LDA Categorization of Security Bug Reports in Chromium Projects

Wajdi Aljedaani
Dept. of Computer Technology
Al-Kharj College of Technology
Al-Kharj, Saudi Arabia
waljedaani@tvtc.gov.sa

Yasir Javed
Dept. of Computer Science
Prince Sultan University
Riyadh, Saudi Arabia
yjaved@psu.edu.sa

Mamdouh Alenezi
Dept. of Computer Science
Prince Sultan University
Riyadh, Saudi Arabia
malenezi@psu.edu.sa

ABSTRACT
Security bug reports (SBR) depict potential security vulnerabilities in software systems. Bug tracking systems (BTS) usually contain huge numbers of bug reports including security-related ones. Malicious attackers could exploit these SBRs. Henceforth, it is very critical to pinpoint SBRs swiftly and correctly. In this work, we studied the security bug reports of the Chromium project. We looked into three main aspects of these bug reports, namely: frequencies of reporting them, how quickly they get fixed and is LDA effective in grouping these reports to known vulnerabilities types. We report our findings in these aspects.

CCS Concepts
• Security and privacy → Software and application security → Software security engineering.

Keywords
Bug repository, bug reports; security bug reports; bug fixing; Google Chromium; empirical studies; topic model.

1. INTRODUCTION
Bug tracking systems (i.e., Jira and Bugzilla) are used by development teams to maintain bug reports [1]. Different types of users such as end-users, developers, test engineers can submit these bug reports. The submitted bug reports can be of different types and aspects. They can relate to software functionality and quality, such as change requests, adding features, compatibility, performance, and security. Specifically, malicious people can exploit security bugs to attack these software system [2]. Security bugs differ from non-security bugs since they represent abusive functionality not insufficient or wrong functionality [1]. Current bug tracking systems do not have the mechanism to automatically identify whether a bug report is a security bug or not.

Developers use bug reports to document issues and bugs to be submitted to BTSs. For instance, the Mozilla bug tracking system has more than 670,000 bug reports with 135 new bug reports added daily [3]. Peters et al. [4] discussed that it is important to correctly identify security bug reports and distinguish them from other nonsecurity bugs reports. Reporting and fixing security bug reports via public bug tracking systems maximizes the probability that a patch is widely available before hackers exploit a vulnerability. Oftentimes, developers or users who report security bug reports lack the security knowledge [3] to know when their bug is normal or when that bug is a security matter. Hence, unfortunately, security bugs are often publicly disclosed before they can be patched [3].

A usual way to identify security bug reports is by reviewing bug reports manually. For example, the Mozilla team has dedicated a special security bug group to handle security bug reports [2]. Still, manually reviewing bug reports needs time and professional knowledge. Consequently, a better solution is to automatically identify security bug reports. In this work, we study the security bug reports of the Chromium project. Specifically, we examined three research questions in our study:

RQ1: What is the frequency of security bugs compared to other projects?

RQ2: How quickly security bug reports get solved across projects?

RQ3: Is LDA effective in clustering bug reports to known vulnerabilities types?

The paper is organized as follows: Section 2 discusses some background material about the discusses topics. Section 3 discusses the study design adopted in this work. Section 4 depicts the results of our study. Section 5 discusses the related work. Section 6 shows the threat to the validity of our experiments. The conclusion is given in Section 7.

2. BACKGROUND
Information retrieval (IR) techniques have been used by software engineering researchers to solve their problems. Latent Dirichlet Allocation (LDA) [5] is one of the famous IR techniques that have been adopted by the software engineering research community [6]. LDA models textual contents into topics based on their distributions. It assumes that documents have been generated using the probability distribution of topics. LDA can group terms into specific clusters or topics. Chen et al. [7] surveyed applying topic modeling when mining software repositories. Some examples of applying LDA to software engineering tasks are feature location [8], software evolution [9] and traceability link recovery [10].

LDA is a generative probabilistic model to find out the topics in the collection of textual documents. Documents are represented as a finite mixture on the topics [11]. Each topic or cluster is a probability distribution over the set of terms of the document collection. In LDA, each document can have several topics, and each topic can exist in more than one document. Topic modeling maps the terms in a bug report to their corresponding topics. Topic names can be manually inferred by looking at the words

ESSE 2020, November 6–8, 2020, Rome, Italy

ACM ISBN 978-1-4503-7762-1/20/11…$15.00

https://doi.org/10.1145/ 3393822.3432335

that are part of the topic which are usually meaningful to humans [12].

Table 1. Statistics of Datasets Used in Our Study

Project	#Bug Reports	Start Date	End Date
Chrome	2,722	27-10-2009	19-10-2019
Windows	2,960	13-11-2008	19-10-2019
Mac	2,100	08-12-2008	19-10-2019
Linux	4,807	12-05-2009	19-10-2019
Total	12,589		

3. STUDY DESIGN

This section outlines our methodology and how we collected and processed the data to answer our three research questions.

3.1 Study Approach

In this section, we present our study approach to security bug reports on Chromium projects. Figure 1 provides an overview approach to our study. In the Google chromium bug repository, we first extract all bug reports from the entire bug repository. Then, we filter the extracted data based on the security-related bug reports and calculate relevant metrics for those bug reports. Next, we analyzed the security bug report metrics to identify the possible security differences between all chromium projects. The selected metrics studied in this paper are explained in detail in the approach subsections of each research question. Lastly, an analysis of the LDA topics was performed on security bug reports.

3.1.1 Data collection

We collected the data of all bug reports of Chromium projects that submitted to its Monorail bug tracker1. In this study, we only consider bug reports that contain security-bug in the bug report type. We have discarded all the bug reports that type as a feature, task, or enhancement. Our research focused on four platforms of Chromium project, Windows, Mac, Linux, and Android. We end up with a total of 12,589 security bug reports from the period of November 13, 2008, to October 19, 2019. We gathered the data on June 3, 2020; however, the monorail bug repository did not display any security bug reports beyond October 19, 20119. Our datasets are presented in detail in Table 1.

3.1.2 Data preprocessing

Unwanted Terms Removal. This step removes unwanted terms in the corpus. This includes repetitive words and words that do not add any value to the meaning.

Tokenization. This approach is widely used in text mining, which used to separate the text into phrases and words [13]. In addition, token elements, usually categorical functions, including punctuation, capitalization, and special characters and symbols, are generally removed [13]. In this paper, we make all words in lower cases as part of the tokenizing process.

Stop-Word Removal. Stop words are common frequent words that are not useful and important to the purpose of classification [13]. Therefore, we discarded these types of words in this phase. An example for stop words: the, an, a, you, he, to, be. In this paper, we used the list of English stop words included with Natural Language Toolkit [14].

Lemmatization. Lemmatization is another method of normalization. It performs the same purpose as stemming [15]. It used to identify the basis of each word. Words can be written in different grammar styles, but knowledge is always the same. During this process, each token shall be removed from appendices, and only the stem will remain.

4. STUDY RESULTS

RQ1: *What is the frequency of security bugs compared to other projects?*

Motivation. This research question focuses on finding security bugs that are reported for each platform within different years. By knowing this we can understand how the existence of bugs has evolved over the years and whether the security awareness has made an effect on security bugs fixing. It will also highlight which project has more security bugs.

Approach. Initially, this research collected all bug reports from selected projects. A total number of 16,882 bug reports were recorded till June 3rd, 2019. Later we executed the cleansing process of the bug reports based on the following criteria: 1 We are considering the bug reports from when all of the projects started reporting bugs and omitted any previous reports for other projects to have a fair comparison. Windows and Chrome contain bug reports in 2008, while Mac and Android do not have any bug reports in the stated year. Thus, we discarded any bug reports that were reported before 2009 allowing us to have a reliable comparison. 2 We have duplicated the bugs reports belonging to the different OS if a similar bug is reported with selected OS. A total of 3,909 bugs were duplicated for all the considered projects. This allowed us to have the actual number of bugs per OS. This resulted in a total of 20,785 bugs. 3 We have removed all reports that do not belong to any OS or other than four selected OS. This resulted in the removal of 8,196 bugs reports. All these criteria resulted in 12,589 bugs as an experimental data set.

Results. It is seen from results that reported bugs have increased over the years at the same time the fixing of bugs has also increased over the years. Such as there were only 12 bugs reported in windows in 2009 compared to 645 in 2019 that is an increase of 53 percent each year. The maximum numbers of bugs that have been reported are of Linux since 2016 reaching a maximum of 1460 in 2018. In 2019, it is seen that all the selected systems have a reduction in bugs.

Discussion. The number of reported bugs doesn't necessarily mean that number of bugs has increased over years but rather it may be one of two cases a community has become strong in reporting bugs that were not the case before 2009 b numbers of attacks on all OS have increased. Our first claim is supported by the result of Linux and Chrome were the number of reported are highest, yet very limited security breaches are reported officially that may be the result of open-source community contribution. Mac is one of the commercial OS and has always the least number of reported bugs compared to other selected OS that.

[1] https://bugs.chromium.org/p/chromium/

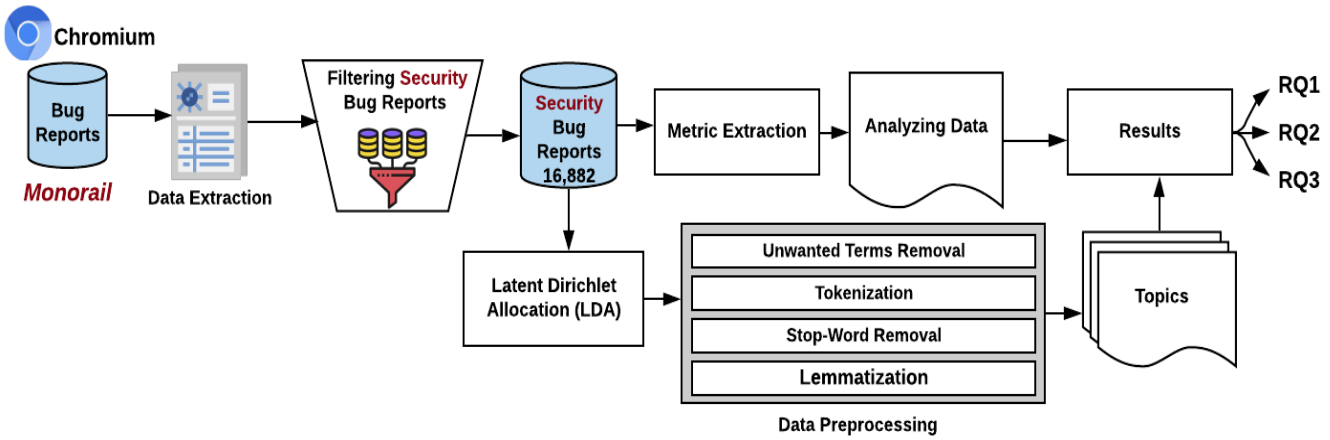

Figure 1. Overview of our approach to extract the metrics of issue reports and analyze the data.

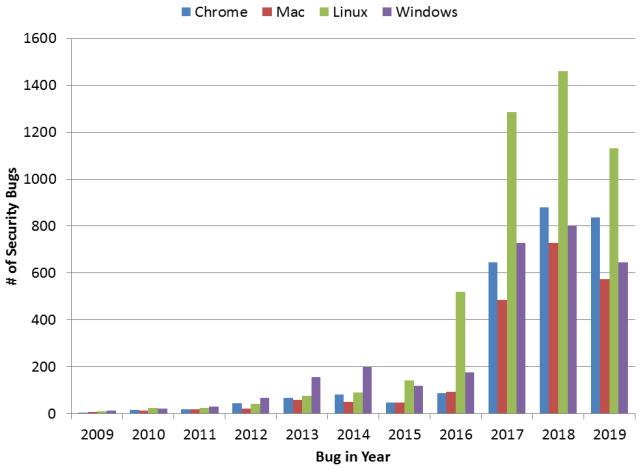

Figure 2. Distribution of the number of security bug reports.

RQ2: How quickly security bug reports get solved across projects?

Motivation. This research question focuses on finding, how long it takes to fix the bugs. This research question also tries to find out whether the open-source OS bugs are solved early or proprietary OS. We want to know how different platforms bugs take the amount of time to be fixed. We find the minimum and maximum fixing time to give users an idea about how long it may take to solve the bug at a particular OS.

Approach. We analyzed the cleaned data in order to find out whether the bug is fixed or not. For this, we checked whether the bug the marked as Fixed or open but only considered the fixed bugs, such as for Mac only 824 were selected out of 2100. For Chrome, we obtained 888 records, Windows 981, and Linux 1,275 fixed bugs. After that, we computed the fixing time of bugs. Wilcoxon test is a non-parametric statistical hypothesis test [16]. in two separate distributions. This research also does the Wilcoxon Rank test to find whether the bug fixing time is similar in different OS or its unrelated. We use the Wilcoxon rank-sum test [17] to evaluate the statistical significance of the difference between fixing time in bug reports in the project dataset (p-value<0.05).

Results. The results shown in Table 2 shows that Windows bug may take up to 2648 days to solve that is maximum, but the average time is equivalent to that of Chrome and Mac that is also evident from Figure 2. The minimum median fixing time is of Linux that is only 10 days. The average time for fixing a bug is maximum for Mac that is 75.70 days while Chrome being second highest with an average time of nearly 73 days. In terms of the Wilcoxon test, the minimum value is for Windows and Mac while Chrome and Mac are also near 0.5 which is 0.48 in actual. The value for Mac and Linux or Chrome and Linux is 1 while Windows and Linux are nearly one.

Table 2. Summary of fixing time of Chromium projects (in #days)

Platform	Min.	Q1	Median	Mean	Q3	Max.
Chrome	0.00	4.00	15.00	72.95	58.00	1805.00
Windows	0.00	4.00	14.00	68.07	51.00	2648.0
Mac	0.00	4.00	15.00	75.70	53.25	1672.0
Linux	0.00	3.00	10.00	52.99	38.00	1739.0

Table 3. P-values for Wilcoxon Rank-Sum tests

X vs. Y	Less	Greater	2-sided
Chrome vs. Windows	0.7792	0.2208	0.4416
Chrome vs. Mac	0.4852	0.5148	0.9704
Chrome vs. Linux	1	< 4.693e-06	< 9.386e-06
Windows vs. Linux	0.9999	< 6.759e-05	0.0001352
Windows vs. Mac	0.2152	0.7848	0.4304
Mac vs. Linux	1	< 5.862e-06	< 1.172e-05

Discussion. The results show that fixing time for Open-source OS is smaller than commercial OS such as Linux shows the minimum time for bug-fixing that can be because of the large community involved in fixing the bugs. If we related the bug fixing to number of years that bug has been reported, it can be seen that release date and short deadline pressure on propriety OS releases have resulted in a higher number of bugs and a similar approach of fixing bugs related to newer OS version thus the average fixing time is increased. The result shows that the bugs that exist in Linux are highly correlated to the bugs that appear in other OS, but if we see the Wilcoxon results for Mac and Windows the bugs are different from each other, while Chrome and Linux have similar kind of bugs that may be of the reason being based on the same kernel.

RQ3: Is LDA effective in clustering bug reports to known vulnerabilities types?

Motivation. This research question focuses on using LDA to cluster bug reports. We want to see if LDA is effective in grouping bug reports according to their reported vulnerabilities. It allows different topics to be clustered together by selecting any independent topics and finding the relation between them.

Approach. We take into account all the bug reports in every project. These bug reports have been analyzed to identify keywords related to security. We used the Latent Dirichlet Allocation (LDA) approach. The description of the bug reports was used as our corpus of documents.

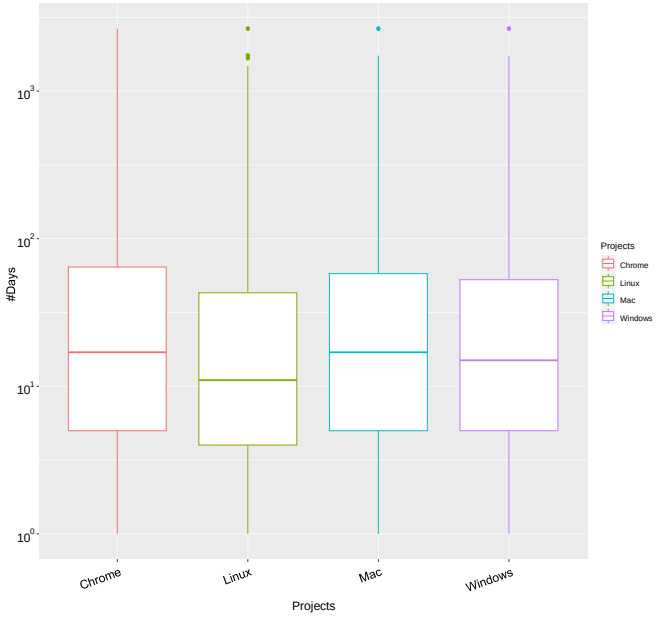

Figure 3. Box-plot of fixing-time (in #days) of security reports for Chromium projects.

We applied the well-known pre-processing steps to the textual description of bug reports as described in detail in Section 3. The LDA identifiers then cluster the words of the text in each bug report to generate results for topics. We use the following LDA parameters:

- Gibbs Sampling: default parameters
- Number of Topics: 10
- Burn-in-Period: 4,000 iterations
- Sampling: 5 times
- Thinning: 500

We applied the LDA analysis on three experiments in 10 topics, 15 topics, and 20 topics. Based on our thorough analysis, we decided to go with 10 topics since the topics are more coherent. 15 and 20 topics generated redundant topics with similar word clusters.

Results. Table 4 shows a list of 10 topic terms selected in Chrome Projects while Table 5 shows the same criteria result for Windows Projects. Table 6 and Table 7 presents 10 topic terms based on MAC and Linux Project respectively. The results of the LDA clustering showed its power in grouping similar security bug reports. In fact, we claim that the resulted topics are very close to well-known vulnerabilities types. This was our hypothesis from the beginning of this research question. All the results show a similarity in issues such as heap, buffer, overflow, redirect

Discussion. If we look at the results of Chrome projects, it can be seen that the first topic set is related to memory while second is related to parsing, the third topic is related to data leakage while fourth is related to secure execution. In terms of Windows first topic is related to content access, the second topic is related to external access. the third topic is related to access attacks, while the fourth topic is related to memory. In the case of Mac, the first topic is related to authentication, the second topic is related to memory, the third topic is related to data leakage and the fourth topic is related to Captcha. In terms of Linux, the first topic is related to memory, the second topic is related to access, the third topic is related to graphics and the fourth topic is related to Captcha. While looking at all results is seen that the topic may overlap in a different clustered topic such as kernel is appearing in parsing issue, driver issues for chrome project. Access and Captcha issues are common among all projects while data leakage or spoofing attack is also a common challenge among selected projects. Thus, a consideration in development must be given in different access types such as external access and internal access while applying various authentication techniques.

5. RELATED WORK

Software bug repositories provide a crucial source of information about development issues and are a significant source of long-term applications. The majority of papers concentrate on the degree of interaction between problem-proneness and committers [18, 19], bug repositories evaluation [20, 21], quality and content assessments of bugs [22] and functionality and developer-level priority [23, 24] and buggy source files [25]. In this section, we discuss related work focused on two aspects in the bug repository, specifically in security bug reports and topic modeling studies.

5.1 Security Bug Reports Studies

Recently, several researched bug reports have focused on recognizing and track security bug studies in the list of software bugs. Text mining bug reports are one of the popular approaches. To order to classify security and non-security bug reports, GosevaPopstojanova et al. [26] carried over an automated identification on software bug reports. They used the industrial dataset "NASA" to apply supervised and unsupervised approaches by using text mining to develop their prediction models. In another study, Gegick et al. [27] suggested a mathematical model method used by NLP to classify security and non-security bug reports.

Table 4. Summary of the terms in 10 topics from the LDA analysis of Chrome projects

S.no	Topic1	Topic2	Topic3	Topic4	Topic5	Topic6	Topic7	Topic8	Topic9	Topic10
1	heap	kernel	url	secur	valu	secur	linux	secur	crash	vulner
2	free	report	via	read	uniniti	file	cve	chrome	secur	report
3	overflow	cve	spoof	pdfium	failur	base	kernel	extens	blink	cros
4	buffer	linux	origin	oob	dcheck	memori	vulner	allow	bypass	lib
5	fuzzer	cros	cross	write	check	alloc	cros	access	user	net
6	intern	vulner	content	cxfa	object	line	report	uaf	bug	media
7	unsign	chromeo	data	confus	intern	uaf	driver	local	app	dev
8	stack	sys	leak	sandbox	type	pointer	qualcomm	code	cast	misc
9	char	creat	site	xfa	array	browser	allow	potenti	bad	tiff
10	skia	parser	request	load	fix	parti	sound	open	mode	libxml

Table 5. Summary of the terms in 10 topics from the LDA analysis of Windows projects

S.no	Topic1	Topic2	Topic3	Topic4	Topic5	Topic6	Topic7	Topic8	Topic9	Topic 10
1	secur	window	secur	free	crash	chrome	overflow	secur	failur	secur
2	origin	file	spoof	heap	intern	password	buffer	pdfium	check	read
3	cross	bypass	url	blink	base	googl	heap	uaf	dcheck	valu
4	content	via	issu	poison	report	browser	stack	content	object	memori
5	site	extens	process	bad	webrtc	vulner	int	possibl	cast	uniniti
6	data	page	request	cpdf	invalid	user	char	address	type	oob
7	allow	open	navig	layoutobject	render	chang	unsign	pdf	fail	write
8	code	download	idn	null	manag	websit	error	cxfa	size	corrupt
9	redirect	access	block	layoutblockflow	wasm	option	fuzzer	type	frame	bound
10	redirect	com	caus	version	invok	extens	integ	xfa	debug	xss

Table 6. Summary of the terms in 10 topics from the LDA analysis of Mac projects

S.no	Topic1	Topic2	Topic3	Topic4	Topic5	Topic6	Topic7	Topic8	Topic9	Topic10
1	secur	free	overflow	blink	failur	crash	secur	read	secur	origin
2	bypass	heap	buffer	poison	check	intern	chrome	valu	url	cross
3	type	pdfium	heap	cast	dcheck	base	uaf	uniniti	spoof	page
4	password	cxfa	stack	bad	object	content	xss	memori	idn	open
5	bound	code	cfx	invalid	debug	browser	attack	data	via	error
6	csp	xfa	size	free	map	worker	googl	oob	extens	request
7	report	test	chrome	possibl	array	leak	user	write	lead	download
8	via	webrtc	mac	integ	fail	manag	api	access	navig	site
9	vulner	expos	autofil	bug	assert	render	file	secur	address	extens
10	heap	domain	secur	skia	isol	invok	confus	char	context	allow

Table 7. Summary of the terms in 10 topics from the LDA analysis of Mac projects

S.no	Topic1	Topic2	Topic3	Topic4	Topic5	Topic6	Topic7	Topic8	Topic9	Topic 10
1	overflow	chrome	valu	blink	secur	intern	base	read	free	failur
2	buffer	cross	uniniti	cast	url	crash	content	type	heap	dcheck
3	heap	origin	gpu	bad	spoof	compil	std	poison	pdfium	check
4	stack	bypass	net	invalid	memori	unsign	char	function	cfx	object
5	webrtc	extens	gles	vptr	address	wasm	void	scope	cxfa	assert
6	global	via	view	sandbox	uaf	int	fuzzer	data	cpdf	map
7	integ	file	error	layoutobject	linux	oper	node	size	xfa	index
8	tabl	access	blit	anonym	password	invok	string	oob	unownedptr	fail
9	add	secur	sse	namespac	vulner	simul	media	code	updat	enabl
10	ccodec	site	quic	bpf	process	decod	test	isol	cach	length

Table 8. Summary of the terms in 10 topics from the LDA analysis of Mac projects

Study	Year	Purpose	Approach	Source of Info	Dataset	Techniques
Gegick et al. [17]	2010	Classifying bug reports	NLP statistical model	Bug reports	Cisco Dataset	Term matrix
Zimmermann [22]	2010	Analyzing trends for CVE data	LDA	CVE entry	CVE database	LDA topic
Behl et al. [9]	2014	Classifying bug reports	Statistical model, Vector space model	Bug reports	Eclipse, Firefox	TF -IDF
Zhang et al.	2016	Bug report assignment	LDA, Heterogeneous network	Bug reports	JDT	DREX, KNN
Peters et al. [3]	2017	Filtering and ranking bug reports	Text mining, Machine learning	Bug reports	Chromium, Wicket, Ambari, Camel, Derby	Tf-idf Terms (RF,KNN,M P LR, NB)
Popstojanova and Tyo [18]	2017	Creating vulnerability profiles	Software fault patterns (SFP)	Bug reports	NASA Dataset	Manual Classificatio n
Alqahtani [7]	2019	Classifying and tagging security concerns	LDA	CVE documents, Bug reports	NVD, Apache	Seeded-LDA
This work		Categorizing security bug reports	LDA	Bug reports	Chromium projects	LDA topics, Wilcoxon test

They used the industrial dataset Cisco and applied the study on the tool named SAS, which is an industrial text-mining tool. Some studies were focused on performing hyperparameter optimization to recognize and classifying security bug reports [16], [28], [20], and other used multi-type features. analysis [2]. A similar study to ours, Camilo et al. [29] assessed the correlation between the pre-release bug, reviewers' experience, and Chromium project post-release vulnerabilities. Their results showed a weak connection between bugs and vulnerabilities.

Our work is different from their work in both purpose and approach. Our purpose is to categorize security bug reports to identify which area is more insecure in Chromium projects across all four projects. Moreover, we also focused on the analysis of security bug reports fixing time to identify if software bug reports repositories can help Chromium to improve and quick the process of fixing process.

5.2 Topic Modeling Studies

Several studies have conducted topic modeling for source code [30], mobile platforms comparison [31], and issue localization [32]. Zaman et al. [33] performed a study to compare the two bug report types security and performance, and how they differ from other bug report types. The study analyzed topics modeling to distinguish these bug report types. They performed statistical metrics to find the developers fixing bugs and the characteristic of each type. In more recent work, Alqahtani [34] studied CVE documents and bug reports to classify and tag security concerns using LDA. The author used Seeded-LDA, Bag of Security Concepts, to be able to automatically tag bug reports. The result showed that 53% for access privilege, 40% SQL injection, and only 6.7% for authentication abuse.

Moreover, Zhang et al. [35] proposed a new approach named BAHA, which used for bug report assignment to automatically assign bug reports to developers. Their method used topics

modeling and heterogeneous network, and they performed their experiment on Eclipse JDT. Their results showed that the proposed approach could improve the recall by 19.19% comparing to previous methods. In another study, Neuhaus and Zimmermann [22] used topic modeling on the CVE database to identify prevalent topics. Their results showed that 19% of all CVEs examined are about CrossSite Scripting. These cutting-edge approaches are summarized in Table 8.

Our study is similar to theirs, but instead of performing the research on CVE dataset, we conducted the study on open-source Chromium projects, we also examine and contrast security bug reports on other Chromium projects. Moreover, we perform LDA topic modeling to identify the top security terms in security bug reports. Therefore, none of the previous security studies did a cauterization on the security dataset.

6. THREATS AND VALIDITY

We only selected Chromium open-source project to experiment on, thus we did not include closed-source project due to their data are not publicly available. We did not consider other projects such as Firefox, Eclipse, etc, due to these projects do not support the information of labeling the bug as security-bug. For future work, we will extend our experiments on other projects and compare their performance to Chromium projects.

The data for this study was gathered from the Monorail archive for all the Chromium projects. For the computation of fixing-time of the security bug reports, where the bug report marked as fixed at the time of gathering the data. In any case, this may incidentally be skewed later on if and when any of these issues might be changed the status to re-opened. In addition, there can be a number of falsepositive in our study as a number of bugs that were reported may be of cosmetic, or coding issue but was reported as a security bug. The analyses were done specifically on projects based on browsers platform, due to the ease of labeling the bug reports and public access for the large datasets.

7. CONCLUSION

We have studied the bug reports of the Chromium project. We looked at three aspects of such reports, their reporting frequency, their resolving speed, and their topics. The experimental study revealed that the number of reported security bugs is increasing over the years. The time to fix these bugs is also increasing over the years. Usually, it takes more than 10 days to fix any security bug. This is an indication that security bug reports have to be triaged and fixed very quickly. The results of the LDA clustering showed its power in grouping similar security bug reports in topics that are very close to well-known vulnerabilities types. Future work includes studying other open-source projects.

8. ACKNOWLEDGMENTS

Our thanks to PSU for supporting the research.

9. REFERENCES

[1] Deqing Zou, Zhijun Deng, Zhen Li, and Hai Jin. 2018. Automatically identifying security bug reports via multitype features analysis. In Australasian Conference on Information Security and Privacy. Springer, 619–633.

[2] Mamdouh Alenezi and Ibrahim Abunadi. 2015. Evaluating software metrics as predictors of software vulnerabilities. International Journal of Security and Its Applications 9, 10 (2015), 231–240.

[3] Rui Shu, Tianpei Xia, Jianfeng Chen, Laurie Williams, and Tim Menzies. 2019. Improved Recognition of Security Bugs via Dual Hyperparameter Optimization. arXiv preprint arXiv:1911.02476 (2019).

[4] Fayola Peters, Thein Tun, Yijun Yu, and Bashar Nuseibeh. 2017. Text filtering and ranking for security bug report prediction. IEEE Transactions on Software Engineering (2017).

[5] David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. Journal of machine Learning research 3, Jan (2003), 993–1022.

[6] Shadi Banitaan and Mamdouh Alenezi. 2015. Software evolution via topic modeling: an analytic study. International Journal of Software Engineering and Its Applications 9, 5 (2015), 43–52.

[7] Tse-Hsun Chen, Stephen W Thomas, and Ahmed E Hassan. 2016. A survey on the use of topic models when mining software repositories. Empirical Software Engineering 21, 5 (2016), 1843–1919.

[8] Lauren R Biggers, Cecylia Bocovich, Riley Capshaw, Brian P Eddy, Letha H Etzkorn, and Nicholas A Kraft. 2014. Configuring latent dirichlet allocation based feature location. Empirical Software Engineering 19, 3 (2014), 465–500.

[9] Stephen W Thomas, Bram Adams, Ahmed E Hassan, and Dorothea Blostein. 2014. Studying software evolution using topic models. Science of Computer Programming 80 (2014), 457–479.

[10] Rocco Oliveto, Malcom Gethers, Denys Poshyvanyk, and Andrea De Lucia. 2010. On the equivalence of information retrieval methods for automated traceability link recovery. In 2010 IEEE 18th International Conference on Program Comprehension. IEEE, 68–71.

[11] Wajdi Aljedaani, Meiyappan Nagappan, Bram Adams, and Michael Godfrey. 2019. A comparison of bugs across the iOS and Android platforms of two open source cross platform browser apps. In 2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft). IEEE, 76–86.

[12] Xin Xia, David Lo, Ying Ding, Jafar M Al-Kofahi, Tien N Nguyen, and Xinyu Wang. 2016. Improving automated bug triaging with specialized topic model. IEEE Transactions on Software Engineering 43, 3 (2016), 272–297.

[13] Ronen Feldman and James Sanger. 2007. Advanced Approaches in Analyzing Unstructured Data. The Text Mining Handbook, Cambrige University (2007).

[14] Steven Bird, Ewan Klein, and Edward Loper. 2009. Natural language processing with Python: analyzing text with the natural language toolkit. " O'Reilly Media, Inc.".

[15] Tuomo Korenius, Jorma Laurikkala, Kalervo Järvelin, and Martti Juhola. 2004. Stemming and lemmatization in the clustering of finnish text documents. In Proceedings of the thirteenth ACM international conference on Information and knowledge management. 625–633.

[16] Mamdouh Alenezi and Shadi Banitaan. 2013. Bug reports prioritization: Which features and classifier to use?. In 2013 12th International Conference on Machine Learning and Applications, Vol. 2. IEEE, 112–116.

[17] Patrick E McKnight and Julius Najab. 2010. Mann-Whitney U Test. The Corsini encyclopedia of psychology (2010), 1–1.

[18] Mario Luca Bernardi, Gerardo Canfora, Giuseppe A Di Lucca, Massimiliano Di Penta, and Damiano Distante. 2012. Do developers introduce bugs when they do not communicate? the case of eclipse and mozilla. In 2012 16th European Conference on Software Maintenance and Reengineering. IEEE, 139–148.

[19] Shadi Banitaan and Mamdouh Alenezi. 2015. Software evolution via topic modeling: an analytic study. International Journal of Software Engineering and Its Applications 9, 5 (2015), 43–52.

[20] Wajdi Aljedaani and Yasir Javed. 2018. Bug Reports Evolution in Open Source Systems. In 5th International Symposium on Data Mining Applications. Springer, 63–73.

[21] Wajdi Aljedaani, Yasir Javed, and Mamdouh Alenezi. 2020. Open Source Systems Bug Reports: Meta-Analysis. In Proceedings of the 2020 The 3rd International Conference on Big Data and Education. 43–49.

[22] Philipp Schugerl, Juergen Rilling, and Philippe Charland. 2008. Mining bug repositories–a quality assessment. In 2008 International Conference on Computational Intelligence for Modelling Control & Automation. IEEE, 1105–1110.

[23] Bader Alkhazi, Andrew DiStasi, Wajdi Aljedaani, Hussein Alrubaye, Xin Ye, and Mohamed Wiem Mkaouer. 2020. Learning to rank developers for bug report assignment. Applied Soft Computing (2020), 106667.

[24] Jifeng Xuan, He Jiang, Zhilei Ren, and Weiqin Zou. 2012. Developer prioritization in bug repositories. In 2012 34th International Conference on Software Engineering (ICSE). IEEE, 25–35.

[25] Nasir Safdari, Hussein Alrubaye, Wajdi Aljedaani, Bladimir Baez Baez, Andrew DiStasi, and Mohamed Wiem Mkaouer. 2019. Learning to rank faulty source files for dependent bug reports. In Big Data: Learning, Analytics, and Applications, Vol. 10989. International Society for Optics and Photonics, 109890B.

[26] Katerina Goseva-Popstojanova and Jacob Tyo. 2018. Identification of security related bug reports via text mining using supervised and unsupervised classification. In 2018 IEEE International Conference on Software Quality, Reliability and Security (QRS). IEEE, 344–355.

[27] Michael Gegick, Pete Rotella, and Tao Xie. 2010. Identifying security bug reports via text mining: An industrial case study. In 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010). IEEE, 11–20.

[28] Rui Shu, Tianpei Xia, Laurie Williams, and Tim Menzies. 2019. Better security bug report classification via hyperparameter optimization. arXiv preprint arXiv:1905.06872 (2019).

[29] Felivel Camilo, Andrew Meneely, and Meiyappan Nagappan. 2015. Do bugs foreshadow vulnerabilities? a study of the chromium project. In 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories. IEEE, 269–279.

[30] Anh Tuan Nguyen, Tung Thanh Nguyen, Jafar Al-Kofahi, Hung Viet Nguyen, and Tien N Nguyen. 2011. A topic-based approach for narrowing the search space of buggy files from a bug report. In 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011). IEEE, 263–272.

[31] Lauren R Biggers, Cecylia Bocovich, Riley Capshaw, Brian P Eddy, Letha H Etzkorn, and Nicholas A Kraft. 2014. Configuring latent dirichlet allocation based feature location. Empirical Software Engineering 19, 3 (2014), 465–500.

[32] Stephen W Thomas, Bram Adams, Ahmed E Hassan, and Dorothea Blostein. 2011. Modeling the evolution of topics in source code histories. In Proceedings of the 8th working conference on mining software repositories. 173–182.

[33] Shahed Zaman, Bram Adams, and Ahmed E Hassan. 2011. Security versus performance bugs: a case study on firefox. In Proceedings of the 8th working conference on mining software repositories. 93–102.

[34] Sultan S Alqahtani. 2019. Automated Extraction of Security Concerns from Bug Reports. In 2019 17th International Conference on Privacy, Security and Trust (PST). IEEE, 1–3.

[35] Wen Zhang, Song Wang, and Qing Wang. 2016. BAHA: A novel approach to automatic bug report assignment with topic modeling and heterogeneous network analysis. Chinese Journal of Electronics 25, 6 (2016), 1011–1018.