWRL). The idea of the ontological approach has been described in works [23] and [24]. Work [11] has offered an ontology driven framework for automatic threat modeling and its use case for cloud systems. Recent researches on the ontology based threat modeling are dedicated to the automation of security analysis of ICT Infrastructures [25], Amazon Cloud Infrastructure [26], and Cyber-Physical Systems [27].

The absence of research datasets and the lack of techniques to create such datasets is a bottleneck of the threat modeling automation. Traditionally, Data Flow Diagrams (DFDs) are commonly used to describe a system structure for security analysis; and relevant publications dealing with DFDs operate only using dozens of diagrams [28, 29]. However, to prove the effectiveness of automatic threat modeling methods, it requires a dataset that includes at least hundreds of diagrams.

Security and threat patterns seem to be an underrated research field [8]. Patterns represent a great way to formalize and use the security knowledge at the requirement and design stages of system life-cycle. However, those procedures typically have to be done by humans via advanced methodologies [10].

Several attempts are known towards the automation of security and threat patterns. Work [30] proposed an ontology based catalog of security patterns. Research [31] aimed to collect and classify security patterns for connected vehicles in a structured way. Work [32] has described a method to detect the security patterns based on a distributed matrix matching technique. Recent work [33] proposes a framework for generation security cases from a list of security requirements based on security argument patterns.

III. The Proposed Framework

To address the work research question, an Ontology-driven Threat Modeling (OdTM) framework has been used (https://github.com/nets4geeks/OdTM). The framework enables semantic representation of security related knowledge by creating OWL ontologies called *Domain Specific Threat Models* (DSTMs), each one enumerates typical components and associated threats for various types of computer systems (e.g., Cloud Computing, Internet of Things, Software Defined Networks).

A structure of a particular computer system (which can be depicted as a diagram) can be represented as a *semantic interpretation* [11]. The semantic interpretation describes components of the system and relations between them as OWL ontology via the DFD terminology and some other domain-specific terminologies.

The crucial item of the framework is a *Base Threat Model* (BTM), implemented as OWL ontology. The model contains necessary terminology to describe the diagram items, threats, and security labels. BTM can be used to create both

semantic interpretations and DSTMs.

Fig. 1 shows how the framework is mapped to the ontology driven analysis.

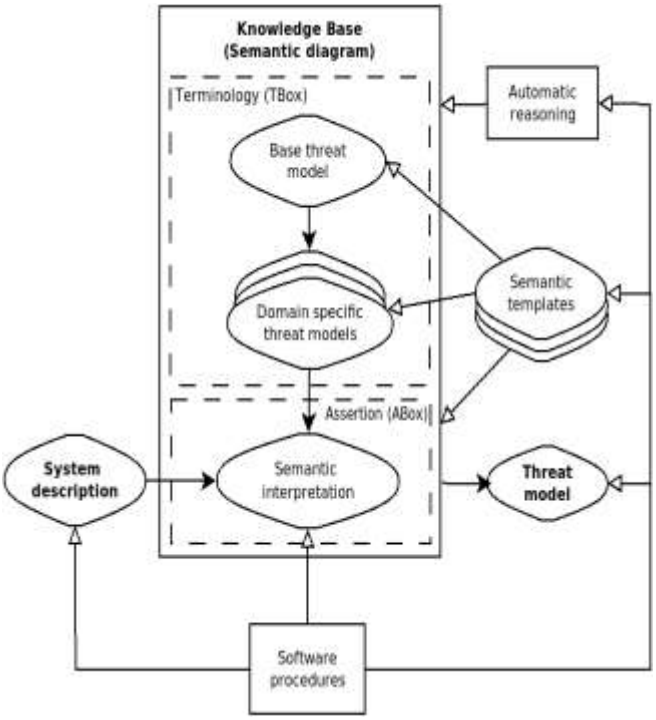

Fig. 1. The framework structure

The ontology driven approach is based on a Knowledge Base (KB). In general, KB consists of Terminology Box (TBox) and Assertion Box (ABox). In proposed framework, TBox includes the Base Threat Model and one or more Domain Specific Threat Models. The semantic interpretation is used as ABox.

The automatic reasoning procedures are used on KB in order to infer extra facts about the diagram. The implicit knowledge can include possible threats to the system components. After automatic reasoning done, the KB, which represents the semantic diagram, can be saved as a knowledge graph in the RDF format. Software procedures are used to manipulate the modeling artifacts in order to enable the threat modeling process and obtain its results.

Semantic templates aim to represent both the structure elements and their properties in a semantic diagram, which indicate the presence of some threat pattern(s) or point out the need to add some security pattern(s) to the system design. Semantic templates can be implemented as either ontology constructions or semantic requests to KB.

To illustrate the used approach, a software deployment model in the form of a Docker Compose configuration file (docker-compose.yml) is considered [34]. Docker Compose enables a declarative description of several containers (services) as a multi-container application in a single file. The docker-compose.yml follows the YAML text format.

Fig. 2 shows a docker-compose.yml file. It includes two services (’web’ and ’mongodb’), each one is implemented as a container. The ’image’ property indicates a basic image for a container (a software set required to run a service): the PHP development environment for ’web’ and the non-relational database MongoDB for ’mongodb’. Also, declarations of the containers may include descriptions of their storage via the ’volumes’ property: ’web’ uses a host storage (’/app’), while ’mongodb’ uses the volume (’dbdata’) provided by the Docker engine.

```
services:
    web:
      image: php:8.0
      volumes:
        - ./app:/var/www/html
      depends_on:
        - mongodb
      ports:
        - 80:80

    mongodb:
      image: mongo:latest
      volumes:
        - dbdata:/data/db
      links:
        - web
```

Fig. 2. An example of software deployment model

Network connections can be described in a configuration file. In this example, the ’web’ container shares the TCP port 80, which indicates that a HTTP server will be run on the container (see the ’ports’ property). Also, relations between services can be described via the ’links’ and ’depends_on’ properties: ’web’ depends on ’mongodb’, and ’mongodb’ has a link to ’web’.

A semantic diagram that may be depicted from the configuration is shown in Fig. 3. Note, no need exists of graphical representation for automatic threat modeling: data in the OWL and RDF formats are only required for that, so Fig. 3 may be used only for understanding of the process.

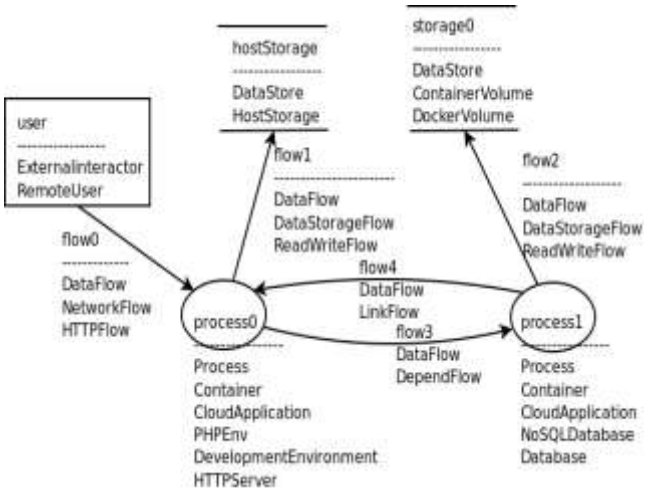

Fig. 3. An example of semantic diagram

To create the ontology by the software, the DFD terminology has been used from the Base Threat Model. The containers have become the ’Process’ instances: ’web’ as ’process0’, and ’mongodb’ as ’process1’, shown as circles in DFD. Storage have been known as the ’Datastore’ instances (’hostStorage’, ’storage0’), shown as two lines. To represent remote interaction of the application, the ’user’ instance has been added, which has been assigned with the ’ExternalInteractor’ and ’RemoteUser’ classes, and shown as a rectangle.

Also, flows have been created. The ’flow0’ flow goes

from 'user' to 'process0' because 'process0' has an open network port. The 'flow1' and 'flow2' flows come from processes to their storages ('process0' - 'hostStorage', 'process1' - 'storage0'). The 'flow3' flow goes from 'process0' to 'process1' because 'process0' depends on 'process1'. The 'flow4' flow comes from 'process1' to 'process0' because of the link between them. All the flows are initially instances of the 'DataFlow' class, and are shown as lines with arrows.

Pieces of domain specific knowledge have been used to enrich the diagram. So, 'process0' has an HTTP server running, and it is associated with the 'HTTPServer' class. For that reason, 'flow0' is labeled as the 'HTTPFlow' instance. Also, 'process0' belongs to the 'PHPEnvironment' class in particular, and to the 'DevelopmentEnvironment' class in general, because it runs a PHP interpreter.

A domain ontology can be used to hold the domain specific knowledge. For example, the port 80 in a configuration indicates 'HTTPServer'; the 'php' keyword points to the 'PHPEnvironment' class; and latter is a child of 'DevelopmentEnvironment'. As a result, we have got the taxonomy that maps keywords from configuration files to classes of semantic diagrams. The domain ontology can be saved as a structured text file (YAML in this case). It also can be included into a Domain Specific Threat Model.

Taking a decision of adding either threat or security pattern for a particular system design (architecture) is called a *pattern contextualization*. The contextualization depends on various factors, including relations between system components, presence of external entities, and properties of components.

Specific interactions between system components, taking into account their properties, are called *design primitive*s. Design primitives characterize various aspects of application architecture.

Semantic templates are used to find design primitives in a diagram in order to automatically analyze its structure and functions. Results of such analysis can be used in the taking a decision for putting either security or threat patterns in the context. Semantic templates can be implemented as OWL axioms (equivalence and class assertions) or as SPARQL requests to the RDF representation of a semantic diagram.

The Base Threat Model has a symmetric object property 'relates' that maps every two components that have a flow between them.

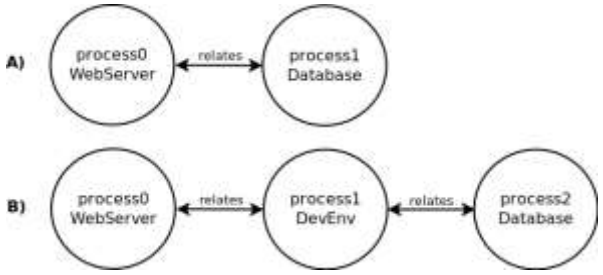

Fig. 4. Web-based design primitives

Fig. 4 shows some design primitives that can be used to analyze an application: (A) a simple web application based on two components, and (B) a complex web application with three components, related to each other.

A semantic template for the (A) case as OWL axiom may be as shown in Listing 1.

Listing 1. OWL based semantic template

```
WebServer and ( relates some Database )
```

A semantic template for the (B) case represented as a SPARQL request may be as Listing 2 shows.

Listing 2. SPARQL based semantic template

```
SELECT ?target ?target1 ?target2
WHERE {
      ?target rdf:type :WebServer .
      ?target1 rdf:type :DevelopmentEnvironment ;
      b :relates ?target .
      ?target2 rdf:type :Database ;
      b :relates ?target1 .
}
ORDER BY ?target
```

IV. THE DATASET OF SEMANTIC DIAGRAMS

To tackle the problem of lack of open datasets that could be used for automatic threat modeling research, a dataset of 200 semantic diagrams has been created. The dataset is based on configurations of real container applications and published via Github (https://github.com/nets4geeks/DockerComposeDataset).

The Docker Compose configurations (docker-compose.yml) have been used. Each docker-compose.yml file is a software deployment model that may include several containers (services). The software deployment models have been obtained from public repositories, such as Github and Gitlab, through Google search requests. Also some enterprise repositories have been used (to keep the privacy and copyrights, source files were not published on the Github repository, only depersonalized artifacts are there). The primary criterion of taking a file into the dataset was the presence of two and more services in an application.

To build the dataset, a special software tool, written in Java, has been used. The tool is based on the OWL API library to manipulate OWL ontologies and RDF knowledge graphs. The Hermit library is used as an automatic reasoner. The Jackson library has been added to process the structured documents in the YAML format.

Fig. 5 shows the outline of processing of a single configuration by the software tool. At the first stage a configuration of a container application is used as an input for software processing based on a domain ontology. A semantic interpretation as OWL ontology is an output there. The domain ontology is used to add extra facts about the configuration to the diagram. The entities may be associated with classes from the ontology that classify services (e.g. 'SQLDatabase', 'WebServer', 'HTTPServer'), storages (e.g. 'HostStorage', 'DockerVolume'), flows (e.g. 'HTTPFlow'). The domain ontology was being created while the dataset was being formed. It is saved in the YAML format (the services2.yml file in the public repository).

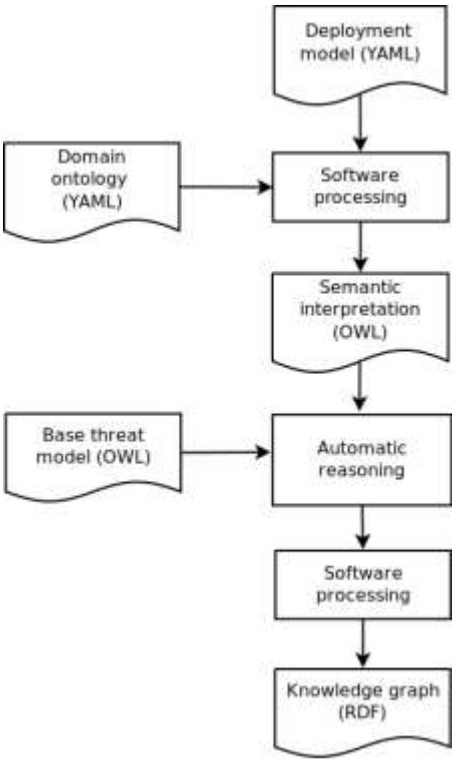

Fig. 5. Creating a semantic diagram

The next stage is the automatic reasoning. The semantic diagram imports the Base Threat Model. Hermit performs reasoning and adds implicit knowledge to the semantic diagram.

At the last stage, the software procedures save the inferred ontology (semantic diagram) as a knowledge graph in the RDF format. The use of RDF makes it easier adding the SPARQL requests in order to examine the diagram by various 'questions' related to the application design and security.

The software tool has been used to process 200 docker-compose.yml files, and a dataset has been created that includes 200 semantic diagrams (the clear2 folder on Github). Each diagram is represented both as OWL ontology with the explicit knowledge and RDF graph with implicit knowledge.

The dataset mitigates lack of research data in the automatic threat modeling field. Existing datasets operate by dozens of diagrams, in this dataset number of diagrams is increased to hundreds. The advantage of the dataset is that the data are represented in strict semantic format what enables their automatic intelligent processing. The dataset has been used in the experiment as a part of this work, and it could be used in various research in the automation threat modeling field.

V. The Experiment and Discussion

To evaluate the effectiveness of the proposed approach, an experimental assessment has been conducted, in particular, it has been researched how to determine application type with semantic templates.

Fig. 6 shows the experimental schema. At the first stage the configuration files have been classified by an expert according predefined criteria (application types). At the second stage the semantic diagrams, obtained from the configurations, have been classified automatically via semantic templates, corresponding to the given criteria.

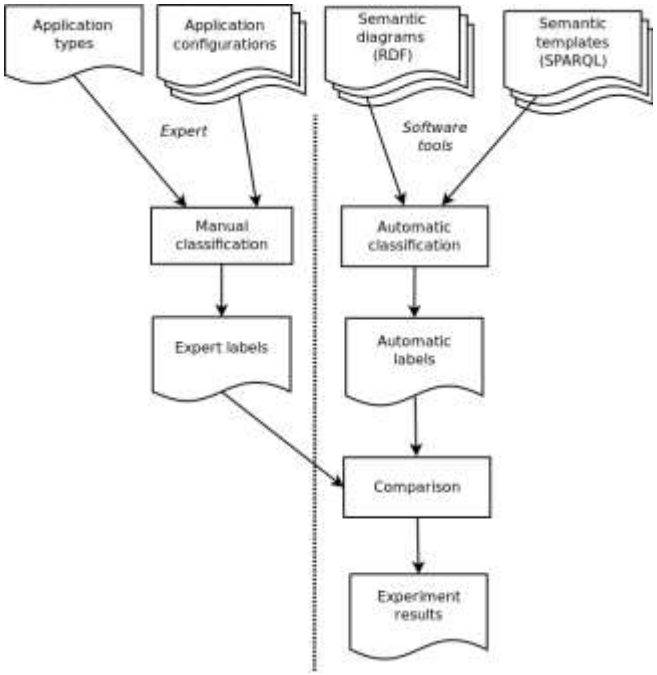

Fig. 6. Schema of the experiment

The results of the experiment (precision and recall of the automatic classification) have been got via comparison with the expert classification.

The expert in software engineering (design of microservices) was given 200 configurations of applications (the docker-compose.yml files). The expert had to map every configuration to predefined criteria. Each criterion (application type) characterizes both functions and structures of applications.

The application types are:

1. *Web Application*. A simple application that accessible via WWW and contains a background database. For example, it may include two containers - Nginx as HTTP/HTTPS server and Mysql as a database.

2. *Complex Web Application*. A sophisticated web application based on database and including extra components like a development environment (PHP, Python, Ruby etc.).

3. *Data Processing*. An application that aims either to collect data (e.g. includes a data collector like Filebeat) or to visualize data (e.g. includes Kibana). Both cases suppose the use of data storage, like document-oriental database (Elasticsearch) or relational database (like Mysql or Postgres).

4. *Complex Data Processing*. A sophisticated data processing application that includes both collecting and visualizing components.

Note, Complex Web Applications (2) have been automatically considered as Web Applications (1), because included their items. The Complex Data Processing (4) criterion has included Data Processing (3) as well. Also, some configurations have not been classified (27%), because they did not fall in any category from the expert viewpoint.

Semantic templates have been used to automatically

determine application functions and structures. As it is shown above, a semantic template allows to recognize relations between items of a diagram and represents some fragment of a knowledge graph of the diagram. Presence of a particular graph fragment in the diagram can indicate that the diagram belongs to some type.

Templates corresponding to the expert criteria are shown in Table 1. The 'qa01-1' and 'qa01-2' templates indicate Web Application. The 'qa02' and 'qa02a' templates point to Complex Web Application. The 'qa03-1' and 'qa03-2' templates relate to Data Processing, and 'qa04' - Complex Data Processing. The 'Semantic template' column contains representations of templates as sequences of relations between the domain concepts based on the 'relates' object property.

Table 1. Semantic templates of container applications

No	ID	Semantic template
1	qa01-1	WebServer - Database
2	qa01-2	DevelopmentEnvironment - Database
3	qa02	WebServer - DevelopmentEnvironment - Database
4	qa02a	DevelopmentEnvironment - DevelopmentEnvironment - Database
5	qa03-1	DataCollector - Database
6	qa03-2	DataVisualizer - Database
7	qa04	DataCollector - Database - DataVisualizer

Automatic recognition of diagram types has been done in following way. Each semantic template was represented as a SPARQL request (see the Github repository, file names of the templates are the same as IDs in Table 1). Then the SPARQL requests were used under the RDF graphs of the diagrams via the Apache Jena tool in order to find the semantic templates. Shell scripts were used to calculate number of automatically detected types and compare them with the expert classification.

Table 2 shows the experiment results. Statistics of the expert classification is depicted both in the 'Type' column that contains the predefined application types, and the "Expert" column that contains amount of the configurations assigned to each application type by the expert. The 'Templates' column shows the semantic templates used to detect each application type.

Table 2. Expert classification vs automatic classification

	Application type	Semantic templates	Expert classification	Detected templates	Positive	Precision	Recall
1	*Web Application*	qa01-1 qa01-2	105	81	79	0.98	0.75
2	*Complex Web Application*	qa02 qa02a	33	26	26	1.00	0.79
3	*Data Processing*	qa03-1 qa03-2	40	30	30	1.00	0.75
4	*Complex Data Processing*	qa04	18	14	14	1.00	0.78

Results of automatic classification are shown in following columns of Table 2. The 'Detected' column counts the diagrams, in which a template has been found by the Apache Jena tool. The 'Positive' column shows amount of coincidences of the automatic classification with the expert classification. 'Precision' is the fraction of the correct results to the number of automatic analysis results. 'Recall' is the fraction of the automatically detected diagrams, containing a template, to the amount of the configurations of that type identified by the expert.

The experiment results demonstrate high precision (up to 100%) of the automatic analysis, and recall equals or more 75% for every application type (see Table 2). This can be treated as a positive answer to RQ regarding the feasibility of automatic analysis of software deployment models. The automatic results have been verified vie the expert 'ground truth'.

The results can be explained following way. The high precision (Table 2) is due to that the SPARQL queries act as deterministic filters, which select primary correct results, if there is a well-structured domain ontology. The exception of two false positive results for Web Applications may be explained by specific view of the expert to some configurations.

Regarding the recall values (Table 2), it can be advocated that false negatives are caused by two reasons. Firstly, there are undetected relations between components. Specification of relations between container in Docker Compose is optional, so the 'DependFlow' and 'LinkFlow' relations are not enough. To recover missed relations various solutions may be added, up to graph link prediction, what is out of scope of this work. Secondly, the reason of false negatives may be a 'narrow' domain ontology. The expert has much more knowledge about application components than the domain ontology represents. So, it can be some trade-off between the quality of the domain ontology and the effectiveness of the analysis.

VI. CONCLUSIONS

The work researches the approach, based on ontologies and knowledge graphs, to automatically determine application functions and structure from software deployment models. Recognition of design primitives, characterizing features of application architecture, using semantic patterns enables farther adding right threat/security patterns to the application design.

Towards answering the research question the ontology-driven framework has been adopted for automatic semantic representation of container applications and learning their architectures. To mitigate the lack of research data in this field, the dataset of 200 semantic diagrams based on multi-container applications (Docker Compose) has been created. The experimental estimation based on created dataset has proven the high precision (up to 100%) and recall equals or more 75% of automatic recognition of application types.

Note, both threat modeling and threat/security patterns automation is currently at initial stage. There are many challenges related to the security analysis automation of software deployment models and managing domain knowledge, in particular, keeping Domain Specific Threat Models, creating machine-readable catalogs of

security/threat patterns, improving methods of pattern contextualization. So, the practical use of the work results depends on factors mentioned above and other ones.

REFERENCES

[1] Murat D., Berkan U., Ali I. An Overview of Secure by Design: Enhancing Systems Security through Systems Security Engineering and Threat Modeling //2024 17th International Conference on Information Security and Cryptology (ISCTürkiye). – IEEE, 2024. – С. 1-6.

[2] Nadifi Z. et al. STRIDE-Based Threat Modeling and Risk Assessment Framework for IoT-enabled Smart Healthcare Systems //International Journal of Online & Biomedical Engineering. – 2025. – Т. 21. – №. 9.

[3] Hammami A. The art of threat modeling //Journal of Computer Sciences and Informatics. – 2024. – Т. 1. – №. 1. – С. 57-57.

[4] Yskout K. et al. Threat modeling: from infancy to maturity //Proceedings of the ACM/IEEE 42nd international conference on software engineering: New ideas and emerging results. – 2020. – С. 9-12.

[5] Erceylan G., Akbarzadeh A., Gkioulos V. Balancing Automation and Human Involvement in Threat Modeling for Optimal Cyber Resilience //International Conference on Human-Computer Interaction. – Cham : Springer Nature Switzerland, 2025. – С. 234-244.

[6] Grosse K. et al. Towards more practical threat models in artificial intelligence security //33rd USENIX Security Symposium (USENIX Security 24). – 2024. – С. 4891-4908.

[7] Steingartner W., Galinec D., Kozina A. Threat defense: Cyber deception approach and education for resilience in hybrid threats model //Symmetry. – 2021. – Т. 13. – №. 4. – С. 597.

[8] Fernandez E. B. et al. Abstract security patterns and the design of secure systems //Cybersecurity. – 2022. – Т. 5. – №. 1. – С. 7.

[9] Cordeiro A., Vasconcelos A., Correia M. A catalog of security patterns //Proceedings of 29th Conference on Pattern Languages of Programs, PLoP. – 2022. – С. 6-8.

[10] Uzunov A. V., Fernandez E. B. An extensible pattern-based library and taxonomy of security threats for distributed systems //Computer Standards & Interfaces. – 2014. – Т. 36. – №. 4. – С. 734-747.

[11] Brazhuk A. Threat modeling of cloud systems with ontological security pattern catalog //International Journal of Open Information Technologies. – 2021. – Т. 9. – №. 5. – С. 36-41.

[12] Qu Z. et al. AdvSQLi: Generating Adversarial SQL Injections against Real-world WAF-as-a-service //IEEE Transactions on Information Forensics and Security. – 2024. – Т. 19. – С. 2623-2638.

[13] Pisu L. et al. HTTP/3 will not Save you from Request Smuggling: A Methodology to Detect HTTP/3 Header (mis) Validations //2024 22nd International Symposium on Network Computing and Applications (NCA). – IEEE, 2024. – С. 97-104.

[14] Ryś A. et al. Model management to support systems engineering workflows using ontology-based knowledge graphs //Journal of Industrial Information Integration. – 2024. – Т. 42. – С. 100720.

[15] Eyvazov F. et al. Beyond containers: orchestrating microservices with minikube, kubernetes, docker, and compose for seamless deployment and scalability //2024 11th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO). – IEEE, 2024. – С. 1-6.

[16] Aung L. H. et al. An Implementation of Web-Based Answer Platform in the Flutter Programming Learning Assistant System Using Docker Compose //Electronics. – 2024. – Т. 13. – №. 24. – С. 4878.

[17] Konev A. et al. A survey on threat-modeling techniques: protected objects and classification of threats //Symmetry. – 2022. – Т. 14. – №. 3. – С. 549.

[18] Usman W., Zappala D. SoK: A framework and guide for human-centered threat modeling in security and privacy research //2025 IEEE Symposium on Security and Privacy (SP). – IEEE, 2025. – С. 2697-2715.

[19] Berger B. J., Plump C. Automatic security-flaw detection-towards a fair evaluation and comparison //Software and Systems Modeling. – 2025. – С. 1-34.

[20] Malakhova D. et al. HarborLang: Enhancing Maritime Operational Safety Through Cyber Threat Simulation and Assessment //International Conference on Business Process Modeling, Development and Support. – Cham : Springer Nature Switzerland, 2025. – С. 290-298.

[21] Rouland Q., Hamid B., Jaskolka J. A model-driven formal methods approach to software architectural security vulnerabilities specification and verification //Journal of Systems and Software. – 2025. – Т. 219. – С. 112219.

[22] Laponina O. R., Kostin R. N. Threat Modeling Software Development for LLM-Agent-Based Systems //International Journal of Open Information Technologies. – 2025. – Т. 13. – №. 6. – С. 132-146.

[23] Venkata R. Y., Kamongi P., Kavi K. An ontology-driven framework for security and resiliency in cyber physical systems //ICSEA. – 2018. – Т. 2018. – С. 23.

[24] Williams I. et al. An automated security concerns recommender based on use case specification ontology //Automated Software Engineering. – 2022. – Т. 29. – №. 2. – С. 42.

[25] De Rosa F. et al. Threma: Ontology-based automated threat modeling for ict infrastructures //IEEE Access. – 2022. – Т. 10. – С. 116514-116526.

[26] Cauli C. et al. Pre-deployment security assessment for cloud services through semantic reasoning //International Conference on Computer Aided Verification. – Cham : Springer International Publishing, 2021. – С. 767-780.

[27] Blanco C. et al. Onto-CARMEN: Ontology-driven approach for Cyber–Physical System Security Requirements meta-modelling and reasoning //Internet of Things. – 2023. – Т. 24. – С. 100989.

[28] Luburić N. et al. A framework for teaching security design analysis using case studies and the hybrid flipped classroom //ACM Transactions on Computing Education (TOCE). – 2019. – Т. 19. – №. 3. – С. 1-19.

[29] Tuma K. et al. Automating the early detection of security design flaws //Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems. – 2020. – С. 332-342.

[30] Pereira-Vale A., Fernandez E. B. An ontology for security patterns //2019 38th International Conference of the Chilean Computer Science Society (SCCC). – IEEE, 2019. – С. 1-8.

[31] Marko N., Vasenev A., Striecks C. Collecting and classifying security and privacy design patterns for connected vehicles: SECREDAS approach //International Conference on Computer Safety, Reliability, and Security. – Cham : Springer International Publishing, 2020. – С. 36-53.

[32] Alvi A. K., Zulkernine M. Security pattern detection through diagonally distributed matrix matching //2022 9th International Conference on Dependable Systems and Their Applications (DSA). – IEEE, 2022. – С. 390-402.

[33] Zeroual M. et al. A Tool Support Methodology for Creating Security Cases Using Argument Patterns //International Conference on Model and Data Engineering. – Cham : Springer Nature Switzerland, 2024. – С. 82-90.

[34] Brazhuk A. I., Olizarovich E. V. Ontological analysis in the problems of container applications threat modelling / Informatika [Informatics], 2023, vol. 20, no. 4, pp. 69–86 (In Russ.)