Cross Site Scripting Vulnerabilities in JAX-RS: A Security Approach

John Velandia*, Jessica Ortiz , Julian Sierra, Roger Guzmán
Faculty of Engineering, Universidad Católica de Colombia, Bogotá, Colombia.

* Corresponding author. Tel.: 573208538533; email: javelandia@ucatolica.edu.co
Manuscript submitted April 10, 2018; accepted October 1, 2019.
doi: 10.17706/jsw.14.11.530-547

Abstract: Restful services are concerned with the integration of software systems using HTTP as base. Research studies addressing security assessments over JAX-RS are scarce, even more in Cross Site Scripting (XSS), which is a sort of attack that consists of stealing data or phishing. Thus, the aim of this paper is to present an assessment of the vulnerabilities over JAX-RS implementations when a XSS attack is involved. The assessment comprises: (1) selection of attack methods, (2) programming and assessing of attacks throughout dynamic programming and recursive methods; (3) identifying the vulnerabilities by means of a mathematical model, which determines the level of security of implementations. As a proof of concept, a prototype is implemented to demonstrate how the guideline is applied. Additionally, controls are proposed for every vulnerability identified.

Key words: JAX-RS, Restful services, vulnerability, security, cross site scripting, dynamic programming, apache CXF, RestEasy, Jersey, Restlet.

1. Introduction

Software vulnerability is defined as a bug in software systems which causes an invalid output or to behave unintended way. The proliferation of restful services to integrate software systems around the world demand the constant assessment of security in restful services to guarantee the availability of services and the confidentiality of data.

Architectural styles have been used to integrate software systems. Representational Style Transfer (REST) is one way to develop an integration. It is visible in different contexts, specially, during financial transactions along different Web sites [1]. For instance, Amazon, during the process of purchasing a product, restful services are called to obtain the products due to sell. Thanks to the sturdiness of performance in REST, millions of financial transactions are possible in these sort of web sites.

In addition to performance, the security issue it is also important to address, in order to guarantee the users' trust and data confidence in RESTful services. However, the security breaches are present in all the architectures, for smaller that these are. Restful services are also vulnerable to Cross Site Scripting Attacks (XSS) [2], which are manners of steal data or phishing base on HTML tags or source code[3], [4].

Research studies addressing security assessments over JAX-RS are scarce, even more in Cross Site Scripting (XSS) [5], [6], thus the aim of this paper is to present a guideline that assesses the vulnerabilities and controls over JAX-RS implementations, specifically when a XSS attack is involved. In addition, the following research question is set out: Current implementations base on JAX-RS consider controls to mitigate Cross Site Scripting attacks or on the contrary, restful services based on JAX-RS are vulnerable to XSS attacks?

1) The principal novelties of this research comprise:
2) The selection of possible attacks methods over restful services, since not all the existing methods are eligible.
3) The programming of the selected methods using dynamic programming and recursively to assess vulnerabilities in the following JAX-RS implementations: that RestEasy, Restlet, Apache CXF and Jersey.
4) Mathematical model is proposed to evaluate the level of security base on the founded vulnerabilities.
5) A software prototype is deployed on Internet in order to simulate the selected attacks methods over any JAX-RS implementation.

2. Fundamentals

Detection of software vulnerabilities are performed throughout different analysis techniques, however the most used are the static and dynamic technique. Static technique encompasses the analysis of the source code without executing it, only analyzing source code it could determine if exist a vulnerability. Dynamic technique consists of running the detection of vulnerabilities at run time to confirm if it is vulnerable; if a vulnerability is found at this case a denial of service could arise.

2.1. REST

Representational State Transfer (REST). It is a style of architecture implemented over HTTP. It is concerned with the integration of software systems. Java API for RESTful Web Services (JAX-RS) is a specification framework that defines how plain Java objects are bound to URIs and HTTP operations using Java annotations[7]. This framework is important since this establishes a standard way to handle incoming and outgoing server requests and how information flows from one restful service to another, consequently JAX-RS facilitates and simplifies restful service implementation. Providers have been risen to implement JAX-RS, supporting the REST principles: *Addressability, uniform interface, content representation, stateless interaction and hypermedia*. In addition, quality attributes such as security, thread-save, concurrency and performance are offered by providers. However, there is not yet research studies regarding which implementation is better than other in terms of these quality attributes. Considering that exist a wide range of quality attributes and each of them is composed of metrics and methodologies to evaluate them, the objective of this paper is to assess only the performance of the following implementations: Jersey, Resteasy, Restlet and CXF, because according to [8] they are the most used it for integrating information systems.

2.2. JAX-RS Implementations

JAX-RS is the acronym for Java API for RESTful Web Services, and it serves as specification for projects dedicate to develop implementations such as Jersey, RESTlet, RESTlet Apache CXF, among others. These implementations facilitate the development of web services in Java, because of its lightweight infrastructure that allows services to be built with minimal tooling (Oracle 2014). Moreover, the aim of these implementations is to integrate systems and applications using restful Web services over HTTP.

Implementations are connected to Plain Old Java Objects (POJOs) through runtime annotations, in this way resources and its actions are defined. For example, @Path annotation is used to locate the Java class that represents a resource. @GET annotation processes HTTP GET requests.

2.3. Cross Site Scripting – XSS

The Cross-Site Scripting attacks are a type of malicious code injection attacks, which it gets any kind of sensitive data stored in a trusted RESTful Service, allowing attackers extract data. XSS attacks are of three types:

Store Cross Site Scripting: is a direct attack, in which the code injection occurs through the use of HTML labels, JavaScript or Flash, the principal characteristic of this attack is the data theft and storage in one or more servers, where the malicious scripts are stored that contains data of users controlled by the attacker, known as the most common and dangerous attack.

Reflected Cross-Site **Scripting:** is a direct attack, occurs through victims' Web browser, likewise using the client's side to obtain the data input while the data is catcher, it's immediately used by the server Web.

DOM-based Cross-Site Scripting: is the less common attack, however, in this case, the attack is executed on the client side, the user page duplicate itself, given the data to other less trusted services.

3. The Assessment Methodology

The proposed methodology encompasses three phases:

Selection of attack methods. A review of the existing methods on HTTP ought to perform at this phase. The set of collected methods are selected based on cutting edge security sources.

1) Programming and assessing of attacks throughout dynamic programming and recursive methods;
2) Identifying the vulnerabilities by means of a mathematical model, which determines the level of security of implementations. Additionally, controls are proposed for every vulnerability identified.

4. Selection of Attack Methods

The selection of the attacks methods is useful to identify vulnerabilities and to determine the type of XSS attack to implement having clarified the similarities and differences between the XSS attack types.

4.1. Identification of Sources

Before the selection process begins, it is important to define the sources of information that are up to date and cutting edge of security attacks. Hence, the following sources are considered as a reference to gather the attacks methods: The Open Web Application Security Project (OWASP), The MITRE corporation, The Computing Technology Industry Association (CompTIA) and the International Information System Security Certification Consortium (ISC^2). These sources have international reputation for technical excellence and innovation in the area of security.

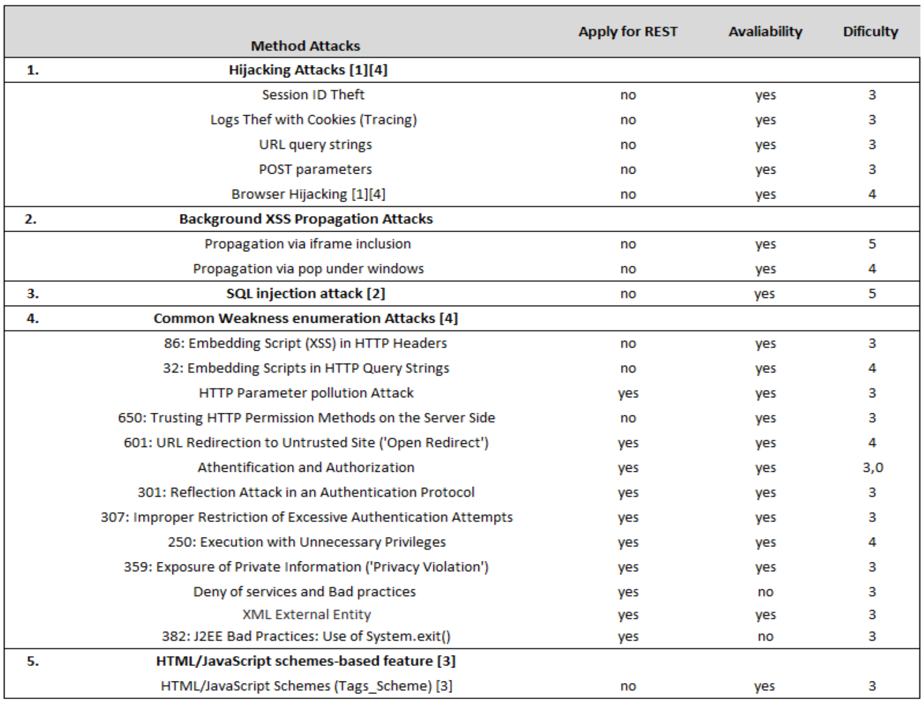
4.2. Classification Criteria

The XSS attack methods are uncountable if they are analyzed from the perspectives of programming language, attack objective and the attacker profile. Thus, the criterion to select the appropriate methods are listed:

Impact on REST. XSS attacks are normally deployed on Web sites, however none research studies present results about which XSS attacks fit restful services implementations. In such way, these criteria prevail over the rest of criterion. The possible values are yes and no. if the attack is yes meaning that the XSS attack apply for restful services implementations, on the contrary, the XSS attack does not apply.

1) Availability. It refers to the availability of the restful service after the XSS attack is executed. The possible values are yes and no. In case the value is yes, it means that the restful service is available despite XSS attack is performed. On the contrary, it means that the restful service is down.
2) Difficulty of implementation. It refers to the level of knowledge The implement difficulty for the attacker, including the script encoding and the strategy made to implement the attack and it is successful.
3) The knowledge level of the users and attackers necessary to prevent and do the attack

Finally, the using the defined a scale measure showed in Fig. 1 for each of these criteria, the methods are classified according to the difficulty level, from the least to the highest [18], [24], [25].



	Method Attacks	Apply for REST	Avaliability	Dificulty
1.	**Hijacking Attacks [1][4]**			
	Session ID Theft	no	yes	3
	Logs Thef with Cookies (Tracing)	no	yes	3
	URL query strings	no	yes	3
	POST parameters	no	yes	3
	Browser Hijacking [1][4]	no	yes	4
2.	**Background XSS Propagation Attacks**			
	Propagation via iframe inclusion	no	yes	5
	Propagation via pop under windows	no	yes	4
3.	**SQL injection attack [2]**	no	yes	5
4.	**Common Weakness enumeration Attacks [4]**			
	86: Embedding Script (XSS) in HTTP Headers	no	yes	3
	32: Embedding Scripts in HTTP Query Strings	no	yes	4
	HTTP Parameter pollution Attack	yes	yes	3
	650: Trusting HTTP Permission Methods on the Server Side	no	yes	3
	601: URL Redirection to Untrusted Site ('Open Redirect')	yes	yes	4
	Athentification and Authorization	yes	yes	3,0
	301: Reflection Attack in an Authentication Protocol	yes	yes	3
	307: Improper Restriction of Excessive Authentication Attempts	yes	yes	3
	250: Execution with Unnecessary Privileges	yes	yes	4
	359: Exposure of Private Information ('Privacy Violation')	yes	yes	3
	Deny of services and Bad practices	yes	no	3
	XML External Entity	yes	yes	3
	382: J2EE Bad Practices: Use of System.exit()	yes	no	3
5.	**HTML/JavaScript schemes-based feature [3]**			
	HTML/JavaScript Schemes (Tags_Scheme) [3]	no	yes	3

Fig. 1. Comparison methods attacks.

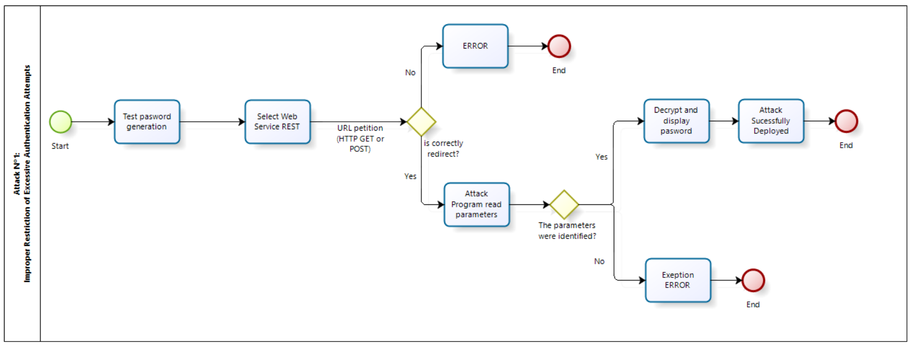

Attack N°1:
Improper Restriction of Excessive Authentication Attempts
Start
Test pasword generation
Select Web Service REST
URL petition (HTTP GET or POST)
is correctly redirect?
No
ERROR
End
Yes
Attack Program read parameters
The parameters were identified?
Yes
Decrypt and display pasword
Attack Sucessfully Deployed
End
No
Exeption ERROR
End

Fig. 2. BMPN attack N1 diagram.

The next step to find the vulnerabilities was to implement a prototype that represents weaknesses in RESTful services using a specific attack. The following are the implemented attacks: Improper Restriction of Excessive Authentication Attempts using Brute force Case 382:[9] and External Entity Attack (XXE) [10]. These attacks were selected because they are common attacks methods APEC [11] and common weaknesses on Internet CWE [12]. In addition, over 97% of the attacks founded it were not implemented because they do not affect the Restful architecture and because it needs web browsers to run, using several resources like cookies, headers of external resources.

5. Programming and Assessing the Attacks

5.1. Improper Restriction of Excessive Authentication Attempts using Brute force

The first vulnerability founded was focused in the number of authentications allowed to the users which is not defined just regulated by the server capacity [9].

To show how the attack works in the prototype in general and detailed terms the following code implementation image and BMPN (Business Process Model and Notation) were added.

An brute force attack works using an alphabet with 27 characters , obtaining all possible combinations given by the password length [13], [14], trying to find out the users password. Using an alphabet with 10 characters, alphabet a to j, the ASCII table [15], the attack try to find out the users password, obtaining all possible permutations is the specified range r given by the password length l [13], [14].

$$r = \begin{pmatrix} a\ b\ c\ d\ e \\ f\ g\ h\ i\ j \end{pmatrix} \quad l = (5\ characters)$$

The brute force algorithm has a time complexity of:

$$T(n) = \frac{1}{n!}\sum_{r=1}^{n} r$$

$$T(n) = \frac{n! + 1}{2}$$

$$T(n) = O(n!)$$

Therefore, the time complexity [16] is **NP (not polynomial)**[17], in consequence the program will do $(10!) = 30240$ iterations in $t = 91\ minutes$ to obtain one password. According to Bellman (1962) [18]a **NP** problem like **TSP (Traveling Salesman Problem) using dynamic programming** is $O(2^n n^2)$.

In order to reduce this complexity, and optimize the algorithm a new approach is gave trying to obtain better results like in the **TPS** case, the algorithm is re-implemented using Dynamic Programming principles [19].

1) Characterize a structure of an optimal solution
2) Recursively define the value of an optimal solution
3) Compute the value of an optimal solution (decision making)
4) Construct an optimal solution for computed information

The following ***Algorithm attackREST***( )

```
1)  validate = false;
2)  response = Failed Attack;
3)  attack(base, length)
4)  if (lenght = 0)
5)      return false;
6)  for (Range := InitialRange to FinalRange)
7)     newBase := base + ASCII;
8)    if (check(newBase) := true)
9)      response := NewBase;
10)     validate := true;
11)  if(validate: = false)
12)     return attack(newBase, length − 1, Maxlength);
```

Algorithm check(result)

```
1)    rd := RESTful Service HTTP response;
2)    line := "";
```

```
3)    while (rd $\neq$ NULL)
4)      if(line $\neq$ ERROR)
5)        return(true);
6)        if(line = ERROR)
7)          return(false)
```

Algorithm 1, was implement in the prototype; the complete code is added in the Appendix.

```
$Algorithm\ attackREST(\,)$
1)   $validate = false;$
2)   $response = Failed\ Attack;$
3)   $\boldsymbol{attack}(base, length)$
4)   $\boldsymbol{if}$ (lenght = 0)
5)        $return\ false;$
6)   $\boldsymbol{for}\ (Range := InitialRange\ to\ FinalRange)$
7)      $newBase := base + ASCII;$
8)     $\boldsymbol{if}\ (check(newBase) := true)$
9)        $response := NewBase;$
10)       $validate := true;$
11)   $\boldsymbol{if}(validate := false)$
12)     $return\ \boldsymbol{attack}(newBase, length - 1, Maxlength);$
```

```
$Algorithm\ \boldsymbol{check}(result)$
1)     $rd := RESTful\ Service$ HTTP $response;$
2)     $line :=$ "";
3)    $\boldsymbol{while}\ (rd \neq NULL)$
4)      $\boldsymbol{if}(line \neq ERROR)$
5)        $\boldsymbol{return}(true);$
6)        $\boldsymbol{if}(line = ERROR)$
7)        $\boldsymbol{return}(false)$
```

Algorithm 1: Attack Improper Authentication Attempts using Brute force

```
The $Algorithm\ attackREST(\,)$
1)   $validate$ = false;
2)   $response$ = Failed Attack;
3)   $attack$(base, length)
4)   if (lenght = 0)
5)        return false;
6)   for ($Range$ := InitialRange to FinalRange)
7)      $newBase$ := base + ASCII;
8)     if ($check$(newBase) := true)
9)        $response$ := NewBase;
10)       $validate$ := true;
11)   if($validate$: = false)
12)     return attack(newBase, length − 1, Maxlength);
```

```
Algorithm check(result)
  1) rd := RESTful Service HTTP response;
  2) line := "";
  3) while (rd ≠ NULL)
  4)   if(line ≠ ERROR)
  5)     return(true);
  6)     if(line = ERROR)
  7)       return(false)
```

Algorithm 1 works with the recursion used for dynamic programming [20]-[22] but not oriented to this by the same nature and behavior of the brute force attack [14], [23]. With initial parameters like the length of the password, an dictionary which is delimited of a set ASCII symbols [15], characters or numbers that can be part of password , in this case letters from a to j.

According to the victim's profile having in count the *length*; as you see in the ***Line 1*** and ***Line 2*** the initial values are *false*, also the *Length* are defined again

In the other hand, using threads the n*ewBase* will increase one by one until get the *Range* according to the *Base* and the ASCII delimited symbols until all the possible combinations are test, this *newBase* is Checked **Line 8**, if the result is *true*, returns like response ***"Failed Attack"*** at same case if the length is 0, because the character was not found in the alphabet defined.

But if the results is *false* the *function Attack* take the values moving a space to the right and starts to combine the possible combinations, starting with a again, Line **3 to 12** and finally when the length of the password is evaluated, returns a *response* to the petition send by the *Class check*.

The time complexity for this algorithm is obtained first calculating the complexity time of the function check which is an algorithm called by the algorithm attack, affecting the computational time complexity:

```
Algorithm check(result)
  1) $C_1 = 1$
  2) $C_2 = 1$
  3) $C_3 = n + 1$
  4) $C_4 = n$
  5) $C_5 = n$
  6) $C_6 = n$
  7) $C_7 = n$
```

$$T(n) = C_1 + C_2 + C_3(n+1) + C_4 + C_5 + C_6 + nC_7$$
$$T(n) = c + bn$$

```
Algorithm attackREST( )
  1) $C_1 = 1$
  2) $C_2 = 1$
  3) $C_3 = 1$
  4) $C_4 = 1$
  5) $C_5 = 1$
  6) $C_6 = n$
  7) $C_7 = n$
  8) $C_8 = n$
  9) $C_9 = n$
  10) $C_{10} = n$
```

11) $C_{11} = \text{n}$
12) $C_{12} = \text{n}$

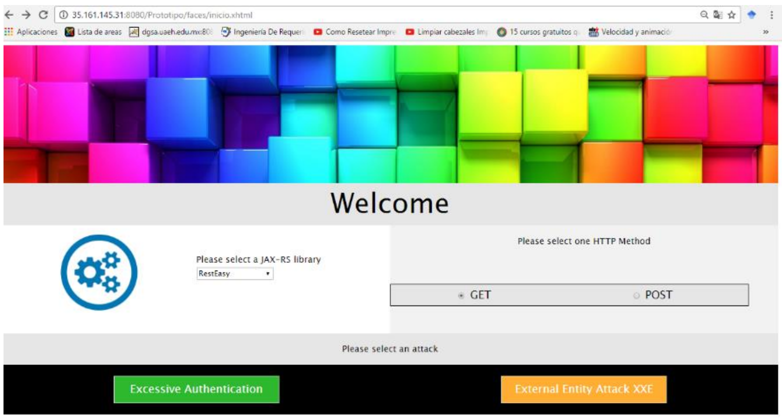
For the better and worst case:

$$T(n) = 0 \; if \; n = 0$$
$$T(n) = T(n-1) + a$$

In that way the cost is defined by:

$$T(n) = \begin{cases} 0 \; if \; n = 0 \\ \displaystyle\sum_{InitialRank}^{FinalRank} T(n) + dn \end{cases}$$

The summation obtained for the complexity time of the new algorithm takes to a cubic or quadratic order with is better than a factorial order. By that reason the response time will be better than the time take before. It is important add, the limited scope which the testing of this attack was done, given by the password length of 5 characters. The execution time for passwords with more characters takes an exponential time depending on the server capacity. Also the attack was tested obtaining login to the URL :(http://35.161.145.31:8080/Prototipo/faces/inicio.xhtml) these results:


35.161.145.31:8080/Prototipo/faces/inicio.xhtml

Welcome

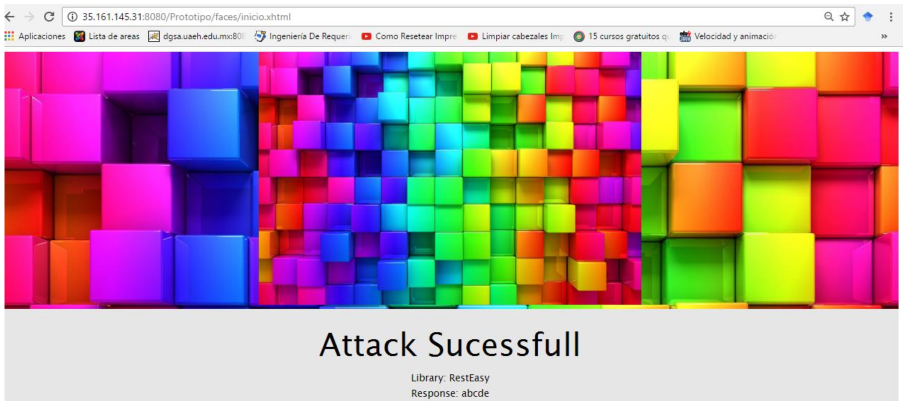
Please select a JAX-RS library: RestEasy

Please select one HTTP Method: [✓ GET] POST

Please select an attack

Fig. 3. Attack N1 testing using GET or POST.

The user chooses one of (4) RESTful services and the method, the service also giving like response the user password `abcde` (burnet in the source code):


35.161.145.31:8080/Prototipo/faces/inicio.xhtml

Attack Sucessfull

Library: RestEasy
Response: abcde

Fig. 4. Attack N1 response from GET/POST HTTP method.

In this case the runtime to break a password with a length of 5 letters was 9 minutes, 540 s approximately (10) times faster than the brute force algorithm standard. The runtime was the same for both transactions, in the case of the other (3) libraries the results were the same. However, according with the iterative nature of the brute force attack for long passwords the time increases in an exponential complexity to response a petition with a password of more than 6 characters or denying the service by the low server capacity.

These observations lead conclude the inefficiency of this attack implementation according to the exponential time complexity of the algorithm.

5.2. Extern Entity Attack (XXE)

The second vulnerability founded allows send malicious codes hidden in variable to extract server side sensitive data [10], [41]. The attack works using a customized variable which has inside XML content and then deceive the server and obtain the sensitive or confidential data saved in the server side [10], in this case throws a file system with content with the IP and networks allowed by Windows [25] with the root C:/Windows/System32/drivers/etc/hosts.

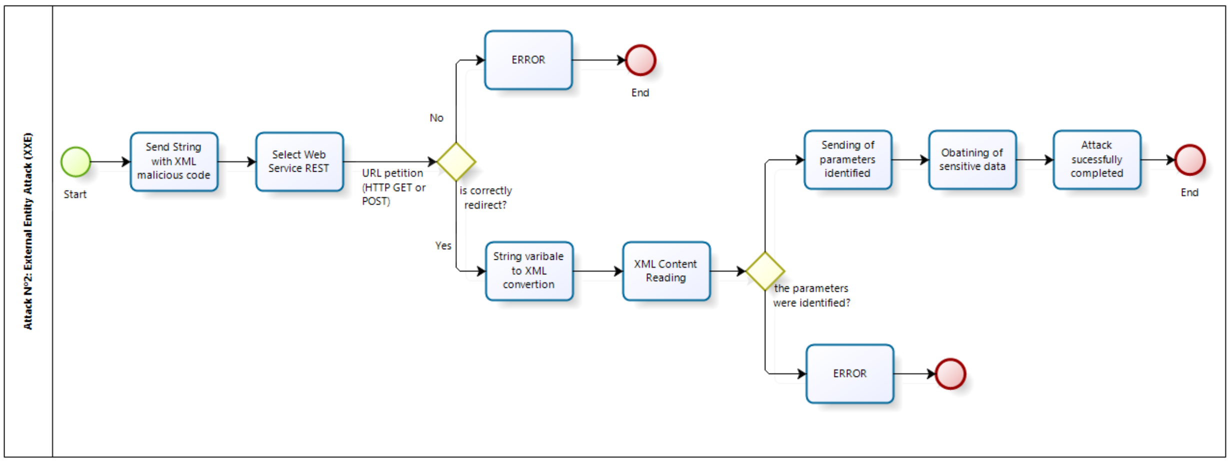

Attack N°2: External Entity Attack (XXE)
Start
Send String with XML malicious code
Select Web Service REST
URL petition (HTTP GET or POST)
No
ERROR
End
is correctly redirect?
Yes
String varibale to XML convertion
XML Content Reading
the parameters were identified?
Sending of parameters identified
Obatining of sensitive data
Attack sucessfully completed
End
ERROR

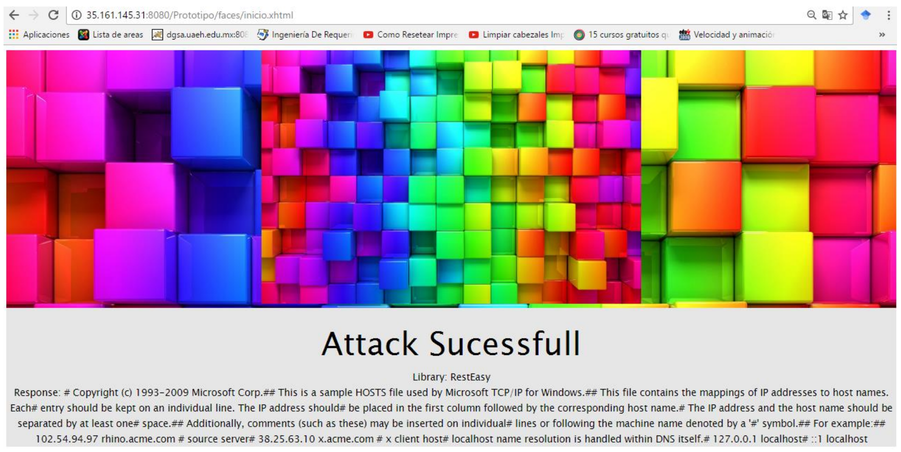
Fig. 5. BPNM attack N2 diagram.

The following source code was implemented in the prototype.

```
1.  $XML \ = < !DOCTYPE\ root\ [ < !ELEMENT\ root\ ANY\ >$
2.  $< !ENTITY\ windowsfile\ SYSTEM$
3.  file:///Text Plain File Path  > ] >
4.  $< root >< info > < boot > \&windowsfile; </boot > </info ></root >$
5.  $result\ =\ "";$
6.  $bufferReader\ rd\ =\ restful\ Service\ HTTP\ Response;$
7.  $line\ =\ "";$
8.  $line\ =\ rd\ LineReader;$
9.  **while** $(line \neq\ NULL)$
10. $result\ =\ result\ +\ line;$
11. **end while**
12. **return** $result;$
```

Algorithm 2. External entity attack (XEE).

The **Algorithm 2** works identifying the path location of a text plain file with system sensitive data Line 3, using *&windowsfile* URI Line 4 to extract with XML commands the file content, while the content is store in the variable *result* before the server reads and translate the HTTP petition *Line 5* to *Line 10,* until there is not more lines to read *Line 9* to *11,*returns the content in the result variable *Line 12.*



Attack Sucessfull

Library: RestEasy

Response: # Copyright (c) 1993-2009 Microsoft Corp.## This is a sample HOSTS file used by Microsoft TCP/IP for Windows.## This file contains the mappings of IP addresses to host names. Each# entry should be kept on an individual line. The IP address should# be placed in the first column followed by the corresponding host name.# The IP address and the host name should be separated by at least one# space.## Additionally, comments (such as these) may be inserted on individual# lines or following the machine name denoted by a '#' symbol.## For example:## 102.54.94.97 rhino.acme.com # source server# 38.25.63.10 x.acme.com # x client host# localhost name resolution is handled within DNS itself.# 127.0.0.1 localhost# ::1 localhost

Fig. 6. Attack N2 response from GET or POST.

6. Evaluation of the Vulnerabilities

To decrease the risk and keep the user trust are the principal objectives of the security, the attacks are made to exploit a set of vulnerabilities that represents a weakness in the software [26], [27], founded in the source code, where the code injection is allowed and be executed. Taking in count the **service**, **the user approach**, the **type of attack** [28] ,the **severity of the attack over REST** and **the impact** [29]**,** are the principal variables to analyze, find and evaluate all the possible risks , their probability, and how mitigate it.

The result obtained of this process is a residual risk to control it [4], [19], [30], using the different security policies and controls given by JAX-RS, these control mechanism [31] are included in the source code, to provide a warrantee for transaction done by the client and the server. To how obtain a score or number that gives basic information about the effectiveness of these controls, and know which controls apply for a specific weakness are necessary a security metric[32] and a documental review in the JAX-RS libraries documentation.

6.1. The Security Metric

This metric is applied to measure the security in a software product, in this case, a RESTful Service[33] . Where:

$$SM(s) = \sum_{n=1}^{m} (P_n \times W_n) \quad (1)$$

s: is the software product the Web Service in this case

W_n: the severity of a set of representstive weakness in the Web Service

P_n: a set of risk identified in the Web Service related with the weakness

The weakness severity W_n is defined by a quantitative score values V_n obtained from a CVSS **Common Vulnerability Scoring System** base scores, for this analysis is used a tool which measure the scope and impact of these weaknesses, and k are the number of vulnerabilities which applied.

$$W_n = \frac{\sum_{i=1}^{k} V_i}{k} \quad (2)$$
$$P_n(i = 1,2 \ldots n)$$

$$P_n = \frac{R}{\sum_{i=1}^{m} R_i} \tag{3}$$

Knowing R like the frequency of the risk R_n, in the same way obtaining the value of R_n from

$$R_n = \frac{k}{M} \tag{4}$$

where k is the number of vulnerabilities and M is the number of months having the following result:

$$\sum_{n=1}^{m} P_n = 1 \tag{5}$$

Taking in count the steps given by the security metric [33], from (2),the conclusion founded is:

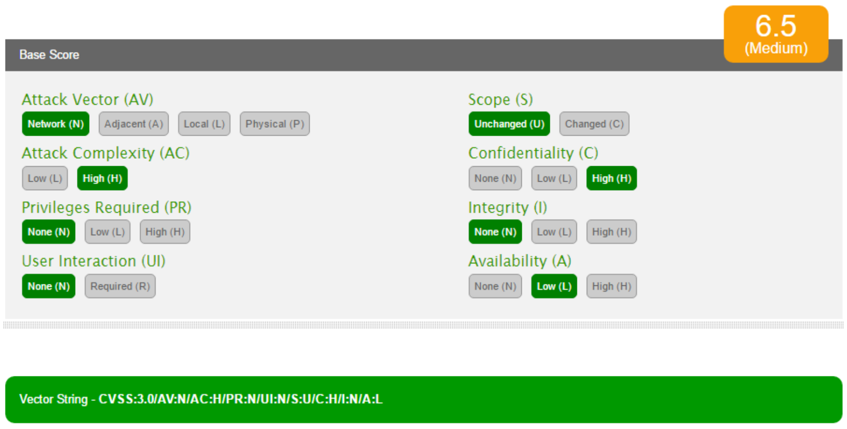
$$C_n = \frac{\sum_{i=1}^{n} C_i}{k} \tag{6}$$

where:

C_i: Are the security controls gived by a library to mitigate W_n

k: The number of security controls founded to mitigate W_n

The security controls C_n given by any JAX-RS library studied in this document is proportional or close to the severity vulnerability and Weakness, that it has over the RESTful Service , in order to be mitigated successfully by the set of available controls C_n.



6.5 (Medium)

Base Score

Attack Vector (AV): [✓ Network (N)] Adjacent (A) Local (L) Physical (P)

Attack Complexity (AC): Low (L) [✓ High (H)]

Privileges Required (PR): [✓ None (N)] Low (L) High (H)

User Interaction (UI): [✓ None (N)] Required (R)

Scope (S): [✓ Unchanged (U)] Changed (C)

Confidentiality (C): None (N) Low (L) [✓ High (H)]

Integrity (I): [✓ None (N)] Low (L) High (H)

Availability (A): None (N) [✓ Low (L)] High (H)

Vector String - CVSS:3.0/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:N/A:L

Fig. 7. CVSS vulnerabilities calculator.

6.2. The Weakness and the Vulnerabilities

A weakness depends of a set vulnerabilities which have based score(s), these are calculated in this punctual case with a software tool [34] provided and developed for CVSS [35] (**Common Vulnerabilities Scoring System**), which allows calculate the based scored in a scale from 1 to 10 according the metrics that allows calculate the based scores V_n, applied to the (2) implemented attacks. The metrics used by **Common Vulnerabilities Scoring System** Version **3.0 Calculator** are:

Access Vector (AV): This metric reflects the context by which vulnerability exploitation is possible.

Attack Complexity (AC) This metric describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Such conditions may require the collection of more information about the target, the presence of certain system configuration settings, or computational exceptions.

Privileges Required (PR) This metric describes the level of privileges an attacker must possess before successfully exploiting the vulnerability. This Base Score increases as fewer privileges are required.

User Interaction (UI) This metric captures the requirement for a user, other than the attacker, to participate in the successful compromise the vulnerable component.

Scope (S) Does a successful attack affect a component other than the vulnerable component? If so, the Base Score increases and the Confidentiality, Integrity and Authentication metrics should be scored relative to the impacted component.

Confidentiality (C) This metric measures the impact to the confidentiality of the information resources managed by a software component due to a successfully exploited vulnerability. Confidentiality refers to limiting information access and disclosure to only authorized users, as well as preventing access by, or disclosure to, unauthorized ones.

Integrity (I) This metric measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of information.

Availability (A) This metric measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. It refers to the loss of availability of the impacted component itself.

Having W_na set of weakness found, and L_n the set of JAX-RS libraries to test

$$W_n = \{W_1, W_2\}$$

where

$$W_1 = Multiple\ failed\ authentication\ attempts\ weakness$$

$W_2 = XML\ data\ sensitive\ abstraction$

And $L_n = \{L_1, L_2, L_3, L_4\}$

Where

$L_1 = Jersey\ Library$

$L_2 = Restlet\ Library$

$L_3 = RestEasy\ Library$

$L_4 = Apache\ CXF\ Library$

As was mentioned in the weakness applies for all the libraries, boating like result:

$$W_n \cup L_n = \begin{vmatrix} W_1L_1 & W_1L_2 & W_1L_3 & W_1L_4 \\ W_2L_1 & W_2L_2 & W_2L_3 & W_2L_4 \end{vmatrix}$$

$$W_n \forall L_n$$

7. Results

For the Attack N1, implemented in the prototype the obtained results using the calculator v. 3.0 were:

$$V_1 = 6.5$$

Considered like a mid-vulnerability defined for the metrics used by the CVSS, this s vulnerability with a residual risk easy to control it.

Appling $V_1$to W_1 result is:

$$W_1 = \frac{\sum_{i=1}^{k} V_i}{k} = \frac{6{,}5}{2} = 3{,}25$$

$$W_1 = 3.25$$

For the Attack Nº2, implemented in the prototype, the results are:

$$V_2 = 5.3$$

Appling V_2result to obtain W_2

$$W_2 = \frac{\sum_{i=1}^{k} V_i}{k} = \frac{5,3}{2} = 2,28$$
$$W_2 = 2,28$$

8. Security Analysis Libraries

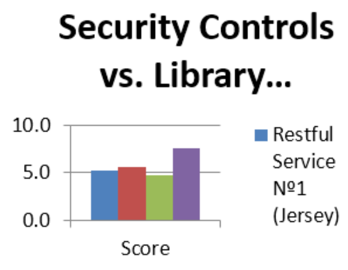
Giving the attack, the libraries and security control libraries relationship is possible identity which controls works to stop the attacks and mitigate the vulnerabilities. Thought additional parameters like the weakness identified and the vulnerability to exploit, the number of knowing parameters and number of mechanism of control implemented for each weakness exploited by each one of the JAX-RS libraries, were used for classified the controls, to calculate a score using the equation (6) defined the next tables and graphics are obtained:

Table 1. Security Controls Parameters Scores Attack Nº1

		SOP			
	Constraints Annotations	HTTP	URL parameters	Roles	Other Controls
Restful Service Nº1 (Jersey)	6,0	10,0	5,0	10,0	
Restful Service Nº2 (RestEasy)	8,0	8,3	10,0	10,0	1,7
Restful Service Nº3 (RestLet)	8,0	8,3	2,5	10,0	
Restful Service Nº4 (Apache CXF)	10,0	10,0	10,0	10,0	

Table 2. Attack Nº1 Final Scores

	Score
Restful Service Nº1 (Jersey)	5,3
Restful Service Nº2 (RestEasy)	5,6
Restful Service Nº3 (RestLet)	4,7
Restful Service Nº4 (Apache CXF)	7,5



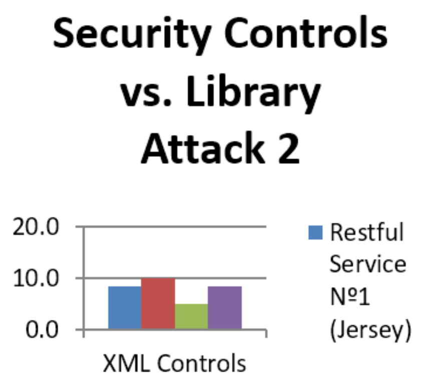
Graph N 1. Attack Nº1 final scores.

Table 3. Attack Nº2 Security Controls Scores Classification

	Attack Nº2	VulnerabilityTo Exploit	Impact in REST Security (Dimenssions)	XML
Restful Service Nº1 (Jersey)	Extern Entity Attack	WEB INF XML	Confidentialy	JAXB ,@XmlRootElement MIME Multipart features @FormDataParam Jetty
Restful Service Nº2 (RestEasy)	Extern Entity Attack	WEB INF XML	Confidentialy	JAXB ,@XmlRootElement FORM Authentification Auth Server Action URL @MultipartForm XML-Binary-Xop
Restful Service Nº3 (RestLet)	Extern Entity Attack	WEB INF XML	Confidentialy	Memory Realem Atom org.restlet.client.ext.xml.DomRepresentation
Restful Service Nº4 (Apache CXF)	Extern Entity Attack	WEB INF XML	Confidentialy	Maven Dependencies XML signtures XML encryption OAuth2 GCM Algorithm

Table 4. Attack Nº2 Security Controls Final Scores

	XML Controls	Score
Restful Service Nº1 (Jersey)	8,3	8,3
Restful Service Nº2 (RestEasy)	10,0	10,0
Restful Service Nº3 (RestLet)	5,0	5,0
Restful Service Nº4 (Apache CXF)	8,3	8,3



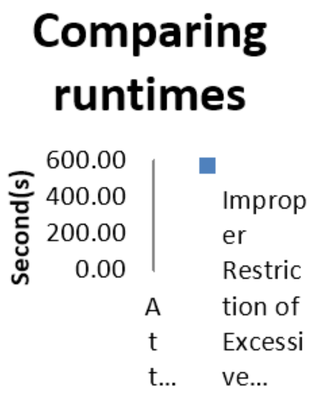
Graph 3. Attack Nº2 security controls final scores.

The following table shows the security controls found to shield the XML documents that contains server side sensitive data, also were added frameworks clasess and anotations used to restric the use of parameters. Formats like JSON are actually used to avoid data injections and abstractions, promiving the simplicity and clarity in the source code, but was not included in this classifcation because are not directled XML warentes or security controls.

Finally to obtain the security level of libraries, both attacks were deployed in each RESTful Service and know which are the libraries the behavior of in front of the attack.

Table 5. Attacks Runtimes

	Attacks Runtime (s)
Improper Restriction of Excessive Authentication Attempts	540
Extern Entity Attack	2,25

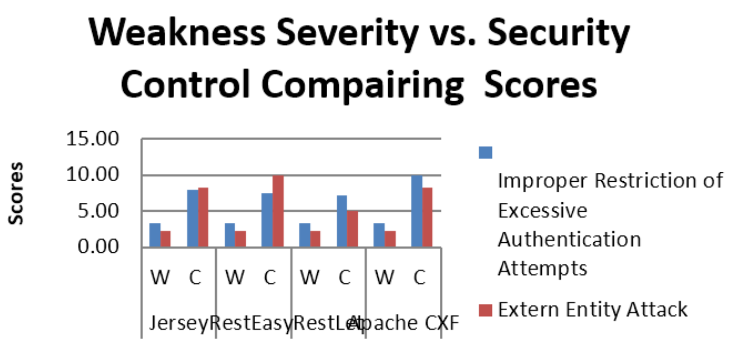


Graph N 4. Comparing attack runtimes.

Comparing the runtimes, of all the RESTful Services were compiled and deployed the attacks in the same time. Considering this the **Attack Nº1: Improper restriction of excessive authentication using a Brute** force takes $2{,}4\,x10^2$ **times more time** to be successful than the **Attack Nº2: Extern Entity Attack**. This allows conclude is easier to exploit the XML vulnerability.

Table 6. Weakness Scores vs Security Controls Scores Comparing

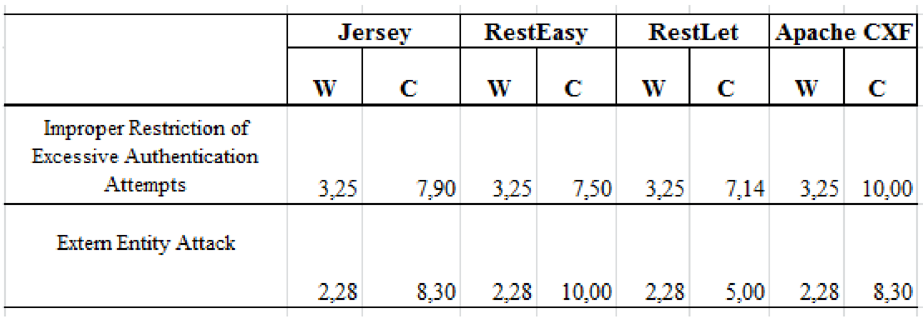
Attack	**Jersey**			**RestEasy**			**RestLet**			**Apache CXF**		
	W	**C**	**t(s)**	**W**	**C**	**t(s)**	**W**	**C**	**t(s)**	**W**	**C**	**t(s)**
Improper Restriction of Excessive Authentication Attempts	3,25	7,90	540 s	3,25	7,50	540 s	3,25	7,14	540 s	3,25	10,00	540 s
Extern Entity Attack	2,28	8,30	540 s	2,28	10,00	540 s	2,28	5,00	540 s	2,28	8,30	540 s



Graph N 4. Weakness severity vs. security control comparing scores

Based in the previously comparison and the graph obtained for the weakness score W_n and the C_n libraries control score calculated for each RESTful Service. Obtaining a score not less than 5 for the controls, which means the libraries have the security controls to mitigated and control these vulnerabilities, so Why is it possible to do this attacks? . The human mistake, in this case the developers experience, knowledge, the lack of documentation review, the bad application of the controls in the source code are residual risks that explain this cases.

Table 7. Final Scored Results Resume Table



	Jersey		RestEasy		RestLet		Apache CXF	
	W	C	W	C	W	C	W	C
Improper Restriction of Excessive Authentication Attempts	3,25	7,90	3,25	7,50	3,25	7,14	3,25	10,00
Extern Entity Attack	2,28	8,30	2,28	10,00	2,28	5,00	2,28	8,30

The level of security given by these libraries taking in count all the variables and preview analysis are: *Apache CXF* with a score of 10 *and 8,3* for the two weakness, followed by *RestEasy* and *Jersey* with a scores *of 7.5 - 10 and 7.9 - 8.3*, and *RestLet* with scores of *7.1 – 5.0*.

9. Conclusions

The 97% of the Cross-site Scripting attacks (XSS) found it are focus in a code injection through tags, using the user interface and the Web Browser to steal data. These attacks do not have directly the REST architecture, and are not successful to implement to exploit the vulnerabilities found it.

The 99% of the attacks found, were mitigated by the APIS so actually do not work, in the international classifications CWE, MITRE, APEC, the analysis and controls are available to implement by the developer.

The two attacks successfully implemented, sends the attack parameters by URL confusing the server to obtain confidentiality data, this may conclude the sending of parameters injected is maybe the only way to attack the RESTful service and then extends the attacks scope lately. While, The brute force attack implemented improved with the dynamic programming takes 9 minutes to give a response 540 seconds, 10 times less than the brute force standard algorithm, with a lower complexity runtime but still is inefficient.

The Billion laughs attack implemented in Jersey RESTful service, allows identify a mechanism of control provided by the Java structure, which the RESTful Web was, build. In that way the RESTful service mitigates the attack, in the moment when library reads the XML and then interpret it, using like a control a maximum limit of entity extensions of 64000, controlling the attack and avoiding the denial of service. The test of the attack using the Unix Shell was, based on those tests we conclude this vulnerability is completely covered.

According with the vulnerabilities found it and the testing done, the weakness severity that represents a directly impact on the RESTful Service are low but representative and dangerous if it is not identified and measured.

Having in count the scores obtained in the security analysis done the libraries, which gives more security controls, are Apache CXF and RestEasy, in that order can be considered the more secure and useful to implement to mitigate the weaknesses found in this work follow by Jersey and RESLET.

Like shows the calculated scores for the library's security controls the (4) libraries have enough controls to mitigate the exploited vulnerabilities, the controls offered by each one are similar, but the attacks still running, because the residual risk commonly the human resource are not correctly controlled.

References

[1] Adamczyk, P., Smith, P. H., Johnson, R. E., & Hafiz, M. (2011). *REST: From Research to Practice*.

[2] Cross-site scripting. *The Open Web Application Security Project*.

[3] Shema, M. (2010). Cross-site scripting. *Seven Deadliest Web Appl. Attacks*, 1–26.

[4] Fogie, S., Grossman, J., Hansen, R. R., & Petkov, P. D. (2007). XSS attacks: Cross site scripting exploits and defense.

[5] Serme, G., Oliveira, A. S. D., Massiera, J., & Roudier, Y. (2012). Enabling message security for RESTful services. *Proceedings of the 2012 IEEE 19th Int. Conf. Web Serv. ICWS 2012* (pp. 114–121).

[6] Wang, X., Jhi, Y., Zhu, S., & Liu, P. (2008). Protecting web services from remote exploit code: A static analysis approach. *Proceedings of the 17th Int. Conf. World Wide Web* (pp. 1139–1140).

[7] Burke, B. (2013). RESTful Java with JAX-RS 2.0.

[8] Velandia, J., Rios, S., Bolivar, H., Vanzina, J., & Almanzar, N. JAX-RS implementations: A performance comparison. *10(1)*, 139–144.

[9] CWE. CWE 382 vulnerability case.

[10] The open web application security project. XML external entity.

[11] MITRE. APEC common parttern attacks enumeration and classification.

[12] XML external entity.

[13] Schneier, B. (1996). Applied cryptography: Protocols, algorithm, and source code in C. *Gov. Inf. Q.*, *13(3)*, 336.

[14]Fellows, M. R., Rosamond, F. A., Fomin, F. V., Lokshtanov, D., Saurabh, S., & Villanger, Y. (1988). Local search: Is brute-force avoidable. 486–491.

[15] Probability, C., *et al*., A. the ASCII code 1.

[16] Reyna, J. A. D., & Lune, J. V. D. (2014). Algorithms for determining integer complexity.

[17] Trakhtenbrot, B. A. (1984). A survey of Russian approaches to perebor algorithms. *Ann. Hist. Comput*., *6(4)*, 384–400.

[18] Stanford encyclopedia of philosophy.

[19] Nunan, A. E., Souto, E., Santos, E. M. D., & Feitosa, E. (2012). Automatic classification of cross-site scripting in web pages using document-based and URL-based features. *Proceedings of the IEEE Symp. Comput. Commun* (pp. 000702–000707),

[20] Harel, D., Tiuryn, J., & Kozen, D. (1984). Dynamic logic. *Handb. Philos. Log*., 497–604.

[21] GitHub. Recursion and dynamic programming.

[22] Jupiter. Programming-and-bayesian-methods-for-hackers.

[23] Choi, J. H., Choi, C., Ko, B. K., & Kim, P. K. (2012). Detection of cross site scripting attack in wireless networks using n-Gram and SVM. *Mob. Inf. Syst*., *8(3)*, 275–286.

[24] Controls, I. S. InfoSec reading room.

[25] Windows documentation host.

[26] Saiedian, H., & Broyle, D. (2011). Security vulnerabilities in the same-origin policy: Implications and alternatives. *Computer (Long. Beach. Calif)*, *44(9)*, 29–36.

[27] Venkat, T., Rao, N., Tejaswini, V., & Preethi, K. (2012). Defending against web vulnerabilities and cross-site scripting. *J. Glob. Res. Comput. Sci*., *3(5)*, 61–64.

[28] Hydara, I., Sultan, A. B. M., Zulzalil, H., & Admodisastro, N. (2015). Current state of research on cross-site scripting (XSS) - A systematic literature review. *Inf. Softw. Technol*., *58*, 170–186.

[29] Hanley, D., & Hatch, A. (2014). *Deloitte Tech Trends 2014: Inspiring Disruption*. Deloitte Univ. Press.

[30] Masood, A., & Java, J. (2015). Static analysis for web service security - Tools amp; techniques for a secure development life cycle. *Technol. Homel. Secur*.

[31] Mead, N. R., & Stehney, T. (2005). Security quality requirements engineering (SQUARE) methodology. *ACM SIGSOFT Softw. Eng. Notes*.

[32] Software engineering institute.

[33] Wang, J. A., Wang, H., Guo, M., & Xia, M. (2009). Security metrics for software systems. *ACM Southeast Reg. Conf*.

[34] CSVSS vulnerabilities software calculator v 3.0.

[35] Common vulnerability scoring system.

John Velandia holds a MSc. from Stuttgart University (Germany). He has been leader of IT projects in the industry, in sectors such as education, food an automotive. He manages his projects based on PMI, TOGAF and ITIL. Moreover, He teaches software architecture and guide bachelor thesis at the Catholic University. He is responsible of a research junior group named GINOSKO. The fields of research area are web technologies, business intelligence and IT frameworks.