Pre-Patch: Find Hidden Threats in Open Software Based on Machine Learning Method

Mutian Yang[1], Jingzheng Wu[1,2](✉), Shouling Ji[3], Tianyue Luo[1], and Yanjun Wu[1,2]

[1] Intelligent Software Research Center, Institute of Software, Chinese Academy of Sciences, Beijing, China
{mutian,jingzheng08}@iscas.ac.cn

[2] State Key Laboratory of Computer Sciences, Beijing, China

[3] NESA Research Lab, Zhejiang University, Hangzhou, China

Abstract. The details of vulnerabilities are always kept confidential until fixed, which is an efficient way to avoid the exploitations and attacks. However, the Security Related Commits (SRCs), used to fix the vulnerabilities in open source software, usually lack proper protections. Most SRCs are released in code repositories such as Git, Github, Sourceforge, etc. earlier than the corresponding vulnerabilities published. These commits often previously disclose the vital information which can be used by the attackers to locate and exploit the vulnerable code. Therefore, we defined the pre-leaked SRC as the Pre-Patch problem and studied its hidden threats to the open source software. In this paper, we presented an Automatic Security Related Commits Detector (ASRCD) to rapidly identify the Pre-Patch problems from the numerous commits in code repositories by learning the features of SRCs. We implemented ASRCD and evaluated it with 78,218 real-world commits collected from Linux Kernel, OpenSSL, phpMyadmin and Mantisbt released between 2016 to 2017, which contain 227 confirmed SRCs. ASRCD successfully identified 206 SRCs from the 4 projects, including 140 known SRCs (recall rate: 61.7% on average) and 66 new high-suspicious. In addition, 5 of the SRCs have been published after our prediction. The results show that: (1) the Pre-Patch is really a hidden threat to open source software; and (2) the proposed ASRCD is effective in identifying such SRCs. Finally, we recommended the identified SRCs should be fixed as soon as possible.

Keywords: Pre-Patch · Open source software · Code repository Vulnerability

1 Introduction

Disclosing vulnerability information before fixing is dangerous since the information is the key factor for the attackers to locate and exploit the vulnerable code [11,13]. Therefore, the vulnerability disclosure policy requires the vulnerability information be remained inaccessible to public before the vulnerability is fixed [24,25]. In particular, CVE and CVE-compatible security vulnerability databases will merely release a reserved CVE number without publishing the details (description, affected products and versions, references, etc.) until it is repaired. Some vendors, like Mozilla, also make effort on keeping secret. For example, CVE-2014-1557 is a vulnerability of Firefox with high severity (CVSS v2 Base Score: 9.3) published on Jul 23, 2014, which may cause remote attack. The details of the vulnerability had not been published until the flaw was fixed. Additionally, the mail-list in bugzilla [1] remains access limited until to May 1, 2016.

However, the premature disclosure of vulnerabilities' information is still possible and even inevitable in real world. We found a flaw in the current vulnerability report and disclosure process: the code changes for repairing the vulnerability, called Security Related Commits (SRCs), are often committed to code repositories such as Github and Sourceforge, released to the public even when the details of the vulnerability are unpublished by the vulnerability databases or vendors. Unfortunately, the publication of SRCs may potentially expose some vital information to the attackers for finding the error cause, locating the vulnerable code and further exploiting. This makes the effort of vendors and security vulnerability databases for protecting the information of vulnerabilities useless. Again, taking CVE-2014-1557 as an example, its SRC 188980-0f2821706cdb was available in code repository [8] on Jun 17, 2014, which is 37 days before the CVE release. Furthermore, its SRC exposed the key information about the vulnerability including the precise location of the vulnerable code, the details of source code and a brief introduction of the error. This flaw introduces hidden threats to open source softwares and we define it as the Pre-Patch problem.

To study the hidden threat of the Pre-Patch problem, we analyze the commits of 4 popular OSS, which are Linux Kernel, OpenSSL, phpMyadmin and Mantisbt, and found that most of the SRCs are released prematurely in the code repositories. This may leak vital information of vulnerable codes before fixing. We further design and implement ASRCD based on a machine learning model to substantiate the possibility of rapidly identifying SRCs in large-scale commits. Specifically, because the SRCs have some specific description exposed in their comments, we use Text Mining method to analyze the existing commits of the 4 OSS, and extract 7,465 features from comment data. Then we trained a Back Propagation Artificial Neural Network (BP-ANN) classifier to identify the suspected SRCs in code repositories, and predict the unpublished vulnerabilities whose SRCs have been released in advance. We evaluated ASRCD with 78,218 real-world commits collected from the 4 OSS released between 2016 to 2017, containing 227 confirmed SRCs. ASRCD successfully identified 206 SRCs including 140 known SRCs and 66 new high-suspicious, where 5 of the new SRCs have been published after our prediction. The results show that: (1) the Pre-Patch is really a hidden threat to open source software; and (2) the proposed ASRCD is effective in identifying such SRCs.

2 Problem

2.1 Pre-Patch Problem

The processes of current vulnerability publishing and SRC releasing are shown in Fig. 1, and we analyze it in detail as follows.

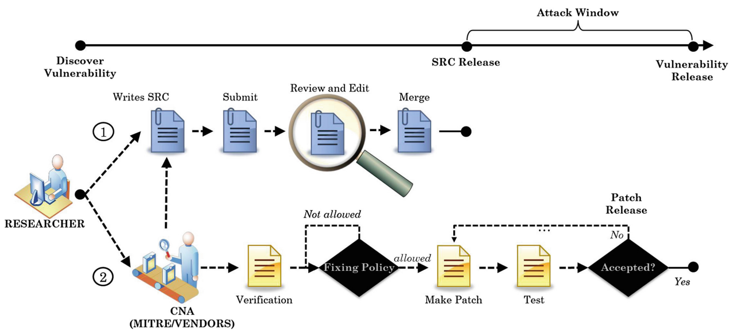


Fig. 1. The publishing process. ① is the SRC release process, ② is the vulnerability publishing process.

Vulnerability Publishing Process. As show in Fig. 1, once a developer/researcher finds a suspicious vulnerability, they notice the CVE Numbering Authorities (or CNA) including the MITRE corporation and some software vendors as soon as possible [2]. Then, the venders and researchers verify the vulnerability according to the description or Proof of Concept (PoC) provided by the notifier. The verified vulnerability will be fixed according to the fixing policy to provide the best security posture to customers. Usually the more severity the vulnerability is, the higher priority it has. Thirdly, the patch for repairing the vulnerability will be tested to ensure the reliability and released according to the patch releasing policy. Finally, the details of the vulnerability will be published to warn the public and offer the solutions. **SRC Release Process.** In the real world, the OSS are maintained in the code repositories. When a vulnerability is discovered and verified, the developer/researcher often submits a SRC to the code repositories to improve the software. According to the vulnerability disclosure policy [2], the disclosures are delayed to give the effected organizations enough time to test, develop and deploy the patch. However, the SRC release process is independent of the policy, leaving a time gap between the vulnerability publishing and SRC release. Unfortunately, the prematurely released SRCs might disclose information about vulnerabilities, such as the directory of the vulnerable file, affected versions, code change, etc., which can be exploited by attackers to locate flaws, analyze the vulnerable code and develop exploits.

2.2 Data Collection

We collected 720,783 commits from 4 popular OSS code repositories [5,7] including Linux Kernel, MantisBT, phpMyAdmin and OpenSSL and picked out 1,079 known SRCs. To build a map from SRCs to vulnerabilities, we search the vulnerabilities' information of the 4 OSS from in vulnerability databases [2,9], identify the urls linking to the pages of SRCs and map the vulnerabilities' information to the SRCs.

2.3 Pre-Patch Analysis

The Pre-Patch problem violates the vulnerability disclosure policy. Therefore, we examine the attack windows and the safe disclosure date (the date when the vulnerabilities disclosed by CVE). Table 1 shows the delay of publishing vulnerabilities in the 4 OSS. The column *Count* indicates the number of collected vulnerabilities, *Delay%* indicates the percentages of vulnerabilities that are published later than their SRCs, and *Delay on avg.* indicates the average delay of the vulnerabilities in each OSS. The result shows that all of the 4 OSS suffer from the Pre-Patch problem, and the majority of vulnerabilities are published later than the corresponding SRCs. The average delay of them (Linux Kernel: 137 days, phpMyAdmin: 42 days, OpenSSL: 29 days, MantisBT: 102 days) are long enough for an attacker to exploit.

Table 1. The delays of publishing vulnerabilities.

OSS	Count	Delay %	Delay on avg.
Linux Kernel	889	97.6%	137 days
phpMyAdmin	89	97.8%	42 days
MantisBT	41	100%	102 days
OpenSSL	60	76.7%	29 days

From Fig. 2, some vulnerabilities are published in relatively short time (1 month) after the SRCs are released. However, the delay of more than 10% of the vulnerabilities is much more than 1 month and worth improving. Particularly, more than 50% of the Linux Kernel SRCs are prematurely released for more than one month. In addition, the delay of Linux Kernel is much slower than the other 3 OSS. It probably because of the difficulty of ensuring all Linux Kernel based softwares have their bug fixed.

The details of the Pre-Patch problem in the 4 OSS are shown in Fig. 3. Take Linux kernel as an example shown in Fig. 3(a), only 21 vulnerabilities are published in 24 h after the corresponding SRCs released. 373 vulnerabilities are published in 1 month after their SRCs. However, the remaining 516 vulnerabilities leave more than 1 month for attackers to find the SRCs, generate exploits and attack unaware users. Furthermore, 90 SRCs were released ahead of the

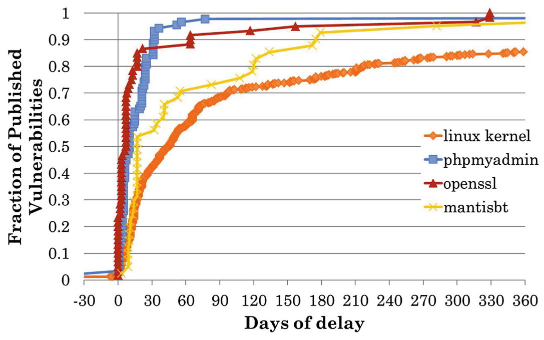
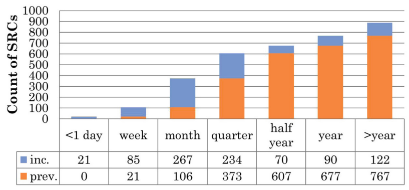
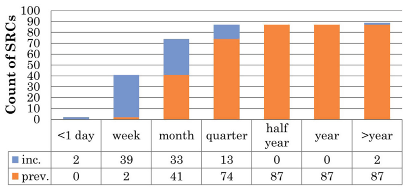


Fig. 2. The statistic of the delay.



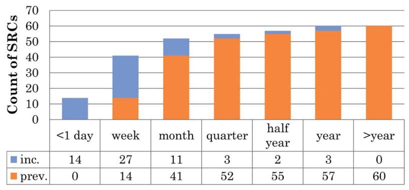
	<1 day	week	month	quarter	half year	year	>year
🟦 inc.	21	85	267	234	70	90	122
🟧 prev.	0	21	106	373	607	677	767

(a) Detail of delay in Linux kernel.



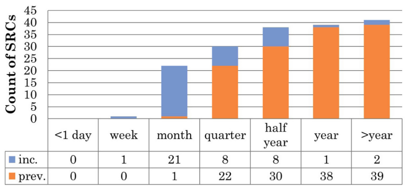
	<1 day	week	month	quarter	half year	year	>year
🟦 inc.	2	39	33	13	0	0	2
🟧 prev.	0	2	41	74	87	87	87

(b) Detail of delay in phpMyAdmin.



	<1 day	week	month	quarter	half year	year	>year
🟦 inc.	14	27	11	3	2	3	0
🟧 prev.	0	14	41	52	55	57	60

(c) Detail of delay in OpenSSL.



	<1 day	week	month	quarter	half year	year	>year
🟦 inc.	0	1	21	8	8	1	2
🟧 prev.	0	0	1	22	30	38	39

(d) Detail of delay in MantisBT.

Fig. 3. The details of the Pre-Patch problem of the 4 OSS.

corresponding vulnerabilities 6 months to 1 year earlier. To make matters worse, 122 vulnerabilities even have a delay of more than 1 year that leave extremely long attack windows. The attack window gives attackers a significant opportunity to attack. The vulnerability release delay in the other 3 OSS (phpMyAdmin, OpenSSL, MantisBT) are shown in Fig. 3(b), (c) and (d) respectively. According to the figures, the majority of vulnerabilities are published 1 quarter after the SRCs release. Comparing with the other 3 OSS, OpenSSL shows a relatively faster vulnerability disclosure, yet it still has 8 vulnerabilities which have a delay of more than 1 month up to half of a year.

3 Text Mining

When submitting a SRC, the author usually offers some description as the comments to explain the details of the vulnerability. This can help other researchers/developers verify and analyze the problem. We believe that SRCs probably contain some special security-related words and phrases in their comments. However, such words and phrases unintentional help attackers locate the SRCs in countless commits. Therefore, we use Text Mining method to extract the features of SRCs and build a machine learning-based system to automatically identify SRCs from large-scale commits.

We present a *Text Mining method* to extract the security related features, and classify the documents into a fixed number of predefined categories [15,16,26]. We propose to use the method to extract the security-related keyword and combine the Back Propagation Artificial Neural Network to classify the description that introduce the SRCs. Our algorithm consists of four steps including *word segmentation*, *word set expansion* and *feature extraction*.

In this section, we leverage 648,149 commits released before Jan 1, 2016 from the code repositories for Linux Kernel [7], OpenSSL [6], phpMyadmin [4] and Mantisbt [3]. We separate the commits into two datasets which are the SRC Dataset (SRCD) and the Other Commits Dataset (OCD). The SRCD consists of the SRCs of known vulnerabilities while the OCD contains the rest of the commits. Generally, each commit has the author information of the commit, committer information, commit id, date and time, title and description of the commit, reviewer information, code changes, and other information. We focus on the title and description and extract features by analyzing the datasets. Finally, we introduce the features and discuss how to extract those features and how to use the extracted features to identify the SRCs.

3.1 Word Segmentation

The first step is to partition the comments/description of each commit into single words, and construct a word set for each of them, called word segmentation. Then we optimize the result by deleting stop words (such as *to, and, is, are*) and merging synonyms (such as *info/information, leak/leaks, fix/fixes*).

Taking SRC 7775142 as an example, it is the SRC of vulnerability CVE-2015-8215, which is a Denial of Service (DoS) vulnerability caused by missing value validation. The comment/description of it is separated into single words, as shown in Table 2.

Comparing the comment/description with the result of word segmentation, we find that the words like "exploit", "remote", "attacker", "lead", and "DoS" in the SRC are used to describe the details of the vulnerability. Some of them are especially applied to express a security related issue and appear in other SRCs.

3.2 Word Set Expansion

Some words are often used simultaneously for describing security related contents. For example, "buffer overflow" is a kind of code flaw which may cause a crash or privilege escalation, while "info leak" means a kind of bug that may lead to privacy leakage.

Table 2. Word segmentation result.

SRC	7775142
CVE	CVE-2015-8215
Words	Daemon, have, like, NetworkManager, running, **exploit**, **remote**, **attacker**, forging, RA, packets, invalid, MTU, possibly, **lead**, **DoS**, currently, only, validates, values, small
Comment	If you have a daemon like NetworkManager running, this may be exploited by remote attackers by forging RA packets with an invalid MTU, possibly leading to a DoS. (NetworkManager currently only validates for values too small, but not for too big ones.)

To find such word groups, we use the Apriori algorithm to expand the results of word segmentation, called *word set expansion*. The Apriori algorithm is presented for frequent item set mining and association rule learning [10]. It is a data mining algorithm which generates candidate item sets of length k from item sets of length $k-1$ by pruning infrequent ones. Here, it combines the words according to their support and confidence.

3.3 Feature Extraction

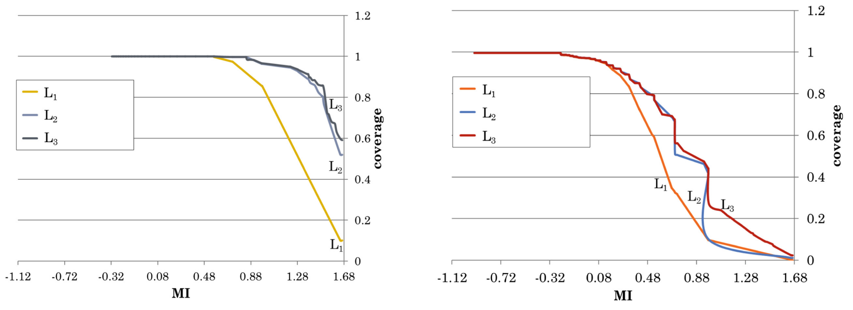
The core step of the Text Mining algorithm is distinguishing security related keywords from the expanded word set. We use Mutual Information (MI) here to calculate the difference between the proportion of a word appeared in SRCs and in other commits, and then to find the words or phrases that are frequently used to describe SRCs rather than other commits.

$$MI(t_k, c_i) = log\frac{P(t_k, c_i)}{P(t_k)P(c_i)} = log\frac{P(t_k|c_i)}{P(t_k)} \tag{1}$$

In Formula 1, t_k means an item consisting of the expanded word set while c_i indicates a security related commit. The formula aims at calculating $MI(t_k, c_i)$ which is the mutual information entropy of t_k according to the probability P. If a word/phrase t_k frequently appears in security related commits c_i rather than in the other commits, the entropy value $MI(t_k, c_i)$ will be high. Therefore, we treat the items in the expanded word set which have high entropy values as the features of SRCs.

According to the Apriori algorithm, the word set expansion step extends the word set by iteration. However, excessive iteration may cause overfitting and thus reduce the quality of features. To find a proper number of iterations, we analyze the coverage rate of the commits including the words/phrases which are contained in the expanded word sets of different levels, as shown in Fig. 4(a) and (b).

Figure 4(a) shows the proportion of the SRCs, which include the items in the expanded word sets of different levels, and Fig. 4(b) shows the proportion of the other commits. In the figures, the x-axis indicates the MI entropy which varies



(a) The coverage rate of SRCs. (b) The coverage rate of other commits.

Fig. 4. The coverage rate based on expanded word sets of different levels

from a fix number to "$-\infty$" and the y-axis is the coverage of commits which contain the items under the corresponding MI entropy. $loop_k$ corresponds to $\bigcup_k L_k$ in Apriori with $k \in \{1, 2, 3\}$. $loop_1$ indicates the expanded word set consisting of single words. $loop_2$ means the combination of the single words and the items including two words, and $loop_3$ includes all of the items whose lengths are no more than 3. As we can see, with the increasing of the number of iterations k, the difference between word sets $loop_k$ becomes smaller. The difference between $loop2$ and $loop3$ is less than that of $loop_1$ and $loop_2$, both in the sets of SRCs and other commits. Many times of iterations do not make too much difference, so we set the number of iterations up to 3.

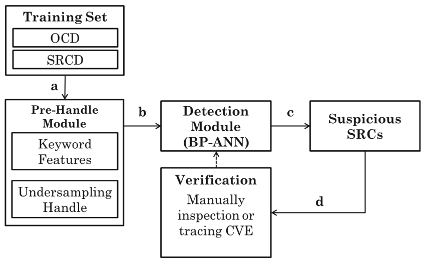
After choosing a proper number of iterations, we give a threshold of MI to extract the features. Thus, we compare the coverage rate of SRCs with other commits given different MI values.

4 Learning-Based Detection

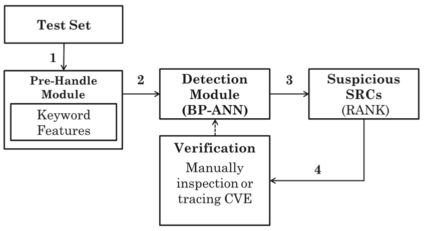
4.1 Overview

Based on the Text Mining methods presented previously, we extract 7,465 keyword features regarded as the features of SRCs. To make these features useful in practice, we develop a learning-based system to automatically train the classifier ASRCD with the features and further identify SRCs in the test set using the trained classifier. The overview of ASRCD is shown in Fig. 5. ASRCD consists of 5 main processes including Training Set (or Test Set), Pre-Handle Module, Detection Module, Suspicious SRCs and Verification.

Figure 5(a) shows the training process. All commits in Training Set are labeled as security-related commits while other commits pre-handled by a Pre Handle Module before identification. The first step of Pre Handle Module is Undersampling, used to mitigate the severe imbalance of current data set. Then, according to the keywords features, the feature matrixes of each commit are



(a) Overview of training process.



(b) Overview of detection process.

Fig. 5. Overview of ASRCD.

established and entered into the Detection Module. After that, a set of Suspicious SRCs are generated and verified according to the label of each commit, all results are feedback to Detection Module to adjust the weight of each feature.

Figure 5(b) shows the process of detection in ASRCD. Separating from training process, the input turn to unlabeled commits and Under-sampling Handle is excluded. All commits in the Test Set are pre-handled by Pre Handle Module to generate feature matrixes of each commit which are then entered into the Detection Module. All the results output from Detection Module are ranked. The results with high ranks are treated as suspicious SRCs. For verification, we manually inspect the results by reviewing them, contacting the authors and tracing the vulnerability information by searching the CVE database. Finally, we label the confirmed SRCs and feedback to the classifier.

4.2 Feature Construction

Data Collection. In this work, we collected 784,959 commits that are released before Jan 1, 2017 from the repositories of Linux Kernel, OpenSSL, phpMyadmin and Mantisbt. Additionally, all merge commits in each OSS are excluded since they are often the duplication of the SRCs. Therefore, the whole size of commits is 648,149. The details of the collected data set are demonstrated in Table 3.

Table 3. Detail of collected data set.

OSS	C(n − m)	SRCD	OCD	Prop.
OpenSSL	18,600	80	18,520	0.43%
phpMyadmin	96,827	153	96,674	0.15%
Linux Kernel	601,317	1,021	600,296	0.17%
Mantisbt	9,632	41	9,591	0.43%

In Table 3, *OSS* means the four open source software, $C(n-m)$ indicates the number of commits without merge commits in each repository, $SRCD$ and OCD

respectively represents the size of SRCs and other commits in the corresponding OSS. *Prop.* means the proportion of $SRCD$ and OCD in each OSS. For instance, Linux Kernel have 601,317 commits in whole data set, 1,021 of them are SRCs while 600,296 of them are other commits. The number of SRCs is only 0.15% of the number of other commits which means that the data set of Linux Kernel is quite imbalance. Therefore, we pose Under-sampling Handle process to eliminate the imbalance in advance to prevent the training process from inaccuracy.

Undersampling Handle. From Table 3, severe imbalance exists in data sets of the four OSS, which probably encumber the accuracy of training process. For this reason, we design a Under-sampling Handle step in Pre-Handle Module to deal with this problem. Especially, for each OSS, we randomly pick several groups of other commits from OCD and ensure that the size of each group is equal to two times of the number of SRCs in the same OSS. ASRCD will be trained several times with different group of other commits and same SRCs until the result turn to stable.

Feature Matrix Generation. For training or detection, we generate a feature matrix for each commit, which will be input into the Detection Module. To generate the feature matrix, we establish several feature vectors for each commit in advance. The feature vector generation is used to establish the relation between the features and commits, based on which the classifier can identify the SRCs. This process can be expressed as $\phi(X) : x \to g(x)$, $x \in X$. X indicates the candidate commit set and $\phi(X)$ indicates the feature matrix generation for the candidate commit set X. x means each commit included in X. The feature vector generation for commit x is indicated by $g(x)$, which is presented below,

$$g(x) = \begin{cases} 1 \; \mathit{if}\, C(x) \cup F_{oss}(MI) \neq \varnothing \\ 0 \qquad\qquad other \end{cases} \tag{2}$$

where x indicates a commit. oss represents the OSS that includes the commit x. MI stands for the threshold of MI. $F_{oss}(MI)$ indicates the keyword features under the threshold MI in the OSS oss. We abstract the process of finding features in the commit x as $C(x)$. Especially, $C(x)$ is defined as all of the words included in the commit x. If $C(x) \cup F_{oss}(MI) \neq \varnothing$, which means the features $F_{oss}(MI)$ exist in the commit x, $g(x)$ will return 1; otherwise return 0.

Combing every feature vector of a candidate commit x, the feature matrix of it is established. For a better understanding, we take the commit 7775142 as an example, the feature matrix of the commit looks as follows.

$$\varphi(x) \to \begin{array}{c} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ \vdots \end{pmatrix} \end{array} \begin{array}{c} vulnerability \\ lead\ DoS \\ remote\ attacker \\ security \\ exploit \\ use\ after\ free \\ \vdots \end{array} \tag{3}$$

The relation between a commit and all the features are demonstrated by a matrix consists of 0 and 1. As previously mentioned, the Word Segmentation step separates the comment of commit 7775142 into several words and phrases. Some keyword features such as “exploit”, “remote attacker” and “lead DoS” are included in the word set. Therefore, the features *lead DoS*, *remote attacker* and *exploit* appearing in the commit are reflected by non-zero dimensions while the unrelated features are labeled as zero dimensions. ASRCD will identify the SRCs in the commits according to the matrix.

4.3 Classification with BP-ANN

Finally, the feature matrix of each commit in the candidate commit set is delivered to the classifier in the detection module for identification. Before identification, the classifier should be trained with plenty of commits, which are labeled as the SRC or the other commit. However, in our case, the label of a commit can be changed - some commits with a label of the other commit is converted to a SRC over time. Changeable commit introduce the noise, which could influence the training performance. Thus, it is necessary to choose a proper algorithm which can be trained with unlabeled samples. For this reason, the module uses a $BP-ANN$ as the final algorithm for integration. $BP-ANN$ is a kind of artificial neural network algorithm using back propagation of errors to resist deviation and refine the identification result. Since it can adjust the references of “neurons” dynamically, the algorithm performs well when classifying a sample set that contains noise.

Another question of this step is the imbalance of the data set. The module trains $BP-ANN$ with the data in OCD and in SRCD whose size is hundreds times larger than OCD.

We implement the $BP-ANN$ algorithm with PyBrain [23] which is an open-source tool integrating most neural network algorithms. The input of $BP-ANN$ for testing is the candidate commit set and the output is a commit with a score ranged from 0 to 1. If the score is close to 1, it indicates that this commit is more likely to be a SRC. Finally, all detection results are ranked by the scores.

5 Evaluation

The Pre-Patch problem violates the mature vulnerability disclosure policy, exposes exploitable information of vulnerable code before the safe publish date, leads to a potential threaten. However, rapidly finding SRCs in large-scale commits seems extremely difficult. In this section, we will break this intuition with ASRCD, evaluate the effectiveness of our approach using several different ways. First, we evaluate the effectiveness of ASRCD in a real-world scenario. We originally constructed training set with SRCs and several groups of other commits in each OSS as described in Sect. 4. We use this dataset for training our Detection Module. Our goal is to find SRCs in large-scale commits. We constructed the test set with all commits released between Jan 1, 2016 and Dec 31, 2016.

Table 4. Datasets constructed to evaluate ASRCD.

OpenSSL			phpMyadmin		
	Training set	Test set		Training set	Test set
SRC	51	29	SRC	84	69
Other	4,080	4,652	Other	18,816	2,662
Linux Kernel			Mantisbt		
	Training set	Test set		Training set	Test set
SRC	895	126	SRC	38	3
Other	107,400	70,061	Other	3,040	843

Table 4 shows the datasets we constructed to evaluate ASRCD. The table consists of 4 parts that indicate the 4 OSS. For each part, rows begin with *SRC* and *Other* indicates the amount of SRCs and other commits, respectively. For instance, the training set of OpenSSL consists of 51 SRCs and 4,080 other commits released before Jan 1, 2016 while the test set combines with 29 SRCs and 4,652 other commits released from Jan 1, 2016 to Dec 31, 2016. Specially, the other commits in training sets are randomly picked from the whole OCD of each OSS and separated into several groups, just like in Sect. 4.

Using the feature extraction methods presented in Sect. 3, we extract 6,934 keyword features from Linux kernel, 365 from phpMyadmin, 114 from OpenSSL and 52 from mantisbt. Then, ASRCD is trained with the training set to predict SRCs in the test set based on the extracted features.

After training, ASRCD uses keyword features of each OSS to identify suspicious SRCs in test set of them.

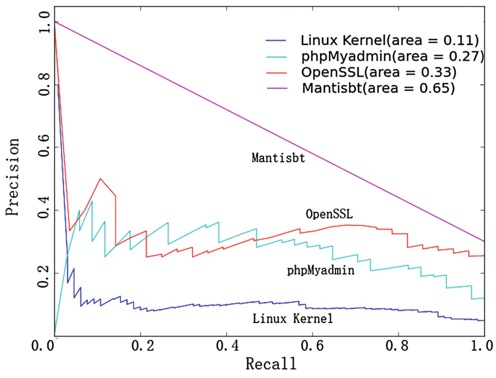


Fig. 6. Identification performance of ASRCD.

Figure 6 shows the precision-recall curves for the results of ASRCD. The curves *Mantisbt*, *OpenSSL*, *phpMyadmin* and *LinuxKernel* indicate the relation between the precision rate and the recall rate of the identification results for Mantisbt, OpenSSL, phpMyadmin and Linux Kernel, respectively. As can be seen, ASRCD performs the best performance when detecting Mantisbt. The performance of the detection for OpenSSL is better than phpMyadmin while the one in Linux Kernel is relatively worse. The result shows that, with ASRCD's help, finding SRCs in Mantisbt is most convenient, identifying SRCs in OpenSSL is more effective than in phpMyadmin and finding SRCs in Linux Kernel is relatively harder than in the other 3 OSS.

Finally, ASRCD identifies 112 suspicious SRCs in OpenSSL, which include 28 known SRCs, 362 suspicious SRCs in phpMyadmin with 34 known SRCs in it, 1,146 SRCs in Linux Kernel with 75 known SRCs and 10 suspicious SRCs in Mantisbt with 3 known SRCs. The detail of result are presented as Table 5.

Table 5. Detection result of four OSS.

OSS	Susp.	SRC	Recl.	Exclude
OpenSSL	112	28	96.6%	97.6%
phpMyadmin	362	34	49.3%	86.4%
Linux Kernel	1,146	75	65.4%	98.4%
Mantisbt	10	3	100%	98.8%

From Table 5, *Susp.* means the number of suspicious SRCs in the OSS and *SRC* represents the number of SRCs in *Susp.*. *Recl.* stands for the recall rate of the result. As can be seen, ASRCD identifies all SRCs in the test set of Mantisbt and find majority of SRCs in OpenSSL and Linux Kernel. The result for phpMyadmin get the lowest *Recl.*. By analyzing the SRCs of phpMyadmin, we find that the SRCs lack plenty comments. In addition, the main purpose of ASRCD is helping rapidly identifying SRCs in large-scale commits. Therefore, the ability of effectively excluding other commits is important. From Table 5, *Exclude* means the percentage of excluded commits, for instance, ASRCD outputs 112 suspicious SRCs in OpenSSL while excludes 4,540 commits that are regarded as other commits, which is 97.6% of all the 4,652 commits in test set. The 97.6% exclusion extremely reduces the time we cost for the further identification and analysis. Furthermore, ASRCD ranks the suspicious SRCs according to their probability scores output by $BP - ANN$.

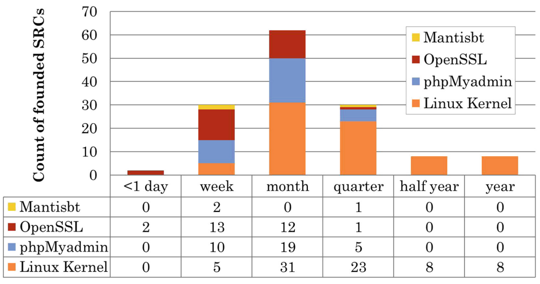
Table 6 indicates the confusion matrix of the system by identifying SRCs in OpenSSL based on the top results, and shows the relation of the precision and recall of ASRCD. In the second row, we select the top 10 suspected results, then 50 in the third row, further 100 in the fourth row and finally 142 in the last row. The above evaluation shows that the ranking is effective. The suspicious results with a higher rank are more likely to be the real SRCs. Among the top 10 results ranked by ASRCD, 4 are SRCs, achieving a precision rate of 40%, which means that the attacker can easily find prematurely released SRCs in high-rank suspected commits. Although the precision rate decreases with the increase of

Table 6. Confusion matrix of OpenSSL detection results order by ranks. T: True, F: False, P: Positive, N: Negative.

	TP	FP	FN	TN	Precision	Recall
Top 10	4	6	25	4,617	0.4	0.14
Top 50	17	33	12	4,590	0.34	0.59
Top 100	27	73	2	4,550	0.27	0.93
Top 142	28	114	1	4,509	0.2	0.97

the recall rate, it still remains 20% when the recall rate approaches 97% in the whole 143 results. This is still a manageable amount of suspected commits for an attacker to review and analyze. More rank results are shown in Appendix.

The Pre-Patch problem violates the vulnerability disclosure policy, prematurely leaks the information of vulnerabilities and extends a potential attack window. To measure the attack window, we calculate the disclosure delay of vulnerabilities whose SRCs are identified, as shown in Fig. 7.



	<1 day	week	month	quarter	half year	year
🟨 Mantisbt	0	2	0	1	0	0
🟥 OpenSSL	2	13	12	1	0	0
🟦 phpMyadmin	0	10	19	5	0	0
🟧 Linux Kernel	0	5	31	23	8	8

Fig. 7. Delay of identified SRCs.

From Fig. 7, all 140 identified SRCs have the Pre-Patch problem. In detail, a large proportion of vulnerabilities of OpenSSL and phpMyadmin are open to the public in one week, leaves a relatively short attack window. However, there still have about half of SRCs that are prematurely released more than one week before the disclosure of corresponding vulnerability. The average delays of OpenSSL and phpMyadmin are 10.3 days and 14.4 days. The probably reason of this situation is that the long delayed vulnerabilities are still unpublished and are treated as suspicious SRCs (not in the 140 SRCs) and therefore fail to take into account. By contrast, the speed of Linux Kernel vulnerability disclosure is slower than the previous 2 OSS. For Linux Kernel, the average delay reaches 61.3 days. Majority of vulnerabilities are published more than one week after the corresponding SRCs release. We do not discuss the result of Mantisbt here since the result is rare.

For all of the 4 OSS, 1.4% vulnerabilities are published by CVE less than one day after SRC release, 22.9% are open to the public after 1 day and in one week. 67.1% are published between one week to one month and 88.6% are disclosed in the period of one month to a quarter. The average delay is 39.5 days which leaves a considerable attack window. Further more, the longest delay reaches 315 days, which leaves a long attack window.

6 Discussion and Future Work

Evaluation Result. In the evaluation, some SRCs of unpublished vulnerabilities fall into FP because of the fact that they are failed to labeled as SRCs. Besides, a small part of links between SRCs and the corresponding vulnerabilities are missed by CVE. Therefore, the unlinked SRCs are failed to labeled as SRCs. For instance, according to the description, commit fcd91dd4 is a SRC to fix the vulnerability CVE-2016-7039. However, the link of the commit is missed in CVE information, which causes a incorrect label. The situations above lead to an underestimated evaluation result of ASRCD, which means the real performance is better than the conservative one shown in evaluation.

Limitations and Future Work. Unlike the attackers who are only concerned with highly suspicious SRCs or those they are familiar with, we handle every suspected result aiming at finding SRCs as many as possible. Thus, precisely verifying every SRC by developing PoC is hard. Instead, in this work, we verify the SRCs by tracing new published vulnerabilities on NVD and confirming with authors of suspected SRCs. All confirmed SRCs are labeled and fed back to our system. In addition, we plan to collaborate with more security experts to speed up the process of SRC verification by analyzing the source code of suspicious SRCs and the corresponding code flaws.

In this work, we focused on the OpenSSL, phpMyadmin, Linux Kernel and Mantisbt, presented and extracted the features from its code repository to analyze the exploitability of the Pre-Patch problem and demonstrate the severity of it. We did not discuss the compatibility of our approach for more OSS. In the future, we will make further effort to apply the method to other OSS.

7 Related Work

The Security Threat of the Information of Vulnerability Leakage. Brumley et al. presented that attackers are able to take advantage of released patch to attack the vulnerable hosts who have not yet received the patch. They further proposed a method of automatic exploit generation based on patch, and showed the security threat of prematurely leaking information of vulnerabilities to attackers [13]. After that, Avgerinos et al. improved the method of automatic exploit generation and presented an end-to-end system for fully automatic exploit generation [11]. Their approach has the ability of finding exploitable bugs according to information of the vulnerability, such as source code, binary and runtime information, which demonstrates the danger of this threat.

Identifying the Potential Vulnerability by Data Mining. Meneely et al. presented that the code that underwent a large number of revisions were more likely to introduce vulnerabilities [20]. In addition, they discussed that code churn and new developers play an important rule in introducing vulnerability-contributing commits (VCCs) [18]. They also found that vulnerable code that was checked by the reviewers who lack security experience and collaborators may escape from the review process and be merged into code repositories [19]. To characterize code changes that contain exploitable vulnerabilities in code repositories, Bosu et al. discussed several characteristics about code churn and author experience [12]. In addition, some previous work was presented to find potentially VCCs in code repositories [21,22]. The previous work has made great effort to characterize and discover VCCs which probably introduce vulnerable code. However, identifying the vulnerabilities in VCCs is still a hard task, since VCCs provide little information about the vulnerabilities. To overcome this obstacle, our approach focuses on characterizing and identifying SRCs in code repositories instead of finding VCCs, since the location and principle of vulnerable code can be intuitively find according to SRCs. To the best of our knowledge, no previous work has been presented to identify the SRCs in code repositories.

Detecting Vulnerabilities Leveraging SRCs. A number of researches took advantage of SRCs to mine potential vulnerabilities in code repositories, such as Jang et al.'s Redebug [14] and Luo et al.'s Patchgen [17]. Those work focused on detecting unpatched vulnerabilities whose SRCs have been given. By contrast, the purpose of our research is to detect unaware SRCs in order to find unpublished vulnerabilities.

8 Conclusions

In this paper, we presented a hidden security threat in OSS: the prematurely released SRCs often leak much information of vulnerabilities which may help attackers locate and exploit the vulnerable code, called the Pre-Patch problem. In addition, we demonstrated the severity and universality of the Pre-Patch problem by analyzing 720,783 commits from 4 popular code repositories. In order to demonstrate the feasibility of the attack based on the Pre-Patch problem, we presented a machine learning-based framework, called ASRCD, to learn the features of SRCs and identify SRCs in large scale code commits. ASRCD firstly extracts several classes of features of SRCs according to data analysis and statistics. Then, it trained a BP artificial neural network classifier to identify SRCs. The evaluation result shows that ASRCD is effective in identifying SRCs in code repository. Our findings have meaningful implications to researchers and vendors in helping them understand the severity of the Pre-Patch problem in OSS and improve the code review and vulnerability disclosure process.

Acknowledgements. This work is supported by the NSFC No.61772507 and Project 2017YFB0801902

References

1. Bugzilla@mozilla (2016). https://bugzilla.mozilla.org/
2. CVE - common vulnerabilities and exposures (CVE) (2016). http://cve.mitre.org/
3. Github - mantisbt (2016). https://github.com/mantisbt/mantisbt/
4. Github - phpmyadmin (2016). https://github.com/phpmyadmin/phpmyadmin
5. Github web-based Git repository hosting service (2016). https://github.com/
6. git.openssl.org git (2016). https://git.openssl.org/
7. Kernel.org git repositories (2016). http://git.kernel.org/
8. Mozilla code repository (2016). https://hg.mozilla.org/mozilla-central/
9. National Vulnerability Database (2016). http://www.nvd.nist.gov/
10. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: VLDB 1994, pp. 487–499 (1994)
11. Avgerinos, T., Cha, S.K., Hao, B.L.T., Brumley, D.: AEG: automatic exploit generation. In: NDSS (2011)
12. Bosu, A., Carver, J.C., Hafiz, M., Hilley, P., Janni, D.: Identifying the characteristics of vulnerable code changes: an empirical study. In: FSE 2014, pp. 257–268 (2014)
13. Brumley, D., Poosankam, P., Song, D.X., Zheng, J.: Automatic patch-based exploit generation is possible: techniques and implications. In: S&P, pp. 143–157 (2008)
14. Jang, J., Agrawal, A., Brumley, D.: ReDeBug: finding unpatched code clones in entire OS distributions. In: S&P, pp. 48–62 (2012)
15. Joachims, T.: Text categorization with support vector machines: learning with many relevant features. In: Nédellec, C., Rouveirol, C. (eds.) ECML 1998. LNCS, vol. 1398, pp. 137–142. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0026683
16. Liu, B., Hsu, W., Ma, Y.: Integrating classification and association rule mining. In: KDD 1998, pp. 80–86 (1998)
17. Luo, T., Ni, C., Han, Q., Yang, M., Wu, J., Wu, Y.: POSTER: PatchGen: towards automated patch detection and generation for 1-day vulnerabilities. In: CCS, pp. 1656–1658 (2015)
18. Meneely, A., Srinivasan, H., Musa, A., Tejeda, A.R., Mokary, M., Spates, B.: When a patch goes bad: exploring the properties of vulnerability-contributing commits. In: ESEM, pp. 65–74 (2013)
19. Meneely, A., Tejeda, A.C.R., Spates, B., Trudeau, S., Neuberger, D., Whitlock, K., Ketant, C., Davis, K.: An empirical investigation of socio-technical code review metrics and security vulnerabilities. In: SSE, pp. 37–44. ACM (2014)
20. Meneely, A., Williams, L.A.: Secure open source collaboration: an empirical study of Linus' law. In: CCS, pp. 453–462 (2009)
21. Mockus, A., Weiss, D.M.: Predicting risk of software changes. Bell Labs Tech. J. **5**(2), 169–180 (2000)
22. Perl, H., Dechand, S., Smith, M., Arp, D., Yamaguchi, F., Rieck, K., Fahl, S., Acar, Y.: VCCFinder: finding potential vulnerabilities in open-source projects to assist code audits. In: CCS, pp. 426–437 (2015)
23. Schaul, T., Bayer, J., Wierstra, D., Sun, Y., Felder, M., Sehnke, F., Rückstieß, T., Schmidhuber, J.: PyBrain. J. Mach. Learn. Res. **11**, 743–746 (2010)
24. Wu, J., Liu, S., Ji, S., Yang, M., Luo, T., Wu, Y., Wang, Y.: Exception beyond exception: crashing android system by trapping in "uncaught exception". In: ICSE 2017, pp. 283–292 (2017)

25. Wu, J., Yang, M.: LaChouTi: kernel vulnerability responding framework for the fragmented Android devices. In: ESEC/FSE 2017, pp. 920–925 (2017)
26. Yang, Y.: An evaluation of statistical approaches to text categorization. Inf. Retr. **1**(1–2), 69–90 (1999)