Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Mgr. Ing. Jean Rosemond Dora

Autoreferát dizertačnej práce

ONTOLOGY FOR CROSS-SITE-SCRIPTING (XSS) IN CYBERSECURITY

na získanie akademického titulu philosophiae doctor
v odbore doktorandského štúdia:

9.1.9 Aplikovaná matematika

Bratislava, 2022

Dizertačná práca bola vypracovaná:
v dennej forme doktorandského štúdia na Katedre aplikovanej matematiky a štatistiky Fakulty matematiky, fyziky a informatiky Univerzity Komenského v Bratislave.

Predkladateľ: Mgr. Ing. Jean Rosemond Dora
Matematický ústav, v. v. i.
Slovenská Akadémia Vied
Štefanikova 49
Bratislava

Školiteľ: doc. RNDr. Karol Nemoga, CSc.
Matematický ústav, v. v. i.
Slovenská Akadémia Vied
Štefánikova 49
Bratislava

Oponenti: Prof. María Isabel González Vasco, PhD.
Catedrática de Universidad, Departamento de Matemática Aplicada, Ciencia e Ingeniería de los Materiales y Tecnología Electrónica (MACIMTE), Universidad Rey Juan Carlos, MADRID
SPAIN

Prof. Ing. Pavol Zajac, PhD.
FEI STU, Ilkovičova 3
812 19 Bratislava

Doc. RNDr. Karol Macák, CSc.
Ministerstvo obrany SR
Kutuzovova 8
832 47 Bratislava

Obhajoba dizertačnej práce sa koná 25. 8. 2022 o 8:30 hod pred komisiou pre obhajobu dizertačnej práce odbore doktorandského štúdia vymenovanou predsedom odborovej komisie dňa 14. 7. 2022 v študijnom odbore 9.1.9 Aplikovaná matematika v seminárnej miestnosti č. 502, FEI STU, Ilkovičova 3. Bratislava, Blok C, Bratislava.

Predseda odborovej komisie:
prof. RNDr. D. Ševčovič, DrSc.
Katedra aplikovanej matematiky a štatistiky
81438 Bratislava

Introduction
Using the semantic concept, we can use deductive inference rules to reason or deduce on a piece of HTTP well-structured scheme.
Firstly, let us briefly indicate to which class each of the following concepts belongs:

HTTP $\sqsubset$ Protocol
GET $\sqsubset$ Method
POST $\sqsubset$ Method
XSS attacks $\sqsubset$ Attack
Request Header $\sqcap$ Response Header $\equiv \perp$
POST $\sqcap$ GET$\equiv \perp$
HTTP Request $\equiv$ User Request

In the diagram, all conceptually subsume ($\sqsubset$) relations are *irreflexive*, *transitive*, and *asymmetric*. The equivalence ($\equiv$) relations are *reflexive*, *symmetric* and *transitive*. Likewise, no conceptually disjoint ($\sqcap$) relations infringe its characteristics of *reflexive*, *symmetric* and *transitive*. We implmented these rules based on how we were able to detect the XSS attacks using burpsuite tool, then apply it 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)* (Transitive)

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

Rule #6: *Attacker(?P)* $\sqcap$ *hasInput(?P, ?Q)* $\sqcap$ *hasPartOf(?R, ?S)* $\sqcap$ *contains(?method, ?openTAG)* $\sqcap$ *contains(?method, ?param)* $\sqcap$ $\exists$*Vulnerability(?R, ?v)* $\sqcap$ *sentBy(?a, ?P)* $\rightarrow$ *detectedBy(?a, ?v)* (Drived)

Rule #7:
maliciousPayload(?i) $\sqcap$ *createdBy(?a, ?i)* $\sqcap$ *HTTPmessage(?m, ?i)* $\sqcap$ *hasPartOf(?R, ?S)* $\sqcap$ $\exists$*Vulnerability(?R, ?v)* $\sqcap$
contains(?method, ?param) $\sqcap$ *isEscaped(?i, ?char)* $\rightarrow$ *isFiltered(?R, ?char)* (Drived)

Rule #8: *if Rule 7 isFiltered(?R, ?ch)* $\sqcap$ $\exists$*Vulnerability(?R, ?v)* $\sqcap$ *infectedBy(?param, ?a)* $\rightarrow$ *exploitedBy(?v, ?a)* (Drived)

Rule #9: *maliciousPayload(?i)* $\sqcap$ *createdBy(?a, ?i)* $\sqcap$ *HTTPMessage(?m, ?request)* $\sqcap$ *contains(?m, ?param)* $\sqcap$ *hasVulnerability(?webAP, ?v)* $\sqcap$ *contains(?webAP, ?request)* $\sqcap$ *isPartOf(?i, ?webAP)* $\rightarrow$ *infectedBy(?m, ?i)* (Drived)

Explanation
Rule 1 is a basic rule that states that if a person has some tools (kali, metasploit, maliciousPayload,...), then that person is an attacker.

This rule 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 "JavaScript" class is a subclass of "Technology", then every instance (tag, characters, payloads...) of the JavaScript class also belongs to the Technology class.

Rule 3: Let us take another example to explain this rule: an attacker crafted his payload and sends it to a victim via a link. So, HTTP Request has part Payload, and Payload has part URL. Logically, the HTTP Request has a part URL as well.

This rule 4, basically indicates that if a URL contains a malicious string, and that malicious string contains a parameter value, then the URL also contains that parameter value.

Rule 5: The HTTP Request has part URL, and the URL contains the payload, then the HTTP Request contains the payload.

Rule 6: If URL has a malicious input, and a vulnerable HTML webpage reflects an open tag and/or character or parameter included in the GET/POST method in the burp suite, then by closing them and relaunch the JavaScript payload into the vulnerable webpage, the system will trigger an alert.

Rule 7: Similarly, the malicious input (i) is created by the attacker, and is embedded in the message. If the HTTP message from the evil link is reflected in the burp while some or all its characters get escaped, then a filter (input validation) has been applied in the victim's environment web application.

Rule 8 indicates that, if there are some characters that are escaped in the Response HTML-DOM in burp, and some not, then through the inference process the vulnerability will get exploited by the attacker using some attack vectors.

For rule 9: This rule indicates that the attacker uses some attack vectors and technology to build his malicious payload and injects it into a user-input field. "m" is an HTTP message which contains a parameter value "param". So, the evil input "i" infects the parameter "param", and becomes a part of the web page. Thus, this implies that the spiteful input "i" also infects the entire HTTP message "m".

0.0.1 Transformation of SWRL Rules to OWL Axioms

We present a theoretical notions employed in our ontology. Let **A, B, C** and **D** be pairwise disjoint, infinite sets of *classes*, (included sub-classes), *properties* (Object and Data), *individuals* and *variables* where $\top$, $\bot \in$ A; the *universal property* U $\in$ B i.e., **owl:topObjectProperty**. A *class expression* is an element of the following grammar F ::= (F $\sqcap$ F | $\exists$B.F | $\exists$.Self | A | {a} where A $\in$ A, B $\in$ **B** and a $\in$ C. Now let us define what an *axiom* is: it is a formula of the form A $\sqsubseteq$ J or $B_1 \circ ... \circ B_n \sqsubseteq B$ with A, J $\in$ F and $B_(i) \in$ **B**. A *rule* is a first-order logic formula usually of the form $\forall \mathbf{p}(\beta(x) \rightarrow \eta(\mathbf{q}))$ with β and η conjunctions of atoms; and **p**, **q** are non-empty sets of terms where $\mathbf{p} \subseteq \mathbf{q}$. *Rules* and *Axioms* are very important in building an ontology using Protégé, they are also referred to as *logical formulas*. Axioms correspond to OWL 2 EL axioms whereas rules correspond to SWRL. Consider some terms m and n and a conjunction of atoms β. We say these two terms m and n are directly linked in β if both terms occur in the same atom in β. We say m and n are linked in β if there is some sequence

of terms m_1...,m_k with $m_1 = m, m_k = n$, and m_{i-1} and m_i are directly linked in β for every i = 2,...,k. Let us say again for rules *rules* of the form $\beta \rightarrow \eta$ that there is exist an interpretation I which *entails rules*. Thus, for every substitution *subst*, we have that *I, subst* $\models \beta$ implies *I, rules* $\models \eta$. That is to say, the semantics of rules follows similarly the standard semantics of the first-order predicate logic. Moreover, we say that two groupings of logical formulas G and G' are equivalent if and only if each interpretation I that entails G and G' are *equivalent* (G $\equiv G'$) and vice-versa. G' is a conservative extension of G if and only if:
Every interpretation that entails G' also entails G.
Each interpretation that entails G' is only defined for the symbols in G can be stretched out to an interpretation entailing G' by appending appropriate interpretations for further signature symbols. Usually, all the variables in the body of a rule are linked together. If two variables for example (v,w) are not in connection in the body of a rule, we could simply add the atom U(v,w) to the body of the rule resulting in a semantically $\equiv$ rule.

Let us take an example to explain the transformation of rules into a axiom:

Example.- Consider the rule Γ = Person(x) $\wedge$ hasChild(x, y) $\wedge$ Female(y) $\rightarrow$ Daughter(y). The following sequence of rules can be produced as follows:
($\exists$hasChild.Person)(y) $\wedge$ Female(y) $\rightarrow$ Daughter(y)
($\exists$hasChild.Person $\sqcap$ Female)(y) $\rightarrow$ Daughter(y)
Rule Δ_Γ from the preceding example can be straight transformed into an axiom as stated in the following lemma.

Lemma 1. *Consider some rule* Γ. *If* Δ_Γ *is of the form A(x)* $\rightarrow$ *B(x), then* Γ *is equivalent to the axiom A* $\sqsubseteq$ *B.*
Since the equivalence relation is transitive, the rule Γ is equivalent to the axiom $\exists$hasTools.Person$\sqcap$Attacks $\sqsubseteq$Attacker.
Proof.- Let a and a' be some rules such that a' results by applying some of the transformations (as in the previous example) to a. By definition, we can conclude equivalency between a and a' and we can show via induction that Γ is equivalent to Δ_Γ. Additionally, if δ ($\alpha \rightarrow \gamma$) is of the form A(x) $\rightarrow$ B(x), then by the definition of the "semantics of rules and axioms", A $\sqsubseteq$ B is $\equiv$ to $\delta(\alpha \rightarrow \gamma)$. Therefore, since the equivalence ($\equiv$) relation is transitive, then γ is $\equiv$ to A $\sqsubseteq$ B.

Lemna 2. Consider some rule Γ. If Δ_Γ is of the form $\bigwedge_{i=2}^{m}(A_i(x_{i-1})) \wedge R_i(x_{i-1}, x_i) \wedge A_n(x_n) \rightarrow$ G(x_1, x_n), then the group of axioms $A_i \sqsubseteq \exists R_{A_i}.Self \mid$ i = 1, ...,m} $\cup$ { $R_{A_i} \circ R_1 \circ ... \circ R_{A_{m-1}} \circ R_m \circ R_{A_m} \sqsubseteq$ G} where all R_{A_i} are the properties unique for each class A_i is conservative extension of the rule Γ. (See 1 to better understand the meaning of the symbols.)
Proof. As shown in proof of Lemma 1, rules Γ and Δ_Γ are in fact equivalent. Therefore, the lemma 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 implemented in Protégé in the ROWLTAB plugin.

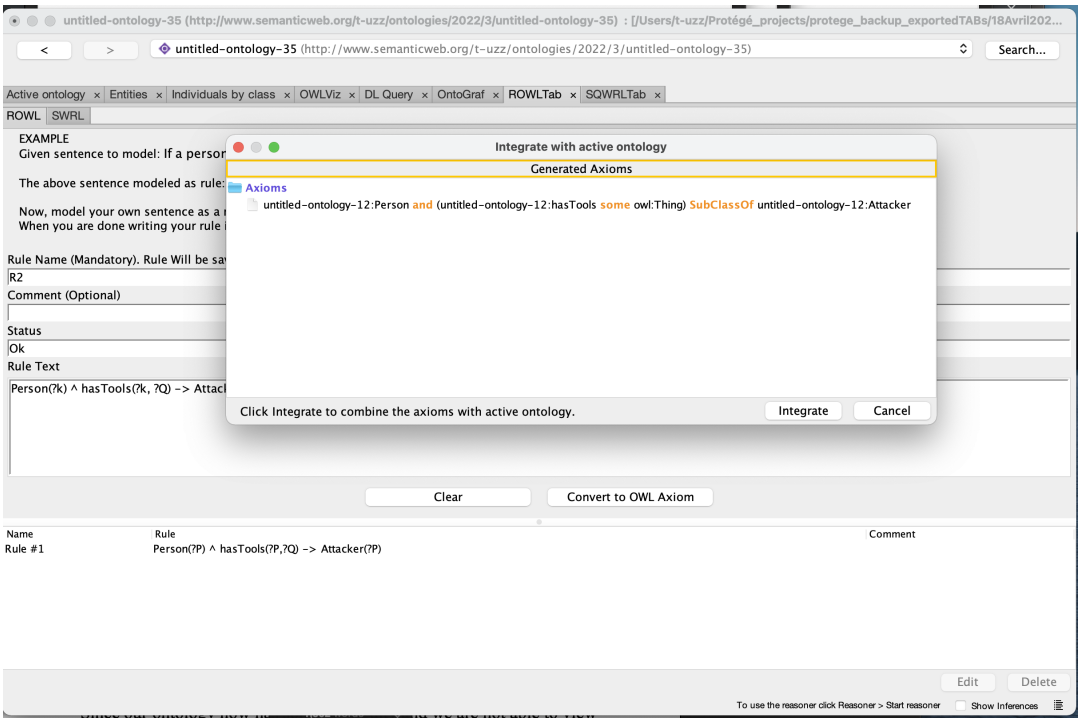

FIGURE 1: Rule is being converted to OWL axiom.

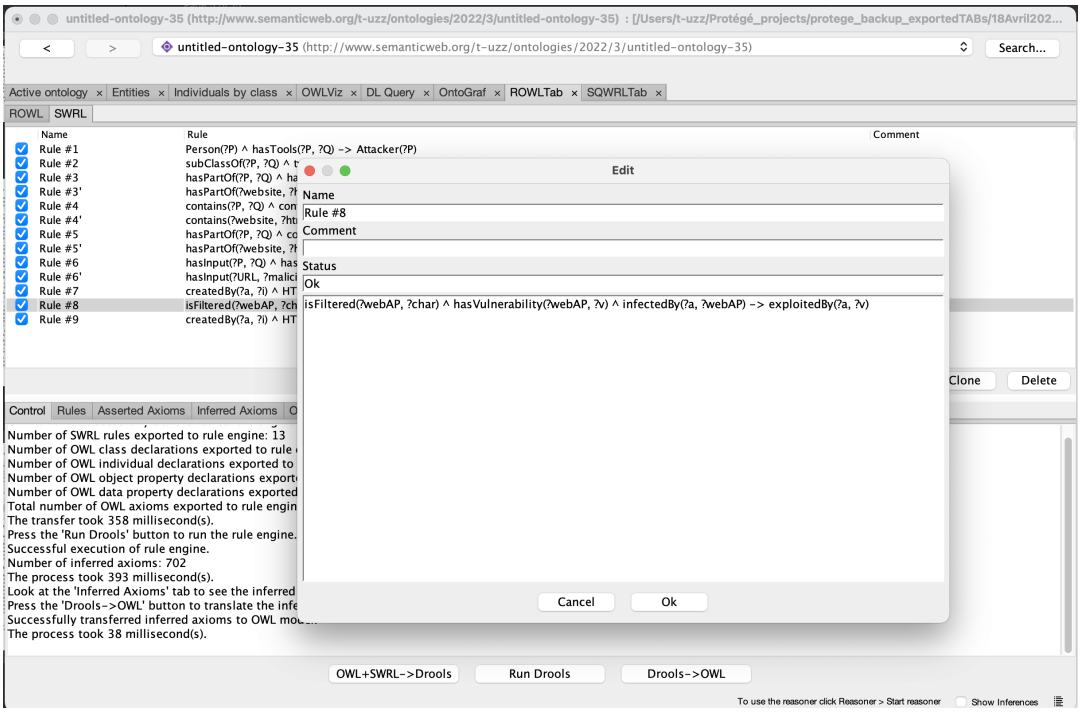

FIGURE 2: The ROWLTab interface with integrated axioms.

Before applying the rules, we had to create instances, object properties, data properties, individuals to co-operate with the classes, sub-classes in the ontology. The following below listed the main classes and sub-classes. However, there are a lot of sub-classes that are not listed in the figure.

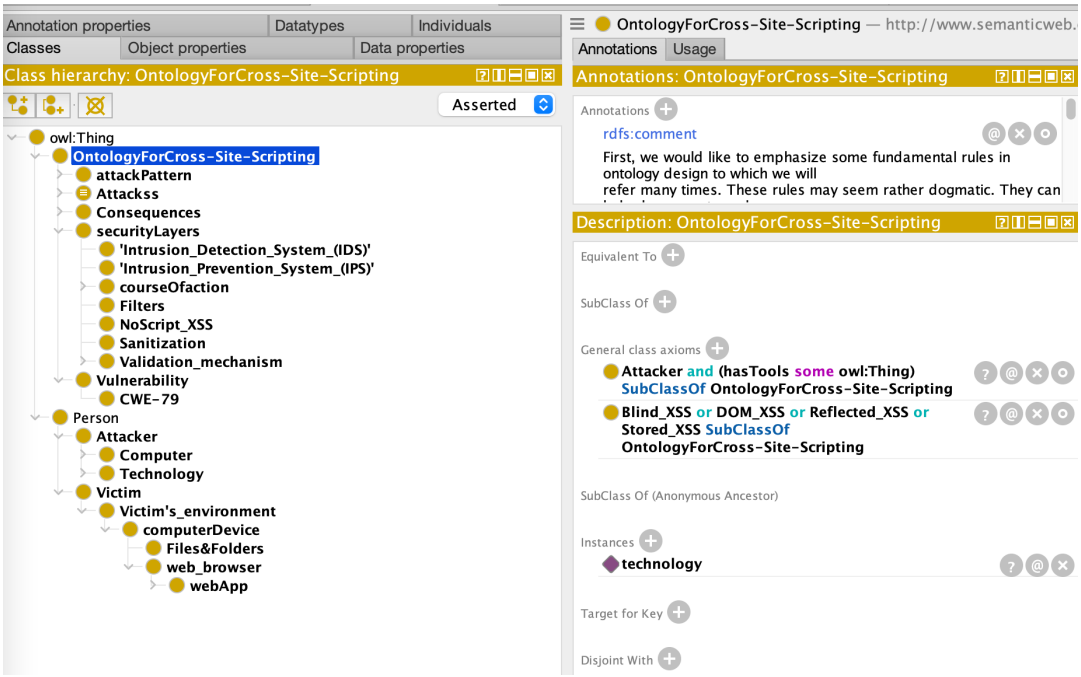

FIGURE 3: Classes and main sub-classes in the ontology development.

The following figure are the sub-classes that can be added to the *attackPattern* ontology.

FIGURE 4: Main sub-classes in the ontology attackPatern class.

The following figure can be represented the object properties of the ontology. Note: If we want to enlarge the ontology in the future, it can be done here in the "Object properties, data properties, individuals" entities.

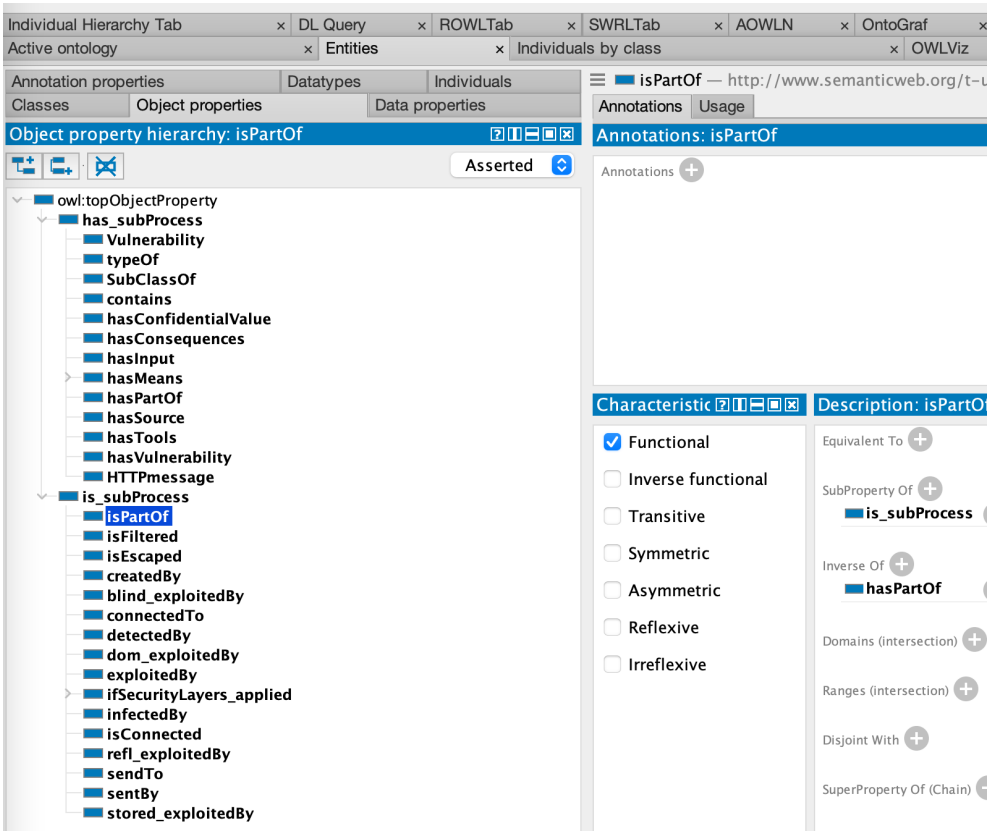

FIGURE 5: Object properties.

The next figure is the data properties which can be built in datatypes (Ranges), for example: xsd:decimal, xsd:double, xsd:float, rdfs:Literal, xsd:dateTime, xsd:hexBinary, etc.

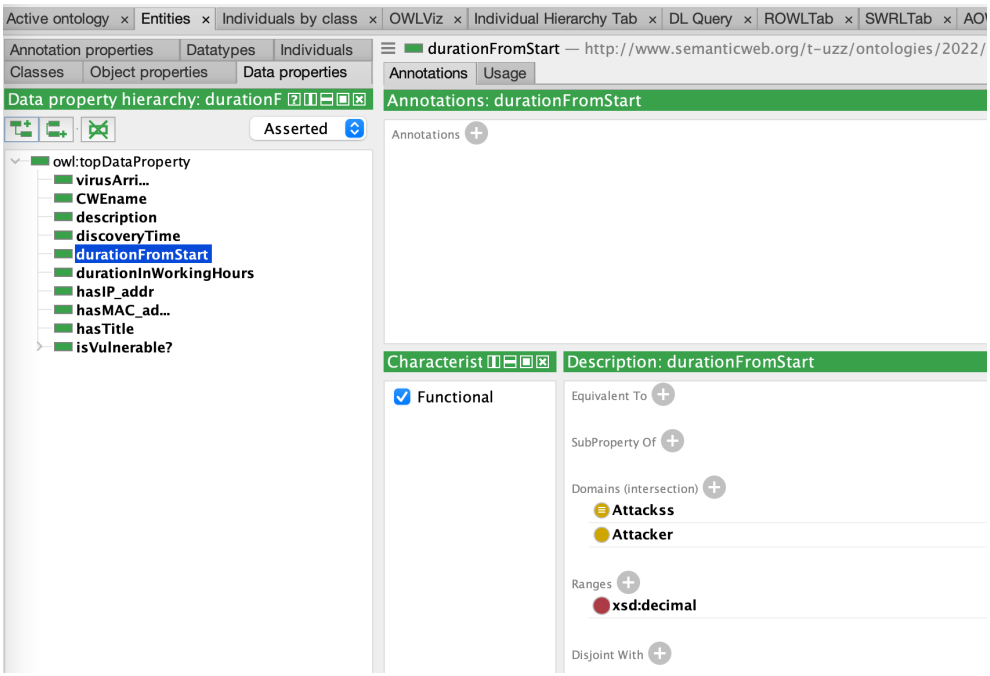

FIGURE 6: Data properties.

The following is the individuals by class, where we can add "data properties assertion, object properties assertion, types, etc".

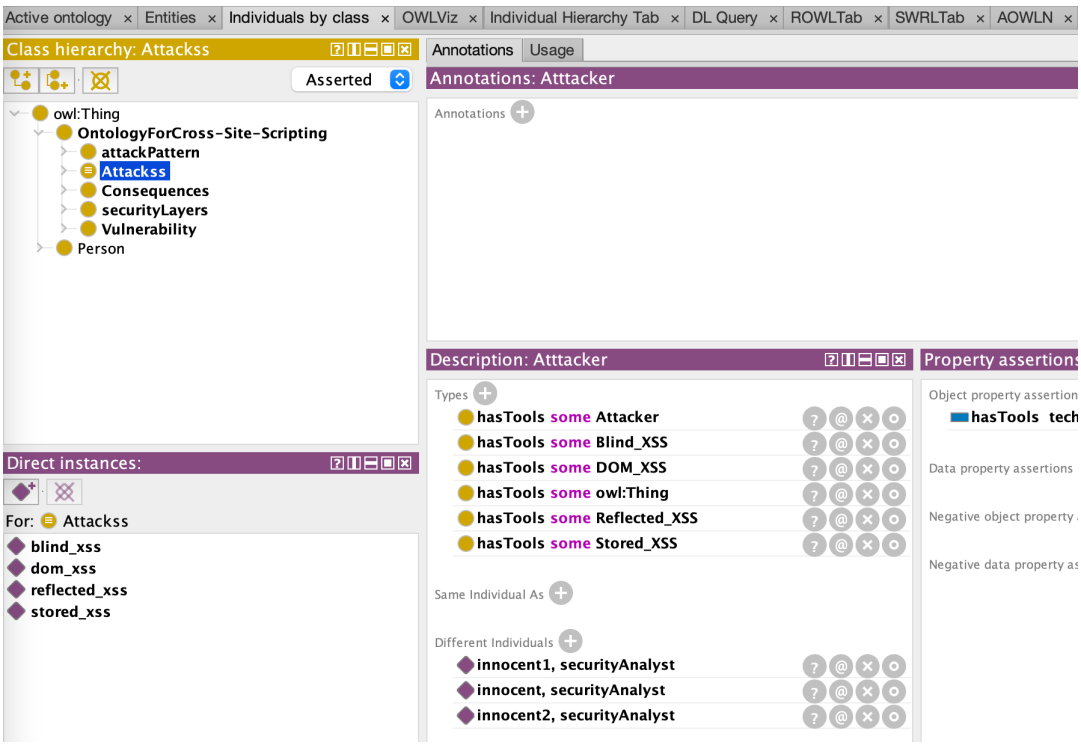

FIGURE 7: Individuals by class.

For rational numbers xsd:decimal is a good choice when using SWRL rules because it is the default for SWRL. When SWRL sees a literal such as 2.0 it assumes the datatype is xsd:decimal. For other datatypes, you need to explicitly specify the datatype in the literal. The property is functional because a Process can only have one value for its slack.

Now, using the Drools rule engine, we can apply the rules on the previous page () 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. We also use ROWLTab, and sub-tabs "ROWL" and "SWRL" to build the rules.
- Pressing the "OWL+SWRL -> Drools" button will transfer SWRL rules and relevant OWL knowledge to the rule engine.
- Pressing the "Run Drools" button will run the rule engine.
- Pressing 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.

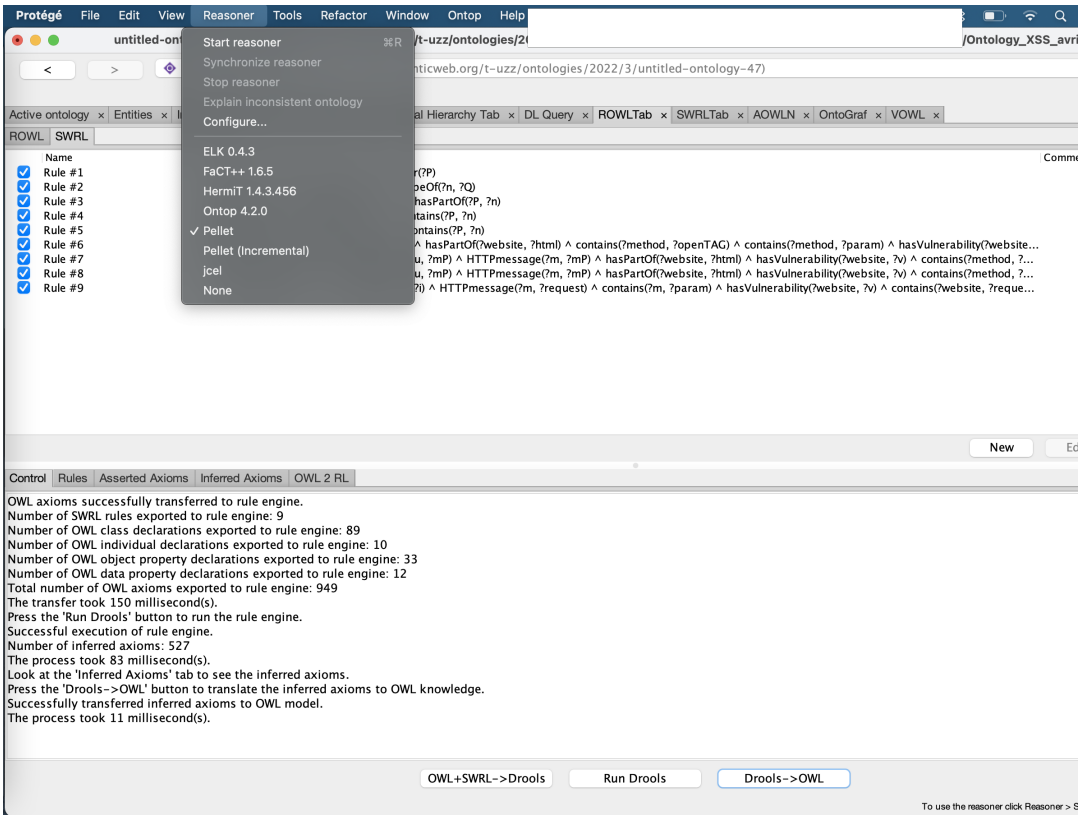

FIGURE 8: Running Reasoner to establish rules.

0.0.2 Ontology Design

In this chapter, we describe the formalization of the main ontology concepts for Cross-site-scripting attacks. First, we introduce the set of terms:
- *Term extraction* or *elicitation* consists of assembling a list of terms that are relevant for a specific domain of knowledge. This can be done by defining and identifying a set of concepts. The relationship, properties, and meaning of concepts should be evaluated before building the class hierarchy. The list of some terms used in our Ontology is as follows:
Cross-site-scripting, attacks, vulnerability, attacker, web application, security layer, tools, technology, payloads, weakness, victim, exploitation, Stored XSS, Blind XSS, Reflected XSS, Reflected XSS.
- *Modules identification* consists of describing the set of individuals that will comply with the ontology system.
- The hierarchy, data properties, object properties, entities, individuals of the ontology modules are designed using the *Description Logics* notation.
The *Attacker* is a class in our ontology that generates the malicious input using some technologies such as MetaSploit, JavaScript, etc to launch an attack against a target victim. This class is further subdivided into several classes.

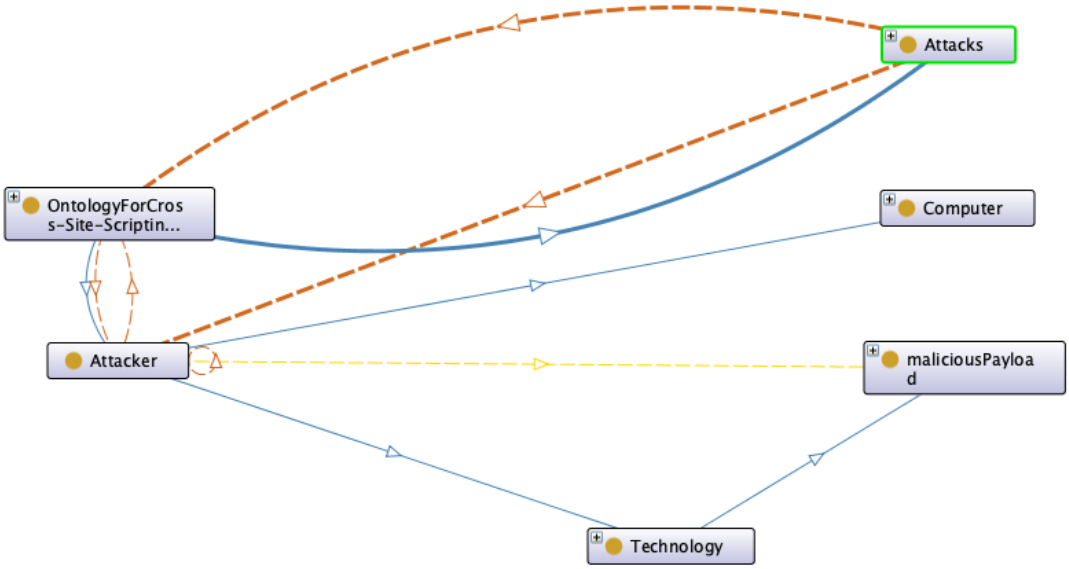

FIGURE 9: Subclasses of the Attacker ontology.

The formal definition of the class *Attacker* scheme is described in description logic (DL) as follows:

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

The *OntologyForCross-Site-Scripting* ontology in the following figure is intended to represent the full range of instances such as (vulnerability, security layers, web browsers, web applications, technology language, types of XSS attacks, objects properties, data properties, etc.) that can be involved in an XSS attack scenario. Every parenting class (Attacker, Attacks, Technology, Victim's environment, Vulnerability, Consequences, attack pattern) is further subdivided into several sub-classes that are interconnected based on their intended functionalities. Figure 11 shows the class hierarchy of the ontology, and figure **??** elaborates more on the functionalities of the ontology.

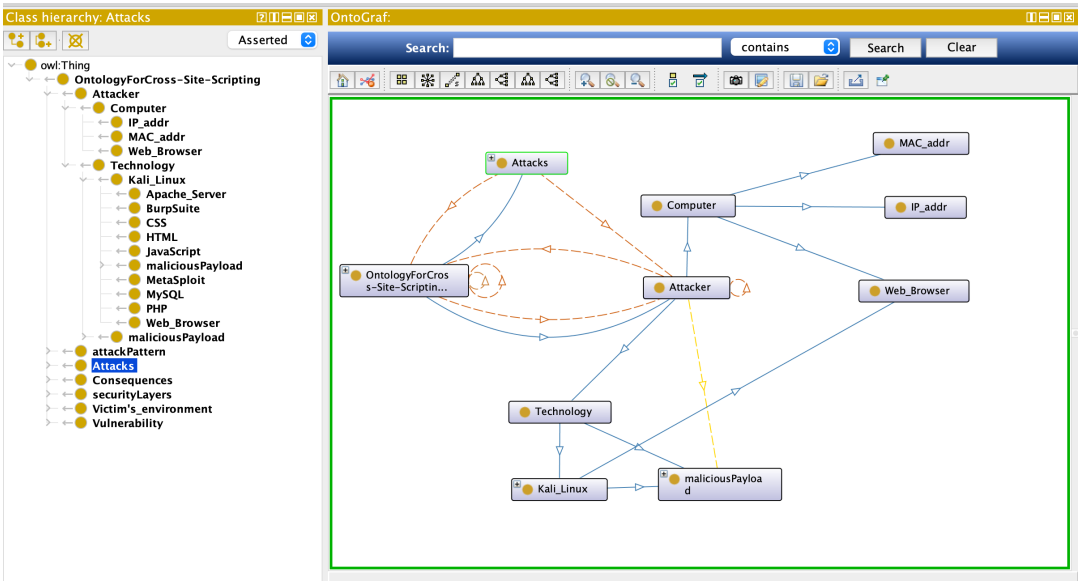

FIGURE 10: Ontology construction: more detail about the Attacker class in Protégé.

- The attacker uses technology such as HTML, JavaScript, Burpsuite, PHP, Apache server, and MetaSploit to build a malicious payload from his machine.
- His purpose is to launch an XSS attack against a target, hence the victim's environment.
- After the *Reconnaissance* phase, the attacker decides to manipulate the victim, while the last is exploring a target vulnerable website.

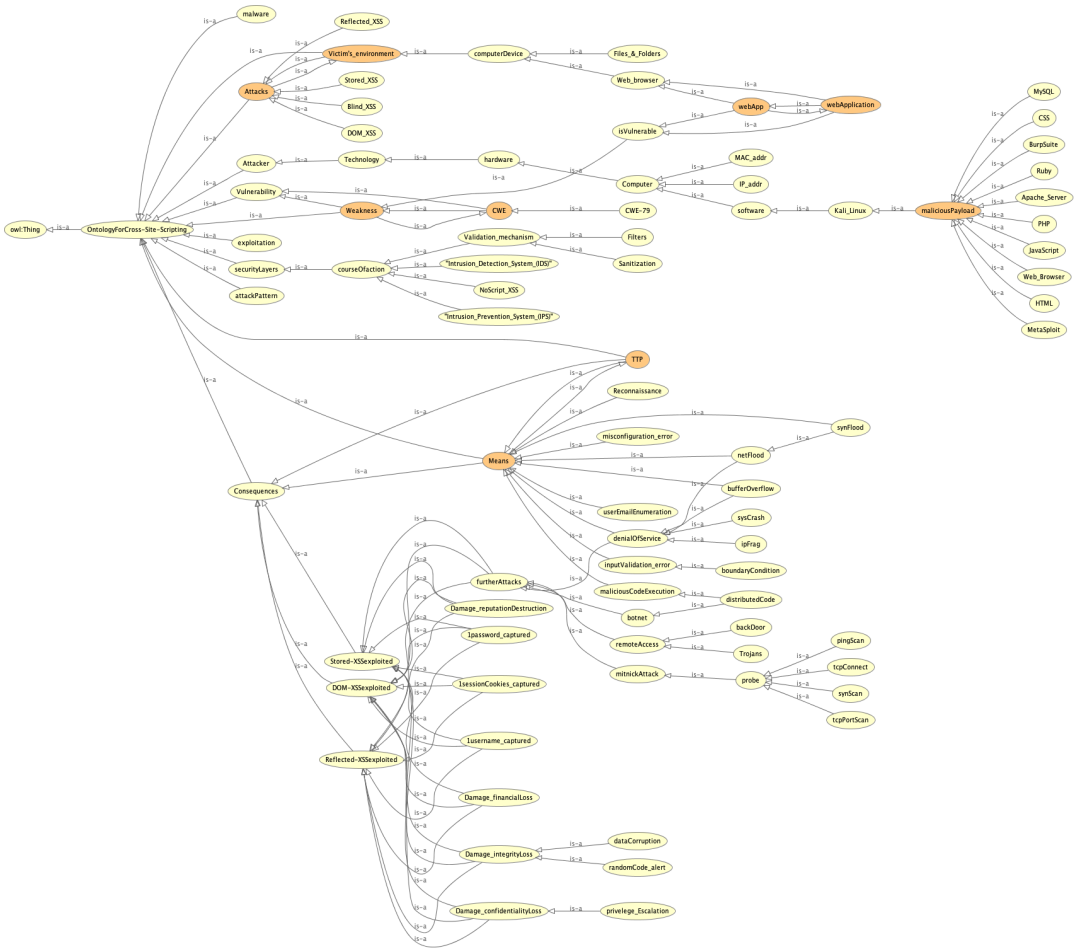

FIGURE 11: Description of the ontology, generated from OWLViz plugin.

Our ontology though describes briefly the security layers, we did not focus too much on the mitigation of the XSS attacks. The approach is more related to detection. 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.

- Therefore, during the testing assessment if the malicious payload does not trigger any alert, does not reflect in the response tab, and/or does not resides in the web application; then, the ontology assumes that some filters, a sanitization were applied from both the client and the server-side of the web application.

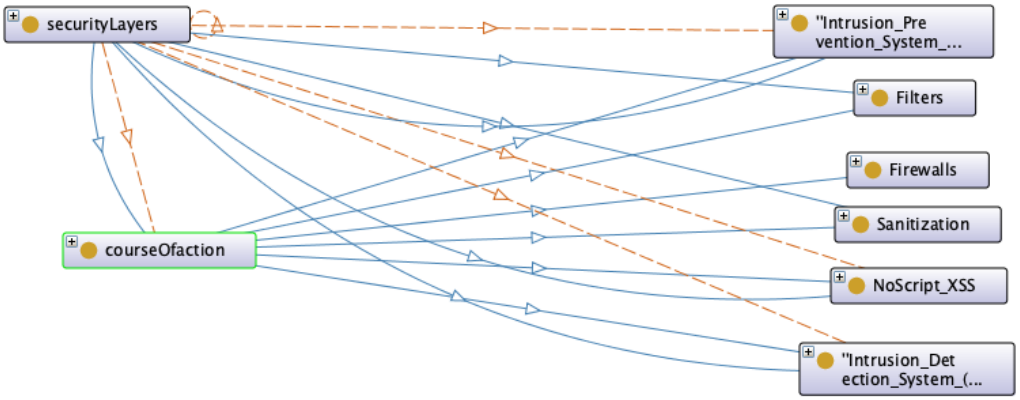

FIGURE 12: Class security layers description

The sub-class *Firewalls* helps the administrator of the website blocks any malicious-looking activity in the website in real-time such as, XSS attacks, SQL injections, etc. The sub-classes *Filters, Sanitization* are meant to be implemented most of the time by the web application developer during the coding process. The other sub-classes can be setup by the administrator of the web program. The class *securityLayers* assumes that a maximum of security measures are met in the web application environment.

$hasInput(?P, ?Q) \sqcap hasPart(?R, ?HTML) \sqcap contains(?method, ?openTAG) \sqcap contains(?method, ?param) \sqcap \exists securityLayers(?R, sec) \sqcap sentBy(?attack, ?P) \rightarrow isDetected(?sec, ?attack)$

- The class *furtherAttacks* is a sub-class of each of the four (4) types of XSS attacks which predicts that if the attacks are exploited, then the attacker may decide to perform additional attacks. The ontology classifies this activity as **consequences** that may result after an attack is successful.
The class *Attacks* and *Victim's_environment* are not subclasses of the class *Attacker*. Thus, for the attacker to launch his tools against the victim's system, he first needs to embed his malicious payload in a link and has the victim click on it. Or, he needs first to store the malicious payload into the vulnerable web application and wait for a victim to visit that web application. As a result, as soon as the victim performs the attacker's intention (with or without knowing), then a connection will be created between the attacker's server and the victim's environment.

0.1 Objectives

The dissertation thesis focuses on three (3) main topics, namely "Cross-site-scripting (XSS) attacks and vulnerabilities", "Penetration Testing" and "Ontology for the detection of XSS".
1. The chapter that elaborates on the XSS attacks and vulnerabilities focuses on the various types of the attacks, the definition of XSS vulnerability and XSS attack, techniques used to detect the vulnerability and the attack, strategies to use to prevent the attacks from occuring, some statistics of how cyber attacks affected industries in the past.
2. The chapter that deals with the penetration testing concept explains different types of the testing, its importance in real world. It also shows the occurrence of the

attacks with a few examples for demonstration purposes. It deals with the discovery of the vulnerability, and exploitation of the vulnerability.
3. The chapter that covers the topic of ontology clearly proves that the ontology itself cannot mitigate, detect any attacks. But, in order to be significant, it has to be properly used with some rules defined and works on a cyclic basis. It explains how the establishment of rules are crucial building an ontology.

0.1.1 Techniques to Detect XSS Vulnerability in a Web Application

We should be aware that the XSS attack is an intense and convoluted topic. Identifying XSS vulnerability in a website requires several penetration tests, based on the difficulty that one may find on each page of that site. The difficulty depends for the most part on how the specific web application is utilizing proper measures to resist the attack. Consequently, examination of reflected codes and payloads in a Document Object Model (DOM) is vital. Utilizing the following steps (fig 13) for checking if a web application is defenseless against XSS attack, (as per the author Gupta), could be an awesome method to start.
- Open up your browser and access a specific web application for the test. On that site, search for parts that require input, similar to search fields, comments, registration, etc.
- Now, enter any string into these spaces and press "enter" to submit that string to the website server.
- Now, we can peruse cautiously to check if the first condition holds. The first condition states that "test the HTTP reaction (or response) website page of the server for the specific string which was submitted by the guest". Thus, if that HTTP response incorporates a similar string, then the website can be exploited by XSS attacks. Else, assuming that the HTTP response does not contain any user-input string, check for the following condition.
This condition states that: "simply enter any JavaScript string, and submit it to the server by pressing 'enter'". E.g. *<script>alert(123)</script>*
- After sending that string to the server, if the server responds with a similar string, at that point the site is exposed to XSS attacks. If there is no such response, go for the next condition.
- The last condition states that: after pressing "enter" in the previous step, check the source code of that site and search if something seems to be like the JavaScript payload entered. If any element of that string is discovered, then the web-page is vulnerable to XSS attacks.

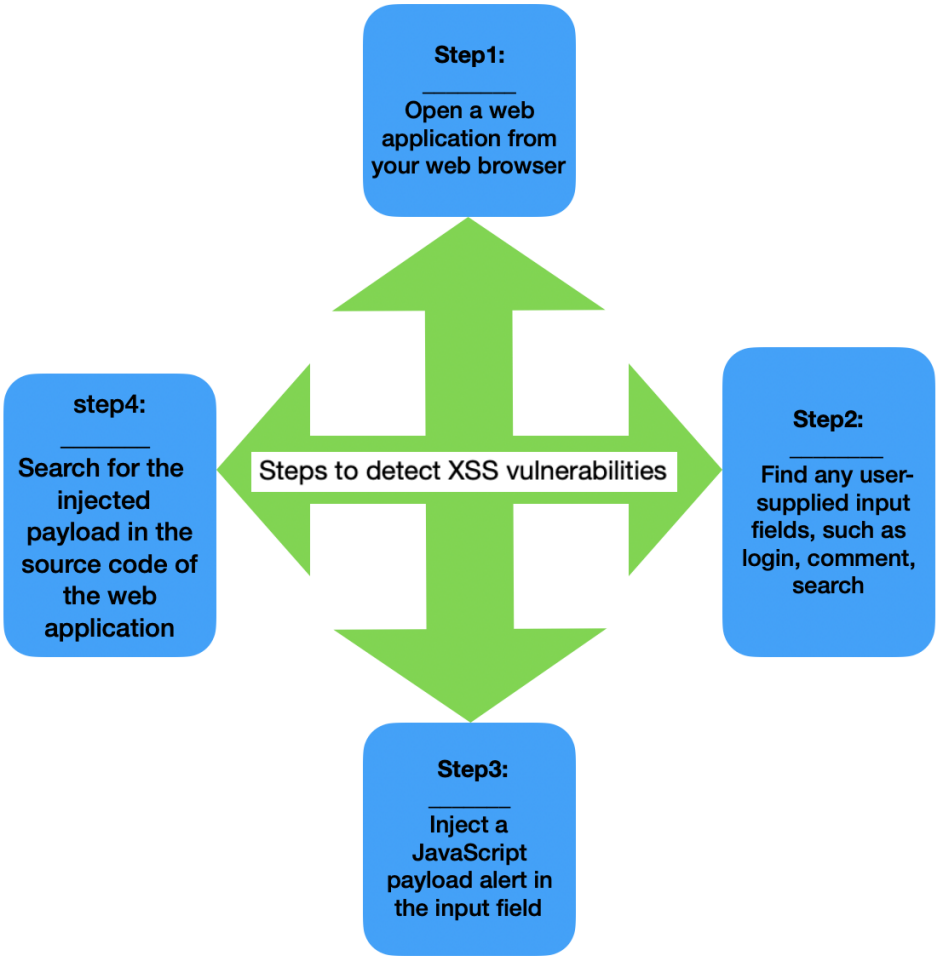

FIGURE 13: Steps to discover XSS attack Jean R. Dora, 2021

There also exists a "Blind XSS". It is a subset of the Persistent-XSS, where an attacker blindly deploys malignant payloads in web application pages that are stored constantly on the target server. Moreover, the stored malicious payload is reflected in various other applications which are connected. Note that, this type of attack only triggers when the adversary's payload is stored by the webserver in a database and runs as an evil script in another part of the application or completely another web application.
The majority of the XSS methods used to identify this vulnerability are not adequate to recognize this type of XSS. A machine learning-based technique can be completely used to identify or detect the blind XSS. Testing results help to identify pernicious payloads that are likely to lay in the database through web applications.

0.1.1.1 Analysis Techniques used for the Detection of XSS on the Client-Side and the Server-Side

The discovery approaches can be: a) Static Analysis, b) Dynamic Analysis, c) Hybrid Analysis, and d) Data-driven Analysis. The settlement of a defense technique on the client-side can be set up either on the user's browser (the client) as filters or plug-ins or even on a proxy server.

a) Static Analysis
As per Chess and McGraw, the static analysis technique mostly centers around the web application's source codes. It examines carefully the codes for potential discovery vulnerabilities.
It is clear that code analysis is an absolute necessity, since XSS attacks happen in web applications, and the whole work is about cybersecurity. In any case, what we need to separate is that the static analysis does not plan for penetration testing, to inject payloads.
XSS filters are a case of a static tool. The idea behind the scene is that in the Reflected XSS attack, the script resides in both HTTP Request and Response which are exchanged between the client and the server. Giorgio Maone presents another XSS filter known as (NoScript, 2020), it works as an add-on. They are broadly available for Firefox and Seamonkey browsers. It facilitates the execution of Java, Flash, JavaScript, and other plug-ins only if they are coming from the acceptable, trusted source chosen by the user. For more information about static XSS detector, please see Li, 2020; Abdalla, 2020.
Rao et al. 2016 proposes a filter known as the XBuster filter, which was used as an extension to the Firefox browser. It fundamentally employs a substring matching algorithm. The primary task of the XBuster filter is to survey the JavaScript and the HTML contents which are in HTTP Request separately.
However, the work of Nguyen et al. Nguyen, Maleehuan, Aoki, et al. (2019) Maleehuan, 2019 demonstrates that static tools have also their downside since they produce false-positive alerts (vulnerabilities detected that do not exist) and false negatives, which means that real vulnerabilities not found. As a consequence, a final audit of a Security Analysis Static Tool (SAST) tool report is required to confirm each security vulnerability.
For more knowledge about benchmarking static analysis tools, *please see page 1558 from the book* Higueral, 2020.
b) Dynamic Analysis
This technique generally focuses on penetration testing. It tries various payloads on the web application's possible injection points, and also makes some extra analysis based on responses.
c) Hybrid Analysis
This category is a combination of the two analyses listed above. It increases the security against XSS efficiently. According to Upasana, 2018; computationally static methods are more expensive and suffer from the ineffectiveness to make decisions. Pan and Mao made the update their framework, in 2017, aiming to propose solutions to DOM-based XSS attacks in the browser extension. That new one is called the DOM-sourced XSS attack, and it proposes to use hybrid analysis.
d) Data-driven Analysis
Besides static, dynamic, and hybrid methods, there exist "data-driven" analysis; it is a new and popular technique developed mainly for cybersecurity analysis. Data-driven analysis is used to scrutinize the XSS payloads instead of analyzing website vulnerabilities.
Apart from DOM-based XSS attacks, the Reflected and Stored occur mostly due to a low-security level on the server side. Again, as described above, it involves the same techniques as for the client-side. Gupta, 2016, introduces XSS-Secure detection for XSS worm propagation. It is a service provided for Online Social Networking (OSN-based) multimedia websites on a cloud platform.

0.2 Brief Methods to Prevent XSS Attacks

Forestalling XSS is banal or trivial in some cases, and again however can be a lot harder relying upon the intricacy of the web application and the manners in which it handles user-controllable information.
Usually, efficiently preventing cross-site-scripting vulnerabilities is likely to involve a combination of the following measures:

1. ***Filter any user-input on arrival***. At the point where input is injected and received, the mechanism filter should be applied as soon as possible.
2. ***Validation***.It is the comparison of input against white list.
3. ***Sanitization***. It is the combination of escaping, filtering and validation mechanisms that assures malicious code cannot be injected into your website.
4. ***Encode information on output***. At the point where any possible response is to be given from the server, the data should be encoded first to stop it from being interpreted as an active substance (active content). Contingent upon the yield setting, this may require applying mixes of HTML, URL, JavaScript, and CSS encoding.
5. ***Use relevant, suitable response headers***.To prevent XSS in HTTP responses that are not planned to contain any HTML or JavaScript, you can employ the Content-Type and X-Content-Type-Options headers to guarantee that browsers interpret or decipher the responses in the way you plan.
6. ***Content Security Policy (CSP)***. As a last line of the guard, you can utilize CSP to lessen the seriousness and the severity of any XSS vulnerabilities that happen. *For more information, please see* PortSwigger, 2021.

0.2.1 Reflected XSS into a Template Literal with Angle Brackets, Single, Double Quotes, Backslash and Backticks Unicode escaped

In this example, we inject any random characters in the "Search" area into the website and press "enter". We use this payload for example:
<script>alert(’1 ”${}Yes”/ ’)</script>
NOTE: We use this payload above just in order to try all the intended characters. You could have used something like this *<anything!@#$%(){/}>’;/*
We then catch its behavior through BurpSuite. After catching it, we send the capture to the "Repeater" to see how it appears in the HTML code DOM-display. In both tabs (Proxy, Repeater), we see that the payload which we have injected does not come into our burp suite that same way. Instead, we observe that the random string is reflected into a JavaScript template block. Some characters are automatically encoded, hence indicating the existence of some filters.
To verify, you can use your burpsuite decoder/encoder to try the incoming text or any online tool made for this purpose. For this specific type of encoded-text online-toolz, we use the online tool to encode the text/payload above.

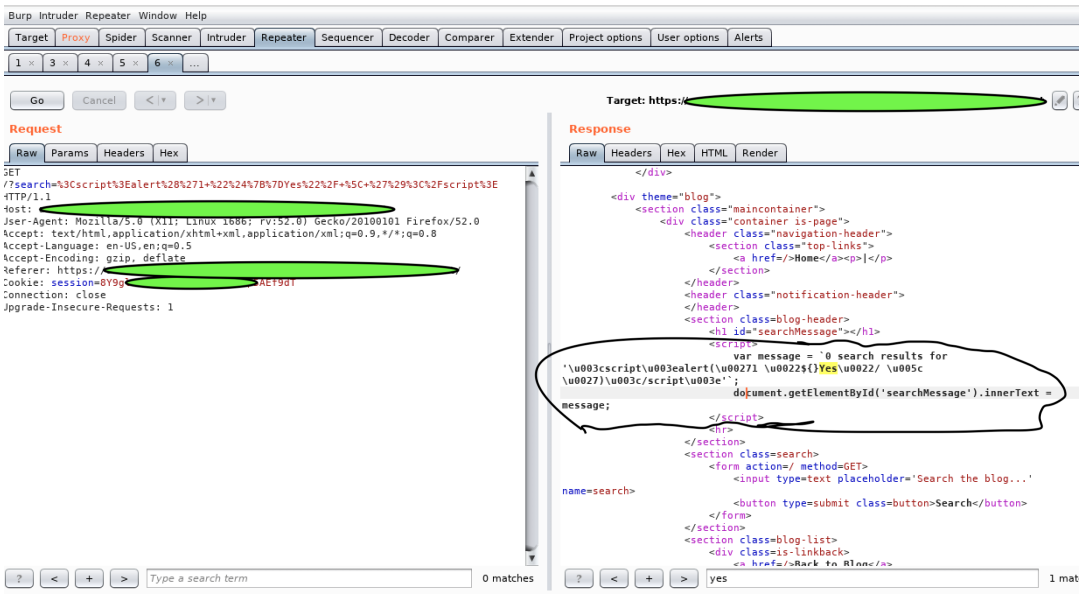

FIGURE 14: Reflected XSS into a template literal with angle brackets

To bypass this filter security and try to exploit the vulnerability, we perform an XSS attack using the non-encoded characters in the payload above **${}** that calls an alert function inside the JavaScript template string. *Please see Hacktricks for more information*. Living your browser still connected with burp suite, when entering the payload above into burp suite "Repeater" and clicking on "Go", the pop-up alert will trigger on your browser when stopping the "intercept". Or simply, we can just right-click on the blank area in the "Repeater" block and choose "Request in browser" –> "In current browser session", then copy the link and paste it to your browser and press "enter/return" to trigger an alert.

0.2.2 Capture Passwords by Exploiting XSS Vulnerabilities from a Website

Once found the vulnerable website, the attacker can decide to exploit the users who visit that site. This is where the importance of "How, where, on what to click" comes into play. Just a single click can jeopardize the user, no further actions are needed. Note that, a vulnerable website is a web application that contains weaknesses. If those weaknesses are exploited, then the reputation of that web application can be jeopardized, as well as the data privacy of users of that application.
The following figure shows a step-by-step process of how this situation can happen.

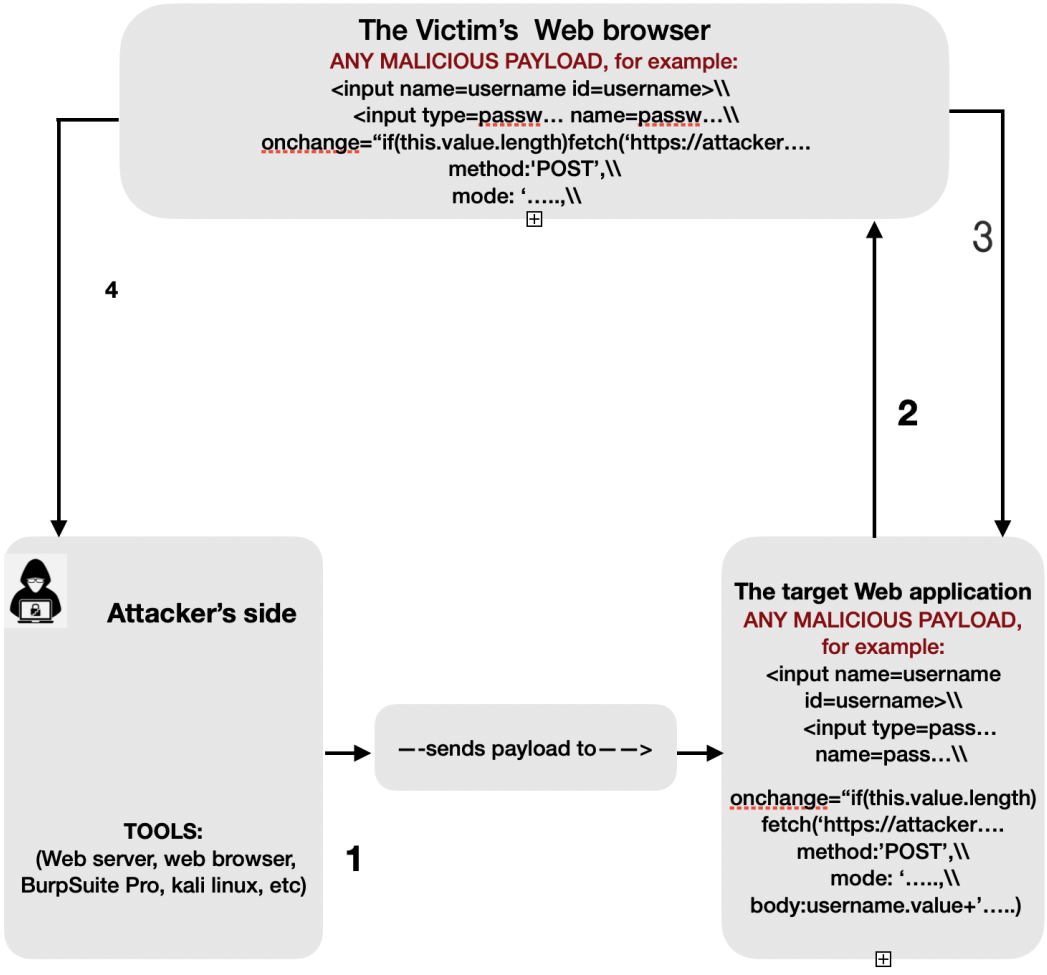

FIGURE 15: Dangerous Stored XSS attack. Jean R. Dora, 2021

Description:
In a blog comment section for example. Assuming that the section is vulnerable to XSS attacks. So the attacker creates a wicked payload, and injects it into that appropriate area of the web application, and posts it.
Step 1: The attacker opens his tool (Kali Linux and Burpsuite Pro), creates his code which includes his server link; then, injects that code or stores that code into the vulnerable section of the web application. The attacker leaves his malevolent server opens all time to keep the connection up.
<input name=username id=username>
<input type=password name=password
onchange="if(this.value.length)fetch('https://Jean-attackerforexample.com',
method:'POST',
mode: 'no-cors',
body:username.value+':'+this.value
);">
Step 2: Now the payload is sleeping onto the vulnerable website in a form of a clickable string. Whenever a user visits that site and reads comments, nothing happens. But, as soon as the user clicks on the particular clickable string to read more, then

the attack gets executed without his/her consent.
Step 3: By clicking on the string, the victim's web browser accepts the request command from the web application and divulges the user's credentials to the website. And since the injected malicious payload was considered as part of the website code, (**Step 4:**) now the website naively responds to the attacker with the credentials of the user who clicks on the clickable string. The attacker will get the response in an HTTP interaction. Thus, the attacker can log in as the victim user by using the victim's username and password.

To detect XSS vulnerability, there should be a combination of different processes, and steps to be undertaken by the attacker depending on the web application in question.
A synthetic approach to XSS attacks is a comprehensive approach and the practical approach to the attack into a single approach. It helps web developers, website administrators, researchers, and any internet user know the cause of this attack and the cause of this vulnerability, the impact it can have on an organization's reputation, and from a business perspective. It also helps us know what an attacker can do, how to minimize the possibility of being exploited as a normal user, and how to minimize vulnerability as a web developer. Moreover, from an organizational point of view, to explicitly inform employees of how this attack can happen.
From the previous chapters, we have seen one of the most popular strategies used by attackers to exploit the XSS vulnerabilities is by crafting a malicious script into a URL format and sending it to a target user. We have also seen how the attacker can store his malicious payload onto a vulnerable website 0.2.2. An attacker searches mainly for web applications that contain user-input (comment field, registration, login, search, ...) to inject his payload to analyze the response he will obtain. A payload can be a file that contains several malicious scripts. Generally, attackers do not inject scripts onto a website one after one to launch the attack on a web application to detect the vulnerability as it can be time-consuming. There are a lot of rigorous tools that can help them in this manner, such as Burpsuite Professional. By reading the previous chapters, now an internet user knows that some best practices can allow him to reduce the possibility of being exploited such as avoiding opening suspicious websites in the same browser with a bank account, avoiding clicking on suspicious links, images, etc.

0.2.3 The Reasons of Using Ontological Approach

Ontology is an interesting proposition for converging the description of a data model and the related rule base into a single application. Ontologies developed in Web Ontology Language derive many benefits afforded by the semantic web stack. The purpose 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 verified for consistency or utilized as a basis for inferences on that peculiar knowledge. Flexibility in defining any concept to the preplanned level of details is a well-known feature of the ontological model. Many reasons can be highlighted, and we listed a few below:
a) To share a customary understanding of the structure of data, information between people or software agents.

To facilitate the reuse of domain knowledge (Domain, in this viewpoint, means the most universal classes, the knowledge base. Classes are divided, subdivided further into many branches of the hierarchy.).
b) To make domain assumptions explicit.
c) To divide domain knowledge from the functional knowledge.
d) To scrutinize domain knowledge.
A diversity of issues may spring out when a non-ontological approach is used.
_ It is ordinarily a good practice to implement or use proactive detection tools. Many web-based detection tools are reactive, that is to say, they are operated according to the specific rules set by the administrator. The attack can only be stopped if the exact signature of the attack is not only recognized by the system scanning but also present.
_ It is easy for an attacker to launch an attack by a slight modification of the signature, since the majority of the existing methods are signature-based, which hold the syntax of the attack.
_ Statistical mechanisms used in Intrusion Detection Systems largely provide a feasible solution for the network layer. However, this solution is not effective at the application layer because it emphasizes the character distribution of the input and does not take into account its contextual nature.

1 Summary

Depending on the project in question, an ontology may be very sizeable, but still understandable. Building a new ontology from scratch can be enigmatic. Fortunately, thanks to the feature of "re-usability", there will be simply, a need of altering an existing ontology to satisfy your goals in the reconstruction. It is obvious that an ontology in cyberspace necessitates the separation of duties; besides, the supervision of the system is of great importance (it can be achieved by the administrator or any trusted individuals) for the process to provide better results in terms of preventing incidents or attacks and enhancing the security.

This thesis presents the structure of the ontology-based semantic web to address the topic of XSS vulnerabilities and attacks. We have implemented some rules that enable the ontology to better address the detection of vulnerabilities and the detection of the attacks through audit logs. The ontology also proposes a technique that requires the set up of security layers to be done correctly if the mitigation of the attack takes into consideration. For instance, the feature of "audit and alert" to the administrative department is used whenever (if in case) a skeptical, suspicious payload penetrates all the other security layers which were set by the security engineers.
In Cross-site-scripting attacks, the analysis is normally done on the input fields in a web application. For example, on a web page where the user is asked to enter his password, the user can inject a JavaScript payload in an encoded format like the following 1. Therefore, when the input field passes through the analysis engine, it is looked over for encoding characters. In case it is encoded, then it would be first decoded and then would be scrutinized for an anomaly. After that, the request is passed on to the validation mechanism and analyzer module. There, semantic rules are being used for checking malicious content. However, the ontology in itself cannot enhance the security of a network institution, thus it has to take into account all the security mechanisms, and be properly used. That is to say, a proper validation mechanism has to be put in place for any user-supplied input, installed software on the server-side has to be up-to-date, and data resources and human resources for the separation of duties have to be well established.

There exist two very critical factors that can impede the security performance, they are "browser" and "user". Which browsers are used (Tor, Brave, Opera, Internet Explorer, Firefox, etc.), and most importantly if they are outdated. The sound judgment of the user is also of great importance when it comes to clicking on links. Therefore, in our future work, the factor of using outdated browsers, and the type of browsers will also be addressed, to efficaciously mitigate the attacks.

The proposed ontology is a modern approach for application of semantic technologies particularly in web application security for XSS attacks. The inference capability facilitates, enables our reference scheme to attain more stamps of approval to such an extent that, it gives ability of detecting complex web attacks. Additionally, by employing semantic rules, the scheme becomes more reliable and more flexible.

Additional Information

%3Cscript%3E%20alert(123)%20%3C%2Fscript%3E

<script>alert(123)</script>

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_ℓ)	$\top \sqsubseteq \forall R.C_i$
[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_ℓ)	$\top \sqsubseteq \forall U.D_i$
[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 1.1: OWL DL axioms and facts Baader, 2010

Bibliography

Abdalla, W. (2020). "Web Application Security: An Investigation on Static Analysis with other Algorithms to Detect Cross Site Scripting." In: *MDPI, Applied Sciences*.

Baader, Franz (2010). "THE DESCRIPTION LOGIC HANDBOOK". In: *United Kingdom at the University Press, Cambridge*. URL: www.cambridge.org/9780521876254.

Gupta, Shashank (2016). "XSS-secure as a service for the platforms of online social network-based multimedia web applications in cloud." In: URL: https://doi.org/10.1007/s11042-016-3735-1.

Higueral, Juan Bermejo (2020). "Benchmarking Approach to Compare Web Applications Static Analysis Tools Detecting OWASP Top Ten Security Vulnerabilities". In: 64, pp. 1555–1577.

Jean R. Dora, Karol Nemoga (May 2021). "Ontology for Cross-Site-Scripting (XSS) attack in Cybersecurity". In: URL: https://www.mdpi.com/2624-800X/1/2/18.

Li, Chenghao (July 2020). "Cross-Site Scripting Guardian: A Static XSS Detector Based on Data Stream Input-Output Association Mining". In: *MDPI, applied sciences*. URL: https://www.mdpi.com/2076-3417/10/14/4740/htm.

Maleehuan (2019). "Reducing false positives of static analysis for sei cert C coding standard." In: *IEEE Computer Society, Montreal, Canada.*

NoScript (2020). In: URL: https://noscript.net/.

PortSwigger (2021). "How to prevent XSS attacks". In: *PortSwigger Web Security Academy*. URL: https://portswigger.net/web-security/cross-site-scripting.

Upasana (Sept. 2018). In: URL: https://www.sciencedirect.com/science/article/pii/S1084804518302042.

Zoznam publikácií a ohlasov

- Ontology for Cross-Site-Scripting (XSS) Attack in Cybersecurity, by Jean R, Dora., Karol, N.

J. Cybersecur. Priv. 2021, 1(2), 319-339; https://doi.org/10.3390/jcp1020018
Received: 4 April 2021 / Revised: 3 May 2021 / Accepted: 18 May 2021 / Published: 25 May 2021

- Clone Node Detection Attacks and Mitigation Mechanisms in Static Wireless Sensor Networks, by Jean R, Dora., Karol, N.

J. Cybersecur. Priv. 2021, 1(4), 553-579; https://doi.org/10.3390/jcp1040028
Received: 5 July 2021 / Revised: 26 August 2021 / Accepted: 15 September 2021 / Published: 24 September 2021

Účasť na konferenciách

- Smolenice June 2018
- Hack In Paris June, 2019, Maison De La Chimie, Paris (France)
- October 2019, Vienna (Austria) "Conference about Ontology"
- February 2022, Paris (France) - International Conference on Cybersecurity and Threats (ICCT 2022: XVI.). ***Certificate of Best Presentation Award Received by Jean Rosemond Dora***
- Smolenice June 2022

Granty

- This "Ontology for Cross-Site-Scripting (XSS) Attack in Cybersecurity" research was funded by the Institute of Mathematics, Slovak Academy of Sciences (MUSAV), Grant VEGA 2/0109/18 and APVV-19-0220.

- This research "Clone Node Detection Attacks and Mitigation Mechanisms in Static Wireless Sensor Networks" was funded by the Institute of Mathematics, Slovak Academy of Sciences (MUSAV), Grant VEGA 2/0109/18 and APVV-19-0220.