INFORMATYKA EKONOMICZNA BUSINESS INFORMATICS 4(30) · 2013
ISSN 1507-3858

Adio Akinwale, Joseph Shonubi, Adebayo Adekoya, Adesina Sodiya, Tosin Mewomo
Federal University of Agriculture, P. M. B. 2240, Abeokuta
e-mail: aatakinwale@yahoo.com, dejishow2yk @yahoo.com, lanlenge@gmail.comsinaronke@yahoo.co.uk, mewomo@yahoo.com

ONTOLOGY OF INPUT VALIDATION ATTACK PATTERNS ON WEB APPLICATIONS

Summary: Web applications have been the main intrusion target, and input errors from the web users lead to serious security vulnerabilities. Many web applications contain such errors, making them vulnerable to remotely exploitable input validation attacks such as SQL Injection, Command Injection, Meta-Characters, Formatting String, Path Traversal and Cross Site scripting. In this paper, we present ontology to represent patterns of input validation attacks on web applications. More specifically, our ontology is based on individual subclasses, properties and inverse functional properties, domain and range of input validation attack patterns. The ontology is implemented and interpreted with the web application development language OWL (Ontology Web Language).

Keywords: input validation attacks, patterns, ontology, web application.

1. Introduction

A web application is typically a client/server software that handles user requests coming from clients such as web browsers. To serve the user requests, it often requires accessing system resources such as databases and files at the server end. The system resources are a part of the trusted environment and often contain security-critical data. Hence these resources need appropriate protection to maintain their confidentiality and integrity, otherwise it cannot be trusted. It cannot be directly used to access the system.

Web applications usually take inputs from users through form, cookies, or some other standard channels, and use these input data in further processing operations, such as querying databases, generating web pages, or executing commands. Because the input data are from the remote users and may contain malicious values, they need to be validated before use. Once a web application fails to do so, attackers can exploit the system's vulnerabilities to launch specific attacks. The following are examples of popular input validation attacks: SQL injections, cross-site scripting, and command injections. These attacks can cause many serious problems, such as the leak of sensitive information and corruption of critical data.

Web applications are designed to present to any user with a web browser a system-independent interface to some dynamically generated content. A user can log into the web database through a web browser, if he/she is using a web database application. These applications normally interact with databases to access persistent data. This interaction is commonly done within a general-purpose programming language, such as Java, through an application programming interface (API) [Su, Wassermann 2009].

However, any user's input that is not handled properly can pose serious security problems. This is because queries are constructed dynamically in an ad hoc manner through low-level string manipulations. This is ad hoc because databases interpret query strings as structured, meaningful commands, while web applications often view query strings simply as unstructured sequences of characters. This semantic gap, combined with the improper handling of user input, makes web applications susceptible to a large class of malicious attacks known as *command injection attacks.*

Web applications are designed to meet business needs due to the demand on developers to push out more and more web applications at a faster rate than the developers could handle. The implication of this is that the developer will be focused more on the development of these applications and less on ensuring the security of the applications.

2. Related works

In Anitha Nalluri and Dulal C. Kar [2005], a Web-based data mining system to analyze intrusions was presented. The system was implemented using all the freeware available in the public domain. The system finds anomalous activity that uncovers a real attack process and identifies long and ongoing patterns. It was used to analyze host-based traffic features, time-based traffic features, protocol-based traffic features, and associated intrusions. With the help of this system, rules were generated to capture the behavior of the intrusions as well as of normal activity.

Jeffery Undercoffer and John Pinkson [Undercoffer, Joshi, Pinkston 2003] present a target-centric ontology for intrusion detection, the ontology presented is based on the analysis of many classes of computer intrusion and their corresponding attacks strategies, which are categorized according to: system component targeted, means of attack, consequence of attack and location of attacker.

They compared the taxonomy and ontology for modeling a computer attack, and found that modeling with ontology gives better understanding and a wider scope than taxonomy. They used the ontology specification language DAML+OIL to implement ontology. They also combined ontology (DAML+OIL) and an inference engine to initiate an event recognition language.

The transformation process, together with its two constituents-pattern detection and an ontology transformation process, were presented by [Sv´ab-Zamazal, Sv´atek 2008]. Pattern detection process is based on SPARQL and the transformation process is based on an ontology alignment representation with specific extensions regarding detailed information about the transformation.

3. Input Validation Attacks

The single biggest cause of web application vulnerabilities is the lack of proper input validation. Through a web application, attackers or intruders can use these lapses to attack rear components, such as a database. Input validation attacks are used to generate information errors, to obtain uninformed data access, execute commands, grab passwords and so on.

3.1. Types of Input Validation attack patterns

SQL injection

A SQL injection attack consists of the insertion of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to effect the execution of predefined SQL commands. SQL injection is a code injection technique that exploits a security vulnerability occurring in the database layer of an application. SQL injection is one of the oldest attacks against web applications.

Guy-Vincent Jourdan [2009], the vulnerability is present when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and thereby unexpectedly executed. This is an instance of a more general class of vulnerabilities that can occur whenever one programming or scripting language is embedded inside another.

Cross-Site Scripting (XSS)

Many websites have options that allow users to enter data and then dynamically generate web pages based on the input. A cross-site scripting vulnerability may occur if the user input is not properly validated. Attackers exploit this vulnerability by embedding malicious script into the generated page. The script is automatically executed on the machine of any user who views the generated page.

Wikipedia (2010), an attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way of knowing that the script should not be trusted, and will therefore execute the script which it thinks came from a trusted source. The malicious script can access any cookies, session tokens, or other sensitive information retained by the users' browser. These scripts can even rewrite the content of the HTML page.

Meta-Character Attack

Hackers can change the behaviour of a web application by inserting meta-characters into an URL-encoded parameter within query strings. Since many meta-characters are interpreted differently by different servers, the risk depends on the operating system, programming languages and workflow of the affected application.

Format String Attack

In a format string attack, an attacker changes the format specification sent to a common program function like prinf(), uncovering information about the system or even executing arbitrary code.

The format string exploit occurs when the submitted data of an input string is evaluated as a command by the application. In this way, the attacker could execute code, read the stack, or cause a segmentation fault in the running application, causing new behavior that could compromise the security or the stability of the system.

Path Traversal Attack

Path traversal vulnerabilities allow a hacker to execute commands or view data outside the intended target path. Path traversal attacks are normally carried out via unchecked URL input parameters, cookies and HTTP request headers.

A path traversal attack aims to access files and directories that are stored outside the web root folder. By browsing the application, the attacker looks for absolute links to files stored on the web server. By manipulating variables that reference files with "dot-dot-slash (../)" sequences and its variations, it may be possible to access arbitrary files and directories stored on the file system, including the application source code, configuration and critical system files, limited by system operational access control. Wikipedia (2010), attacker uses "../" sequences to move up to root directory, thus permitting navigation through the file system.

4. Ontology

Ontology is an integration of a specific domain of knowledge in a common vocabulary, which provides basic concepts in a domain and the relations among the concepts. Noy and McGuinnes [2002] gave reasons why someone would want to develop an ontology, they are :

- to enable the reuse of domain knowledge,
- to make domain assumptions explicit,
- to separate domain knowledge from operational knowledge,
- to analyze domain knowledge.

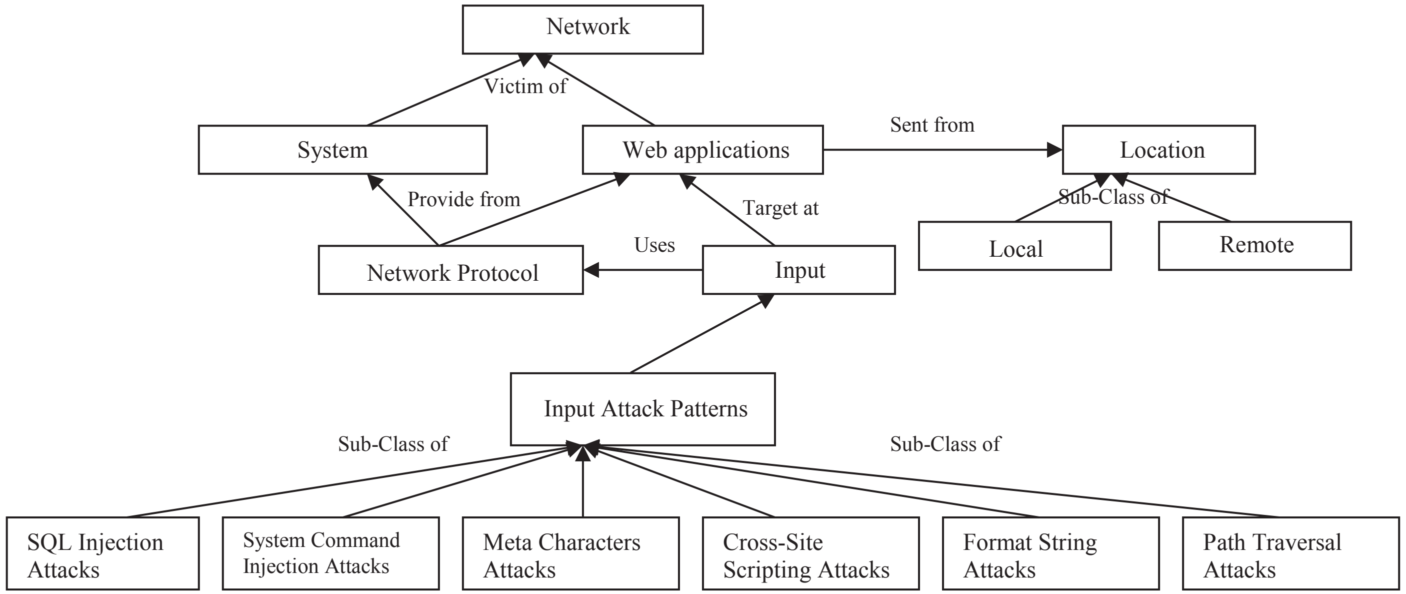

Figure 1. Ontology of Input Validation attack patterns

Source: own elaboration.

Ontology consists of concepts (classes) in the domain of knowledge, difference features and the attributes are used to the describe concept in the ontology using properties to show relations among classes and subclasses of the domain in Figure 1.

System and web applications are the victim of network attacks or intrusions. The heart of the ontology is where we introduce the Input that is Input Attack Patterns.

Network protocols are the means of attack (Uses). The attack can be from remote or within (local) which are subclasses of Location. SQL Injection, System Command Injection, Meta Characters, Cross-Site Scripting, Format String and Path Traversal Attacks are the patterns of input validation attacks on web applications, also sub--classes of our super class (Input attack patterns).

4.1. Class ontology of Input Validation attack patterns

Individuals

Individuals represent objects in the domain in which we are interested, the individuals in our ontology are SQL, DOS, FSA, MC, CSS and PTA inside INPUT VALIDATION classes. In Figure 2, individuals are represented by diamond symbols. OWL classes are interpreted as sets that contain individuals

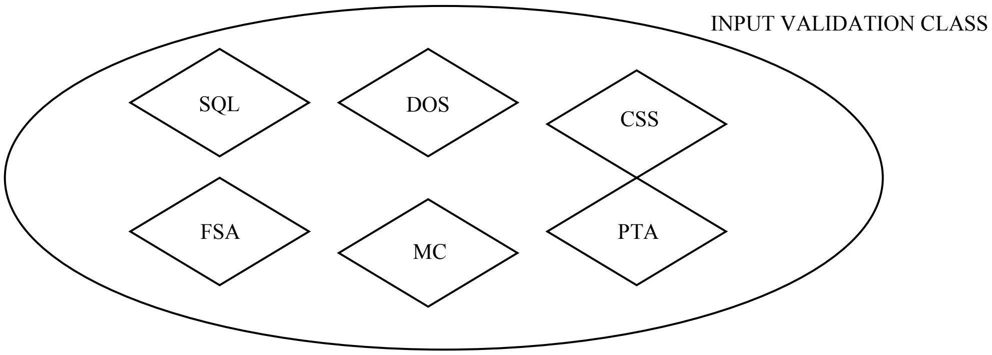

Figure 2. Representation of Individuals

Source: own elaboration.

Subclasses

Classes are organized into a superclass-subclass hierarchy, which is known as taxonomy. Subclasses are subsumed by their superclasses, for example class command injection attack and classes HTML form, Cookies, and URL parameters, are subclasses of class command injection attack which means that any such input would result in command injection attack.

In Figure 3 subclasses are represented by diamond symbols inside circles representing classes.

Properties and Inverse

Properties are a binary relationship on an individual (i.e. properties link two individuals together)

The property isCommandOn links individual SQL injection attacks to the individual SQL command or the property isInsertingOn links the individual Command Injection attacks to the individual URL.

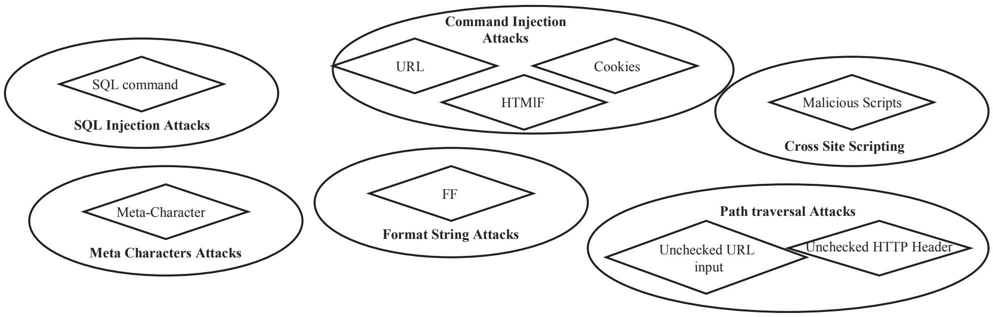

Figure 3. Representation of classes and subclasses

Source: own elaboration.

5. Sample Codes, OWL/RDF Classes, Sub-classes and Properties Representations

Sample Codes for SQL Injection Attack

In SQL: select id, firstname, lastname from intrusionbase

If one provides: Firstname: shonubi'al , lastname: joseph

The query string becomes:

select id, firstname, lastname from intrusionbase where firstname = 'shonubi' al' and lastname ='joseph' , which the database attempts to run as incorrect syntax near al' as the database tried to execute shonubi.

SQL injection attack resulted from an injection of some codes into SQL commands, Guy-Vincent Jordan [2009].

Sample Codes for Path Traversal Attack Patterns

This attack can be executed with an external malicious code injected on the path, like the Resource Injection attack. To perform this attack it is not necessary to use a specific tool- attackers typically use a spider/crawler to detect all URLs available.

This attack is also known as "dot-dot-slash", "directory traversal", directory climbing" and "backtracking".

Sample: In many operating systems, null bytes %00 can be injected to terminate the filename. For example, sending a parameter like:

?file=secret.doc%00.pdf , will result in the Java application seeing a string that ends with ".pdf" and the operating system will see a file that ends in ".doc". Attackers may use this trick to bypass validation routines, Guy-Vincent Jordan [2009].

OWL/RDF Representing Class, Subclasses and Properties SQL Injection Attack Patterns

In Figure 4 Class (SQL Injection attack) has sub-class (SQL command codes) and property (isResultOf) that shows the relationship between the class and subclass. Figure 5 below shows the OWL/RDF presentation.

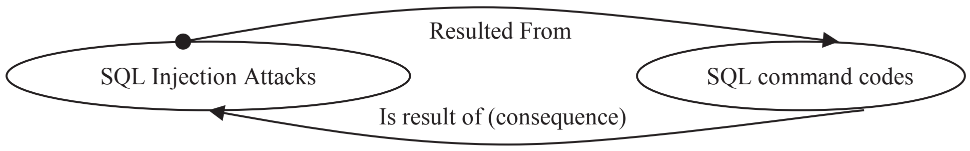

Figure 4. Class, sub-class and properties of SQL injection attack patterns.

Source: own elaboration.

```
<owl:Class rdf:about="&sqlcommand;SQLInjection">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="&sqlcommand;ResultedFrom"/>
<owl:someValuesFrom rdf:resource="&sqlcommand;sqlCommands"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
<!-- http://www.semanticweb.org/ontologies/sqlcommand.owl#sqlCommands -->
<owl:Class rdf:about="&sqlcommand;sqlCommands">
<rdfs:subClassOf rdf:resource="&sqlcommand;SQLInjection"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="&sqlcommand;isResultOf"/>
<owl:someValuesFrom rdf:resource="&sqlcommand;SQLInjection"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
</rdf:RDF>
```

Figure 5. OWL/RDF representing class, subclasses and properties SQL injection attack patterns

Source: own elaboration.

OWL/RDF representing Class, subclasses and properties Path Traversal attack patterns

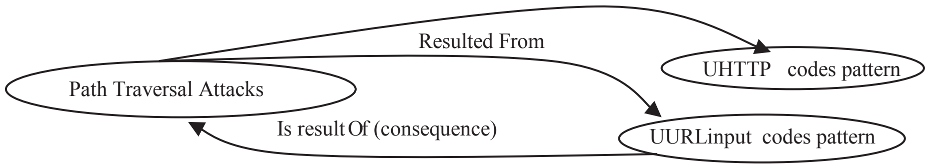

Figure 6. Class, sub-class and properties of Path Traversal attack patterns

Source: own elaboration.

In Figure 6 Class (Path Traversal Attacks) has sub-classes (UHTTP_ codes pattern, UURLinput_codes pattern) and property (isResultOf) that shows the relationship between the class and subclasses. Figure 7 below shows the OWL/RDF presentation.

```
<owl:Class rdf:about="&pathtraversal;Path_Traversal_Attacks">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="&pathtraversal;Resultedfrom"/>
<owl:someValuesFrom rdf:resource="&pathtraversal;UURLinput_pattern"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="&pathtraversal;Resultedfrom"/>
<owl:someValuesFrom rdf:resource="&pathtraversal;UHTTP_pattern"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
<!-- http://www.semanticweb.org/ontologies/pathtraversal.owl#UHTTP_pattern -->
<owl:Class rdf:about="&pathtraversal;UHTTP_pattern">
<rdfs:subClassOf rdf:resource="&pathtraversal;Path_Traversal_Attacks"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="&pathtraversal;Resultedfrom"/>
<owl:someValuesFrom rdf:resource="&pathtraversal;Path_Traversal_Attacks"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
<!-- http://www.semanticweb.org/ontologies/pathtraversal.owl#UURLinput_pattern -->
<owl:Class rdf:about="&pathtraversal;UURLinput_pattern">
<rdfs:subClassOf rdf:resource="&pathtraversal;Path_Traversal_Attacks"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="&pathtraversal;isResultOf"/>
<owl:someValuesFrom rdf:resource="&pathtraversal;Path_Traversal_Attacks"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
</rdf:RDF>
```

Figure 7. OWL/RDF representing Class, subclasses and properties Path Traversal attack patterns

Source: own elaboration.

Sample code for Cookie grabber attack patterns

If the application does not validate the input data, the attacker can easily steal a cookie from an authenticated user. All the attacker has to do is to place the following code in any posted input (i.e. message boards, private messages, user profiles):

<SCRIPT type=”text/javascript”>
var adr = ‘../evil.php?cakemonster=’ + escape(document.cookie);
</SCRIPT>

The above code will pass an escaped content of the cookie (according to RFC content must be escaped before sending it via HTTP protocol with GET method) to the evil.php script in “cakemonster” variable. The attacker then checks the results of his/her evil.php script (a cookie grabber script will usually write the cookie to a file) and use it.

In some instances, a hacker can execute operating system commands by injecting them via HTML forms, cookies or a URL parameter. Using this type of attack the hacker is able to execute system-level functions such as removing and copying files, sending emails, and calling operating system tools to modify the application’s input and output. From Wikipedia (2010)

OWL/RDF representing Class, subclasses and properties System Command injection attack patterns

In Figure 8, Class (System command attacks) has sub-classes (cookies_ codes pattern,

HTMF _codes pattern and URL codes pattern) and property (isResultOf) that shows the relationship among the class and subclasses. Figure 9 below shows the OWL/RDF presentation.

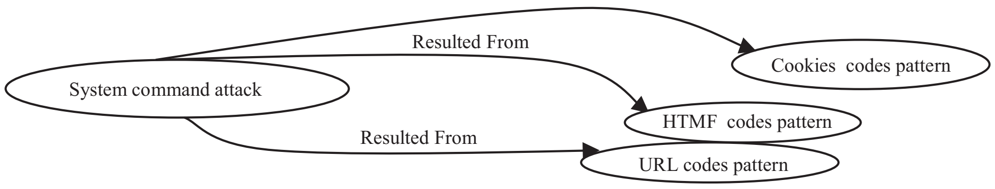

Figure 8. Class, sub-class and properties of System Command attack patterns

Source: own elaboration.

```
<owl:Class rdf:about="&commandinjection;Cookies_pattern">
<rdfs:subClassOf rdf:resource="&commandinjection;commandInjection"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="&commandinjection;isResultOf"/>
<owl:someValuesFrom rdf:resource="&commandinjection;commandInjection"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
<!-- http://www.semanticweb.org/commandinjection.owl#HTMIF_pattern -->
<owl:Class rdf:about="&commandinjection;HTMIF_pattern">
<rdfs:subClassOf rdf:resource="&commandinjection;commandInjection"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="&commandinjection;isResultOf"/>
<owl:someValuesFrom rdf:resource="&commandinjection;commandInjection"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
<!-- http://www.semanticweb.org/commandinjection.owl#URL_pattern -->
<owl:Class rdf:about="&commandinjection;URL_pattern">
<rdfs:subClassOf rdf:resource="&commandinjection;commandInjection"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="&commandinjection;isResultOf"/>
<owl:someValuesFrom rdf:resource="&commandinjection;commandInjection"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

Figure 9. OWL/RDF representing class, subclasses and properties of System Command attack patterns

Source: own elaboration.

Sample codes for Format String attack in Denial of Service

In this case, when an invalid memory address is requested, normally the program is terminated for example, printf (userName);

The attacker could insert a sequence of format strings, making the program show the memory address where a lot of other data are stored. Then the attacker increases the possibility that the program will read an illegal address, crashing the program and causing its non-availability.

printf (%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s);

OWL/RDF representing Class, subclasses and properties Format String patterns

In Figure 10 Class (Format String attacks) has sub-class (FF_Codes) and property (isResultOf) that shows the relationship between the class and subclasses. Figure 11 below shows the OWL/RDF presentation.

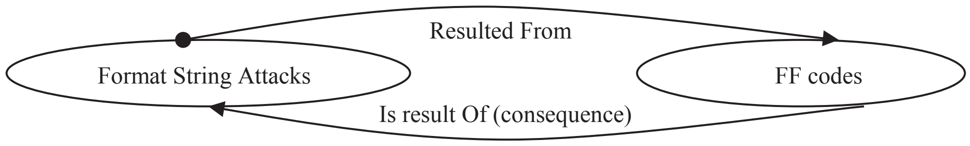

Figure 10. Class, sub-class and properties of Format String attack patterns

Source: own elaboration.

6. Conclusion and further work

The main contribution of this paper, so far, is the identification of classes of input validation attack patterns on web applications with their sample codes of launching attacks on web applications. The paper focuses more on intrusion patterns of by-pass network intrusion detection and access to network applications.

```
<owl:Class rdf:about="&FormatStringpatterns;FF_pattern">
<rdfs:subClassOf rdf:resource="&FormatStringpatterns;Format_String_patterns"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="&FormatStringpatterns;ResultedFrom"/>
<owl:someValuesFrom rdf:resource="&FormatStringpatterns;Format_String_patterns"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="&FormatStringpatterns;Format_String_patterns">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="&FormatStringpatterns;isResultOf"/>
<owl:someValuesFrom rdf:resource="&FormatStringpatterns;FF_pattern"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
</rdf:RDF>
```

Figure 11. OWL/RDF representing class, subclasses and properties of System Command attack patterns

Source: own elaboration.

Ontology was used with individual, classes, sub-classes, relations and properties of our domain ontology. OWL was used to implement our ontology. Further research should be carried out on how to use ontology to model a reliable and efficient intrusion detection system in both network-based and web-based attacks and intrusions.

Also, rules can be designed to cater for identified attack patterns in the classes of input validation attack patterns.

Reference

Crosbie M., Price K., Curry D.A., *Intrusion Detection Systems,* www.cerias.purdue.edu/about/history/coast_resources/idcontent/ids.html [accessed: 9.03.2004].

Fernandez D., *Detección De Intrusos En GNU/Linux*, 2007, www.emagister.com [accessed: 19.07.2011].

Intrusion Detection System (2009), www.cerias.purdue.edu/coast/intrusion-detection/ids.html [accessed: 21.06.2011].

Jordan G.-V., *Command Injections*, School of Information Tech. and Engineering University of Ottawa, Ottawa 2009.

Nalluri A., Kar D.C, *A Web-Based System for Intrusion Detection*, CCSC: South Central Conference, 2005.

Noy N.F., McGuinnes D.L., *Ontology Development 101(2002): A Guide to Creating Your First Ontology*, Technical Report, Stanford University, http: //protege.stanford.edu/publications/ontology development/ontology101-noy-mcguinness.html.

Scarfone K., Mell P., *Guide to Intrusion Detection and Prevention Systems (IDPS)*, "Computer Security", February 2007.

Su Z., Wassermann G., *The Essence of Command Injection Attacks in Web Applications*, University of California, Davis 2009.

ˇSv´ab-Zamazal O., Sv´atek V., *Pattern-Based Ontology Transformation Service*, Online Paper, 2008.

Category of Web-Based Attacks (2010), www.mediawiki.com [accessed: 17.02.2012].

Undercoffer J., Joshi A., Pinkston J., *Modeling Computer Attacks: An Ontology for Intrusion Detection*, 2003.

Understanding the Cause and Effect of CSS Vulnerabilities (2009),www.technicalinfo.net/papers/CSS.html [accessed: 20.02.2012].

Varshovi A., Sadeghiyan B., *Ontological Classification of Network Denial of Service Attacks: Basis for a United Detection Framework*, 2004.

ONTOLOGIA WZORCÓW SPRAWDZANIA ATAKÓW DOTYCZĄCYCH POPRAWNOŚCI DANYCH WEJŚCIOWYCH W APLIKACJI SIECI WEB

Streszczenie: Aplikacje internetowe są głównym celem włamań, a błędy wejściowe użytkowników internetowych prowadzą do poważnych luk w bezpieczeństwie. Wiele aplikacji internetowych zawiera takie błędy, co czyni je podatnymi na zdalnie przeprowadzane ataki niepoprawnymi danymi wejściowymi (*input validation attacks*), takie jak: SQL Injection, wstrzykiwanie poleceń, metaznaków, łańcuchy formatujące, skrypty międzyścieżkowe (*path traversal scripting*) i skrypty międzyserwerowe (*cross site scripting*). W niniejszym artykule przedstawiamy ontologię do reprezentacji wzorców ataków niepoprawnymi danymi wejściowymi. Prezentowana ontologia opiera się na indywidualnych podklasach, właściwościach oraz odwróconych właściwościach funkcjonalnych, domenach i zakresach wzorców ataków niepoprawnymi danymi wejściowymi. Ontologia jest implementowana i interpretowana przez język tworzenia aplikacji internetowych OWL (*Ontology Web Language*).

Słowa kluczowe: *input validation attacks*, wzorce, ontologia, aplikacje internetowe.