On Mining Data across Software Repositories*

Prasanth Anbalagan[1] Mladen Vouk[2]
Department of Computer Science , North Carolina State University, Raleigh, NC 27695, USA
[1]panbala@ncsu.edu [2]vouk@ncsu.edu

Abstract

Software repositories provide abundance of valuable information about open source projects. With the increase in the size of the data maintained by the repositories, automated extraction of such data from individual repositories, as well as of linked information across repositories, has become a necessity. In this paper we describe a framework that uses web scraping to automatically mine repositories and link information across repositories. We discuss two implementations of the framework. In the first implementation, we automatically identify and collect security problem reports from project repositories that deploy the Bugzilla bug tracker using related vulnerability information from the National Vulnerability Database. In the second, we collect security problem reports for projects that deploy the Launchpad bug tracker along with related vulnerability information from the National Vulnerability Database. We have evaluated our tool on various releases of Fedora, Ubuntu, Suse, RedHat, and Firefox projects. The percentage of security bugs identified using our tool is consistent with that reported by other researchers.

1 Introduction

Mining software repositories is an important activity when analysing projects. Mining information across multiple data sources is one of the challenges [8, 5]. We observed that relevant information from one repository can complement the mining activity on another repository. For example, the Bugzilla bug tracker for *Mozilla*[1] and *Red Hat* projects[2] contain custom "keywords", textual tags, that help identify specific categories of bugs in the database. The keyword *security* relates to a security bug[3]. This could have been used to identify the security bugs in projects like *Fedora*[4], *Firefox*[5] etc. But the usage of the keyword was not consistent across bug reports. The Common Vulnerability Exposure[6](CVE) site maintains information about publicly known vulnerabilities. The vulnerabilities tagged by CVE are contained in the *National Vulnerability Database*[7](NVD), a U.S. government repository devised to manage vulnerability data. The NVD database lists vulnerabilities specific to different types of products including *Fedora (Red Hat)*, *Firefox (Mozilla)* etc. The *external resource* section of each vulnerability listed in the NVD has a mapping or link to the bugs in their respective bug-tracking system. This implies that information from the NVD can be utilised in mining security bugs in projects like *Firefox*, *Fedora* etc.

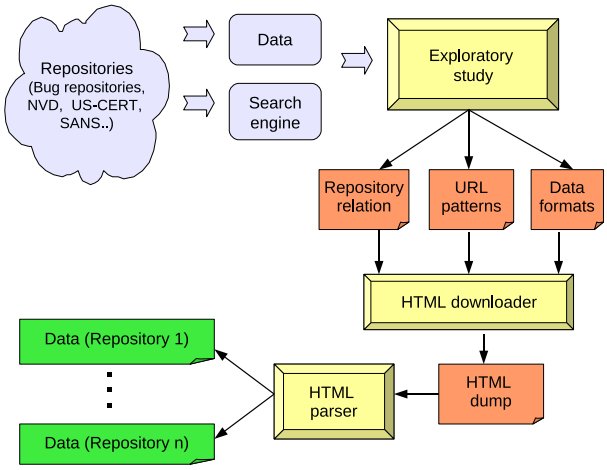

Figure 1. Framework

For projects, like Ubuntu, that deploy the Launchpad bug tracker, the search engine allows to search for bugs with CVE tags. These CVE tags in turn can be used to collect linked vulnerability characteristics in terms of the nature of exploits, impacts, and their metric values present in the the NVD. Extracting such information would give additional context to information available through individual repositories. Manually extracting such information is tedious. In this paper we

*This work is supported by the U.S. Army Research Office (ARO) under grant W911NF-08-1-0105 managed by NCSU Secure Open Systems Initiative (SOSI).

[1]https://bugzilla.mozilla.org/
[2]https://bugzilla.redhat.com/
[3]https://bugzilla.redhat.com/describekeywords.cgi
[4]http://fedoraproject.org/
[5]http://www.mozilla.com/en-US/firefox/
[6]http://cve.mitre.org/
[7]http://nvd.nist.gov/

- Discuss a framework that helps in automatically mining linked information across repositories.
- Discuss its implementation in the context of identifying security bugs in projects that use the Bugzilla bug tracker by mining linked information from the NVD.
- Discuss its implementation in the context of mining security bugs from repositories that deploy the Launchpad bug tracker along with related vulnerability information from the NVD.
- Discuss the use of the tool on the *Fedora*, *Ubuntu*, *Suse*, *RedHat*, and *Firefox* projects.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 discusses the framework. Section 4 discusses implementations of the framework. Section 5 discusses results from application of our tool. Section 6 concludes the paper.

2 Related Work

Robles et al. [7] propose an architecture for automated retrieval and analysis of data from open source repositories like CVS and archives of source packages. The authors discuss on including support for bug tracking systems and mailing lists as future work. Similarly, Neuhaus et al. [6] retrieve vulnerability information from bug databases and CVS repositories. Hassan [5] discusses the challenges in mining software respositories. The author identifies linking data across repositories as one of the open challenges in this area. In this paper, we address this limitation in mining data from individual repositories as well as linked information across repositories.

3 Framework

Figure 1 shows an overview of our framework. It consists of: the exploratory study, the HTML downloader, and the HTML parser. we discuss the components in detail in the following sections.

3.1 Exploratory study

Exploratory study refers to the preliminary work done to identify the relation between the repositories under consideration and collect details required to extract data from the repositories. The output of this study would be the *repository relation identifier*, *URL patterns*, and the *data formats*.

We define a repository relation as an identifier obtained from one repository that would help in search of information in another repository. For example, the vulnerability summary provided by the NVD contains a mapping or link to individual bugs in the project's bug repositories. Consider the vulnerability summary for the *Firefox* vulnerability CVE-2007-3656[8] in the NVD.

[8]http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2007-3656

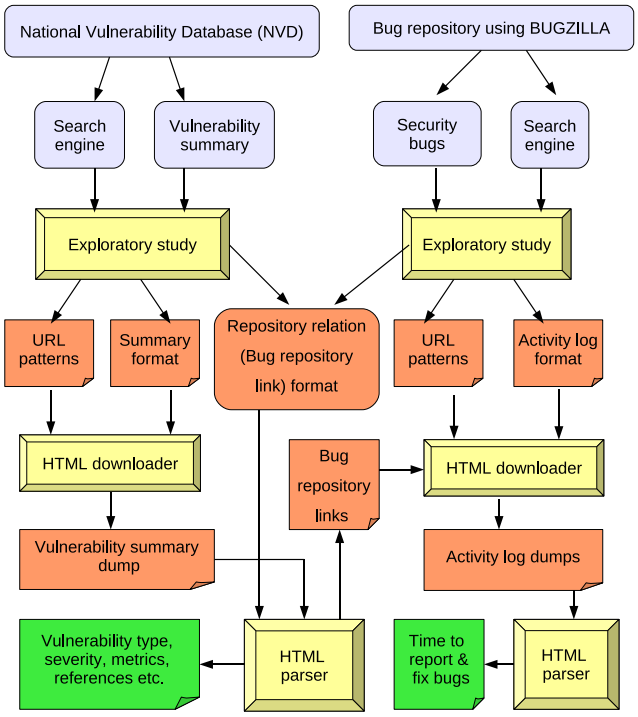

Figure 2. NVD and Bug repositories using Bugzilla

The external source section of this summary has the link *"https://bugzilla.mozilla.org/show_bug.cgi?id=387333"* which maps to the *Firefox* bug repository. We call this mapping or link a repository relation. Also, the vulnerabilities listed in NVD have a unique CVE tag (eg.CVE-2007-3656) associated with each of them. We observed that bug repositories use this CVE tag either in the bug summary or the bug description section. This unique CVE id could also be used as a repository relation. Projects like *Ubuntu* quote the CVE tag in their bug reports. This may not be true for all projects. It is necessary to carefully examine the repositories, understand their usage of keywords or tags, and make sure the usage is consistent across bug reports.

URL patterns refer to the patterns observed in the HTTP or HTTPS requests used by the search engines of the respective project's bug tracker. In our implementation, the patterns are simple URL links observed in the address bar of the browser when one uses the search engines but without the parameter values. For example, consider the URL used by the search engine in the Redhat's Bugzilla "$https://bugzilla.redhat.com/buglist.cgi?query\_format = advanced\&classification = Fedora\&product = Fedora\&bug_status = NEW$". This URL is used to list all *NEW* bugs for the *Fedora* product. Here the *URL pattern* would include the url but without the parameters *Fe-*

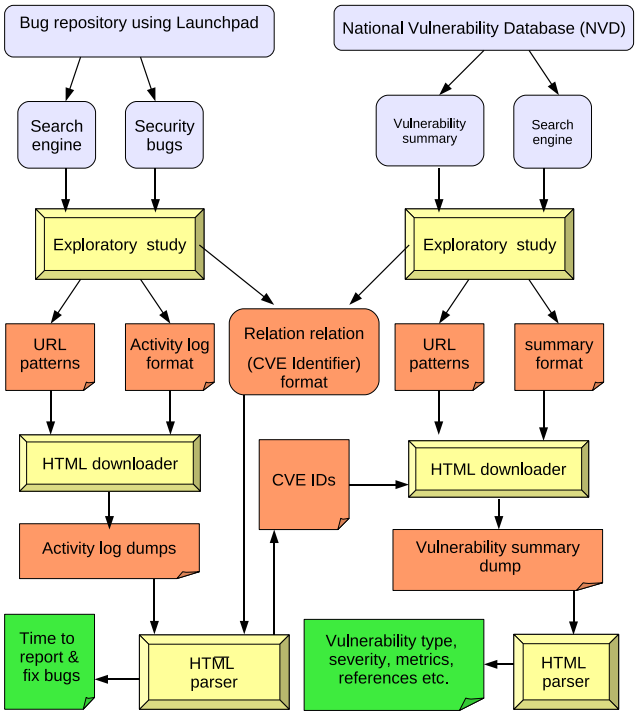

Figure 3. NVD and Bug repositories using Launchpad

dora and *New*. Parameters will be passed as arguments to the *HTML downloader* based on the URL pattern and type of data or product required. We need to identify the *URL pattern* for generic search engine usage as well as for viewing individual sections in a bug report. The latter is required to extract information for each bug while the former is required to collect the list of all the bugs in the bug repository using generic search requests. For example, "$https://bugzilla.redhat.com/show\_activity.cgi?id=$" refers to the *URL pattern* to view the activity log of each bug report. As the bug ids are collected, this *URL pattern* will be appended with the individual bug ids and executed to view their activity log.

Data formats refer to the formats in which the required data is stored in the repositories. In our implementation, we collected information in terms of calendar time when the bug was reported and fixed. For example, Redhat's Bugzilla maintains the timing using *Year-month-day hour-minute-seconds* format (Example: 2007-01-11 08:20:30). Vulnerability characteristics in NVD are storied in the format (AV: x /AC: x /Au: x /C: x /I: x /A: x), where the attribute AV is the access complexity, Au is the authentication complexity, C is the confidentiality impact, I is the integrity impact, A is the availability impact, and x is the respective attribute's value.

3.2 HTML downloader and HTML parser

The *HTMl downloader* uses the *URL patterns* to automatically collect information from the repositories based on the parameter values passed during execution. The output of the *HTML downloader* is the HTML dump of the search result returned for each of the HTML requests automatically invoked. In other words, the output is the HTML version of the webpage that would be displayed when a user searches for information using the search engine. The *HTML parser* uses the *data formats* obtained from the *exploratory study* component to parse this HTML output and extract the required information. We used libraries from HTMLScraper[9], an open source project, to implement the *HTML downloader* and *HTML parser* components of the tool.

4 Implementation

Figures 2 and 3 show the implementation of our framework between the NVD and projects that deploy BUGZILLA and Launchpad bug trackers respectively. Figure 2 shows the implementation where information from the NVD is used to identify security bugs in the respective project's bug repositories. Figure 3 shows the implementation where security bugs are directly mined from project bug repositories that deploy Launchpad bug tracker and then collect related vulnerability characteristics from the NVD.

In our first implementaion, a list of CVE ids for vulnerabilities specific to a project is given as input to the *HTML downloader*. The *HTML downloader* retrieves the HTML data of the vulnerability summary for each vulnerability in the list using their CVE ids. In this case, each CVE id is appended to the URL pattern "$http://web.nvd.nist.gov/view/vuln/detail?vulnId=$" to retrieve the vulnerability summary. Then the *HTML parser* identifies the mapping or link to the bug repositories from the HTML output. Next, the *HTML downloader* uses the identified links and retrieves the HTML data of the activity log section of each bug report in the respective bug respoitory. The activity log contains the timestamp when the bug was reported and fixed. Based on the data format in which the timestamps were stored, the *HTML parser* identifies the date and time when each bug was reported and fixed. This implementation was evaluated on projects like *Firefox* and *Fedora*. We collected the timestamps for use in our published works [4, 3].

In our second implementation, the *HTML downloader* automatically retrieves the list of bugs from the bug repository using the URL pattern similar to in Section 3.1 for projects that use Launchpad bug tracker. Launchpad bug tracker allows to search for bugs tagged with CVE id. The

[9] http://sourceforge.net/projects/HTMLscraper/

CVE ids are displayed along with the bug ids when one uses the search engine. The *HTML parser* parses this output and extracts the bug ids, and the CVE ids associated with them. Once the bug ids and CVE ids are extracted, the activity log data and the vulnerability characteristics were extracted by using the *HTML downloader* and the *HTML parser* in a similar fashion as in the first implementation. This implementation was evaluated on the *Ubuntu* project.

5 Discussion

We collected time specific information for security bugs from the individual bug repositories along with the data from the NVD. Table 1 shows the data collected in terms of the number of security and non-security bugs for *Fedora* (releases 1 to 8), SuseLinux (releases 10.1, 10.2) OpenSuse (releases 10.2, 10.3), Ubuntu (releases 4.10 to 8.10), and RedHat Enterprise Linux (releases 2.1 to 5). From the table, we find that security bugs account for roughly 0.5% to 5% of the total number of bugs. This is consistent with the rates reported by other researchers [2, 1]. The security bugs have been identified using the approaches explained in this paper. We have successfully utilised the tool in our published works on reliability analysis of open source software [4, 3]. We plan to make the tool open source software and release it on Sourceforge[10].

It is possible to find more than one repository relation and retrieve data across repositories. In our implementation, we identified the relation as the mapping or links present in the NVD vulnerability summary. Since these links could be used directly by the HTML downloader, we considered this repository relation. We could also search for CVE ids in the summary section of the bugs, and there by identify security bugs. In this case, we need to collect the CVE ids from each of the bugs and then retrieve the corresponding vulnerability information from the NVD. We have implemented this approach for Suse and RedHat projects.

6 Conclusion

We have studied data across the NVD and repositories of projects that deploy *Bugzilla* and *Launchpad* bug trackers, and identified that relevant information from one repository can help in mining activity in other repositories. We have proposed a general framework to help in mininig linked data across repositories. We implemented and evaluated our framework for the projects *Fedora*, *Ubuntu*, and *OpenSuse*. Our results show that the tool can help in simplifying the task of automatically mining data from individual repositories as well as linked data across repositories.

[10] http://sourceforge.net/

Table 1. Tool Statistics

Project	Non-security	Security	Total	Percentage
Fedora	48077	908	48985	1.85
Ubuntu	71408	1086	72494	1.49
OpenSuse	8747	66	8813	0.75
SuseLinux	6975	56	7031	0.80
Firefox	8506	367	8139	4.30
RedHat (RHEL)	22496	822	23318	3.52

References

[1] O. H. Alhazmi and Y. K. Malaiya. Modeling the vulnerability discovery process. In *ISSRE '05: Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering*, pages 129–138, Washington, DC, USA, 2005. IEEE Computer Society.

[2] O. H. Alhazmi and Y. K. Malaiya. Application of vulnerability discovery models to major operating systems. *Reliability, IEEE Transactions on*, 57(1):14–22, March 2008.

[3] P. Anbalagan and M. Vouk. Security failure estimation for open source software: An empirical approach. In *WDSE '08: Proceedings of the 1st Workshop on Dependable Software Engineering*, Seattle, WA, USA, 2008. IEEE Computer Society.

[4] P. Anbalagan and M. Vouk. Student paper:on reliability analysis of open source software-fedora. In *ISSRE '08: Proceedings of the 19th IEEE International Symposium on Software Reliability Engineering*, Seattle, WA, USA, 2008. IEEE Computer Society.

[5] A. Hassan. The road ahead for mining software repositories. In *In Proceedings of Frontiers of Software Maintenance, 2008. FoSM 2008*, pages 48–47, 2008.

[6] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller. Predicting vulnerable software components. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 529–540, October 2007.

[7] G. Robles and J. Carlos. Gluetheos: Automating the retrieval and analysis of data from publicly available software repositories. In *In Proceedings of the International Workshop on Mining Software Repositories*, pages 28–31, 2004.

[8] T. Zimmermann. Knowledge collaboration by mining software repositories. In *Proceedings of the 2nd International Workshop on Supporting Knowledge Collaboration in Software Development*, pages 64–65, September 2006.