Data Leakage and Validation Bypass in SHACL

Davan Chiem Dao[1,*], Christophe Debruyne[1]

[1]*Montefiore Institute of Electrical Engineering and Computer Science, University of Liège, Liège, Belgium*

Abstract

This paper describes security vulnerabilities when using SHACL for data validation in knowledge graphs (KGs) and enhance security awareness in SHACL and KG technologies. We employ a KG developed in a prior study in the social security domain to illustrate potential security issues. The vulnerability identified is SHACL's inability to distinguish user input from historical information in the KG, resulting in two potential exploits: validation bypass and data leakage. This study improves our understanding of security risks in SHACL and KG technologies, providing insights in this underexplored area in literature.

Keywords

Knowledge Graphs, SHACL, Security

1. Introduction

Knowledge Graphs (KG) have emerged as versatile tools for representing and organizing complex relationships between entities in various domains [1]. As applications utilizing KGs continue to proliferate, ensuring the accuracy and reliability of its data becomes increasingly important. One technique is the Shapes Constraint Language (SHACL), which provides a standardized way to validate RDF graphs.

Despite the increase in popularity of KGs, there is limited research on related security issues. In 2023, [2] stated that security risks of KG (reasoning) remain largely unexplored and provided one of the first systematic studies on these risks. However, this study and others such as [3, 4] present security issues that are mostly related to machine learning on top of KGs rather than problems pertaining to KG technologies.

Motivated by this gap in the literature, this paper presents a case study to uncover security vulnerabilities inherent to KG, focusing on SHACL utilization for data validation. SHACL is often used as a simple data validation tool, either internally or for users to check the structure of their documents. However, it is flexible enough to be considered in more complex contexts where multiple stakeholders might interact with the KG. We employ a KG developed in a prior study to validate social security declarations and illustrate potential security issues, showcasing how this flexibility, while beneficial, also introduces risks.

SEMANTiCS 2023: 20th International Conference on Semantic Systems, September 17–19, 2024, Amsterdam, The Netherlands

*Corresponding author.

0009-0004-8139-1927 (D. Chiem Dao); 0000-0003-4734-3847 (C. Debruyne)

2. SHACL

The Shapes Constraint Language (SHACL) [5] is a standardized language for expressing conditions that RDF graphs must adhere to, commonly referred to as “shapes.” The set of shapes forms an RDF graph called a “shapes graph.” On the other hand, the RDF graph undergoing validation against these shapes is called a “data graph,” and the RDF graph reporting the result is called a “validation report.”

The validation process works as follows. For each shape, the focus nodes are selected based on their target among the nodes of the data graph. Then, these focus nodes are validated against the shape’s constraints. Depending on the validation results, the validation report indicates either that the data graph conforms when all nodes conform or that the graph does not conform if at least one node fails. In the latter case, the cause of the failing node is reported.

The types of constraints that can be expressed include value type constraints, cardinality constraints, value range constraints and logical operators, among others. These constraints, along with the general workings of the validation, form SHACL-CORE. In addition to SHACL-CORE, SHACL-SPARQL extends its capabilities by enabling the creation of custom constraints using SPARQL queries. This component allows one to define more complex validation rules that may not be expressible using standard SHACL constraints alone.

3. Social Security Artifact

In a prior study [6], a validation system using KG technologies was built to validate social security declarations stored as XML. This study demonstrated the feasibility of SHACL in expressing complex domain rules where current practice can only validate the syntax and structure of these XML documents with an XSD schema. The built system allows users to have a more complete and transparent validation of their declarations (i.e., user input) as SHACL is more expressive than XSD[1]. Moreover, the shapes can be shared with the public and used in several applications.

The constraints in the shapes graph were developed by translating constraints from an XSD schema and a well-documented glossary into SHACL. The kind of constraints that were expressed include simple constraints expressed with SHACL-CORE constraints such as datatype, value range, cardinality constraints, regular expressions, and complex constraints expressed with SHACL-SPARQL such as checksums, value part of an enumeration, and value unicity[2]. Listing 1 shows an example of a shape. This shape constrains a field to be a two-digit integer (lines 5), occur at most once (end of line 4), and be part of an enumeration (lines 6-11).

For the enumeration constraint, the user input must be complemented with additional data for the data graph, e.g., to check whether the provided values are valid w.r.t. rules documented elsewhere. We will refer to these data as *public data* as it concerns information that is publicly available in the glossary’s annexes. It describes the values allowed for some specific fields and their meaning. For instance, the position code in Listing 1[3] specifies the code for a person’s job:

[1]e.g., SHACL allows arithmetic computation, rules between different graphs and recursion
[2]Value unicity refers to ensuring that each value following a specific path in the KG does not occur more than once.
[3]More detailed examples are available at https://github.com/Ikeragnell/shaclExploits

```
ont:OccupationShape a sh:NodeShape;
    sh:targetClass ont:Occupation;
    sh:property [
        sh:path ont:PositionCode; sh:maxCount 1
        sh:minLength 2; sh:maxLength 2; sh:minInclusive 0; sh:maxInclusive 99; sh:datatype xs:integer;];
    sh:sparql [
        sh:prefixes <> ;
        sh:select """SELECT $this ?value WHERE {
                        $this $PATH ?value.
                        OPTIONAL{?p a an9:PositionCode ; an9:Code ?value.}
                        FILTER(!BOUND(?p))}"""] .
```

Listing 1: Shape example requiring public data

e.g., 58–barman, 61–hunter, 62–valet. As not all values are allowed and allowed values can vary over time, the public data contains an enumeration of the currently allowed values.

Similarly, some rules may require additional data to be included in the data graph, but that should be *private*. This information concerns user data relevant to social security, such as the number of working hours, amount of contributions, and amount of social rights. For example, Listing 2 states that one cannot be employed and have unemployment benefits simultaneously.

```
ont:NaturalPersonShape a sh:NodeShape ;
    sh:targetClass ont:NaturalPerson ;
    sh:sparql [
        sh:prefixes <> ;
        sh:select """SELECT $this WHERE {
                $this ont:INSS ?nbr; ont:R_90017_90012 ?workerRecord.
                ?workerRecord a ont:WorkerRecord.
                ?person ont:INSS ?nbr; ?person ont:R_90017_901234 ?unemploymentBenefit.
                ?unemploymentBenefit a ont:UnemploymentBenefit.}"""] .
```

Listing 2: Shape example requiring private data

4. Exploits

In the previous section, we explained that the data graph included not only the transformed user input but also public and private data. Despite these components being distinguished before they are provided to the SHACL processor, it is critical to note that the SHACL processor treats the data graph as a whole. This aspect becomes a potential source of vulnerability, as SHACL cannot differentiate user input from historical and trusted information.

This lack of distinction between user input and historical data potentially enables attackers to manipulate the validation process or extract sensitive information from the KG. The following sections present two exploits leveraging this vulnerability: validation bypass and data leakage.

While some access control mechanisms might help mitigate these risks, current access control solutions for KGs are not yet mature [7] and, to the best of our knowledge, there are no thorough implementations yet. Moreover one might argue that it is the responsibility of KG engineers to manage these risks by implementing their own solution on top of SHACL. However, by identifying these issues at the SHACL level, we can envisage solutions that are more standardized and applicable across various projects.

4.1. Validation Bypass

Validation bypass occurs when a declaration that should be erroneous is reported as conforming to the shapes. This exploit capitalizes on SHACL's inability to differentiate between user-provided data and historical information within the KG. Essentially, attackers add additional triples that do not concern the declaration being evaluated[4]. This additional data pretends to be trusted data and masks the underlying inaccuracies in the provided data.

To illustrate this exploit, we will now describe how to bypass the shape in Listing 1, specifically the constraint of being part of an enumeration. For example, the value 31 is not part of the enumeration allowed for a position code. In normal circumstances, when a declaration containing this value is validated, the validation report generated states the node does not conform as it has a position code that does not exist.

An attacker can make value 31 valid by creating a position code with this value and adding it to the original declaration. Listing 3 shows the poisonous triples to be added. This kind of data should only be in the public data. However, as there is no distinction between the user input and the public data in the data graph, the SHACL validation process considered all asserted data as true.

In this case, the vulnerable constraint is a SPARQL constraint. SPARQL constraint component can utilize any paths to express complex rules even those beyond closed SHACL shapes. Thus, closing shape does not solve the problem. Moreover, this SPARQL constraint can be expressed with SHACL-CORE which shows that this exploit does not depend on SHACL-SPARQL.

```
<http://maliciousInput.be/PositionCode31> a an9:PositionCode ; an9:Code 31.
```

Listing 3: Poisonous data

4.2. Data Leak

Data leakage occurs when sensitive information stored within the KG is exposed during validation. Users only interact with the SHACL processor and do not have direct access to private data. However, some rules require private data to be included in the data graph. Thus, an attacker could forge declarations and deduce sensitive data based on their validity.

To illustrate this exploit, we will now describe how the shape in Listing 2 can leak some sensitive information. Suppose that the data in Listing 4 is some private data, this data states that someone with the social security number 77101500172 is unemployed. We will refer to this person as Bob. In normal circumstances, no employer would declare Bob as an employee.

```
ont:UnempPers0 ont:INSS 77101500172.
ont:UnempPers0 ont:R_90017_901234 ont:UnempBen0.
ont:UnempBen0 a ont:UnemploymentBenefit.
```

```
ont:NatPers0 a ont:NaturalPerson.
ont:NatPers0 ont:INSS 77101500172.
ont:NatPers0 ont:R_90017_90012 ont:WorkRec0.
ont:WorkRec0 a ont:WorkerRecord.
```

Listing 4: Private data (left) and forged declaration (right)

[4]Note that poisoned data should not concern the declaration; otherwise, only fraud would occur instead of an exploit. For instance, if a declaration is erroneous due to a missing field, adding some fake data can make it pass the validation. This would be equivalent to fraudulently filling out a declaration.

An attacker can forge a declaration stating that he employs Bob. The relevant data from the forged declaration can be seen in Listing 4. The attacker can determine whether Bob is employed depending on the validation result. Bob is unemployed if the validation report states that the declaration does not conform (which is the case with this forged declaration). On the other hand, if the declaration conforms, then Bob is employed.

5. Conclusion

This paper marks a first step in understanding the vulnerabilities present in SHACL by describing exploits such as validation bypass and data leakage. We highlight the risks stemming from SHACL's inability to differentiate triples coming from different graphs. Moving forward, we will develop and implement effective strategies to address these security concerns. In the social security domain, where personal or sensitive data is used, convoluted approaches can be conceived in which validation reports are not communicated to users or where humans are kept in the loop in the validation process. Such solutions delegate all responsibilities to the applications built on the KG. Ideally, approaches are embedded into and part of the KG. Our goal is to address these declaratively using KG technologies. One potential venue is to use SHACL extensions[5]. This study is important to understand the vulnerabilities inherent in KGs. Through these endeavors, we aim to enhance the overall understanding of security issues of KG technologies and fortify them against potential exploits.

References

[1] S. Tiwari, F. N. Al-Aswadi, D. Gaurav, Recent trends in knowledge graphs: theory and practice, Soft Computing 25 (2021) 8337–8355.

[2] Z. Xi, T. Du, C. Li, R. Pang, S. Ji, X. Luo, X. Xiao, F. Ma, T. Wang, On the security risks of knowledge graph reasoning, 2023. arXiv:2305.02383.

[3] P. Bhardwaj, J. Kelleher, L. Costabello, D. O'Sullivan, Adversarial attacks on knowledge graph embeddings via instance attribution methods, in: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 2021, pp. 8225–8239.

[4] H. Zhang, T. Zheng, J. Gao, C. Miao, L. Su, Y. Li, K. Ren, Data poisoning attack against knowledge graph embedding, 2019. arXiv:1904.12052.

[5] Shapes constraint language (SHACL), 2017. URL: https://www.w3.org/TR/shacl/.

[6] D. Chiem Dao, C. Debruyne, P. Stijfhals, Using knowledge graphs and shacl to validate declaration forms: An experiment in the social security domain to assess shacl's applicability, in: 2nd EuropeaN Data conference On Reference data and SEmantics ENDORSE 2023, 14-16 March 2023 - Proceedings, Publications Office of the European Union, 2023, pp. 85–96.

[7] M. Valzelli, A. Maurino, M. Palmonari, B. Spahiu, Towards an access control model for knowledge graphs (discussion paper), in: Proceedings of the 29th Italian Symposium on Advanced Database Systems, SEBD 2021, Pizzo Calabro (VV), Italy, September 5-9, 2021, volume 2994 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, pp. 339–346.

[5]For instance, https://afs.github.io/shacl-datasets.html