Studying Reuse of Out-dated Third-party Code in Open Source Projects

Pei Xia Makoto Matsushita Norihiro Yoshida Katsuro Inoue

Using existing source code as third-party code to build new software systems becomes very popular in these days. However, many existing code is keeping on updating during their life circle. Different versions of code, even out-dated, is reused by other software and spreading all over the world. This paper presents an empirical study on the reuse of out-dated third-party source code of several famous open source libraries. Given target source code, using repository mining techniques and file clone detection techniques, we identified the different versions of code in other user projects, and discovered the vulnerability information of the out-dated versions. We also investigated how user projects manage their code. The result shows that a large proportion of open source projects are reusing out-dated third-party code, and many of them are not well managed.

1 Introduction

Nowadays, using existing software to build new software systems becomes very popular. More and more high quality software are becoming open source. Even software in the industry increasingly reuse open source systems due to their reliability and cost benefits. [2]

Integrating third-party code is an important approach in code reuse. Third-party code is a set of reusable software components developed to be either freely distributed or sold by an entity other than the original vendor of the development platform, and they are wildly reused by developers all over the world [10]. However, while enjoying the benefits, developers also have to concern about the risks brought by reusing third-party code. If the reused code contain critical vulnerabilities, it will bring damage to the software system.

Out-dated third-party code in this paper represents that code of older versions containing known vulnerabilities such as software vulnerabilities which should be fixed by upgrading them to a newer version. As far as we know, currently there is few research focus on the out-dated third-party code reuse and management behavior. Our work is to do an empirical study in this area and collect quantitative and qualitative data to answer following research questions:

RQ1 What is the proportion of out-dated third-party code reused in the open source projects?

RQ2 What are the potential vulnerabilities caused by such reuse?

RQ3 How do developers manage those out-dated third-party code?

Answering these questions would be helpful in understanding the open source software, evaluating the quality of the software which reuses reused third-party code, and predicting some of the potential vulnerabilities in open source software. Also it would make developers aware of the importance of third-part code management.

Based on the code clone detection, repository mining techniques, and open source code search engines, here we propose an approach to detect-

オープンソースプロジェクトを対象としたレガシーコードの保守漏れに関する調査.

夏 沛, 松下 誠, 井上 克郎, 大阪大学大学院情報科学研究科コンピュータサイエンス専攻, Department of Computer Science, Graduate School of Information Science and Technology, Osaka University.

吉田 則裕, 奈良先端科学技術大学院大学情報科学研究科, Graduate School of Information Science, Nara Institute of Science and Technology.

コンピュータソフトウェア, Vol.30, No.4 (2013), pp.98–104.
[研究論文 (レター)] 2013 年 3 月 27 日受付.

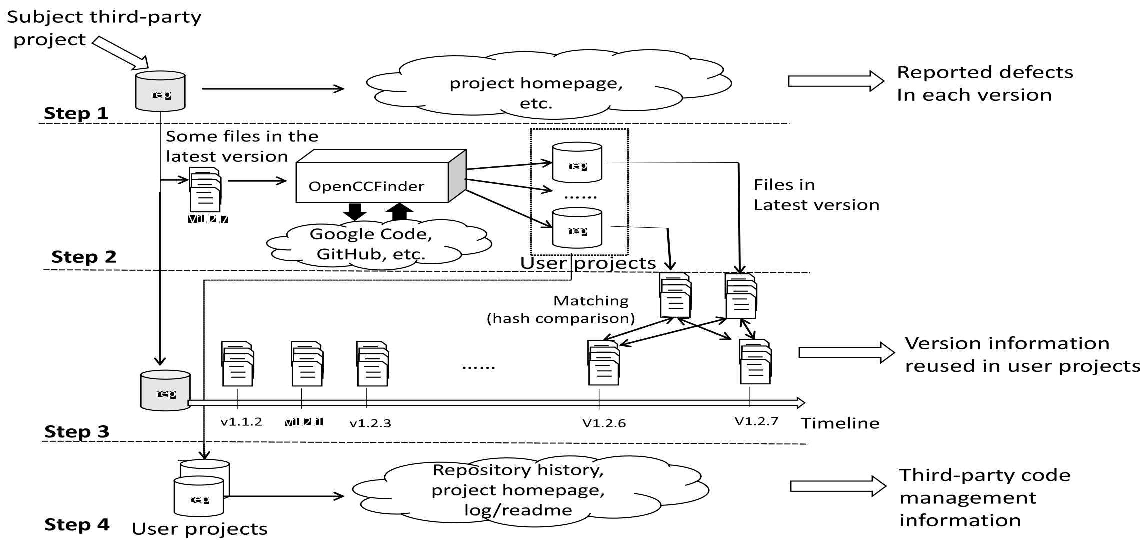
Subject third-party project
Step 1
project homepage, etc.
Reported defects In each version
Some files in the latest version
OpenCCFinder
Google Code, GitHub, etc.
Files in Latest version
User projects
Step 2
Matching (hash comparison)
Version information reused in user projects
v1.1.2
v1.2.3
V1.2.6
V1.2.7
Timeline
Step 3
Repository history, project homepage, log/readme
Third-party code management information
Step 4
User projects

Fig. 1 Study Approach Overview

ing out-dated third-party code reuse for certain libraries in open source software.

2 Study Approach

Fig.1 shows the proposed approach of this study, composed of Step 1 to 4.

2.1 Step 1: Vulnerabilities Information Collection

To explore the potential vulnerabilities of given third-party project, we did not analyze the project directly. Instead, the vulnerabilities information is manually collected from those two sources: the *National Vulnerability Database* [5] (*NVD*) and project homepage announcement.

NVD is the U.S. government repository for the vulnerability management data. It contains famous resources such as *Common Vulnerabilities and Exposures* and *CERT Vulnerability Notes Database*. Searching with keywords such as project name or filename, *NVD* returns a list of vulnerabilities information including vulnerability id, summary, published date, and CVSS Severity score. For example, one result of using keyword "libpng"[†1]:

> CVE-2011-3464 (07/22/2012): Off-by-one error in the png_formatted_warning function in pngerror.c in libpng 1.5.4 through 1.5.7 might allow remote attackers to cause a denial of service (application crash) and possibly execute arbitrary code via unspecified.

From project homepage announcement, we can also get some vulnerabilities information. Usually, when some critical vulnerabilities are found in some versions of an open source project, there would be an announcement on the project's homepage. Taking the zlib homepage as example[†2], we can find announcement such as:

> Version 1.2.3 (July 2005) eliminates potential security vulnerabilities in zlib 1.2.1 and 1.2.2, so all users of those versions should upgrade immediately.

The vulnerabilities information from these two sources are considered to be reliable.

2.2 Step 2: User Projects Searching

Nowadays, open source software hosting facilities are becoming popular. Millions of open source projects are hosted on the Internet. Google Code

†1 National Vulnerability Database (CVE-2011-3464), http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-3464

†2 zlib Home Site, http://zlib.net/

and GitHub are some of the most popular open source hosting sites. From such open source software hosting facilities, we find a list of projects that reused the subject third-party code. In this step, we use a code searching system *OpenCCFinder* (Open Code Clone Finder) helping us to do this [4] [7]. *OpenCCFinder* is a system to explore code fragments from open source repositories. This system takes a query code fragment as input, and returns the files containing code clones with the query code, along with information including project repository URL, code similarity and so on.

In this study, we select several source files from the subject third-party projects and give them to *OpenCCFinder*, and merge the search results together. Then we get a list of open source projects reusing the code of the subject third-party project.

There are a number of experimental personal projects also hosted on the open source project hosting facilities, which are not well managed or abandoned, so we manually filter out the projects considered not appropriate. The user projects in this study are formally published and actively maintained by stable organizations.

2.3 Step 3: Version Identification

The idea of identify the version number is to compare the content of each file in the user projects returned by *OpenCCFinder* with those of third-party projects. If each file of a certain version (e.g. v1.0) of a third-party code exactly matches with the files in a user project returned by OpenCCFinder, this project is probably reusing v1.0 of the third-party code.

However, when a user project tries to reuse source code from third-party project, it is possible to modify the original code. In this study, we ignore trivial modifications between source code such as adding or removing comments, blank lines, line breakers or space characters. We also ignore the rename refactoring, since such modifications would not affect the potential vulnerabilities in source code. In order to ignore the effect of these trivial modifications, each source file is tokenized before the hash values calculation and matching. Fig.2 explains the process of the hash value calculation.

We check out all the source files of each versions in history of third-party code from its repository and calculate the tokenized hash value for each file.

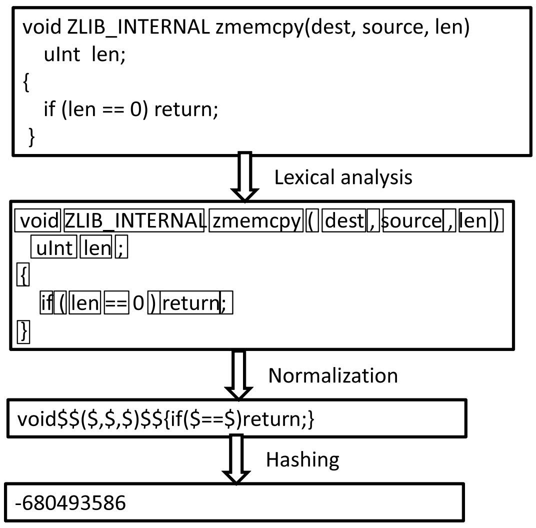
void ZLIB_INTERNAL zmemcpy(dest, source, len)
uInt len;
{
if (len == 0) return;
}
Lexical analysis
void ZLIB_INTERNAL zmemcpy (dest , source , len)
uInt len ;
{
if (len == 0) return ;
}
Normalization
void$$($,$,$)$${if($==$)return;}
Hashing
-680493586

Fig. 2 File Hash Calculation

Also, we check out the latest version of user projects and calculate the tokenized hash values. By comparing those hash values, we are able to identify the version numbers of the third-party code which are reused in the user projects precisely. In addition, if there are some user project files which do not match any files in third-party code history, we would manually look into the files to confirm whether they are modified by the user projects.

2.4 Step 4: Management Information Collection

At last, we manually investigate how the developers of the user projects have managed the third-party code.

Firstly, we check the repository commit message history to see whether the developers have ever updated the third-party code. Secondly, we check the “readme.txt” and “changelog.txt” etc. under the directory of the third-party code to find how the developers have managed the version information of the third-party code. At last, we check if there are any extra changes that might cause difficulty of management. For example, some developers change the package or directory name of the third-party libraries, or they change some of the filenames, or they mix the third-party code with their own code. Such behaviors are considered harmful for management.

Table 1 Subjects Third-party Code

Project name	zlib	libcurl (curl/lib)	libpng
Domain	data compression	file transfer	graphics
Project homepage	http://www.zlib.net/	http://curl.haxx.se/libcurl/	http://www.libpng.org/
Repository url	https://github.com/ madler/zlib.git	https://github.com /bagder/curl	git://libpng.git.sourceforge. net/gitroot/libpng/libpng
Earliest version found	v0.71	v6.5	v0.71
Release date of the earliest version found	April 1995	December 1999	July 1995
Latest Version	v1.2.7	v7.28.1	v1.5.13
Latest Release date	May 2012	November 2012	September 2012
# of version tags	65	134	150
# of source files (.c or .h) in latest version	26	222	24
Totle size of source files (.c .h) in latest version	482KB	2.77MB	1.06MB

3 Case Study

3. 1 Subject Third-party Code

Currently we have chosen three subject third-party code in different domain. These are zlib, libcurl, and libpng. They are all well-known and widely reused open source libraries. Table 1 is information of these three subject third-party projects.

3. 2 Case Study Statistics

Using OpenCCFinder, we have collected 45 user projects for zlib, 28 user projects for libcurl, and 50 user projects for libpng. The detailed data can be found in [6]. †3

3. 2. 1 Proportion of Out-dated Third-party Code

In our study, a number of versions of third-party libraries even including out-dated unsafe code are found in the user projects. Only a few projects are using the latest version libraries. Fig.3 shows the number of projects that reuse the third-party code of the different versions. Those libraries without any known software vulnerabilities are in light gray columns; the ones with warnings from the third-party project homepage are in dark gray columns; and the ones containing software vulnerabilities are in black columns. We can observe that:

- For zlib, 45 projects in this study reuse 9 different versions of zlib code. 14 projects (31.1%) are using out-dated zlib code containing potential vulnerabilities, while 6 projects have upgraded to the latest version. V1.2.3 seems to be a stable version, and it is the most reused one.
- For libcurl, 28 projects in this study reuse 20 different versions of libcurl code. 24 projects (85.7%) are reusing out-dated libcurl code. Only 4 projects are using the newer versions of code without vulnerabilities and only one project is using the latest version.
- For libpng, 50 projects in this study reuse 37 different versions of libpng code. 46 projects (92%) are reusing out-dated libpng code. Only 4 projects are using newer versions of code without vulnerabilities and 2 projects are using the latest version.

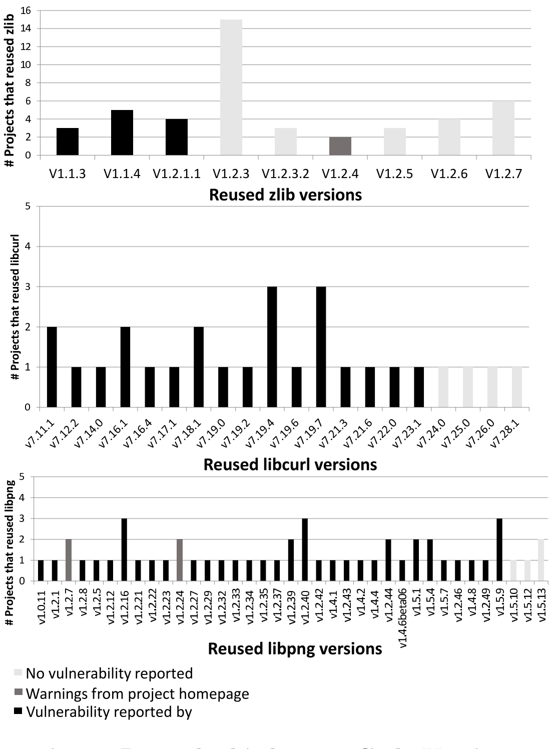

Fig. 3 Reused Third-party Code Versions

†3 All these case studies have been performed between Nov. 2012 and Jan. 2013.

In all the 123 projects, 84 projects (68.3 %) are reusing the out-dated third-party code. This result indicates that a large number of open source software are containing code with vulnerabilities.

3. 2. 2 Third-party Code Management Information

For the third-party code management information:

- In all the 123 projects we studied, 27 projects (22.0%) have modified the third-party code. The left 96 projects (78.0%) have reused the third-party code without any modification.
- In all 123 projects, only 18 projects have managed the third-party code well and update them frequently; while 83 projects did not update the third-party code at all, and in these 83 projects, 23 projects (18.7%) have no version information of the third-party code. Those projects have imported only the source code of the third-party libraries but they have not included the readme or changelog files. The users of these user projects cannot know which version of the third-party code is reused.
- In all 123 projects, 6 projects have changed the directory names or mixed the third-party code with other code, which could lead to difficulty in third-party code management. In addition, 2 projects have reverted the third-party code from the new versions to older versions.

3. 3 Case Study Results

Based on the case study statistics, we answer the raised questions as follows:

RQ1 What is the proportion of out-dated third-party code reused in the open source projects?
In this study, 68.3% open source projects are reusing out-dated third-party code.

RQ2 What are the potential vulnerabilities caused by such reuse?
Software vulnerabilities of the third-party code could cause potential vulnerabilities in open source software. The detailed information could be checked in *NVD*. In the case of reusing zlib, libcurl and libpng, according to the vulnerabilities descriptions, denial of service and execute arbitrator code are some examples of the potential vulnerabilities.

RQ3 How do developers manage those out-dated third-party code?
Broadly speaking, more than a half of the open source projects did not manage the third-party very well. Many of them just "copied&pasted" third-party code to their project without any modification. A large number of developers only imported third-party code into their projects. After those code started working properly, they left those code alone and did not tough them anymore. Some projects lost the version information of the third-party code and were not able to manage them anymore. Also, a few projects changed directory names or mixed the third-party code with their own code.

4 Discussion

4. 1 Reason for No Updating

We randomly selected 5 user projects that have reused out-dated third-party code, and sent emails to the project owners to ask for reasons that out-dated third-arty code is not updated. 4 of them have replied to us.

Owner of "openjpeg" project[†4] reusing libpng said, they have notified users that these code are from third-party and they do not recommend user to use these code.

Owner of "mtasa-blue" project[†5] reusing libpng said, they thanked our notification and agreed that there might be potential vulnerabilities. But they only update when any problem happens to avoid introducing new bugs.

Owner of "Angel-engine" project[†6] reusing libpng said, they cannot update because of multiple dependencies and compatibility issues.

Owner of "PCSX2" project[†7] reusing zlib said, they did not have time to go over everything. They updated the code immediately after reading our email and thanked to our contribution.

In our view, what the developers firstly care about is the functional implements but not the security vulnerabilities. Thus, the priority of updat-

†4 openjpeg, http://www.openjpeg.org/

†5 mtasa-blue, https://code.google.com/p/mtasa-blue/

†6 Angel-engine, https://code.google.com/p/angel-engine/

†7 PCSX2, http://pcsx2.net/

ing their code would not be high, unless some security problems really happen. Moreover, there are risks for the developers to update the code currently working, just as mentioned in the case of "mtasa-blue" project.

4. 2 Threat to Validity

Although many vulnerabilities of third-party libraries are reported, the reproducibility depends on how people reuse code. Taking libpng for example, many vulnerabilities are reproducible in a condition of reading a crafted picture. If a user project only uses this library to read their own pictures, these vulnerabilities would not be problems. However, if the software, such as a web browser, using libpng library to read pictures from external users, they have to take these vulnerabilities seriously. Anyway, to use the newer version of the third-party library would be a safer choice.

Totally 123 projects are studied in this study. To a certain extent the results from these projects can reflect how open source projects reuse and manage third-party code. However, as we know, there are millions of open source projects in the world. The candidate projects returned by *OpenCCFinder* are only in a very small subset of them. These projects are from Google Code, GitHub, and search[code]. Since we do not know the detailed searching and ranking algorithms of those external search engines, there might be selection bias in this study.

Moreover, currently only the third-party libraries written in C were studied. It is possible that we will get different results if we study the libraries in other language or in the form of binary code.

5 Related Work

There are existing researches on third-party evolution impact analysis. Kotonya et al. [3] proposed approach of assuming a black box view on integrated components. They also use an architecture description language and process-based approach to manage evolving third-party components. Klatt et al. [1] coped with the trend of integrating open source components that provide access to source code and software management information with further possibilities for the impact and development reliability analysis. Clarksen [8] et al. and Bohner [9] used dependency analysis in source code based impact analysis. However, they mainly focus on the third-party components themselves. None of them had done study on how out-dated third-party code are reused by the user open source projects.

6 Conclusion

In this paper, we have proposed an approach for studying the reuse of out-dated third-party source code in open source projects. We have conducted case studies on zlib, libcurl and libpng libraries and found that a large proportion of open source projects are reusing out-dated third-party code, and many of the reused third-party code are not well managed.

In the future, we would like to develop a support tool for third-party code version detection, vulnerability prediction and update support.

Acknowledgments This work was supported by KAKENHI (No.21240002, No.23650015).

References

[1] Klatt, B., Durdik, Z., Koziolek, H., Krogmann, K., Stammel, J. and Weiss, R.: Identify Impacts of Evolving Third Party Components on Long-living Software Systems, in *16th European Conference on Software Maintenance and Reengineering* (*CSMR*), 2012, pp. 461–464.

[2] Ebart, C.: Open Source Software in Industry, *IEEE Software*, Vol. 25, No. 3(2008), pp. 52–53.

[3] Kotonya, G. and Hutchinson, J.: Analysing the Impact of Change in Cots-based Systems, *COTS-Based Software Systems*, 2005, pp. 212–222.

[4] Inoue, K., Sasaki, Y., Xia, P. and Manabe, Y.: Where Does This Code Come from and Where Does It Go?—Integrated Code History Tracker for Open Source Systems, in *Proc. 34th ICSE*, 2012, pp. 331–341.

[5] National Vulnerability Database. http://nvd.nist.gov/.

[6] Xia, P.: An Empirical Study of Out-dated Third-party Code in Open Source Software, *Master's thesis*, Graduate School of Information Science and Technology of Osaka University, 2013.

[7] Xia, P., Manabe, Y., Yoshida, N. and Inoue, K.: Development of a Code Clone Search Tool for Open Source Repositories, コンピュータソフトウェア, Vol. 29, No. 3(2012), pp. 181–187.

[8] Clarkson, J., Simons, C. and Eckert, C.: Predicting Change Propagation in Complex Design, *Journal of Mechanical Design*(*Transactions of the ASME*), Vol. 126, No. 5(2004), pp. 788–797.

[9] Bohner, S.: Extending Software Change Impact Analysis into Cots Components, in *Proceedings of the 27th Annual NASA Goddard Software Engi-*

neering Workshop, 2002, pp. 175–182.
[10] Haefliger, S., Krogh, G. and Spaeth, S.: Code Reuse in Open Source Software, *Management Science*, Vol. 54, No. 1(2008), pp. 180–193.

夏　　沛

2010 年年上海交通大学ソフトウェア工学部卒業．2013 年大阪大学大学院情報科学研究科博士前期課程修了．在学中、コードクローン分析やコード再利用に関する研究に従事．現在 GREE に勤務.

松下　誠

1993 年大阪大学基礎工学部情報工学科卒業．1998 年同大学大学院博士後期課程修了．同年同大学基礎工学部情報工学科助手．2002 年大阪大学大学院情報科学研究科コンピュータサイエンス専攻助手．2005 年同専攻助教授．2007 年同専攻准教授．博士（工学）．ソフトウェア開発環境，リポジトリマイニングの研究に従事．日本ソフトウェア科学会，ACM 各会員.

吉田則裕

2004 年九州工業大学情報工学部知能情報工学科卒業．2009 年大阪大学大学院情報科学研究科博士後期課程修了．2010 年奈良先端科学技術大学院大学情報科学研究科助教．博士（情報科学）．コードクローン分析手法やリファクタリング支援手法に関する研究に従事．ソフトウェア科学会，情報処理学会，電子情報通信学会，人工知能学会，IEEE，ACM 各会員.

井上克郎

1984 年大阪大学大学院基礎工学研究科博士後期課程修了（工学博士）．同年、大阪大学基礎工学部情報工学科助手．1984〜1986 年、ハワイ大学マノア校コンピュータサイエンス学科助教授．1991 年大阪大学基礎工学部助教授．1995 年同学部教授．2002 年大阪大学大学院情報科学研究科教授．2011 年 8 月より大阪大学大学院情報科学研究科研究科長．ソフトウェア工学、特にコードクローンやコード検索などのプログラム分析や再利用技術の研究に従事.