


# Threat Modeling Knowledge for the Maritime Community

Bernhard J. Berger  
University of Bremen  
Bremen, Germany  
 0000-0001-6093-9229

Christian Maeder  
University of Bremen  
Bremen, Germany  
c.maeder@uni-bremen.de

Salva Daneshgadeh Çakmakçı  
University of Bremen  
Bremen, Germany  
salva@uni-bremen.de

**Abstract**—Architectural risk analysis is a manual technique to identify architectural security flaws that undermine a software system’s security concept. The Architectural Security Tool Suite ArchSec automates this process by applying static analyses to automatically extract architectural security views and employing a knowledge base to automatically detect application-independent architectural security flaws. This paper presents how ArchSec was extended to check existing BPMN diagrams for application-dependent security flaws. Therefore, a model-driven approach is used to generate a knowledge base automatically. This generated knowledge base hosts rules checking an authorisation policy raised for a port community system (PCS). Then, the application-independent and the application-specific knowledge base were applied to BPMN diagrams of the mentioned PCS. The check successfully identified problems within the analysed system. Nevertheless, the generated knowledge base is application-specific, but it is transferable to other software systems interacting with the PCS system, creating a first domain-specific knowledge base for the port community.

**Index Terms**—software architecture, risk analysis, threat modeling, BPMN

## I. INTRODUCTION

Attacks on software systems changed over the last decades. At the beginning of interconnected systems, they were carried out at the network level and tried to attack an operating system’s network stack. With more robust network stacks and firewalls in place, attackers started to use widespread implementation-level bugs, mainly related to missing input validation, such as SQL injections, cross-site scripting, or path traversal attacks. Challenged with these new kinds of attacks, practitioners developed new techniques and software libraries that are not vulnerable to these kinds of attacks. Furthermore, static code analyses have been developed to detect these kinds of vulnerabilities automatically. Consequently, the attacks shifted once more. Whilst the previously mentioned attacks were application-independent, attackers nowadays

focus on application-specific attacks and look for weaknesses in an application’s security concept. To mitigate these kinds of attacks, architectural risk analysis comes into play. Microsoft’s Threat Modeling is a commonly used technique here to identify such architectural security flaws [1].

Microsoft’s Threat Modeling is part of the Secure Development Lifecycle [2] and aims at hardening the software architecture’s security. It is a manual attacker-centric process. During the assessment for each part of the system is discussed how an attacker may attack it. The STRIDE approach guides the assessment and encourages the participants to think about different kinds of attacks, e.g., spoofing, tampering, and information disclosure. Being attacker-centric is necessary since attackers usually think different than regular developers. While Microsoft’s Threat Modeling is a beneficial tool for securing software systems, it also has disadvantages. First, the results of the threat process depend on the people involved in the assessment. Second, the effort for conducting a threat assessment is high. Different sources state that automating the threat process would be very beneficial [3]–[6]. The Architectural Security Tool Suite<sup>1</sup> [7] deals with these aspects to some degree. Therefore, it automatically extracts architectural security views in the form of extended dataflow diagrams and applies different knowledge bases to identify architectural security flaws. Consequently, it reduces the time necessary to create architectural views and makes the result less human-dependent. So far, ARCHSEC’s knowledge base focuses on general security flaws applicable to all kinds of applications. Nevertheless, they do not cover all possible security flaws of an application since there always are domain- or even application-specific flaws. This paper presents an approach to specify an authorisation matrix and automatically generate a knowledge base from the matrix. First, we use this approach to capture a domain-specific authorisation policy for a port community system. Then, ArchSec uses the generated knowledge base to validate a set of given BPMN diagrams successfully.

The German Federal Ministry of Education and Research (BMBF) supported this work under the grant 16KIS0583 (PortSec project) and the German Federal Ministry of Transport and Digital Infrastructure (BMVI) under the grant 19H18012E (SecProPort project).

<sup>1</sup>ARCHSEC is available at <https://www.archsec.de>.

The contributions made by the research presented in this paper are:

- 1) An approach to automatically generate knowledge bases for detecting authorisation-related security flaws.
- 2) An evaluation of the approach using an existing port community system.
- 3) A domain-specific but transferable knowledge base for port community systems.

Section II first describes the background of this work. Afterwards, Section III presents the approach used to specify and generate the knowledge base. Next, Section IV describes the results of the security flaw detection. Continuing, Section V relates this work to other publications. Finally, Section VI discusses the approach and concludes the paper.

## II. BACKGROUND

Microsoft’s Threat Modeling uses dataflow diagrams for modelling a software system’s architecture. The main reason for employing them is their ease of use due to their informal nature, which allows creating them in assessments using a whiteboard. A dataflow diagram is a graph structure consisting of five node types, edges representing dataflows, and trust areas. However, a downside of their informality is a lack of expressiveness which complicates automatic security flaw detection. To overcome this problem, ARCHSEC introduced extended dataflow diagrams, which are very similar to traditional ones. However, while the different node types are pre-defined in traditional diagrams, extended dataflow diagrams support a dynamic schema. The schema allows defining hierarchical types for nodes, edge, data, and trust areas. Additionally, it is possible to define attribute types for the elements mentioned above. Each of the elements allows binding scalar values to attribute types. Furthermore, the element’s types provide means to imply attribute bindings. ARCHSEC’s default schema provides types for different *computers*, *software systems*, *software components*, *interacting users*, *intra-process communication*, and *inter-process communication*. At the same time, attributes for the different protection goals of information security exist, such as *confidentiality*, *reliability*, and *non-repudiation*. Additionally, there are attribute types focusing on security measures, e.g., *encryption*, *authorisation*, and *authentication*. The default schema, for instance, defines the data type *credentials*, which implies the attribute binding from *confidentiality* to the value `true`. A second example is the edge type *HTTPS*, which implies a binding of the *transport encryption* attribute to the value `true`. Consequently, data of type *credentials* can be transferred using an *HTTPS* connection since the data is encrypted.

ARCHSEC has two key features. First, it automatically extracts extended dataflow diagrams, and second, it automatically searches for security flaws [8]. The extraction

mechanism uses different static analyses to extract architectural views of component-based software systems, such as JavaEE-based or Android-based software. Thus, it divides the implementation into several architectural components, detects the interface of these components and the usage of other components. Furthermore, it identifies the use of security libraries and APIs, extracts the employed features, and maps them to the architecture. The extracted extended dataflow diagrams allow the automatic detection of security flaws.

Knowledge bases capture information on architectural security flaws in ARCHSEC. There are three groups of security rules. The first group contains transferable security rules since they deal with aspects like the unprotected submission of sensitive information. While identifying sensitive information is a manual and system-specific task, the security flaw looks the same in all systems. The second group of rules applies to a specific domain since all applications belonging to this domain must adhere to the same rules, such as legal rules. Lastly, the third group contains application-specific rules. ARCHSEC reduces the actual security flaw detection to the subgraph isomorphism problem. Consequently, the threat and mitigations patterns are subgraphs that ARCHSEC looks for. Each of the patterns is described using the graph query language Cypher. Since the graph query language works on property graphs, consisting of typed and attributed nodes and edges, ARCHSEC first lowers the extended dataflow diagrams to property graphs. Afterwards, it locates matches of the patterns. The matches, in turn, form the resulting threat model for the software under investigation.

While this approach is flexible and allows adding new patterns very quickly, it requires the security expert to create Cypher queries by hand, which can be complicated depending on the pattern. Adding custom patterns is necessary for different reasons. First, many security aspects are application-specific, such that they require some security policy. Second, there are even domain- and application-specific security properties that cannot be foreseen.

## III. APPROACH

While ARCHSEC focuses on reverse engineering architectural views from a software’s implementation, the situation was different in the research projects PortSec and SecProPort. Here, architectural views in the form of large business process model notation (BPMN) diagrams were present. One research topic, among others, in these projects, was whether these models adhere to an authorisation policy or if there are policy violations. Figure 1 shows the outline of our approach to check the authorisation policy.

Domain experts created the access control list (ACL) using a custom web tool during the PortSec project. The ACL encodes which subjects of the port community system are allowed to access which information. For example,

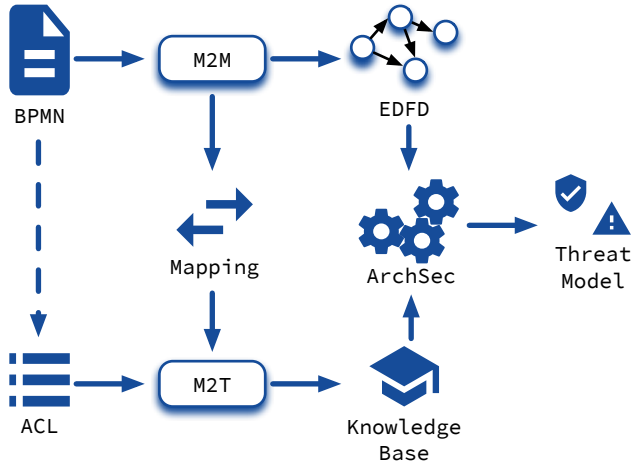


Fig. 1. Transformation and checking process

a subject may be customs or a hauler. Additionally, the passed messages are mapped to assets, which may be sensitive. Therefore, it is possible to automatically extract the subjects and the assets of the ACL from the given BPMN diagrams.

In the first step, a model-to-model transformation converts the given BPMN diagrams into an extended dataflow diagram. The diagrams exist in the form of bizagi Modeler files, which are XML-based model files and therefore computer processable. During this mapping, the transformation encodes different assumptions of the BPMN diagram's structure, such as pools correspond to subjects and swimlanes represent different programs running under the control of the subject. Besides the extended dataflow diagram, this step also produces a mapping of the BPMN diagram elements to the extended dataflow diagram elements.

During the second step, a model-to-text transformation converts the access control list (ACL) to a knowledge base suitable for ARCHSEC. Therefore, it reads the access control list and the mapping generated during the first step. Next, the transformation generates two knowledge base rules for each asset present in the ACL. The first marks any element processing the asset as a possible threat to the system's security, and the second does the same for every channel transporting the asset to another process. Then, a corresponding mitigation rule is generated for both rules that mark the threat as mitigated. For the first rule, it is sufficient that the process is allowed to process the asset, and for the second rule, the target process of the communication channel needs permission to access the asset. Otherwise, the data is sent to a process that is not allowed to access the asset.

In a final step, we refined the generated extended dataflow diagram. In this step, additional knowledge of the domain experts was used to make the diagram more precise, e.g., by changing the channel types to proper types,

such as HTTPS, SSL connection, or e-mail. Unfortunately, this information is not present in the BPMN diagrams and, therefore, it is not possible to extract it automatically.

Listing 1. Query pattern for channel-based threats

```

1 MATCH
2 (d : Data {"uid" : "«unique-data-id»"}),
3 (src : Element {type : subtypeof("Process")})
4 -[c : Channel {type : "inter-process communication",
5   ↪ data : contains(d)} * 1]->
6 (tgt : Element {type : subtypeof("Process")})
7 WHERE
8   src <> tgt
9 RETURN
  d AS host, src AS source, tgt AS target

```

Listing 1 shows a generated query that identifies channels that transport specific data between two processes. Please note that data is the extended dataflow diagram term for assets. The query matches the particular asset by its unique identifier in line 2 of the query. Lines 3 to 5 search for a subgraph of the extended dataflow diagram that consists of two elements of type Process and a connecting channel of length one that transports the data matched in line 2. The concluding where-clause filters out all channels that start and end at the very same process. As a result, ARCHSEC marks all channels that transport the specific asset as a threat.

Listing 2. Mitigation pattern for channel-based threats

```

RETURN
  target.uid IN ["«unique-element-id-1»",
  ↪ "«unique-element-id-2»"] AS mitigated

```

At this point, the mitigation pattern gets crucial since not all transmissions of the asset are a potential threat. Listing 2 shows the used mitigation query. The pattern essentially checks if the target's unique identifier is in the list of unique identifiers allowed to access the information. If the unique identifier is found, the query returns true and ARCHSEC marks the threat as mitigated.

Please note that the given unique identifiers in the shown listings are placeholders for the actual identifiers of the extended dataflow diagram elements. The generated rule and mitigation for processes are similar to the given listings. Therefore, we omit them here. Besides detecting the threats, ARCHSEC generates descriptions of the threats. Table I shows the description template for the shown example.

The shown description template contains different information. First of all, the rule has a name and a description.

TABLE I  
DESCRIPTION TEMPLATE OF THE SHOWN RULE

<b>Rule</b>	Violation of authorisation Policy		
<b>Severity</b>	High	<b>Exploitability</b>	Likely
<b>Categories</b>	Information disclosure Spoofing Tampering		
<b>Description</b>	Process “«source.name»” sends data protected by the authorisation policy to process “«target.name»” using a channel. Since data “«data.name»” is part of the authorisation policy, process “«source.name»” can leak sensitive data. However, the threat might be mitigated if the channel’s target is permitted to access the data, which is checked by a corresponding mitigation rule.		

The description is a template that ARCHSEC expands. The expressions between an opening and a closing guillemet are evaluated using the actual match in the extended dataflow diagram. Furthermore, the description template assigns a severity and an exploitability category to the threat. The severity of these threats is high, and it is very likely that the threat is exploited since the data are transmitted by the application’s design. The threatened protection goals are information disclosure, spoofing, and tampering.

#### IV. FINDINGS

We used the presented approach to check a business process model notation diagram that matched the assessed authorisation policy. We successfully transformed the diagrams and the policy could into an extended dataflow diagram and a knowledge base. ARCHSEC was then successfully able to check the knowledge base within several seconds. The result showed that there were some minor problems, which turned out to be problems with the authorisation policy. The domain experts did not have all aspects in mind while creating the policy. After refining the policy accordingly, there were no policy violations left threatening the system’s security.

Nevertheless, it turned out that ArchSec’s built-in knowledge base could identify threats based on the refined extended dataflow diagrams. It correctly identified communication channels transmitting sensitive information which did not employ channel or message encryption to protect the transmitted data. In particular, customs sent some export-related information to the port community systems using e-mails. These e-mails are neither encrypted nor digitally signed. Consequently, an attacker can read, discard, or manipulate the sent information.

#### V. RELATED WORK

There are different publications on automating threat modelling. Abi-Antoun et al. use static analysis to extract object-ownership graphs, which can be compared to dataflow diagrams. They use these graphs to detect information flows within software systems that violate a given authorisation policy [9]–[11]. However, their approach is not as flexible as the presented approach since it requires the developers to annotate the source code with additional information to provide hints for analysing which

trust areas exist and to which trust area a created object belongs. Furthermore, the approach is not easily extensible to new rules and to sources different from source code.

Antonino et al. presented an early research prototype for indicator-based architecture-level security evaluation for service-oriented architectures, called SiSOA, in 2010. Their approach consists of three steps. First, the extraction phase uses reverse-engineering techniques to create a system model that comprises static class-level information from the source code and configuration information. Second, the identification phase employs a knowledge base to tag the system model. The tag rules can either add abstract information on the program, such as identified SOA components, or security information. Finally, a severity and a credibility value are calculated and assigned to the tag added. These metrics quantify the quality and correctness of the placed tags. The calculated tags are then used to check the conformance to manually defined security goals, such as message integrity [12].

Lastly, Tuma et al. automate the detection process of threats in dataflow diagrams, as well. Their early papers focused on a guided way of manually creating threat models and reducing the size of the resulting threat model. Therefore, the authors used a guide to make the threat modelling process easier for non-security experts. Later, they started to automate the detection of some threats, similar to the way ARCHSEC does. Instead of using graph queries, they use VIATRA queries. However, their work does not focus on authorisation-related security problems [13], [14].

#### VI. CONCLUSIONS

The findings for the analysed port community system showed that the system architecture conforms to the defined authorisation policy. While this means that the analysis could not detect authorisation policy-related vulnerabilities, it was able to validate the re-documented authorisation policy successfully. Thus, it verified that only permitted participants could access sensitive information within the port community system. Nevertheless, the detection of threats using the built-in knowledge base shows the usefulness of the automated threat modelling approach.

Additionally, the generated knowledge base provides a more significant benefit. I stores architectural security knowledge for the maritime community and can be reused to analyse other system parts. It is even possible to transfer the knowledge to other port community systems since the communication with customs is the same for all port community systems. Furthermore, it is possible to use the knowledge base when checking concrete applications. Consequently, it is necessary to extract a dataflow diagram for the application and identify the processed and transmitted sensitive information. Extracting dataflow diagrams from an application's source code allows verifying if the implementation, which might differ from the planned architecture, conforms to the rules.

#### REFERENCES

- [1] F. Swiderski and W. Snyder, *Threat Modeling*. USA: Microsoft Press, 2004.
- [2] M. Howard and S. Lipner, *The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software*. Microsoft Press, May 2006.
- [3] S. Rehman and K. Mustafa, "Research on Software Design Level Security Vulnerabilities," *SIGSOFT Softw. Eng. Notes*, vol. 34, no. 6, pp. 1–5, Dec. 2009. [Online]. Available: <https://doi.org/10.1145/1640162.1640171>
- [4] M. Almorsy, J. Grundy, and A. S. Ibrahim, "Automated Software Architecture Security Risk Analysis Using Formalized Signatures," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. IEEE Press, 2013, pp. 662–671. [Online]. Available: <https://doi.org/10.1109/ICSE.2013.6606612>
- [5] E. Khalaj, R. Vanciu, and M. Abi-Antoun, "Is There Value in Reasoning about Security at the Architectural Level: A Comparative Evaluation," in *Proceedings of the 2014 Symposium and Bootcamp on the Science of Security*, ser. HotSoS '14. New York, NY, USA: Association for Computing Machinery, 2014. [Online]. Available: <https://doi.org/10.1145/2600176.2600206>
- [6] A. Shostack, *Threat Modeling: Designing for Security*, 1st ed. Wiley Publishing, 2014.
- [7] B. J. Berger, K. Sohr, and R. Koschke, "The Architectural Security Tool Suite — ARCHSEC," in *19th International Working Conference on Source Code Analysis and Manipulation, SCAM 2019, Cleveland, OH, USA, September 30 - October 1, 2019*. IEEE, 2019, pp. 250–255. [Online]. Available: <https://doi.org/10.1109/SCAM.2019.00035>
- [8] —, "Automatically Extracting Threats from Extended Data Flow Diagrams," in *Engineering Secure Software and Systems - 8th International Symposium, ESSoS 2016, London, UK, April 6-8, 2016. Proceedings*, ser. Lecture Notes in Computer Science, J. Caballero, E. Bodden, and E. Athanasopoulos, Eds., vol. 9639. Springer, 2016, pp. 56–71. [Online]. Available: [https://doi.org/10.1007/978-3-319-30806-7\\_4](https://doi.org/10.1007/978-3-319-30806-7_4)
- [9] M. Abi-Antoun and J. M. Barnes, "Analyzing Security Architectures," in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '10. New York, NY, USA: Association for Computing Machinery, 2010, pp. 3–12. [Online]. Available: <https://doi.org/10.1145/1858996.1859001>
- [10] R. Vanciu and M. Abi-Antoun, "Finding Architectural Flaws in Android Apps is Easy," in *Proceedings of the 2013 Companion Publication for Conference on Systems, Programming, & Applications: Software for Humanity*, ser. SPLASH '13. New York, NY, USA: Association for Computing Machinery, 2013, pp. 21–22. [Online]. Available: <https://doi.org/10.1145/2508075.2514574>
- [11] —, "Finding Architectural Flaws Using Constraints," in *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE'13. IEEE Press, 2013, pp. 334–344. [Online]. Available: <https://doi.org/10.1109/ASE.2013.6693092>
- [12] P. Antonino, S. Duszynski, C. Jung, and M. Rudolph, "Indicator-Based Architecture-Level Security Evaluation in a Service-Oriented Environment," in *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, ser. ECSA '10. New York, NY, USA: Association for Computing Machinery, 2010, pp. 221–228. [Online]. Available: <https://doi.org/10.1145/1842752.1842795>
- [13] K. Tuma, D. Hosseini, K. Malamas, and R. Scandariato, "Inspection Guidelines to Identify Security Design Flaws," in *Proceedings of the 13th European Conference on Software Architecture - Volume 2*, ser. ECSA '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 116–122. [Online]. Available: <https://doi.org/10.1145/3344948.3344995>
- [14] L. Sion, K. Tuma, R. Scandariato, K. Yskout, and W. Joosen, "Towards Automated Security Design Flaw Detection," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW)*, 2019, pp. 49–56. [Online]. Available: <https://doi.org/10.1109/ASEW.2019.00028>