ICTCK 2015

The International Congress on Technology, Communication and Knoeledge

NOV2015, Islamic Azad University, Mashhad Branch

ICTCK2015.IR

Islamic Azad University

Mashhad Branch

Certification of Acceptance

It is certified that

Ala EkramiFard, Mohsen Kahani

Presented a paper, titled

Providing a Source Code Security Analysis Model Using Semantic Web Techniques

During the second International Congress on Technology, Communication and Knowledge (ICTCK 2015), held on November 11-12 Mashhad Branch, Islamic Azad University.

Dr. Saeed Toosizadeh	**Dr. Mahdi Yaghoobi**	**Dr. Mehrdad Jalali**
Vice Chancellor for Research and Technology	Congress President	ICKIS Co-Chair
Islamic Azad University, Mashhad Branch		

ICTCK

IEEE IRAN SECTION

ISC

Sponsored and Indexed by CIVILICA We Respect the Science

ICTCK2015.IR

ICTCK2015.IR

Providing a Source Code Security Analysis Model Using Semantic Web Techniques

Ala EkramiFard
Dept. of Computer Engineering
Ferdowsi University of Mashhad
Mashhad, Iran
ekramifard@staff.um.ac.ir

Mohsen Kahani
Dept. of Computer Engineering
Ferdowsi University of Mashhad
Mashhad, Iran
kahani@um.ac.ir

Abstract— **Security is one of the main issues in all phases of the software life cycle. Since most software vulnerabilities occur in coding phase, so the secure implementation is very important. Semantic Web ontology expresses the concept of a specific area. According to variety of software systems and manufacturing techniques, the Semantic Web can be effective in production of software systems. Anthology helps to review security holes and bugs in source code and produces appropriate reports.**

To overcome the problem of variety of source code language, in this paper, an ontology approach for source code security analysis model has been used. In this model, the source code is represented in terms of the RDF triples. The security error patterns are provided in the form of SPARQL queries. The result shows that this approach is promising and can effectively find the security flaw patterns in source codes. Experimental evaluations demonstrate that this approach is feasible and finds bug patterns that implemented. The main advantage of this method is the independence of code analysis and error inference sections so each parts can be developed.

Keywords: Security analysis, source code, semantic web ontology

I. Introduction

Current computer systems are connected to each other by global networks. It can provide more threats for software systems. Software engineering helps to develop a qualified computer application in some steps: Analysis, Design, Implementation, Testing and Maintenance. Security as a measure of software quality should be considered at all steps of software development life cycle. Thus, in all steps of software engineering, security engineer helps to create a secure application. Nowadays in the computer world there are many complex attacks and threats. Safe software design includes use of static analysis tools, security inspections and security testing. Static analysis, by examining the code, identifies and reports security holes and vulnerabilities.

Vulnerability, a weakness in software system, can be exploited in security attacks [16]. Since the bulk of security weaknesses is created in the implementation step, using source code analysis tools help to identify and reduce security holes in applications. Manual auditing, as a static analysis method, is very time-consuming and static analysis tools are so faster. Various methods have been introduced for static code analysis. Static security analysis tools provide testers with a variety of methods to evaluate code for common vulnerabilities. Static code analysis tools are classified by the languages that it covered or how it works.

Semantic web as a new generation of the web, by semantic code analysis help to create automated tools for software development and systems analysis. Semantic web presents new structure for concept modeling that helps to infer new results and reuse facilities. We can do security analysis on code using semantic web techniques. In these methods, parser creates an internal representation of the source code by analyzing the code. This representation is stored as RDF or OWL, and vulnerability patterns are presented as semantic query. Then, the inference or query tools find vulnerability patterns in this intermediate code.

The main advantage of this method is the possibility of developing the errors patterns and reusing the tool in future. Semantic analysis and vulnerable point presentation using ontology can help us to discover new and more complex error patterns.

In rest of this paper, first in Section II related works in code analysis and code security analysis are briefly reviewed. Then, in Section III the proposed model, which includes a parser and the inference engine is introduced and each part is explained. Finally, in Section IV the model was evaluated and in Section IV a summary of the paper and future works are presented.

II. Related Work

Static analysis of source code

Code analysis consists of three main components: parser, internal representation and analysis. Parser identifies components of the code using lexical or syntactic parsing and converts the source code to one or more internal representations. The second part, designed based on the programming language compiler, provides a suitable display for automated analysis. The last part of the code analysis tool does the intended analysis on the internal representation. Static analysis tools look for a fixed set of patterns or rules, in the code so they can't find all security problems. Advanced tools allow to added new rules over time [1].

Many automatic and semi-automatic static code analysis methods have been introduced. Binkley introduced static analysis as process of extracting program data from source code [3]. Static analysis is used in many fields of software engineering. Architecture discovering, automation software engineering, concept understanding, model based development, optimization techniques for software engineering, performance analysis, program evaluation, review of security etc. [3]. In security analysis source code is explored and security bugs or vulnerable functions are identified without running the code.

Some analytical tools work based on lexical analysis like Grep [1]. This method processes the source code and converts it to symbols. This symbols' flow is compared with a library of vulnerabilities. Grep, based on the easiest method of static analysis, does a simple string search on the text file and extracts the necessary information.

ITS4 [11], static vulnerability scanning tool for C and C++ code, helps to identify security holes in programs by simple rules. ITS4 breaks source file into a series of lexical tokens, and matches patterns in that stream to find non-regular patterns. For example some functions, like *sprint* and *strcat*, lead to buffer overflow problem. This tool scans the code to fine them.

Flawfinder [6], a database-centric analysis tool, identifies security holes in C and C++ source code and prioritizes them based on the risk level. Flawfinder uses a database of problematic function calls to detect and report possible vulnerabilities.

RATS [10] is an open source review tool security in programming source code like C, C++, Perl, PHP and Python, and discovers vulnerabilities such as buffer overflow.

Other types of code analysis tools, work based on syntax analysis. Some of them create an *Abstract Syntax Tree* (AST) form code, and then execute a semantic analysis on the code [1]. AST is a tree representation of the abstract syntactic structure of source code written in a programming language. Each node of the tree shows a construct occurring in the source code. Abstract syntax trees are a data structure widely used in compilers, due to their property of representing the structure of program code. An AST is usually the result of the syntax analysis phase of a compiler.

Livshits and Lam [20] use syntax analysis to identify security vulnerabilities in Java. Their tool identifies HTTP splitting, XSS and SQL injection vulnerability as the most common web application security holes.

Codeminer tool [7] converts PHP source code into AST intermediate model. Abstract Syntax Tree represents syntax structure of source code as tree view and each node represents a contract in code. Then, a security analyst reviews all defined vulnerabilities in tool and finally reports security holes in the code.

FindBugs [5], open source static analysis tool for Java, uses byte code analyzing. This tool auditing java byte code using BCEL libraries then identifies and reports common vulnerabilities like XSS and SQL injection.

Code analysis using Semantic Web techniques

Paydar and Kahani [19] propose an approach for design pattern detection from source code by semantic web. Design pattern of a software system helps to development and reuse the code. It is based on the semantic data model as the internal representation, and on SPARQL query execution as the analysis mechanism.

Zheng et al. [9] propose a parser, a combination of lexical and syntactic analysis, that converters C source code to XML intermediate model and also safety rules are translated into vulnerability patterns. Finally XQuery phrases, the most common XML query language, compares XML intermediate model with vulnerability patterns and security holes are identified.

Zhang et al. [21] propose an approach to audit and reason about the security concerns based on ontology. In this method, source code conceptual understanding is an iterative process of concept recognition and relationship discovery in code.

Yu et al. [13] propose a static analysis approach to identify vulnerabilities in code based on ontology model. In *Security Vulnerability Detection* (SVD) prototype of this tool, the source code is converted to OWL language, then compared with the vulnerable patterns that are modeled by SWRL rules and finally code security holes are reported.

The work of this paper has similarities with [19], but it is different in use and proposes new task to security bug detection.

The main difference of the method introduced in this paper to [13] and [21] is the independence of code analysis and error inference sections. Each sections can expand and reused in similar tools.

III. Proposed Approach

Our model is implemented based on [19] and display code elements and the relationships by semantic model. Parser converts source code to RDF triples and SPARQL queries identify security holes in the code.

This model consists of two parts: Parsing and Reasoning.

1. Parsing : Parser gets a source code as input and makes Abstract Syntax Tree for it. Then, it discovers code components and relationships between them by tree nodes navigation and produces RDF triples. Triples are stored in an

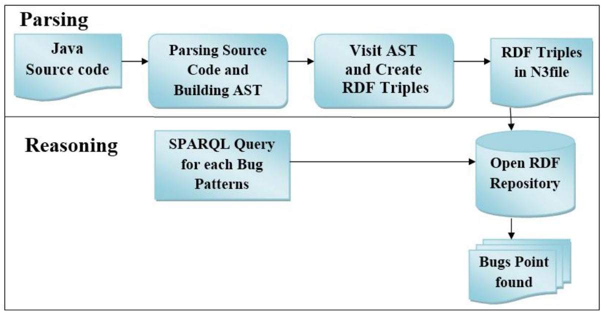
Parsing
Java Source code
Parsing Source Code and Building AST
Visit AST and Create RDF Triples
RDF Triples in N3file
Reasoning
SPARQL Query for each Bug Patterns
Open RDF Repository
Bugs Point found

Figure 1. Proposed model architecture

n3 file, and can be loaded in the RDF repository and the inference runs on it.

2) Reasoning : A SPARQL query is designed for any security holes that may exist in the code and runs on the RDF repository of the previous step. Bugs points are found and reported.

Fig. 1 shows the proposed model architecture. The parser, queries structure and inference are explained in more details, in here. In rest main components are explained.

Parser

This parser is developed by extending the *J2RDF*, in Web Technology Lab[1], and implemented as an Eclipse plugin. The parser reviews certain parts of the code, and the proposed parser added new features to the base code.

WTLab parser identifies classes, methods and code control expression, but not the concepts such as *interface*, *inheritance* and *implemented* classes. The proposed model implements the functions required to extract these concepts.

The parser, provided by the class *ASTParsetr*, uses the Eclipse *Abstract Syntax Tree (AST)* API for parsing source code. This parser gets Java program as project and analyzes it. The result of the analysis of the code is a syntax tree that stores elements of the code and the relationships between them in their nodes.

J2RDFVisitor class is defined to visit AST nodes. Method *visit* navigates the resulting tree nodes and analyzes the code, identifies the components of the code and discovers the relationships between them. Every subclass of *ASTNode* contains specific information for the Java element.

Then, the parser identifies elements of the code, converts them to RDF triples and stores in a repository. Fig. 2 shows sample output of code analysis.

1 http://wtlab.um.ac.ir

```
j2rdf: rulessecurity    j2rdf:Interface
j2rdf: rulessecurity _SER2.

j2rdf: rulessecurity _SER2     rdf:type
j2rdf:Interface.

j2rdf: rulessecurity _SER2  j2rdf:SuperInterface
" java.io.Serializable "
```

Figure 2. Output of code analysis

Error Patterns

Various tools are provided to identify security holes in code. FindBugs [5], a syntax analysis tool, gets a Java binary file as input and analyzes components. Then checks 11 introduced error patterns and reports the vulnerable point in code. PMD [17] introduces just 2 templates to identify security error code. PMD gets Java source code and identifies components by AST, then visits functions identify error patterns. Jtest tool [12] identifies 6 security holes patterns in Java code.

The proposed model implements 3 patterns of Jtest, 2 patterns of FindBugs and 1 pattern of PMD. A semantic query in SPARQL is implemented for each error pattern and runs on the RDF repository. In the following one of these patterns has been described.

SecuritySer pattern disallows classes that implement the '*java.io.Serializable*' "interface" that may be cases security hole.

Hackers can read the internal state of the object by examining a byte array that contains the series object. It then can access private fields and the object referred. Fig. 3 shows the query that is implemented to identify this pattern.

```
SELECT  *  WHERE {
        ?cls   rdf:type    j2rdf:class.
        ?cls   j2rdf:Implements   ?intf .
        ?intf  rdf:type      j2rdf:interface .
        ?intf  j2rdf:interface  "Serializable".}
```

Figure 3. SecuritySer pattern query

Inference

After converting source code to the semantic model and extracting RDF triples, the inference and errors discovery is done. SPARQL queries, representing error patterns in code that are defined in the previous step, are executed on the RDF repository.

In this step, OpenRDF Sesame [15] is used to manage and manipulate the RDF repository. OpenRDF Sesame is a standard framework for RDF data processing, including facilities for analysis, storing RDF repository, conclusion and implementation of the SPARQL queries. A new repository is defined and n3 file from the analysis is loaded, then the defined queries run on it.

IV. Experimental Evaluation

In order to evaluate the proposed model, a test case contains a variety of studied errors, is implemented and analyzed by FindBugs and our model. Because our model is still in the development phase, we only focus on the bug patterns that our model has implemented. TABLE I shows the experimental results of bugs found by these tools.

As Tab. I shows, our model can finds three bug patterns: *ConstantDBPass, MReInternalArray, SecuritySer* in case study and FindBugs finds *ConstantDBPass*.

TABLE I. Bug Pattern Detection Capability

Bug Pattern Name	Proposed Model	FindBugs
ConstantDBPass	√	√
MReInternalArray	√	×
SecuritySer	√	×

√. Detect
×. Not Detect

To conduct our experiment, open source project JHotDraw7.0.6 has been studied. The Parser gets the source code as input and produces the RDF repository of code. The result file is stored in JHote.n3 file. Then a new RDF repository is added in openrdf-workbench and triples extracted from the code are loaded. Queries related to defined error patterns are executed on repository. Tab. II shows the results.

TABLE II. Number Of Bugs Detected In JHotDraw7.0.6

# of RDF Triples	# of Bugs (Bug Pattern Name)		
	SecuritySer	SecuritySer2	MReInternalArray
19012	6	7	12

According to the evaluation, the proposed model on reviewed projects, 19,012 RDF triples are extracted that produce ontology of the source code. The proposed model detects 6 errors of *SecuritySer* error patterns, 7 errors of *SecuritySer2* error patterns and 12 errors of *MReInternalArray* error patterns that it had.

Tab. III shows the number of bugs found by the proposed model and PMD on *MReInternalArray* pattern. Both tools detected 12 errors on *MReInternalArray* pattern.

TABLE III. Bug Detection Comparison

Bug Pattern Name	# of Bugs Proposed Model	# of Bugs PMD
MReInternalArray	12	12

V. Conclusion and Future Works

Security is one of the most important requirements in the software production process. Static analysis at the implementation step, by examining the structure of the code, identifies and reports security holes. This paper proposes a semantic model for source code security analysis. Java source code is converted to an intermediate model, then semantic SPARQL queries audit the code.

Proposed parser uses syntax analysis that unlike lexical analysis method is more efficient. Lexical approaches just review elements in the code without identify and analyze the relationships between elements. Proposed method uses syntax analysis, audits elements such as classes, methods, relationships, control statements, also analyzes the relationship between them.

The main advantage of this method in compare with [13] and [21], is the independence of code analysis and error inference sections. The proposed parser traces the code and stores in form of RDF triple. This intermediate representation of the code can be reused in other processes of evaluation or pattern identification. We can identify more elements in code and complete relationships between them by developing a better parser. Also inference step can be improved by adding new error patterns as SPARQL queries.

VI. References

[1] B. Chess, G. McGraw, “Static analysis for security”, *Security & Privacy, IEEE 6*, 2004.

[2] B. Motik, A. Maedche, R.Volz, “Ontology Representation & Querying for Realizing Semantics-driven Applications”, *FZI Research Center for Information Technologies at the University of Karlsruhe*, Germany, 2003.

[3] D . Binkley, “Source code analysis: A road map”, *Future of Software Engineering*, 104-119, 2007.

[4] D. Milano, “Ontology Representation & Reasoning, Maurizio Lenzerini”, Antonella Poggi Dipartimento di Informatica e Sistemistica Antonio Ruberti Universit di Roma La Sapienza , Roma, Italy, 201of2.

[5] FindBugs, *Find Bugs in Java Programs*, Available: http://findbugs.sourceforge.net, 2013.

[6] Flawfinder, *Home Page,* Available:http://dwheeler.com/flawfinder, 2012.

[7] G. Agosta, A. Barenghi, A. Parata, , G. Pelosi, “Automated Security Analysis of Dynamic Web Applications through Symbolic Code Execution”, *Ninth International Conference on Information Technology: New Generations (ITNG)*, 189 – 194*)*, 2012.

[8] H. J. Happel, S. Seedof, “Applications of Ontologies in Software Engineering”, 2nd International Workshop on Semantic Web Enabled Software Engineering, 2006.

[9] H. Zheng, K. Zhou, X. Lai, C. Liu, “Research on XML Based Static Software Security Analysis”, *Second World Congress on Software Engineering (WCSE)*, 141 - 144 , 2010.

[10] HP Fortify, *Rough Auditing Tool for Security*, Available: https://www.fortify.com/ssa-elements/threat-intelligence/rats.html, 2012.

[11] J. Viega, J. Bloch, Y. Kohno, G McGraw, “ITS4: a static vulnerability scanner for C and C++ code”, *16th Annual Conference on Computer Security Applications*, Dulles, USA , 2000.

[12] Jtest, *Built-in Static Analysis Rules*, Available: http://www.technology.heartland.edu/courses/ Computer Science/Programming/Java Courses/JTest Users Guide/r_index.htm, 2013

[13] L. Yu, SZ. Wu, T. Guo, GW. Dong, CC. Wan, “Ontology model-based static analysis of security vulnerabilities”, Find out how to access preview-only contentInformation and Communications SecurityLecture Notes in Computer Science Volume 7043, 330-344, 2011.

[14] M. Howard , S. “Lipner,The Security Development Lifecycle”, *MSPress*, May 2006.

[15] OpenRDF, *home of Sesame*, 2013 ,Available: http://www.openrdf.org , 201

[16] OSVDB, *Open Sourced Vulnerability Database*, Available: http://www.osvdb.org, 2014.

[17] PMD, *Security Code Guidelines*, Available: http://pmd.sourceforge.net/pmd-5.0.5/rules, 2014

[18] R.K. McLean, “Comparing Static Security Analysis Tools UsingOpen Source Software”, *Sixth International Conference Software Security and Reliability Companion (SERE-C)*, 2012.

[19] S. Paydar, M. Kahani, “A Semantic Web based approach for design pattern detection from source code” , *2nd International eConference on Computer and Knowledge Engineering (ICCKE)*, 2012.

[20] V. Livshits, M. Lam, “Finding security vulnerabilities in Java applications with static analysis”, *14th conference on USENIX Security,* 2005.

[21] Y. Zhang, J. Rilling, V. Haarslev ,”An ontology-based approach to software comprehension-reasoning about security concerns”, *30th Annual International Computer Software and Applications Conference(COMPSAC),* 333 – 342, 2006.