IJCSE International Journal of Computer Sciences and Engineering Open Access
Review Paper Volume-4, Issue-11 E-ISSN: 2347-2693

Efficient Code Clone Analysis to Detect Vulnerability in Dynamic Web Applications

Vineetha K R[1]* and N. Santhana Krishna[2]

[1,2]Department Of Computer Science
AJK College of Arts And Science, Coimbatore
(Affiliated To Bharathiar University, Coimbatore & Approved By Govt. Of Tamil Nadu)
vpvprakash@gmail.com and seema.80.sk@gmail.com

Available online at: www.ijcseonline.org

Received: Oct/26/2016 Revised: Oct/31/2016 Accepted: Nov/22/2016 Published: Nov/30/2016

***Abstract*—** In this system an approach to clone analysis and Vulnerability detection for Web applications has been proposed together with a prototype implementation for web pages. Our approach analyzes the page structure, implemented by specific sequences of HTML tags, and the content displayed for both dynamic and static pages. Moreover, for a pair of web pages we also consider the similarity degree of their java source. The similarity degree can be adapted and tuned in a simple way for different web applications. We have reported the results of applying our approach and tool in a case study. The results have confirmed that the lack of analysis and design of the Web application has effect on the duplication of the pages. In particular, these results allowed us to identify some common features for the web pages that could be integrated, by deleting the duplications and code clones. Moreover, the clone analysis and Vulnerability detection of the pages enabled to acquire information to improve the general quality and conceptual/design of the database of the web application. Indeed, we plan to exploit the results of the code clone analysis method to support web application reengineering activities.

***Keywords*—** Vulnerability Detection, Code Clone, Dynamic Webpages, Duplication

I. INTRODUCTION

Code Vulnerabilities are similar program structures of considerable size and significant similarity. Several studies suggest that as much as 20-50 percent of large software systems consist of Vulnerability code . Knowing the location of Vulnerabilities helps in program understanding and maintenance. Some Vulnerabilities can be removed with refactoring , by replacing them with function calls or macros, or we can use unconventional meta level techniques such as Aspect-Oriented Programming or XVCL to avoid the harmful effects of Vulnerabilities.

Vulnerability detection is an active area of research, with a multitude of Vulnerability detection techniques been proposed in the literature . One limitation of the current research on code Vulnerabilities is that it is mostly focused on the fragments of duplicated code (we call them simple Vulnerabilities), and not looking at the big picture where these fragments of duplicated code are possibly part of a bigger replicated program structure. We call these larger granularity similarities structural Vulnerabilities. Locating structural vulnerabilities can help us see the forest from the trees, and have significant value for program understanding, evolution, reuse, and reengineering.

Vulnerability detection tools produce an overwhelming volume of simple vulnerabilities' data that is difficult to analyze in order to find useful vulnerabilities. This problem prompted different solutions that are related to our idea of detecting structural vulnerabilities. Some vulnerability detection approaches target large-granularity vulnerabilities such as similar files, without specifying the details of the low-level similarities contained inside them. For example, the authors consider a whole webpage as a "vulnerability" of another page if the two pages are similar beyond a given threshold, computed as the Levenshtein distance. Without the details of the low-level similarities in the large-granularity vulnerabilities, it is not always straightforward to take remedial actions such as refactoring or creating generic representation, as these actions require a detailed analysis of low-level similarities. Moreover, Vulnerability Miner goes a step ahead in vulnerability analysis, by looking at the bigger similarity structures consisting of groups of such highly similar files.

Organization of the Paper

The Papers is organized as the following chapters
Chapter 1 includes the Abstract of the proposed paper
Chapter 2 includes the Introduction of the research
Chapter 3 includes the Proposed System
Chapter 4 includes the Methodology of the Research
Chapter 5 includes The Conclusion of the Research
Chapter 6 includes the references made for the research

3. PROPOSED SYSTEM

Problem Definition

To find if automatically produced Vulnerability report summaries can help a developer with their work, the system conducted a task-based evaluation that considered the use of summaries for Vulnerability report duplicate detection tasks. system found that summaries helped the study participants save time, that there was no evidence that accuracy degraded when summaries were used and that most participants preferred working with summaries to working with original Vulnerability reports. Many existing text summarizing approaches exist that could be used to generate summaries of Vulnerability reports. Given the strong similarity between Vulnerability reports and other conversational data

Objective

We propose an approach to automatically testing and refactoring modern web applications and detect duplicated pages in dynamic Web sites and on the analysis of both the page structure, implemented by specific sequences of HTML tags, and the displayed content. In addition, for each pair of dynamic pages we also consider the similarity degree of their scripting code. The similarity degree of two pages is computed using different similarity metrics for the different parts of a web page based on the code duplication string edit distance. We have implemented a prototype to automate the clone detection process on web applications developed using technology and used it to validate our approach

4. METHODOLOGY

Reusable Mechanism and Code Consistency

A code Vulnerability is a code portion in source files that is identical or similar to another. It is common opinion that code Vulnerabilities make the source files very hard to modify consistently. Vulnerabilities are introduced for various reasons such as lack of a good design, fuzzy requirements, undisciplined maintenance and evolution, lack of suitable reuse mechanisms, and reusing code by copy-and-paste. Thus, code Vulnerability detection can effectively support the improvement of the quality of a software system during software maintenance and evolution.

The Internet and World Wide Web diffusion are producing a substantial increase in the demand of web sites and web applications. The very short time-to-market of a web application, and the lack of method for developing it, promote an incremental development fashion where new pages are usually obtained reusing (i.e. "cloning") pieces of existing pages without adequate documentation about these code duplications and redundancies. The presence of Vulnerabilities increase system complexity and the effort to testing and refactoring , maintain and evolve web systems, thus the identification of Vulnerabilities may reduce the effort devoted to these activities as well as to facilitate the migration to different architectures.

This project proposes an approach for detecting Vulnerabilities in web sites and web applications, obtained tailoring the existing methods to detect Vulnerabilities in traditional software systems. The approach has been assessed performing analysis on several web sites and web applications.

Software Environment and Maintenance

Maintaining software systems is getting more complex and difficult task, as the scale becomes larger. It is generally said that code Vulnerability is one of the factors that make software maintenance difficult. This project also develops a maintenance support environment, which visualizes the code Vulnerability information and also overcomes the limitation of existing tools.

Collaborative Structures and Message Passing

One limitation of the current research on code Vulnerabilities is that it is mostly focused on the fragments of duplicated code (we call them simple Vulnerabilities), and not looking at the big picture where these fragments of duplicated code are possibly part of a bigger replicated program structure. We call these larger granularity similarities structural Vulnerabilities. Locating structural Vulnerabilities can help us see the forest from the trees, and have significant value for program understanding, evolution, reuse, and reengineering. The samples are abstracted from Vulnerabilities found in Project Collaboration portals developed in industry using ASP and JEE and a PHP-based portal developed in our lab study. Structural Vulnerabilities are often induced by the application domain design technique or mental templates used by programmers. Similar design solutions are repeatedly applied to solve similar problems

Detection granularity structural Vulnerabilities

Reuse only what is similar, knowing Vulnerabilities helps in reengineering of legacy systems for reuse. Detection of large-granularity structural Vulnerabilities becomes particularly useful in the reuse context . While the knowledge of structural Vulnerabilities is usually evident at the time of their creation, we lack formal means to make the presence of structural Vulnerabilities visible in software, other than using external documentation or naming conventions. The knowledge of differences among structural Vulnerability instances is implicit too, and can be easily lost during subsequent software development and evolution. The limitation of considering only simple Vulnerabilities is known in the field . The main problem is the huge number of simple Vulnerabilities typically reported by Vulnerability detection tools. There have been a number of attempts to move beyond the raw data of simple Vulnerabilities. It has been proposed to apply classification, filtering, visualization, and navigation to help the user make sense of the cloning information. Another way is to detect Vulnerabilities of larger granularity than code fragments. For example, some Vulnerability detectors can detect Vulnerability files, while others target detecting purely conceptual similarities using information retrieval methods rather than detecting simple Vulnerabilities.

Template Extraction and Code Stealing

Generally speaking, templates, as a common model for all pages, occur quite fixed as opposed to data values which vary across pages. Finding such a common template requires multiple pages or a single page containing multiple records as input. When multiple pages are given, the extraction target aims at page-wide information. When single pages are given, the extraction target is usually constrained to record wide information, which involves the addition issue of record-boundary detection. Page-level extraction tasks, although do not involve the addition problem of boundary detection, are much more complicated than record-level extraction tasks since more data are concerned. A common technique that is used to find template is alignment: either string or tree alignment. As for the problem of distinguishing template and data, most approaches assume that HTML tags are part of the template, while EXALG considers a general model where word tokens can also be part of the template and tag tokens can also be data. However, EXALG's approach, without explicit use of alignment, produces many accidental equivalent classes, making the reconstruction of the schema not complete.

Traditional Classification, Filtering, Visualization

Detection of large-granularity structural Vulnerabilities becomes particularly useful in the reuse context. While the knowledge of structural Vulnerabilities is usually evident at the time of their creation, we lack formal means to make the presence of structural Vulnerabilities visible in software, other than using external documentation or naming conventions. The knowledge of differences among structural Vulnerability instances is implicit too, and can be easily lost during subsequent software development and evolution. The limitation of considering only simple Vulnerabilities is known in the field. The main problem is the huge number of simple Vulnerabilities typically reported by Vulnerability detection tools. There have been a number of attempts to move beyond the raw data of simple Vulnerabilities. It has been proposed to apply classification, filtering, visualization, and navigation to help the user make sense of the cloning information. Another way is to detect Vulnerabilities of larger granularity than code fragments. For example, some Vulnerability detectors can detect Vulnerabilityd files, while others target detecting purely conceptual similarities using information retrieval methods rather than detecting simple Vulnerabilities. The approach described in this paper is also based on the idea of applying a follow-up analysis to simple Vulnerabilities' data. We observed that at the core of the structural Vulnerabilities, often there are simple Vulnerabilities that coexist and relate to each other in certain ways. This observation formed the basis of our work on defining and detecting structural Vulnerabilities. From this observation, we proposed a technique to detect some specific types of structural Vulnerabilities from the repeated combinations of collocated simple Vulnerabilities.

Extracting Structured Data from Web Pages

Many web sites contain large sets of pages generated using a common template or layout. For example, Amazon lays out the author, title, comments, etc. in the same way in all its book pages. The values used to generate the pages (e.g., the author, title,...) typically come from a database. In this paper, we study the problem of automatically extracting the database values from such template generated web pages without any learning examples or other similar human input. We formally define a template, and propose a model that describes how values are encoded into pages using a template. We present an algorithm that takes, as input, a set of template-generated pages, deduces the unknown template used to generate the pages, and extracts, as output, the values encoded in the pages. Experimental evaluation on a large number of real input page collections indicates that our algorithm correctly extracts data in most cases.

Proposed Methodology

The WWW distribution are generating a considerable boost in the order of web sites and web applications. A code similarity is a code portion in source files that is matching or similar to another. It is general view that code clones make the source files very hard to modify constantly. Clones are launched for various reasons such as lack of a good design, fuzzy requirements, disorderly protection and evolution, lack of suitable reuse mechanisms, and reusing code by copy-and-paste. Thus, code clone detection can effectively support the improvement of the quality of a software system during software preservation and growth.

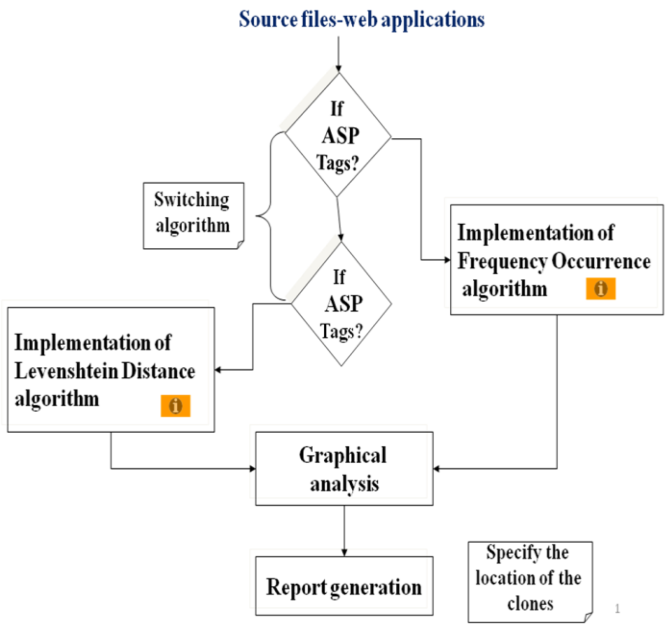
Source files-web applications
If ASP Tags?
Switching algorithm
Implementation of Frequency Occurrence algorithm
If ASP Tags?
Implementation of Levenshtein Distance algorithm
Graphical analysis
Report generation
Specify the location of the clones

Figure 1: Architecture Diagram [12]

The very short time-to-scope of a web application, and the need of method for developing it, support an increase al expansion fashion where new pages are usually obtained reusing (i.e. "cloning") pieces of existing pages without sufficient documentation about these code duplications and redundancies. The presence of clones increase system difficulty and the effort to test, maintain and change web systems, thus the identification of clones may reduce the effort devoted to these activities as well as to facilitate the migration to different architectures.

5. CONCLUSION

The survey focus on describing our approach for function clone detection for detecting Vulnerabilities to web applications with the main goal to assess the effectiveness and efficiency of the approach, and measure the extent Vulnerability detection opportunities. The research tries web applications from the public domain, for which we did not have expectations about how much duplication and Vulnerabilities exists, and one web application from the research domain for which it was known that there were many duplicated tag functions. The first static web application, is a basic auction application that can be integrated into other web sites to add simple auctions features. The second dynamic web applications and third is dynamic web application with different programming language applications, respectively, are both web-based Levistein Distance.

REFERENCES

[1] J. Anvik, L. Hiew, and G.C. Murphy, "Coping with an Open Vulnerability Repository," Proc. OOPSLA Workshop Eclipse Technology eXchange, **2005.**

[2] J. Anvik, L. Hiew, and G.C. Murphy, "Who Should Fix This Vulnerability?" Proc. 28th Int'l Conf. Software Eng. (ICSE '06), **2006.**

[3] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Duplicate Vulnerability Reports Considered Harmful; Really?" Proc. IEEE 24th Int'l Conf. Software Maintenance (ICSM '08), **2008.**

[4] J. Davidson, N. Mohan, and C. Jensen, "Coping with Duplicate Vulnerability Reports in Free/Open Source Software Projects," Proc. IEEE Symp. Visual Languages and Human-Centric Computing (VL/HCC '11), **2011.**

[5] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of Duplicate Defect Reports Using Natural Language Processing," Proc. 29th Int'l Conf. Software Eng. **2007**

[6] A.J. Ko, B.A. Myers, and D.H. Chau, "A Linguistic Analysis of How People Describe Software Problems," Proc. IEEE Symp. Visual Languages and Human-Centric Computing (VL-HCC '06), **2006**

[7] N. Bettenburg, S. Just, A. Schr€oter, C. Weiss, R. Premraj, and T. Zimmermann, "What Makes a Good Vulnerability Report?" Proc. 16th Int'l Symp. Foundations of Software Eng. (FSE '08), **2008**

[8] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann, "Information Needs in Vulnerability Reports: Improving Cooperation between Developers and Users," Proc. ACM Conf. Computer Supported Cooperative Work (CSCW '10), **2010**

[9] R.J. Sandusky and L. Gasser, "Negotiation and the Coordination of Information and Activity in Distributed Software Problem Management," Proc. Int'l ACM SIGGROUP Conf. Supporting Group Work (GROUP '05), **2005**

[10] D. Bertram, A. Voida, S. Greenberg, and R. Walker, "Communication, Collaboration, and Vulnerabilities: The Social Nature of Issue Tracking in Small, Collocated Teams," Proc. ACM Conf. Computer Supported Cooperative Work (CSCW '10), **2010**.

[11] R. Lotufo, Z.Malik, andK. Czarnecki, "Modelling the 'Hurried' Vulnerability Report Reading Process to Summarize Vulnerability Reports," Proc. IEEE 28th Int'l Conf. Software Maintenance (ICSM'12), **2012.**

[12] S. Mani, R. Catherine, V.S. Sinha, and A. Dubey, "AUSUM: Approach for Unsupervised Vulnerability Report Summarization," Proc. ACM SIGSOFT 20th Int'l Symp. the Foundations of Software Eng. (FSE '12), article 11, **2012**

[13] S. Haiduc, J. Aponte, L. Moreno, and A. Marcus, "On the Use of Automated Text Summarization Techniques for Summarizing Source Code," Proc. 17th Working Conf. Reverse Eng. (WCRE '10), pp. 35-44, **2010**

[14] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Vijay Shanker, "Towards Automatically Generating Summary Comments for Java Methods," Proc. 25th Int'l Conf. Automated Software Eng. (ASE '10), pp. 43-52, **2010**

[15] Jyotsnamayee Upadhyaya, Namita Panda and Arup Abhinna Acharya "Attack Generation and Vulnerability Discovery in Penetration Testing using Sql Injection " International Journal of Computer Science and Engineering ,Volume-2, Issue-3 ,E-ISSN: 2347-2693 , **2014**

AUTHORS PROFILE

Vineetha Prakash, is currently doing her M.Phil in Computer Science, Department of Computer Science, AJK college of arts and Science , Coimbatore, her research area includes Knowledge and web Mining.
vpvprakash@gmail.com