A Novel Hybrid Approach for Detecting Recurring Code Clones in Open Source

Nitin Gothwal*

Department of Computer Science & Engineering, Swami Keshwanand Institute of Technology, Jaipur Rajasthan, India

Copyright 2018 © Nitin Gothwal. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Abstract
Open source criterion and their paradigms are rapidly now becoming a well entrenched approach in the development process of applications and information systems. Copy paste programming approaches may cause severe effects on the recently developing system as it may lead to concealed dependencies and vulnerabilities in distinct parts of the system. The detection of vulnerabilities in source code is a central issue and a primary key for the computer system security. A minor flaw in a code base can severely weakens the security of a computer system and makes it an easy victim for the attackers. Manual detection of these vulnerabilities is very tedious, time consuming and also not feasible. Several detection techniques have been proposed to detect the code clones ranging from tokens, plain text to parse tree and program dependency graph along with some kind of hybrid approaches. So, there arises a need of an approach which automatically detects the bugs and reports them to the developer thereby enhancing the security of computer system. This work proposes a hybrid approach for detecting recurring bugs in large software systems. The novel aspect of proposed hybrid approach takes the advantage of token based and fingerprint based techniques for recurring bug detection.

Keywords: Vulnerability, Open Source Software (OSS), Common Vulnerability Exposure (CVE), National Vulnerability Database (NVD), Finger Print, Normalizations, Tokenization.
1 Introduction

Open source criterion and their paradigms are rapidly now becoming a well entrenched approach in the development process of applications and information systems. Now a days, computers and internet technologies has changed the way of work, study, and interaction, yet there are multiple things about computers and software which the developers and researchers find dissatisfying. Bugs, security flaws, and other vulnerabilities appear in all the trusted programs regardless of whether it is an open source or a proprietary one. The software maintenance community view cloning as a bad activity because it unnecessarily increases the size of software which makes it more complex and increases maintenance cost. According to Common Vulnerability Exposure (CVE) and National Vulnerability Database (NVD) reports, there are vast numbers of vulnerabilities existing in today’s Open Source Software (OSS). When the new versions of these OSS are created, some part of the previous version is also copied by the programmers. So this copying and paste of the vulnerable code leads to vulnerable code clones in the later versions also which is harmful. Although, there are number of vulnerability detection tools available but they have their own limitations. So it is necessary to detect and remove the vulnerable code clones in different versions of OSS which enhances the security of OSS and to make them enterprise ready.

2 Related Work

Bellon et al. [1] classified clones into three types according to the modifications of the clones: type-I clones are exact copies, type-II clone are syntactically identical copies except for identifier names, and type-III clones are further modified copies. Johnson et al. [2] proposed a text based clone detection approach which uses fingerprints on substrings of the source code. Firstly, fixed number of lines of a code fragments are hashed and then a combination of fixed length sliding window with an incremental hash function is used to locate similar hash sequence values as clones. Ducasse S. et al. [3] proposed a lightweight approach that uses minimal parsing and simple string matching methods for comparison. J. Wiley et al. [4] proposed a three step string based approach consists of the following three steps that are Eliminate noise, Comparison and Filtration. The source code is transformed into persuasive portions by removing comments, white spaces, and other uninteresting artifacts. Optional code normalizations are also performed to rectify recall of clones. Li et al. [5] proposed a token based approach that uses subsequent mining techniques to identify code clones. Lexical analysis of source code is performed and then a token stream is generated in which all the similar types of identifiers and variables are remains same. Then, each token is hashed and the source code is converted into a sequence of hash values and then searched for similar hash values. Jiyong Jang et al. [6] developed a tool named ReDeBug which uses syntax based approach for finding unpatched code clones quickly in the entire open source systems. The original code snippet is extracted and then normalization is performed for removing white spaces, comments and then a window size is selected for tokenization of the normalized code. Bloom filter is used for finding similar hashed tokens for the entire normalized source code. Roy et al. [7] proposed an efficient text comparison technique that accurately finds near miss clones with high precision and high recall. They implemented the proposed approach and developed a clone detection tool NICAD, a loose acronym for Accurate Detection of Near-miss Intentional Clones. N. Yoshida et al. [8] proposed a method based on identifiers similarity between a query and a target source code fragment to obtain similar code fragments. They perform the lexical analysis of both the input code fragment and the target source code. Only identifiers are extracted from each of the token sequence and after performing normalization the identifiers are arranged in a list as Identifier Lists.
R. Smith et al. [9] proposed a new technique for finding similar code blocks and quantifying their similarity. Their technique can be used to find clone clusters, similar sets of code blocks and present them in rank wise according to their similarity of threshold. Code clones are detected at the block level, based on statement level fingerprints computation. All the detected similar code clone blocks are arranged rank wise according to their similarity score and similarity distance.

3 Proposed Method

A combination of token based technique having a different approach of generating fingerprint of each buggy code fragment and then queried with the target source code will produce all similar code fragments. Subsequently, manual filtration can also be applied on the detected code fragments which will help in reducing the false positive rate of the vulnerabilities in the open source system. So an automated approach is proposed to support the detection of Type-I and Type-III recurring vulnerabilities in open source system. The approach creates a knowledge base of the priory known vulnerabilities and identifies them in source code of a given system that is similar to the knowledge base.

In this paper, we propose a criterion for finding the clones of practical interest, not only syntactically same portions, but also similar portions of little variance with the actual buggy code fragment. Moreover, we have built a tool for finding Type-I with Type-III clones and implemented the proposed criterion. The effectiveness of the criterion is evaluated through an experiment.

In the proposed model, a semi-automated approach for detecting the clones is developed. A knowledge base of prior known/reported vulnerabilities and vulnerable code fragments is maintained for the detection and resolution purpose. The proposed approach is illustrated in figure 1.

In order to build the knowledge database, vulnerable code fragments with available patches are collected from the Open Security Vulnerability Databases. From CVE and NVD repositories, the vulnerable and patched code fragments are then extracted and stored in the knowledge database as code fragments. The vulnerable code fragments and the target source code are then feed into the proposed model. This model helps developers by pointing out percentage of matched vulnerable buggy code fragment tokens, so that developer can resolve it with the available patch.

4 Proposed Model

In proposed approach a semi-automated tool for detecting recurring buggy code fragments is developed. The proposed model is a two phase model.
Figure 1: Proposed approach for code clone detection

The Phases are as follows.

**Phase-1: Transformation**

Steps involved in Phase-1 are as follows:

1. **Remove spaces:** All the spaces and comments present in the code fragment are removed and the code fragment is converted into a single contiguous string.
2. **n-Gram Tokenization:** The string of code fragment is now divided into stream of n-length tokens. Here we set the minimum length of tokens i.e. MinT as 7.
3. **Hash calculation:** After tokenization, hash of each token is calculated. We used inbuilt java hashCode() function to calculate the hash of each token.
4. **Associate hash to each tokens:** In this step, hash value of each token is associated to the corresponding token as a unique identity. This is done because hash is a one-way function and we cannot obtain the original string from the hash. So by associating hash values to tokens, tokens can be re-obtained from hash values by which the buggy code fragment string can be obtained.
5. **w-length sliding window for fingerprint generation:** After generating sequence of hash values, a sliding window of w-length moves over the hash sequence. From each window, a single hash value is selected based on the proposed fingerprint generation algorithm. Thus a fingerprint pattern is created from each window. The procedure is applied for both the buggy code fragment and the target source code file. Thus, fingerprint pattern reduces the number of hashes in similarity comparison which reduces time and space complexity.

**Phase-2: Comparison**

Steps involved in Phase-2 are as follows:

6. **Identifying similar hash subsequences:** After creating fingerprint pattern of both the files, a fingerprint to fingerprint similarity matching is performed to get similar hash values in both the files.

7. **Mapping identical subsequences to target source code:** As tokens are associated with their hash values, so tokens are obtained from the hash values with their corresponding index position and line number in the target source code file.

8. **Percentage of similar tokens found in the buggy code fragment and the common fingerprint:** This step confirms the presence of buggy code fragment in the target source code file on the basis of predefined Similarity Threshold i.e. SimT.

9. **Re-obtaining buggy code fragment string:** In this step, tokens are regained from their hash values present in the common fingerprint pattern to form an approximate buggy code fragment string.

**5 Proposed Algorithm For Fingerprint Generation**

Steps involved in proposed fingerprint generation algorithm are as follows:

1. A sliding window of w-length slides over the sequence of hashes.
2. From the current window, the smallest and the largest hash value is selected to get the range of hash values in that particular window. This is done so that certain calculations can be performed on the hashes to get a unique hash value.
3. Now, take the average or mean of these two values and select the hash value just smaller than the obtained mean value.
4. If a previous hash value is again repeated from another window, then discard it. This will give only unique and non repeated hash values from the sliding window due to which more number of different hashes are obtained in the fingerprint pattern and will give more accurate result in clone detection.

**6 Result**

The experiment is conducted with the proposed approach to detect the clone types in open source. For evaluation, large open source systems such as the Linux kernel and Eclipse are chosen for cloned vulnerability and bug evaluation. The purpose behind choosing these projects is that these exhibits a variety of different programming languages such as C and Java, which helps in evaluating the commonality and language independency of the proposed approach. The target source code fragment is a Debian Squeeze argyll package file and buggy code fragment is a CVE-2006-3459 reported vulnerability file fragment. The threshold is assumed to set 85%. Above all to this percentage of detection, the clones are categorized as Type-I clone. This is taken for checking the effectiveness of proposed approach as same kind of vulnerability is also detected by the existing approach [5]. The result shows that our proposed approach identifies the cloned vulnerability exact identical to the source code and we categorised it as
Type-I code clone vulnerability. Same experiment is also performed on similar code clones to identify the Type-III vulnerability. The Result of detection is given in table 1.

<table>
<thead>
<tr>
<th>Package Name</th>
<th>File Name</th>
<th>CVE Number</th>
<th>Proposed Approach Detection</th>
<th>Vulnerability Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>argyll</td>
<td>tiff_next.c #1</td>
<td>CVE-2006-3459</td>
<td>100%</td>
<td>Type-I</td>
</tr>
<tr>
<td>argyll</td>
<td>tiff_thunder.c #1</td>
<td>CVE-2011-1167</td>
<td>89%</td>
<td>Type-I</td>
</tr>
<tr>
<td>avfs</td>
<td>decompress.c #1</td>
<td>CVE-2010-0405</td>
<td>63%</td>
<td>Type-III</td>
</tr>
<tr>
<td>glhack</td>
<td>gifread.c #4</td>
<td>CVE-2011-1782</td>
<td>88%</td>
<td>Type-I</td>
</tr>
</tbody>
</table>

7 Conclusion

Software cloning is a very common practice in software engineering. It is basically caused by the programmer’s copy and paste activities. This is also a reason of recurring bugs in large software system. These bugs propagate from one version to another version during software development. The work presented here tries to identify such kind of recurring bugs and resolves them in real software systems. In this paper, a novel hybrid fingerprint based approach is proposed which uses token based technique for detection and reporting of recurring bugs. Thus our proposed approach method is used to represent the detected vulnerable code fragments. Therefore, this work can be considered for further research using this approach for code clone detection research or similar studies.

8 Future Work

Our approach and proposed algorithm can be implemented to detect recurring code clones in open source software. The MinT and SimT parameters can be set to different values to check the accuracy and extent of our approach. A number of different types of vulnerabilities can be used to detect the presence of recurring bugs in open source software. A comparison case study can also be conducted with our proposed approach to the existing recurring clone detecting techniques.

References


https://doi.org/10.1109/TSE.2007.70725


https://doi.org/10.1109/ICSM.1999.792593


https://doi.org/10.1002/smr.317


