Computing and Informatics, Vol. 42, 2023, 480–500, doi: 10.31577/cai_2023_2_480

# ONTOLOGY FOR BLIND SQL INJECTION

Jean Rosemond Dora, Ladislav Hluchý

*Institute of Informatics*
*Slovak Academy of Sciences*
*Dúbravská cesta 9*
*845 07 Bratislava, Slovakia*
*e-mail:* {jeanrosemond.dora, Ladislav.Hluchy}@savba.sk

Karol Nemoga

*Institute of Mathematics*
*Slovak Academy of Sciences*
*Štefánikova 49*
*811 04 Bratislava, Slovakia*
*e-mail:* nemoga@mat.savba.sk

**Abstract.** In cyberspace, there exists a prevalent problem that heavily occurs to web application databases and that is the exploitation of websites by using SQL injection attacks. This kind of attack becomes more difficult when it comes to blind SQL vulnerabilities. In this paper, we will first make use of this vulnerability, and subsequently, we will build an ontology (OBSQL) to address the detection of the blind SQL weakness. Therefore, to achieve the exploitation, we reproduce the attacks against a website in production mode. We primarily detect the presence of the vulnerability, after we use our tools to abuse it. Last but not least, we prove the importance of applying ontology in cybersecurity for this matter. The mitigation techniques in our ontology will be addressed in our future work.

**Keywords:** SQL injection, blind SQL, vulnerability, weakness, ontology, semantic web, information security, cyber threats, website security, web application vulnerabilities, attack detection

## 1 INTRODUCTION

The significance of the website at present, and its constant use, make it a niche for evildoers to obtain confidential data of users. According to recent research from `https://www.verizon.com/business/en-gb/resources/reports/dbir/` (2022), web application attacks are involved in 26 % of all breaches, making the second most common attack pattern. Knowing this information is extremely crucial for both the attackers and the IT security engineers, as it involves security concerns.

The detection of the blind SQL injection is a high-level scenario. It involves advanced tools and techniques to be used by the attacker. First and foremost, a web application developer does not mean to be a cyber security analyst; therefore, it may be hard for him to detect vulnerabilities. Moreover, during the programming phase of the web application, the may use some common techniques in PHP, Java, Python, etc. to mitigate SQL vulnerability. However, detecting the blind SQL injection will require a deep analysis of the codes and the use of penetration testing tools. Manually, it is very time-consuming to detect the vulnerability and attack its corresponding database. One of the reasons is that the injection payloads that should be used differ from a database to a database. For example, we need at least two (2) different payloads to abuse SQLite3 and MySQL databases.

Regarding this web application vulnerability, an important factor that requires great attention is when it accepts user input. For example, if it accepts users to be registered, submit, log in, comment, etc. Publishing the website without submitting it to a penetration testing phase, will be presented to attackers like a doorless house for thieves. Additionally, it is also vital to expand the awareness of the need to comprehensively evaluate what kinds of information the web application will reveal as output when a request is made, and which mechanisms we use to break down the request to keep up the whole security framework high.

To approach this idea, we resort to the methodology that necessitates knowing the followings:

- what kind of response,
- who utilizes,
- for what purposes,
- if this, then that.

Hence, a question may be risen: "How can we use semantic languages, (semantic axioms and rules) to help us understand the structure of a possible vulnerability?" Thus, comes the importance of ontology.

Briefly, an ontology is a well-structured diagram consisting of a tree of classes (sub-classes) or simply classes inheritance, attributes, and relationships. The construction of an ontology relies on the establishment of rules.

The rest of this paper is then organized as follows: Section 2 provides the definition and impact of the Blind SQL injection. It also summarizes some related works. Sections 3 and 4 embrace the attack scenario, the detection, and the exploitation of the vulnerability. Section 5 provides information on the usability of ontology in cybersecurity, the semantic web, axioms, and rules implementation. Section 6 provides our future work and concludes the paper.

Note: *For privacy reasons, we are obliged to hide some confidential data in the figures (public address and more), since it is a live website.*

## 2 DESCRIPTION AND IMPACT OF BLIND SQL INJECTION

Here, we will describe SQL injection from two angles:

1) **SQL vulnerabilities** – It is when a web page suffers from SQL weakness. This kind of vulnerability cannot be detected by simply reading the website source code only, most of the time it has to go through a testing phase.

   SQL injection is a code injection method that may destroy the database of a web application if wrongly used. Imagine a scenario where a bank stores all its clients' data in a specific database. Having access to that database can help the attacker find relevant and sensitive information, such as clients' names, hashed or plaintext passwords, telephone numbers, home addresses, signed contracts, etc. We also have to note that such companies have a lot of databases on their systems. Thus, manually exploiting this vulnerability can be problematic, as the attacker may have no clues about the name of the database management system (DBMS), databases name, tables name, and columns name. Utilizing automatic tools is more appropriate for this particular detection and exploitation of the vulnerability. It helps us reduce the time of the attack since all the requests will produce latency of the target server.

   Usually, this attack works by inserting malicious code in an SQL statement. Wherever there are some parameters, then it is possible to inject any payload for detection purposes. SQL injection is one of the most common web application vulnerabilities on the OWASP checklist.

2) **SQL attacks** – commonly known as SQL injection, it is when the SQL vulnerability of a web page is being exploited by an attacker, or by a penetration tester.

   Therefore, blind SQL injection arises when a web application is vulnerable to SQL injection, but its HTTP responses do not include the results of the relevant SQL query or the information of any database errors.

   With this type of vulnerability, many techniques such as UNION attacks are not efficient, since they rely on being able to see the results of the injected query within the website's responses.

However, it is still possible to exploit the blind SQL injection to access unauthorized data, but different techniques must be applied. From a boolean operation case, it asks the database true or false questions and determines the answer based on the response of the application. This attack scenario is often used when the website is configured to show generic error messages but has not diminished the code that is vulnerable to SQL injection.

From a latency viewpoint, the payload request aims to slow down the time the server takes to respond to the query. That being said, an attacker can double-check how the server reacts in time when a payload is injected.

The impact of exploiting the SQL vulnerabilities is greatly significant since the attacker can steal confidential data from the database of the web application (username, table name, user passwords, etc.). For more information, please the following related works: [1, 2, 3, 4, 5, 6, 7, 8].

In the following section of the practical part (Detection & Exploitation), we had to force the server response to slow down to detect the presence of the blind SQL vulnerability from the target website.

## 3 DETECTION OF BLIND SQL VULNERABILITIES

Detecting the blind SQL weakness from a website can be difficult using the manual inputs (payloads) method to a query field. Sometimes, it requires to the attacker hundreds of payloads to inject into the user-input field of the web page. Saying so, attempting to inject commands one by one by an individual is drastically not a good practice. Therefore, hackers or penetration testers usually resort to manual tools or some automated tools to achieve this goal.

By injecting some characters into the user input of the web page, we found a table name "X". And the text on the page provides more information about another table name "Y" and its column "Y1". However, some of this data was dummy, fake. We had to find a way somehow to abuse what we have obtained from the response. From the following source (*please see* `https://hackersonlineclub.com/sql-injection-cheatsheet/`), We realized that the present blind SQL is a "Generic Time-Based SQL Injection" by invoking the ASCII char. When we used 500000000/1 (please see below), the server response comes up automatically. But using /2, it takes 2 seconds to pop up.

Next, by modifying the payload, we forced the server to give us a response at the time of our choice.

We have seen clearly how we were able to detect the presence of this vulnerability. The next section will demonstrate how we were able to exploit it.

## 4 EXPLOITATION OF BLIND SQL VULNERABILITIES

The attempt to exploit a system is usually the subsequent action after detecting the presence of a vulnerability. To proceed with the exploitation, we first double-

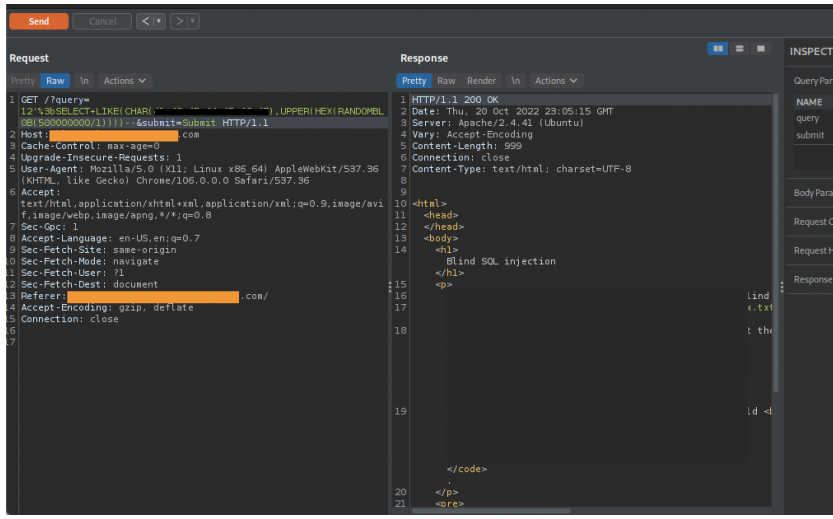



Figure 1. No latency has occurred from the server response

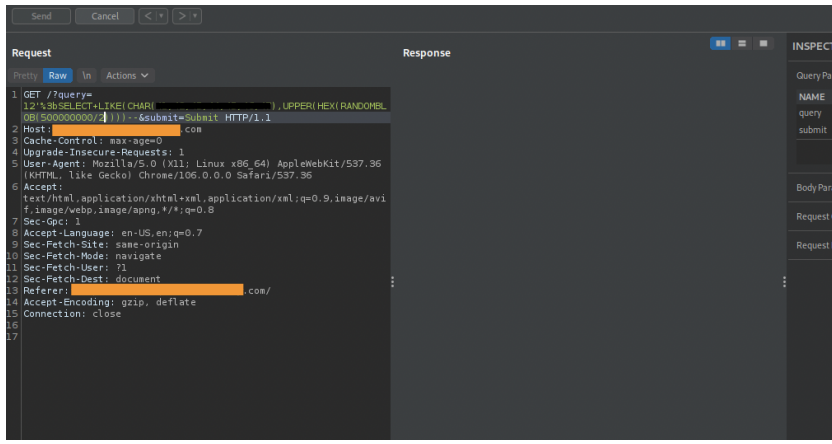



Figure 2. 2s of latency has been occurred from the server response

checked if the target web application is behind a firewall. We also made sure that we had full right to do anything we want to exploit the vulnerability from that target website. As we can see below, the command we used is very aggressive and dangerous as the "–risk 3" and "–level 4" might disrupt or erase the target database.

We realized that the target was behind a firewall, so we had to find a way to bypass this type of protection. To do so, we used "–tamper=space2comment" command.


Figure 3. Bypassing firewall

When everything is set up properly (your full right to attack the target, your tools), then it is time to launch the attack. The attack vigorously sends multiple requests to the website server trying to obtain the exact database management system (DBMS) it uses. As we can see in the following figure, we were able to extract the database name, the tables name, and the rows and columns.

For demonstration purposes, we have chosen our target website for which we already know the number of its databases and which do not contain hundreds of tables. It helps avoid time-consuming attacks.

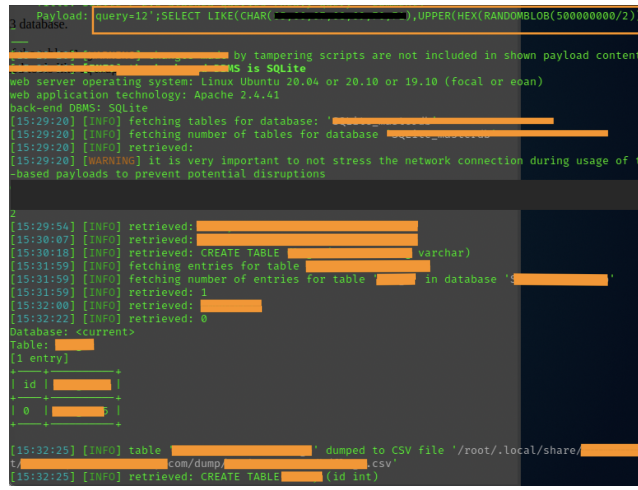



Figure 4. Successful blind SQL attack of the target website database

We have seen how we successfully hacked into a web application and gain information from its database system. Now, let us use the ontology approach to see how it can help when it comes to cybersecurity.

## 5 NOVEL APPROACH FOR THE DETECTION OF SQL INJECTION ATTACKS (ONTOLOGY)

From the previous sections, we have demonstrated a few examples of how SQL injection (blind) weaknesses can be detected by an attacker (or any individual, penTester for example). We have also seen how he can make use of those vulnerabilities by injecting some payloads to jeopardize the target system. Therefore, is extremely important thus imperative to fight against the adversary by implementing

significant methodologies (approaches and steps) to strengthen security and mitigate the attacks. The term ontology approach is a powerful mechanism which we can start with. For more information about other ontological approaches, please see [9, 10].

Usually, the word ontology can be defined as a formal and explicit specification of a set of concepts in a specific field of interest. The clear specification of those concepts is usually presented in a shape of a well-structured scheme composed of classes inheritance and sub-classes, relationships, and attributes.

### 5.1 Ontology and Semantic Web

Ontology can be designed to facilitate data to be shared and reused across multiple applications, institutions, organizations, and so on. Based on the field of interest, security experts can use ontology to enhance their systems. In medicine, for instance, IT security engineers can use ontology for pregnancy, covid-19, diabetes, Alzheimer's etc. Please see [11, 12] for some related works.

To apply the concept of ontology in a field, some components should be put into question. The typical ontology components are:

**Categories:** concepts, i.e., types of objects;

**Individuals:** situations or things (in this case, individuals are also known as "first-order objects");

**Relationships:** ways in which individuals and groups can communicate;

**Limitations (Constraints):** The formal and steady description of what must be true until some inputs are accepted;

**Features:** classes, properties, aspects, parameters, or instances that objects and categories can contain;

**Axioms:** assertions, or statements in a logical and understandable form that form together with the perceivable theory that is illustrated and demonstrated by the ontology in their domains.

Before we dive into the construction of our ontological approach, let us first define its importance.

#### 5.1.1 The Reasons of Implementing Ontological Approach in Cyberspace

Ontology is an exciting approach for linking up the description of a data model and the related rules into one application. Ontologies developed in Web Ontology Language (OWL) acquire many benefits afforded by the semantic web stack. The goal of OWL is to represent complex knowledge of entities in a domain through a logic-based language, via a computational, such that the knowledge encapsulated can be ascertained for consistency or utilized as a basis for inferences on that specific knowledge.

- To share a comprehensive structure of data, and information between people. The ontology also allows the reuse of domain knowledge.
- To split domain knowledge from operational knowledge.
- To make domain assumptions obvious.
- To carefully analyze domain knowledge.

  It is good practice to install and configure or use proactive detection tools. Many web-based detection tools are reactive, i.e., they function according to the specific rules set by the administrator.

  The attack can only be prevented if the exact signature of the attack is not only recognized by the scanning tools but also present.

  – It is easy for a malicious entity to launch an attack altering the signature since the majority of the existing techniques are signature-based, which hold the syntax of the attack.
  – Additionally, statistical mechanisms used in Intrusion Detection Systems (IDS) largely provide an attainable solution for the network layer. However, this solution is not efficient at the application layer since it focuses on the character distribution of the input and does not take into account its contextual nature.

### 5.1.2 Ontology Model – Communication Protocols

The communication protocols, as its name says, allow the transfer of messages from one point to another. It is shaped as semantic networks. The essential part of this activity relies on the "Protocol" concept, which can be classified as the main class of the following sub-classes *FTP*, *SMTP*, *HTTPS*, *HTTP*. This classification subsequently involves three (3) other concepts: *Message*, *Request*, *Response*.

One of the finest benefits of the ontology approach is that it comes up with inference potentiality and the required constructs that enable software systems to reason over the knowledge base.

The following example will produce a response latency of two (2) seconds from the web server response in the attacker's environment, (taken from Figure 2):

*query=12';SELECT LIKE(CHAR(22,23,...,28,29),UPPER(HEX(RANDOM BLOB(500000000/2))))–*

To illustrate the inference activity and flexibility in semantic rules, the query string carries the detection payloads which forces the web server to respect its request. Instead of inserting a single parameter in the user-input field, (12 for example), we added a SELECT + LIKE of a RANDOMBLOB command there to experience the latency.

The referrer is in line 13, the request does not involve any cookies. All the other lines from the "Request" tab are irrelevant to us. The inference of the ontology

yields all the numerous activities using a general semantic rule. Generally, the rules give a focal point if the malicious payload infects the parameter values. Additionally, the rules describe the inference structure through transitive features.

### 5.2 Implementation of Rules

By applying the semantic concept, we can use deductive inference rules to reason on a piece of HTTP well-constructed diagram.

Let us describe to which class hierarchy each method and protocol are belonging:

- GET $\sqsubset$ Method,
- POST $\sqsubset$ Method,
- HTTP $\sqsubset$ Protocol,
- SQL injection attacks $\sqsubset$ Attack,
- Request Header $\sqcap$ Response Header $\equiv \perp$,
- POST $\sqcap$ GET $\equiv \perp$.

In our proposed approach, all subsume ($\sqsubset$) relations are *transitive, irreflexive* and *asymmetric*. But the equivalence ($\equiv$) relations are *reflexive, symmetric* and *transitive*. Likewise, no conceptually disjoint ($\sqcap$) relations contravene its properties of *symmetric, reflexive* and *transitive*. We established these rules based on how we were able to detect the SQL injection vulnerabilities, then applied them to our ontology.

**Rule 1:** *Person(?P)* $\sqcap$ *hasTools(?P, ?Q* $\rightarrow$ *Attacker(?P)*(Transitivity),

**Rule 2:** *SubClassOf(?P, ?Q)* $\sqcap$ *typeOf(?n,?P)* $\rightarrow$ *typeOf(?n, ?Q)*(Transitivity),

**Rule 3:** *hasPartOf(?P, ?Q)* $\sqcap$ *hasPartOf(?Q, ?n)* $\rightarrow$ *hasPartOf(?P, ?n)*(Transitivity),

**Rule 4:** *contains(?P, ?Q)* $\sqcap$ *contains(?Q, ?n)* $\rightarrow$ *contains(?P, ?n)*(Transitivity).

From rules 3 and 4, the $5^{\text{th}}$ becomes:

**Rule 5:** *hasPartOf(?P, ?Q)* $\sqcap$ *contains(?Q, ?n)* $\rightarrow$ *contains(?P, ?n)*(Transitivity),

**Rule 6:** *Attacker(?P)* $\sqcap$ *hasInput(?P, ?Q)* $\sqcap$ *hasPartOf(?webAp, ?HTML)* $\sqcap$ *contains(?query, ?method)* $\sqcap$ *contains(?method, ?param)* $\sqcap$ $\exists$*Vulnerability(?webAp, ?v)* $\sqcap$ *is_sentBy(?P, ?a)* $\rightarrow$ *is_detectedBy(?a, ?v)* (Drived),

**Rule 7:** *IF Rule 6*$\neg$*is_detectedBy(?a, ?v)* $\rightarrow$ *continue(?Q, ?a2)*(Drived).

Rule 7 becomes:
Attacker(?P) $\wedge$ hasInput(?P, ?Q) $\wedge$ hasPartOf(?webAp, ?HTML) $\wedge$ contains(?query, ?method) $\wedge$ contains(?method, ?param) $\wedge$ hasVulnerability(?webAp, ?v) $\wedge$ is_sentBy(?P, ?a) $\wedge$ notFound(?response, ?v) $\wedge$ Payload(?a2) $\rightarrow$ continue(?Q, ?a2)).

**Rule 8:** *IF Rule 6 is_detectedBy(?a,?v) OR Rule 7 is_detectedBy(?a2, ?v)* $\sqcap$ $\exists$*Vulnerability(?webAp, ?v)* $\sqcap$ *infectedBy(?param, ?a) OR infectedBy(?param, ?a2)* $\rightarrow$ *exploitedBy(?P, ?v)*(Drived).

**Interpretation of the rules**

- The first rule is a basic rule that states that if someone has some tools (Kali, Metasploit, maliciousPayload, ...), and uses them illegally, then that person is an attacker.
- Rule number 2, indicates that if class P is a sub-class of Q, then each instance of class P also belongs to class Q. For example: if the "Tools" class is a subclass of "Technology", then every instance (browsers, Kali Linux, Metasploit, ...) of the Tool class also belongs to the Technology class. The similar paradigm for the rule 3.
- This Rule 4, basically indicates that if the request contains a malicious string, and that, the malicious string contains a parameter value, then the request also contains that parameter value.
- Rule 5: The HTTP Request has part Referer, and the Referer contains the payload, then the HTTP Request also contains the payload.
- Rule 6: The HTML webpage allows user input. The attacker uses his method built with a parameter of his choice to query the request. If the server responds with a latency defined by the payload, then the vulnerability will be detected by the attacker.
- Rule 7: If the server is not responding (the request does not produce any latency), it does not mean that the web application is not vulnerable. Therefore, continue the attack process.
- Rule 8: If the response produces latency (from rule 6), then through the inference process the vulnerability will be possibly exploited by the attacker using some attack vectors.

### 5.3 Transformation of SWRL Rules to OWL Axioms

We present a theoretical idea applied to our ontology. Let **E**, **F**, **G** and **H** be some pairwise disjoint, infinite sets of *classes*, *sub-classes*, *properties* (Object and Data), *individuals* and *variables* where $\top$, $\bot \in \mathbf{E}$; the *universal property* U $\in \mathbf{F}$ i.e., **owl:topObjectProperty**. A *class expression* is an element of the following grammar I ::= (I $\sqcap$ I | $\exists$**F**.I | $\exists$.Self | **E** | {a} where $\mathbf{E} \in \mathbf{E}$, $\mathbf{F} \in \mathbf{F}$ and a $\in \mathbf{G}$.

Let us now resort to the definition of what an *axiom* is: it is a formula of the form E $\sqsubseteq$ K or $F_1 \circ \cdots \circ F_n \sqsubseteq F$ with E, K $\in$ I and $E_(i) \in \mathbf{F}$. A *rule* is a first-order logic formula ordinarily of the form $\forall \mathbf{p}(\beta(\mathrm{x}) \rightarrow \eta(\mathbf{q}))$ with $\beta$ and $\eta$ conjunctions of atoms; and **p**, **q** are subsequently non-empty sets of terms where $\mathbf{p} \subseteq \mathbf{q}$. *Rules* and *Axioms* expressions are very significant in building an ontology. Furthermore, they are referred to as *logical formulas*. Axioms correspond to OWL2 whereas rules correspond to SWRL.

Consider some terms $w$ and $z$ and a conjunction of atoms $\beta$. We say these two terms $w$ and $z$ are directly connected, or joined in $\beta$ if they occur in the same atom in $\beta$. We say $w$ and $z$ are connected in $\beta$ if there is some sequence of terms $w_1$, $\ldots$, $w_k$ with $w_1 = w$, $w_k = z$, and $w_{i-1}$ and $w_i$ are directly linked in $\beta$ for every $i = 2, \ldots, k$.

Additionally, for rules *rules* of the form $\beta \rightarrow \eta$; there exists an interpretation *it* which *entails rules*. Therefore, for every substitution *subst*, we have that *it*, *subst* $\models \beta$ implies *it*, *rules* $\models \eta$. That means, the semantics of rules here follows analogous standard semantics of the first-order predicate logic. From the same perspective, we say that two groupings of logical formulas S and $S'$ are equivalent if and only if each interpretation *it* that calls for $S$ and $S'$ are *equivalent* ($S \equiv S'$) and vice-versa.

On the same current of idea, $S'$ is a conservative extension of $S$ if and only if: Every interpretation that calls for $S'$ also calls for $S$.

Each interpretation that entails, calls for $S'$ is only expressed for the symbols in $S$ can be extended to an interpretation calling for $S'$ by adding appropriate interpretations for further signature symbols. Normally, all the variables in the body of a rule are connected. If two (2) variables for example $(a, b)$ are not linked with the body of a rule, then we could simply append the atom $\mathrm{U}(a, b)$ to the body of the rule resulting in a semantically $\equiv$ rule.

Using a fundamental example in the ontology can help us explain the transformation of rules into an axiom.

**Example 1.** Consider the rule $\Gamma$ = Person(p) $\wedge$ hasChild(c, c') $\wedge$ Female(c') $\rightarrow$ Daughter(c'). The following sequence of rules can be produced as follows:

$(\exists \mathrm{hasChild.Person})(\mathrm{c'}) \wedge \mathrm{Female(c')} \rightarrow \mathrm{Daughter(c')}$
$(\exists \mathrm{hasChild.Person} \sqcap \mathrm{Female})(\mathrm{c'}) \rightarrow \mathrm{Daughter(c')}$

Rule $\Delta_\Gamma$ from the above example can be now transformed into an axiom as stated in the following lemma.

**Lemma 1.** Consider some rule $\Gamma$. If $\Delta_\Gamma$ is of the form $\mathrm{A(p)} \rightarrow \mathrm{B(p)}$, then $\Gamma$ is equivalent to the axiom $\mathrm{A} \sqsubseteq \mathrm{B}$.

Since the equivalence relation is transitive, the rule $\Gamma$ is equivalent to the axiom $\exists \mathrm{hasTools.Person} \sqcap \mathrm{Attacks} \sqsubseteq \mathrm{Attacker}$.

**Proof.** Let $r$ and $r'$ be some rules such that $r'$ results by using some of the transformations (as in the previous example) to $r$. By definition, we can conclude that there is an equivalency between $r$ and $r'$. We can also demonstrate through induction that $\Gamma$ is equivalent to $\Delta_\Gamma$. Additionally, if $\delta$ $(\alpha \rightarrow \gamma)$ is of form $\mathrm{E(p)} \rightarrow \mathrm{F(p)}$, then by the definition of the semantics of rules and axioms, $\mathrm{E} \sqsubseteq \mathrm{F}$ is $\equiv$ to $\delta$ $(\alpha \rightarrow \gamma)$. Therefore, since the equivalence $(\equiv)$ relation is transitive, then we can safely say that $\gamma$ is $\equiv$ to $\mathrm{E} \sqsubseteq \mathrm{F}$. $\square$

**Lemma 2.** Furthermore, let us consider some rule Gamma (Γ). If $\Delta_\Gamma$ is of the form $\bigwedge_{t=2}^{m}(A_t(x_{t-1})) \wedge R_t(x_{t-1}, x_t) \wedge A_n(x_n) \rightarrow \mathrm{G}(x_1, x_n)$, then the group of axioms $A_t \sqsubseteq \exists R_{A_t}.\mathrm{Self} \mid t = 1, \ldots, m\} \cup \{R_{A_t} \circ R_1 \circ \cdots \circ R_{A_{m-1}} \circ R_m \circ R_{A_m} \sqsubseteq \mathrm{G}\}$ where all $R_{A_t}$ are the properties unique for each class $A_t$ is conservative extension of the rule Γ.

**Proof.** As illustrated in Lemma 1, rules Γ and $\Delta_\Gamma$ are equivalent. Thus, the lemma which follows the set of rules presented in the statement of the lemma is a conservative extension of Γ. See the following figure to see the preprocessing axiom generated in Protégé in the ROWLTAB plugin. □

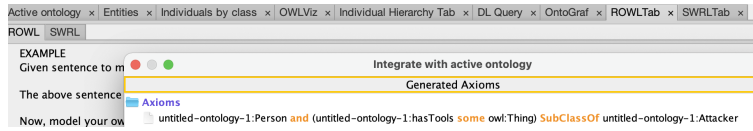



Figure 5. Rule is being converted to OWL axiom

Before implementing these rules, we had to create instances, data properties, object properties, and individuals to cooperate with the classes, and sub-classes; without that, the ontology will not understand your intention. The below listed the main classes and sub-classes. However, there are a lot of sub-classes that are not listed in the figure.

The figure 8 presents the individuals by class, where we can add "data properties assertion, object properties assertion, description types, etc."

For rational numbers xsd:decimal is of the best practice when using SWRL rules because it is the default for SWRL (Figure 7). When SWRL sees a literal such as 2.0 it draws the inference that the datatype is xsd:decimal. For other data types, you need to explicitly define the datatype as the literal. The property is functional because a Process can only have one value for its slack.

After that, we use the Drools rule engine to apply the rules in Section 5.2 to our ontology. If the rules are matched the properties you have established in the software protégé, then running the program using Pellet or HermiT plugins will generate the inferred classes along with their characteristics.

Note that, in the Protégé application, you can install several plugins to suit your needs, and add them to your tabs. After installing, you can simply go to "Window → Tabs" and select your desired one to add to your project. For more information about the software, here is a practical guide to building OWL ontology `https://www.researchgate.net/publication/351037551_A_Practical_Guide_to_Building_OWL_Ontologies_Using_Protege_55_and_Plugins`. The author very well described the concept of "Description Logic Reasoner to check the consistency of the ontology, data properties". He also introduces the Semantic Web Rule Language (SWRL) and a walk-through of creating SWRL and SQWRL rules.

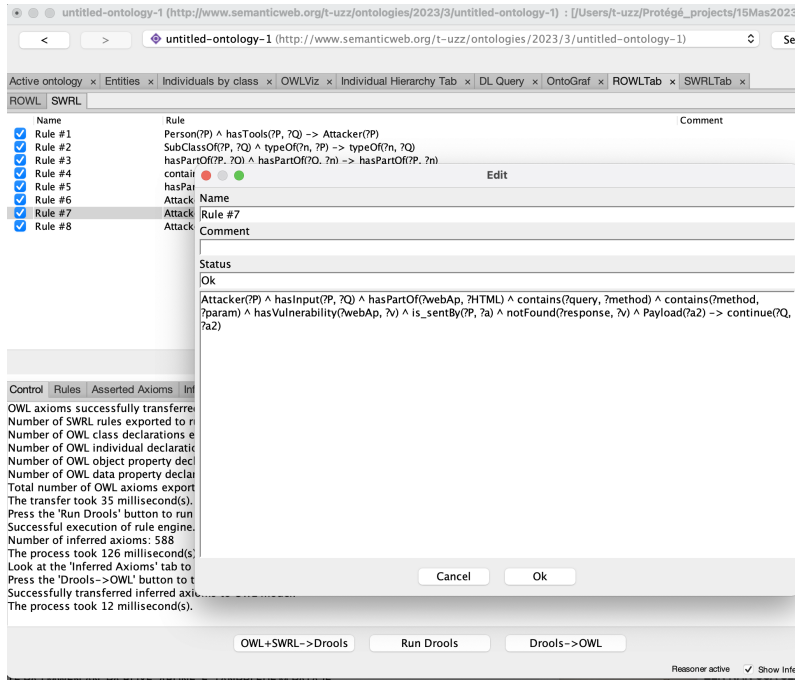



Figure 6. The ROWLTab interface with integrated axioms

We subsequently used the ROWLTab, “ROWL” and “SWRL” tab options to build the rules.

- Clicking on the “OWL + SWRL → Drools” button will transfer SWRL rules and relevant OWL knowledge to the rule engine.
- Likewise, clicking on the “Run Drools” button will run the rule engine.
- Clicking on the “Drools→OWL” button will transfer the inferred rule engine knowledge to OWL knowledge.

The SWRLAPI supports an OWL profile called OWL 2 RL and uses an OWL 2 RL-based reasoner to perform reasoning. An example is given in the following figure.

### 5.4 Ontology Design

In this section, we define the formalization of the core ontology concepts for SQL injection attacks. First, we introduce the set of terms:

- Term extraction consists of gathering a list of terms together that are relevant for a specific domain of knowledge. This can be done by defining a set of concepts. The properties, relationships, and meaning of concepts should be

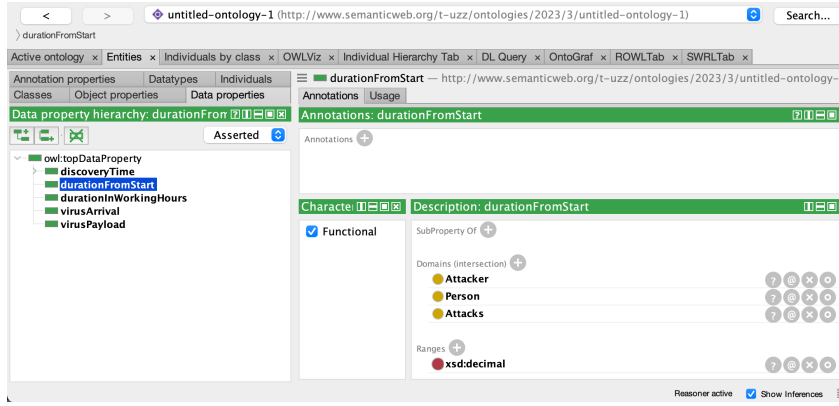



Figure 7. Data properties

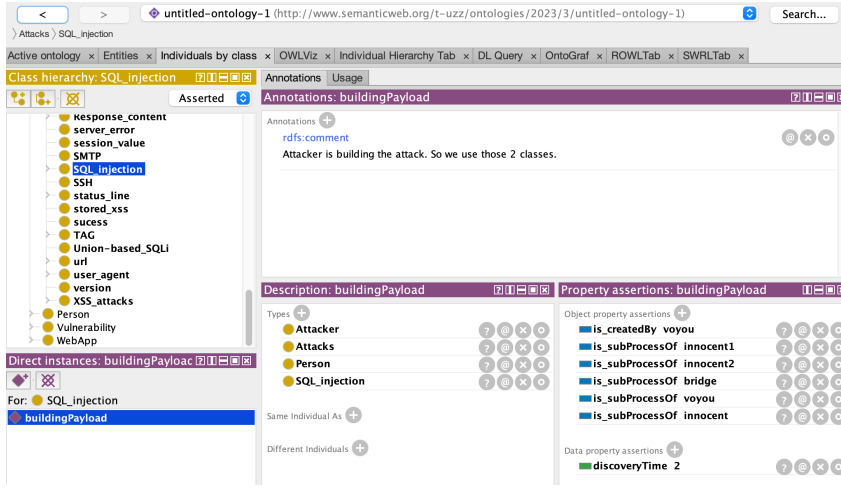



Figure 8. Individuals by class

evaluated before building the class hierarchy. To build our ontology, we make use of the following terms: SQL injection, Blind SQLi, attacks, vulnerability, weakness, attacker, web application, security layer, tools, technology, payloads, victim, exploitation.

- *Modules identification* consists of defining the set of individuals that will comply with the ontology scheme.
- The entities, data properties, object properties, and individuals of the ontology modules are designed using the Description Logics (hence DL) notation.

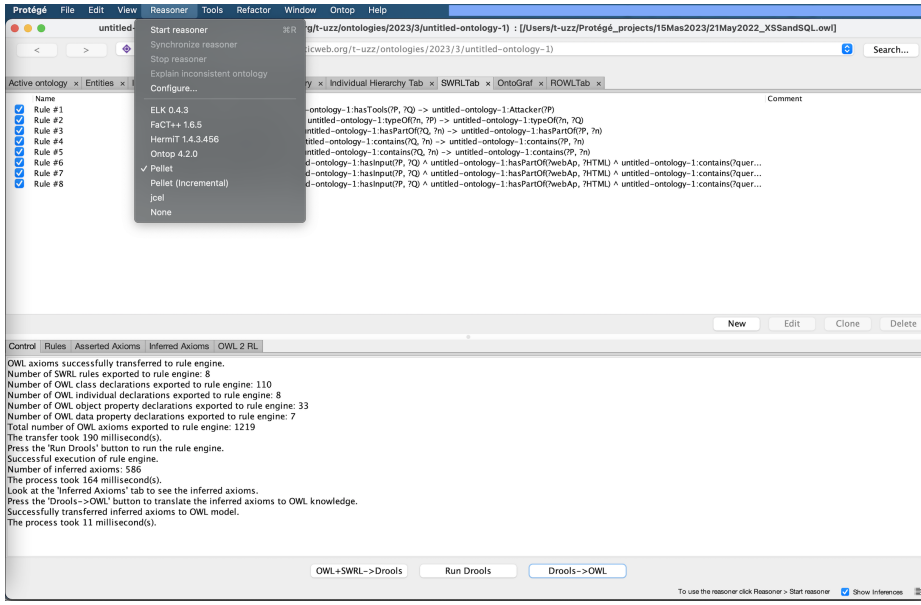



Figure 9. Running Reasoner to establish rules

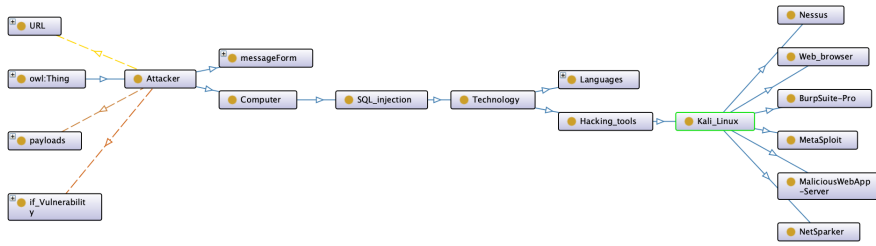



Figure 10. Subclasses of the Attacker ontology

The *Attacker* is a class in our ontology that generates the malicious payload using some technologies to launch the attack against a target victim. This class is further subdivided into several classes.

The following description logic (DL) represents the formal definition of the class *Attacker*.

Our ontology below describes briefly the security layers as a class, but we did not emphasize the mitigation of the SQL attacks. The approach is more related to the detection of the vulnerability. However, to encompass all the important concepts of the attack scenario, we also addressed some mitigation techniques that can be used to reduce these types of attacks.

$$\begin{gathered}\textit{Attacker} \equiv \\ \exists\ \textit{hasTitle.xsd:string} \sqcap \\ \textit{hasTools.xsd:string} \sqcap \textit{typeOf.xsd:string} \sqcap \\ \textit{hasDescription.xsd:string} \sqcap \\ \textit{discoveryTime.xsd:dateTime} \sqcap \\ \textit{discoveryTime.xsd:dateTimeStamp,} \sqcap \textit{isComposedOf(Attacker, Technology),} \\ \forall \textit{isComposedOf.Technology} \sqcap \exists \textit{isVulnerable.xsd:boolean,} \\ \textit{Technology} \subseteq \textit{Attacker,} \\ \textit{Computer} \subseteq \textit{Attacker}\end{gathered}$$

Figure 11. DL of the Attacker ontology

- The sub-class *Security* may include *Firewall*, *IDS*, *IPS* helps the administrator of the website to log and block any malicious-looking activity in the website in real-time such as SQL injections, XSS attacks, etc. The sub-classes *validation_Mechanism* may also include *Filters*, *Sanitization* are meant to be implemented most of the time by the web application developer during the coding process.
- *if_Vulnerability* exists, then a response from the target web server may alert the client (hence, the attacker's web browser). If the alert does not occur with a string response, then mostly it may occur in a form of latency.
- The class *SQLi_Attacks* contains several subclasses and sub-subclasses; it is the main class for the penetration testing phase. This is where all the attempts (SQL malicious payloads) occurred.

## 6 CONCLUSIONS

From now, our level of thinking about security risks and privacy – specifically about how our confidential data is stored online, should be well-oriented more seriously. In this paper, we briefly talked about the cybersecurity offensive. However, having this kind of knowledge about how users' data can be extracted by attackers abusing SQL injection vulnerabilities leads us into digging into how we can prevent this from happening. We briefly demonstrated through the establishment of semantic rules how we can make use of the ontology to detect vulnerabilities. Due to the required size of this paper, a deeper description will be elaborated on in our future work. Therefore, in our next paper, we will dive into the work performance of our ontology approach for the detection of SQL vulnerabilities and will be more oriented to the mitigation techniques.

### Abbreviations and Acronyms

**SQLi:** SQL injection,

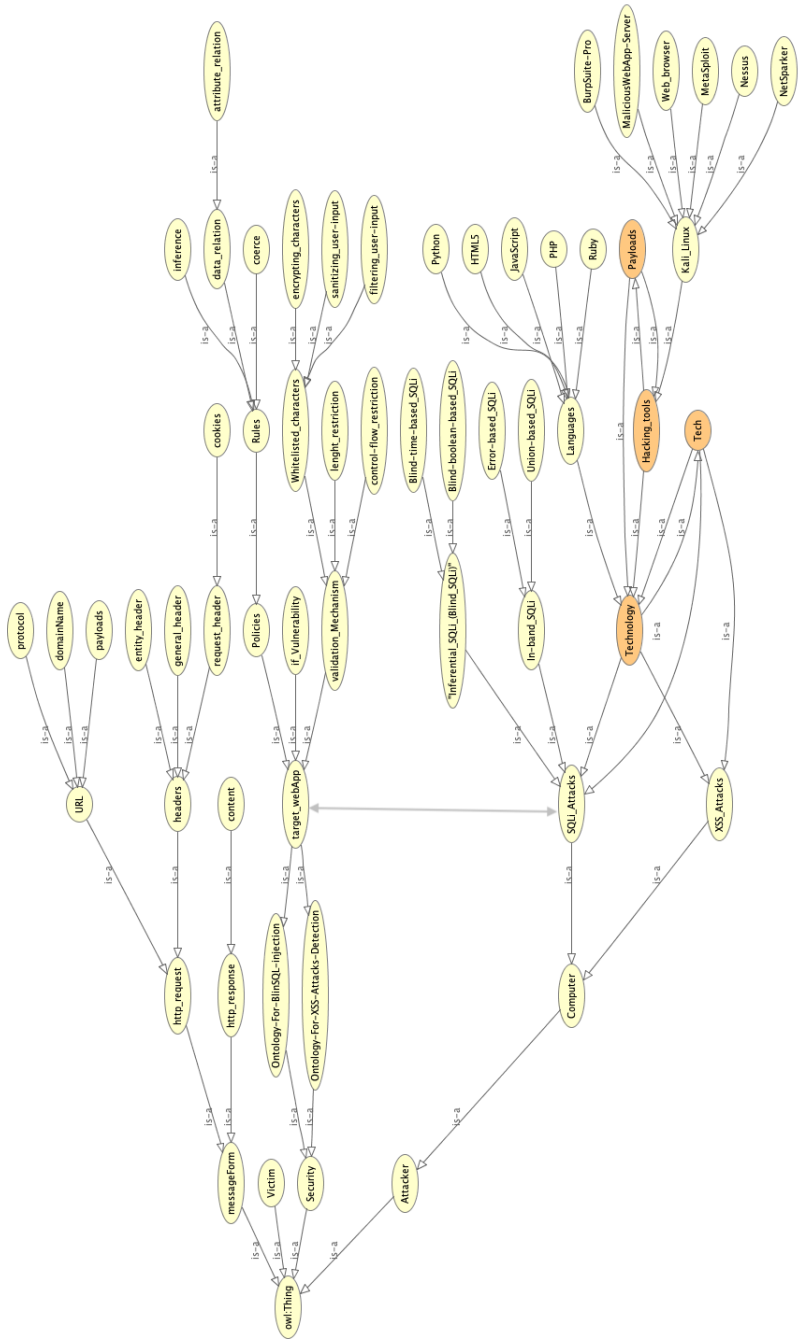



Figure 12. Description of the ontology, generated from OWLViz plugin

**DL:** Description Logic,

**ICS:** Industrial Control System,

**OWASP:** Open Web Application Security Project,

**OWL:** Web Ontology Language.

### Mathematical Symbols

| Abstract Syntax | DL Syntax |
|---|---|
| Class($A$ partial $C_1 \dots C_n$) | $A \sqsubseteq C_1 \sqcap \dots \sqcap C_n$ |
| Class($A$ complete $C_1 \dots C_n$) | $A \equiv C_1 \sqcap \dots \sqcap C_n$ |
| EnumeratedClass($A$ $o_1 \dots o_n$) | $A \equiv \{o_1\} \sqcup \dots \sqcup \{o_n\}$ |
| SubClassOf($C_1$ $C_2$) | $C_1 \sqsubseteq C_2$ |
| EquivalentClasses($C_1 \dots C_n$) | $C_1 \equiv \dots \equiv C_n$ |
| DisjointClasses($C_1 \dots C_n$) | $C_i \sqcap C_j \sqsubseteq \bot, i \neq j$ |
| Datatype($D$) | |
| ObjectProperty($R$ super($R_1$)...super($R_n$) | $R \sqsubseteq R_i$ |
| domain($C_1$)...domain($C_m$) | $\geqslant 1\, R \sqsubseteq C_i$ |
| range($C_1$)...range($C_\ell$) | $\top \sqsubseteq \forall R.C_i$ |
| [inverseOf($R_0$)] | $R \equiv R_0^-$ |
| [Symmetric] | $R \equiv R^-$ |
| [Functional] | $\top \sqsubseteq \leqslant 1\, R$ |
| [InverseFunctional] | $\top \sqsubseteq \leqslant 1\, R^-$ |
| [Transitive]) | $Tr(R)$ |
| SubPropertyOf($R_1$ $R_2$) | $R_1 \sqsubseteq R_2$ |
| EquivalentProperties($R_1 \dots R_n$) | $R_1 \equiv \dots \equiv R_n$ |
| DatatypeProperty($U$ super($U_1$)...super($U_n$) | $U \sqsubseteq U_i$ |
| domain($C_1$)...domain($C_m$) | $\geqslant 1\, U \sqsubseteq C_i$ |
| range($D_1$)...range($D_\ell$) | $\top \sqsubseteq \forall U.D_i$ |
| [Functional]) | $\top \sqsubseteq \leqslant 1\, U$ |
| SubPropertyOf($U_1$ $U_2$) | $U_1 \sqsubseteq U_2$ |
| EquivalentProperties($U_1 \dots U_n$) | $U_1 \equiv \dots \equiv U_n$ |
| AnnotationProperty($S$) | |
| OntologyProperty($S$) | |
| Individual($o$ type($C_1$)...type($C_n$) | $o \in C_i$ |
| value($R_1$ $o_1$)...value($R_n$ $o_n$) | $\langle o, o_i \rangle \in R_i$ |
| value($U_1$ $v_1$)...value($U_n$ $v_n$)) | $\langle o, v_i \rangle \in U_i$ |
| SameIndividual($o_1 \dots o_n$) | $\{o_1\} \equiv \dots \equiv \{o_n\}$ |
| DifferentIndividuals($o_1 \dots o_n$) | $\{o_i\} \sqsubseteq \neg\{o_j\}, i \neq j$ |

Figure 13. OWL DL axioms and facts [13]

**Acknowledgement**

This work was supported by the Slovak Research and Development Agency under the Contract No. APVV-20-0548 (ARIEN), also by the Slovak Scientific Grant Agency VEGA 2/0125/20, VEGA 2/0119/23 and APVV 19-0220.

## REFERENCES

[1] Gomez-Valades, A.—Martinez-Tomas, R.—Rincon, M.: Integrative Base Ontology for the Research Analysis of Alzheimer's Disease-Related Mild Cognitive Impairment. Frontiers in Neuroinformatics, Vol. 15, 2021, Art. No. 561691, doi: 10.3389/fninf.2021.561691.

[2] Zouri, M.—Ferworn, A.: An Ontology-Based Approach for Curriculum Mapping in Higher Education. 2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC), 2021, pp. 0141–0147, doi: 10.1109/CCWC51732.2021.9376163.

[3] Karimi, S.—Iordanova, I.—St-Onge, D.: An Ontology-Based Approach to Data Exchanges for Robot Navigation on Construction Sites. 2021, doi: 10.48550/arXiv.2104.10239.

[4] Singels, L.—Biebuyck, C.—Maluleke, L.: A Formal Concept Analysis Driven Ontology for ICS Cyberthreats. In: Gerber, A. J. (Ed.): Proceedings of the First Southern African Conference for Artificial Intelligence Research (SACAIR 2020). 2020, pp. 247–263.

[5] Sattar, A.—Ahmad, M. N.—Surin, E. S. M.—Mahmood, A. K.: An Improved Methodology for Collaborative Construction of Reusable, Localized, and Shareable Ontology. IEEE Access, Vol. 9, 2021, pp. 17463–17484, doi: 10.1109/ACCESS.2021.3054412.

[6] Aguado, E.—Sanz, R.: Using Ontologies in Autonomous Robots Engineering. Robotics Software Design and Engineering, IntechOpen, 2021, doi: 10.5772/intechopen.97357.

[7] Lu, D.—Fei, J.—Liu, L.: A Semantic Learning-Based SQL Injection Attack Detection Technology. Electronics, Vol. 12, 2023, No. 6, 1344 pp., doi: 10.3390/electronics12061344.

[8] Crespo-Martínez, I. S.—Campazas-Vega, A.—Guerrero-Higueras, Á. M.—Riego-DelCastillo, V.—Álvarez-Aparicio, C.—Fernández-Llamas, C.: SQL Injection Attack Detection in Network Flow Data. Computers and Security, Vol. 127, 2023, Art. No. 103093, doi: 10.1016/j.cose.2023.103093.

[9] Nallusamy, S.—Hoo, M. H.—Zulkifle, F. A.: Controlled Experiment for Assessing the Contribution of Ontology Based Software Redocumentation Approach to Support Program Understanding. Computing and Informatics, Vol. 40, 2021, No. 5, pp. 1025–1055, doi: 10.31577/cai_2021_5_1025.

[10] Yakhyaeva, G.—Karmanova, A.—Ershov, A.: Application of the Fuzzy Model Theory for Modeling QA-Systems. Computing and Informatics, Vol. 40, 2021, No. 6, pp. 1197–1216, doi: 10.31577/cai_2021_6_1197.

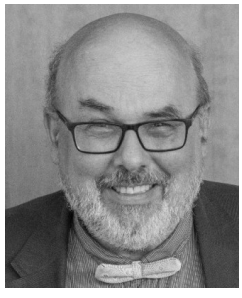
[11] Dora, J. R.—Nemoga, K.: Ontology for Cross-Site-Scripting (XSS) Attack in Cybersecurity. Journal of Cybersecurity and Privacy, Vol. 1, 2021, No. 2, pp. 319–339, doi: 10.3390/jcp1020018.

[12] Dora, J. R.—Nemoga, K.: Clone Node Detection Attacks and Mitigation Mechanisms in Static Wireless Sensor Networks. Journal of Cybersecurity and Privacy, Vol. 1, 2021, No. 4, pp. 553–579, doi: 10.3390/jcp1040028.

[13] Baader, F.—Calvanese, D.—McGuinness, D.—Patel-Schneider, P.—Nardi, D.: The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, 2003.

**Jean Rosemond Dora** works in the CAI Editorial Office as Reviewer, as well as in UI SAV. He is a penetration tester, focusing primarily on detecting and exploiting (upon request) vulnerabilities from a given environment. He holds certificates among which, Cybersecurity and Infrastructure Security Agency (CISA) from the U.S. Department of Homeland Security, Industrial Control System (ICS); Certified Ethical Hacker (CEH), Practical Network Penetration Testing (PNPT), Security Awareness Foundations and Training. He holds Ph.D. degree obtained from the Institute of Mathematics, Slovak Academy of Sciences (MUSAV). Has Master's degree from the Faculty of Electronics and Informatics, Slovak University of Technology (FEI-STU) in Bratislava. Holds a second Master's degree in computer science from the Faculty of Education in Ružomberok. He is also employed in internal/external, web applications, and wireless network assessments in penetration testing as an independent contractor. Additionally, He is an online instructor, teaching ethical hacking courses on Udemy, Thinkific, and Teachable platforms.

**Ladislav Hluchý** is Senior Research Scientist and Manager with more than 20 years of experience in leading national and international research projects and teams of 5 to 20 researchers. He is a competent scientist in the area of high-performance computing, multi-cloud computing, parallel and distributed information processing, and knowledge management. His research also focuses on data flow management through abstract language mechanisms. In the past, Ladislav Hluchý, Associate Professor, has participated in several cooperations with industry, which is beneficial for the transfer of the project results into practice.



**Karol Nemoga** Director of the Institute of Mathematics Slovak Academy of Sciences. He graduated from the Faculty of Mathematics and Physics of Charles University in Prague. In 1976, he joined the Institute of Mathematics Slovak Academy of Sciences. He has been its director since 2015. He works as University Teacher. He specializes in cryptology, computational number theory, and coding theory. He published about 30 scientific articles and about ten teaching texts. He is also working in the Association for Computing Machinery, and the Institute of Electrical and Electronics Engineers (IEEE). He is a member of the Union of Slovak Mathematicians and Physicists, the Slovak Gas Society, and the International Association for Cryptology.