The Journal of Systems and Software 117 (2016) 218–228

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Mining trends and patterns of software vulnerabilities

Syed Shariyar Murtaza[a,*], Wael Khreich[b], Abdelwahab Hamou-Lhadj[b,*], Ayse Basar Bener[a]

[a] Data Science Lab, Mechanical and Industrial Engineering, Ryerson University, 350 Victoria Street Toronto, M5B 2K3, Canada
[b] Software Behaviour Analysis (SBA) Research Lab, Department of Electrical and Computer Engineering, Concordia University, 1515 St. Catherine West, Montreal, Quebec H3G 2W1, Canada

ARTICLE INFO

Article history:
Received 9 August 2015
Revised 9 February 2016
Accepted 28 February 2016
Available online 4 March 2016

Keyword:
Software vulnerabilities
Vulnerability prediction
Vulnerability trends

ABSTRACT

Zero-day vulnerabilities continue to be a threat as they are unknown to vendors; when attacks occur, vendors have zero days to provide remedies. New techniques for the detection of zero-day vulnerabilities on software systems are being developed but they have their own limitations; e.g., anomaly detection techniques are prone to false alarms. To better protect software systems, it is also important to understand the relationship between vulnerabilities and their patterns over a period of time. The mining of trends and patterns of vulnerabilities is useful because it can help software vendors prepare solutions ahead of time for vulnerabilities that may occur in a software application. In this paper, we investigate the use of historical patterns of vulnerabilities in order to predict future vulnerabilities in software applications. In addition, we examine whether the trends of vulnerabilities in software applications have any significant meaning or not. We use the National Vulnerability Database (NVD) as the main resource of vulnerabilities in software applications. We mine vulnerabilities of the last six years from 2009 to 2014 from NVD. Our results show that sequences of the same vulnerabilities (e.g., buffer errors) may occur 150 times in a software product. Our results also depict that the number of SQL injection vulnerabilities have decreased in the last six years while cryptographic vulnerabilities have seen an important increase. However, we have not found any statistical significance in the trends of the occurrence of vulnerabilities over time. The most interesting finding is that the sequential patterns of vulnerability events follow a first order Markov property; that is, we can predict the next vulnerability by using only the previous vulnerability with a recall of approximately 80% and precision of around 90%.

1. Introduction

Software systems are perhaps among the most vulnerable engineering systems. Everyday new vulnerabilities are discovered and their exploitations for malicious goals continue to threaten the use of software systems. The literature has also shown that zero-day vulnerabilities are increasing dramatically (Frei et al., 2006, Check Point Ltd. 2015)— the number of attacks exploiting these vulnerabilities has increased to 142 million in 2014 from 83 million in 2013 and 34 million in 2012 (Check Point Ltd. 2015). These zero-day vulnerabilities are unknown to software vendors and a rise of more than 300% them from 2012 to 2014 is alarming. Most of the attacks exploiting these vulnerabilities are not diagnosed by modern anti-malware tools because the tools can only detect the patterns (signatures) they have seen before (using signatures and pattern matching techniques) (Murtaza et al., 2014, Murtaza et al., 2013). These new attacks continue to rise despite the fact that vulnerabilities in software systems are regularly patched and the systems are increasingly made secure. According to a recent report (Check Point Ltd. 2015), approximately every half an hour an unknown malware is downloaded (causing an attack) in an enterprise and sensitive data are sent outside the organization. This shows that the analysis of software vulnerabilities—the causes of attacks (exploits)—should be of paramount importance to the research community.

To protect modern software systems from attacks, the sole monitoring of software applications with the goal to develop new security approaches does not seem to be sufficient, e.g., anomaly detection systems are prone to false alarms (Murtaza et al., 2013, Murtaza et al., 2014, Khreich et al., 2010). We need to be proactive when it comes to the protection of software systems rather than reactive. To better understand vulnerabilities in a software system, we need to understand the relationship among vulnerabilities across different software applications. In addition, we need to understand how vulnerabilities occur over a period of time. This information can help in making informed decisions about the

[*] Corresponding author. Tel.:+1 514 848 2424 x.7949.
E-mail addresses: syed.shariyar@gmail.com, syed.shariyar@ryerson.ca (S.S. Murtaza), wkhreich@ece.concordia.ca (W. Khreich), abdelw@ece.concordia.ca (A. Hamou-Lhadj), ayse.bener@ryerson.ca (A.B. Bener).

http://dx.doi.org/10.1016/j.jss.2016.02.048
0164-1212/© 2016 Elsevier Inc. All rights reserved.

occurrence of vulnerabilities. For example, we can use the relationship between vulnerabilities to predict future vulnerabilities in an application and take appropriate measures to avoid them. Similarly, we need to spend more resources on a type of vulnerability that is not significantly decreasing or increasing over a period of time despite all the prior countermeasures.

There exist studies that aim to estimate the risk level of vulnerabilities (Houmb and Nunes Leal Franqueira, 2009), to estimate the time it takes to discover zero-day (new) vulnerabilities (Frei et al., 2006), to determine the time to fix vulnerabilities (Zaman et al., 2011), to predict bugs that will turn into vulnerabilities (Wijayasekara et al., 2012, Wijayasekara et al., 2013, Wijayasekara et al., 2014), and to automatically discover types of vulnerabilities (Neuhaus and Zimmermann, 2010). In addition, several studies have been conducted to show that trends of different vulnerabilities in software applications have been decreasing or increasing (Christey and Martin, 2007, Symantec 2013 trends, April 2014, Barlowe and Blackbird, 2012). In this paper, we focus on exploring the relationship between vulnerabilities and the significance of trends of the software vulnerabilities over a period of time. The intuition is that many vulnerabilities occur in an application but there can be common patterns of vulnerabilities across applications that can be leveraged to improve the vulnerability prediction. Also, the trends in frequency of vulnerabilities can provide useful information, such as the measuring the significance of vulnerability trends can help us in understanding whether the trends are really increasing or decreasing. In particular, we address the following research questions:

(RQ1) How significantly is the trend of software vulnerabilities increasing or decreasing over a period of time?

The analysis of trends of software vulnerabilities across various software applications can help in determining the predominant type of vulnerabilities over a period of time. If the same vulnerabilities are exploited over many years in a similar manner without any significant change, then this may be an indication that attackers are using the same types of attacks for software applications despite the recent advances in software security. In this question, we investigate whether the trends in the frequency of vulnerabilities across all software applications are really significantly changing or not.

(RQ2) What is the trend of vulnerabilities over time within a software application?

Computer science is evolving at a rapid pace. Software applications are becoming more mature and developers are securing software applications from known exploits by using new protection mechanisms. In RQ2, we investigate the significance of trends in the frequency of vulnerabilities within a particular software application. This could help determine whether the improvements in protection mechanisms in a software application have really reduced the vulnerabilities within that application or not. To answer this question, we examine vulnerabilities in approximately 15,000 software applications that are reported in NVD.

(RQ3) What are the common patterns of software vulnerabilities across different software applications?

New software vulnerabilities are being discovered every day in software applications (Frei et al., 2006). Many of these vulnerabilities are repercussions of vulnerabilities diagnosed earlier. For example, a buffer error vulnerability may be followed by an illegal privilege access vulnerability in an application. Identification of common sequential patterns of vulnerabilities could help security experts and developers in understanding the relationship between vulnerabilities and in securing their systems from commonly occurring vulnerabilities. RQ3, therefore, investigates the frequent patterns of vulnerabilities that are common across software applications.

(RQ4) How can we predict the type of vulnerability in a software application?

The sequential patterns of vulnerabilities may also be used to estimate the occurrence of future vulnerabilities in a software application. For example, if two vulnerabilities have occurred frequently together in the past, then a new software application which experiences the first vulnerability may also experience the second one. Therefore, this question investigates how can we predict vulnerabilities in software applications by using sequential patterns of vulnerabilities.

To investigate these questions, we use the National Vulnerability Database (NVD) (National Vulnerability Database (NVD): [Online]: http://nvd.nist.gov). NVD is maintained by the support of the U.S. government and it contains comprehensively documented information about all the vulnerabilities in approximately all known software applications. To answer our research questions, we examine 25,915 vulnerabilities in 15,076 software applications that are reported in NVD from 2009 to 2014. Due to the size and authenticity of this dataset of vulnerabilities, we believe that significant conclusions can be drawn for our research questions. Prior researchers focusing on analysis of vulnerabilities have also used NVD as a comprehensive resource (Christey and Martin, 2007, Frei et al., 2006, Houmb and Nunes Leal Franqueira, 2009, Neuhaus and Zimmermann, 2010, Symantec 2013 trends, April 2014, Barlowe and Blackbird, 2012).

To the best of our knowledge, these research questions have not been addressed in the literature before. To answer the first two research questions, we measure the significance of the trends of vulnerabilities by applying the Cox Stuart trend test (Cox and Stuart, 1955). We solve the third question by extracting the patterns of vulnerabilities using N-grams, and we answer the fourth research question by using those N-grams to predict future vulnerabilities. Our results show that the frequency of vulnerabilities is not really significant and future vulnerabilities in an application can be predicted with approximately 80-90% accuracy by using the last known vulnerabilities in that application. Using the answers of our research questions, software vendors may improve the security of their systems by using additional measures to prevent expected vulnerabilities. In addition, if attackers are exploiting an application in a similar manner as in the past, then this would be a good indication that there is a need to further enhance the software quality (e.g., testing, code review, etc.) process to uncover the specific type of vulnerabilities.

The rest of the paper continues as follows. In Section 2, we present the background and related work. In Section 3, we present the research methodology. Section 4 discusses the results by answering research questions. Section 5 illustrates the threats to validity, followed by a conclusion in Section 6.

2. Background and related work

National Vulnerability Database (NVD) contains information about all publicly known software vulnerabilities (National Vulnerability Database (NVD): [Online]: http://nvd.nist.gov) (Common Vulnerability and Exposure (CVE): [Online]: http://cve.mitre.org/). NVD is a comprehensive repository of information that combines all public sources of vulnerabilities (e.g., weekly alerts from security focus (Security Focus Vulnerability Database: [Online]: http://www.securityfocus.com/)) and vendor specific vulnerabilities (e.g., Microsoft's alerts about vulnerabilities)[1]. Each vulnerability is assigned a unique identifier, referred to as the Common Vulnerabilities and Exposures (CVE) identifier (Common Vulnerability Exposure (CVE): http://cve.mitre.org/). The information included with

[1] Vulnerabilites of all known software applications exist in it; see CVE FAQs for details (Common Vulnerability and Exposure (CVE): [Online]: http://cve.mitre.org/).

Table 1
Example of a reported vulnerability in NVD.

CVE ID	CVE-2010-1146
Date	04/12/2010
Description	The Linux kernel 2.6.33.2 and earlier, when a ReiserFS filesystem exists, does not restrict read or write access to the .reiserfs_priv directory, which allows local users to gain privileges by modifying (1) extended attributes or (2) ACLs, as demonstrated by deleting a file under .reiserfs_priv/xattrs/
CVSS	6.9
Access complexity	Medium
Confidentiality impact	Complete
Integrity impact	Complete
CWE	264 (permission and privilege access vulnerability)

each CVE identifier consists of, but not limited to, the affected software product and sub-products, different versions of the product, the description of the vulnerability, the impact of exploitation of vulnerability, and a vulnerability score calculated using a standardized vulnerability scoring mechanism called the Common Vulnerability Scoring System (CVSS).

A vulnerability has also a type, which is referenced by another identifier, called the Common Weakness Enumeration (CWE) identifier. CWE is a predefined hierarchical list of vulnerability types developed by security experts (Common Weakness Enumeration (CWE): http://cwe.mitre.org/). Each CVE identifier includes a CWE identifier. In most cases, CWE is included with a CVE identifier of a vulnerability. The standardized evaluation measures in the NVD repository allow evaluating different software applications on the same scale. Table 1 shows an excerpt of a vulnerability reported on the NVD. The table shows a vulnerability in the Linux kernel (2.6) where the system does not restrict read and write access to private directories, making it easier for an attacker to gain unauthorized access privileges. The example shows the unique CVE ID, the date of release, a description of the vulnerability, a CVSS score, exploitability, and impact metrics (access complexity, confidentially impact, integrity impact), and a CWE identifier (i.e., a vulnerability type).

The NVD repository has been used in several studies in the past. Frei et al. (2006) used data in NVD and other similar repositories to quantify the differences between the time of discovery of vulnerabilities, the time of disclosure of attacks, and the time of availability of patches. They also identified that exploits (attacks) of zero-day vulnerabilities are available immediately in NVD on the date of disclosure of vulnerabilities but software vendors are slow to provide software patches. They also found that attacks associated with zero-day vulnerabilities are increasing dramatically.

Houmb and Nunes Leal Franqueira (2009) estimated the risk level of vulnerabilities by examining the impact and frequency of vulnerabilities in NVD. They have used CVSS metrics to estimate the misuse frequency and misuse impact of vulnerabilities—i.e., severity of consequences after attacks by exploiting vulnerabilities. They also employed a Markov model to estimate the risk level of vulnerabilities at different times.

Zaman et al. (2011) performed an exploratory study on the comparison of security bugs and performance bugs in Firefox web browser. They used the Firefox bug and CVE repositories to uncover security bugs. They found that security bugs are fixed faster than design bugs and they can be reopened multiple times in a bug repository.

Neuhaus and Zimmermann (2010) used the CVE vulnerability record of up until 2009 to automatically classify vulnerabilities into different types. They argued that there are far too many CWE vulnerability types (approximately 700) and these are beyond human comprehension. They also explained that NVD effectively classifies software vulnerabilities by using very few CWE vulnerability types (approximately 30 types are found in our six years data). The authors proposed an alternative mechanism to automatically categorize CVE vulnerabilities into vulnerability types by using an unsupervised machine learning algorithm, called Latent Dirichlet Allocation (LDA) (Blei et al., 2003). They applied LDA on description of vulnerabilities (see Table 1).

Christey and Martin (2007) published a technical report on vulnerabilities in NVD in 2007. They found that the total number of web application attacks, such as the PHP remote file inclusion has significantly increased in 2006. They also found that buffer overflow is still the number one vulnerability in 2007. Similarly, Symantec technical report published in 2014 (Symantec 2013 trends, April 2014) and the Microsoft's 10 years security report (Barlowe and Blackbird, 2012) have employed the NVD repository to mine trends of different types of vulnerabilities. For example, the reports show the identification of top zero-day vulnerabilities (Symantec 2013 trends, April 2014), and trends in hardware and software vulnerabilities (Barlowe and Blackbird, 2012).

Wijayasekara et al. (2012) mined vulnerabilities that appeared long after the bug has been made public, called hidden impact vulnerabilities. The authors found from CVE and bug databases that Linux kernel and MySQL software applications had 32% and 62% of hidden impact vulnerabilities from 2006 to 2011. They also proposed a text mining classifier to identify the hidden impact vulnerabilities from bug databases. In the text mining classifier, they used text in bug reports and static code measures as features of the classifier. However, they reported a high false positive rate in prediction of vulnerable bugs due to an imbalance in normal bugs and bugs causing vulnerabilities. Wijayasekara et al. (2013) extended their earlier work (Wijayasekara et al., 2012) by proposing the combination of information gain and genetic algorithms to reduce the number of features for a classification algorithm. They evaluated their approach on the same vulnerability and bug database of Linux and MySQL as in Wijayasekara et al. (2012), and reported significant improvement in accuracy. Wijayasekara et al. (2014) evaluated their approach for the detection of hidden impact vulnerabilities (Wijayasekara et al., 2012) using three different classifiers: Naïve Bayes, Naives Bayes Multinomial, and C4.5 Decision Tree. They again used Linux vulnerabilities and bugs for their experiments. In this study, they used only textual information present in the bug reports as features, and found that three classifiers could detect hidden impact vulnerabilities better than a random guess.

In this paper, we perform an exploratory study on NVD. We use information present in NVD to mine trends of vulnerabilities and predict future vulnerabilities in an application. Our methodology for prediction does not require the vulnerability to be identified as bugs first, unlike the earlier studies, which focus on hidden impact vulnerabilities (Wijayasekara et al., 2012, Wijayasekara et al., 2013, Wijayasekara et al., 2014). In addition, we focus on all types of vulnerabilities and not just hidden impact vulnerabilities. The four research questions explained in the previous section are novel. In the case of the first two research questions, we are looking for significances in the trends of vulnerabilities that have not been discussed before in earlier studies. Prior researchers only looked at the trends to report the rise and fall of vulnerabilities. To our knowledge, the last two research questions have not been discussed in the literature at all. In addition, we analyze the latest information up to 2014, whereas, most of the prior work in the literature covers older data.

3. Research methodology

Fig. 1 shows an overview of our approach. We first download the NVD repository for the period of 2009 to 2014 and measure the

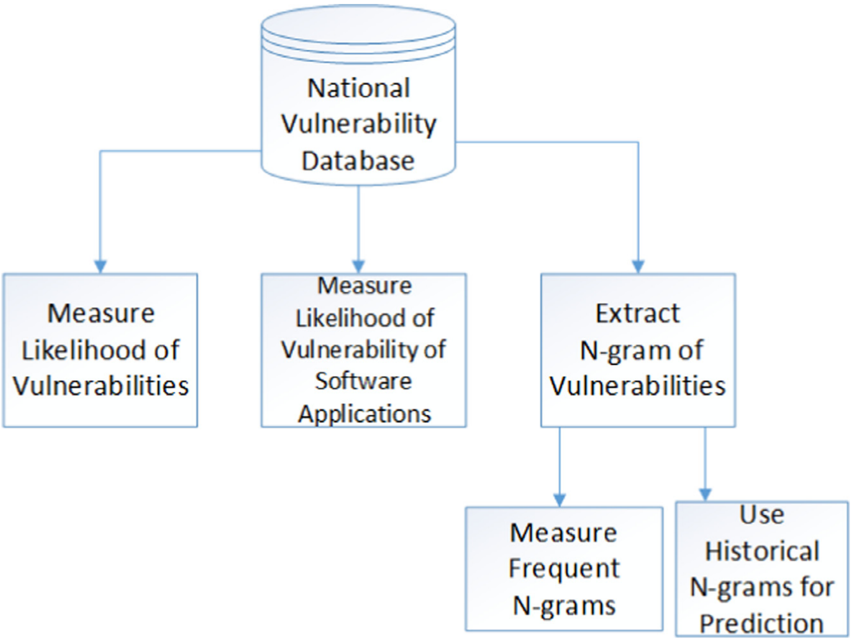
National Vulnerability Database
Measure Likelihood of Vulnerabilities
Measure Likelihood of Vulnerability of Software Applications
Extract N-gram of Vulnerabilities
Measure Frequent N-grams
Use Historical N-grams for Prediction

Fig. 1. An Overview of our research.

likelihood of each vulnerability in each year. We apply a statistical test on this likelihood for different years to determine whether the trend in vulnerabilities is really significant or not. This allows us to determine the answer to RQ1. Second, we measure the likelihood of vulnerabilities within individual software applications from 2009 to 2014. Again using a statistical measure, we estimate from the likelihood whether the trend in vulnerabilities in an application is significantly changing or not (RQ2). Third, we extract N-gram patterns of vulnerabilities that occur in a software application. Each vulnerability has a timestamp associated with it and we use this to temporally order vulnerabilities. In N-gram pattern extraction, patterns of a given length are extracted from a sequence of vulnerabilities by sliding a window over vulnerabilities one by one. For example, if *D, F, R, D* are four vulnerabilities in a sequence, then three patterns of length 2 can be extracted from this sequence, namely *DF, FR*, and *RD*. We measure the most frequent N-grams of vulnerabilities across all applications to determine the common patterns of vulnerabilities. This allows us to find the answer of RQ3. Fourth, we match historical N-grams of vulnerabilities in applications with the most recent vulnerabilities in an application to investigate if the next vulnerability in an application can be predicted or not. This is to determine the answer to RQ4. We describe these steps in detail in the following subsections.

3.1. Measuring the significance of the trends of vulnerabilities across software applications (RQ1)

To determine the significance in the trends of vulnerabilities, we first measure the likelihood of each vulnerability in a year according to Eq. (1). Eq. (1) measures the likelihood of a vulnerability, v_i, in a given year, *y*, by counting the occurrence of that vulnerability in a year and dividing it by the total number of all vulnerabilities in that year. In Eq. (1), *V* is a set of all vulnerability types in a given year and $v_i \in V$. Second, we measure this likelihood of vulnerabilities for each of the years from 2009 to 2014 and apply the Cox Stuart trend test (Cox and Stuart, 1955) to determine if there is any significant difference or not in the likelihood across years. The Cox Stuart trend test is used to determine if a trend of a variable over a period of time is really increasing or decreasing with a significant difference. It has also been used in other studies in software engineering (Barua and Stephen, 2014).

$$\mathrm{Likelihood}(\mathrm{v_i}, \mathrm{y}) = \frac{\mathrm{count}(\mathrm{v_i}, \mathrm{y})}{\sum_{\mathrm{v}\epsilon\ \mathrm{V}} \mathrm{count}\ (\mathrm{v}, \mathrm{y})} \quad (1)$$

3.2. Measuring the significance of the trends of vulnerabilities within a software application (RQ2)

To determine the likelihood of vulnerabilities within a software application over a period of time, we measure the likelihood of vulnerabilities in an application for a given year according to Eq. (2). In Eq. (2), a_i represents an application for which we measure the vulnerability likelihood in a given year y, V is a set of all vulnerabilities in an application in a given year y, *and* A is a set of all applications in a given year y.

$$\mathrm{Likelihood}(\mathrm{a_i},\ \mathrm{V},\ \mathrm{y}) = \frac{\mathrm{count}(\mathrm{a_i},\ \mathrm{V},\ \mathrm{y})}{\sum_{\mathrm{a}\epsilon\ \mathrm{A}} \mathrm{count}\ (\mathrm{a},\ \mathrm{V},\ \mathrm{y})} \quad (2)$$

Eq. (2) is actually used to measure the likelihood of vulnerability of an application by counting the number of occurrences of all vulnerabilities in an application during a year and dividing it by the total number of occurrences of all vulnerabilities in all the applications in that year. We measure this likelihood for the vulnerabilities in every individual application from 2009 to 2014. We then apply Cox Stuart trend test (Cox and Stuart, 1955) to determine if there is any significant difference or not in the likelihood of vulnerability of applications across years.

3.3. Measuring frequent N-grams to determine common patterns of software vulnerabilities (RQ3)

In order to extract patterns of vulnerabilities, we use an N-gram pattern extraction algorithm. The N-gram algorithm is used to extract sequential pattern of events (vulnerabilities) from a sequence of events by sliding a window of a given length, as explained earlier in the lead text of Section 3.

We extracted N-gram patterns of vulnerabilities for each software application and for each year. Suppose, {$\mathrm{A_1}$, $\mathrm{A_2}$, ….., $\mathrm{A_k}$} is a sequence of *k* vulnerabilities that occur successively within an application in a year. We can then extract length 5 N-gram (i.e.,

Table 2
Example N-grams of length 2 (2-grams).

Example 2-grams	Frequency
Buffer error, Command injection	2
Numeric error, Buffer error	1
Command injection, Buffer error	1

5-grams) patterns of vulnerabilities for that application in that year as: {(A_1, A_2, …..A_5), (A_2, A_3, …..A_6), (A_3, A_4, …..A_7)…..(A_{k-4}, A_{k-3}, …..A_k)}. The length of N-grams can vary and is determined by the user. For example, consider {Numeric Error, Buffer Error, Command Injection, Buffer Error, Command Injection} are the vulnerabilities that occur in a sequence on an application. (The sequence can be easily determined by ordering the vulnerabilities according to the date of their occurrences in NVD). The 2-gram patterns from this sequence can be extracted as: {(Numeric Error, Buffer Error), (Buffer Error, Command Injection), (Command Injection, Buffer Error) and (Buffer Error, Command Injection)}. The last pattern, “Buffer Error, Command Injection”, has occurred twice and we only need to extract this pattern only once. To determine how many times a unique pattern has occurred in an application, we also count its frequencies. Therefore, the final 2-grams along with their frequencies for this example are shown in Table 2 in the descending order of their occurrences.

To answer the research question RQ3, we mine length 2, 3, 4 and 5 N-grams (i.e., 2-grams, 3-grams, 4-grams and 5-grams) vulnerability patterns for each individual software application and for each year. Afterwards, we determine the most frequent N-grams patterns of vulnerabilities across different software applications and across all years.

For example, suppose a 5-gram “Numeric Error, Buffer Error, Command Injection, Buffer Error, Command Injection” occurred 10 times in one application and 5 times in another application across all the years. The total frequency of occurrence of this 5-gram will be 15. The most frequent 5-gram pattern of vulnerabilities will be the one that occurred more than the others. The measurement of the most frequent 5-grams of vulnerabilities across all applications allows us to determine the common patterns of vulnerabilities across all applications and in turn finding the answer of RQ3. Similarly, we also determine the 2-grams, 3-grams, and 4-grams patterns of vulnerabilities. We limit ourselves up to 5-grams because higher values of N require exponentially larger datasets to identify sequences of vulnerabilities as the number of permutations increases exponentially with N.

3.4. Using historical N-grams to determine the expected vulnerability in a software application (RQ4)

N-gram patterns of historical vulnerabilities in different software applications from NVD can also be used to predict the next new vulnerability in a software application. We use a simple technique by matching the first few vulnerabilities in an application with the historical N-grams to predict the next vulnerability. We extract 2-grams, 3-grams, 4-grams and 5-grams of vulnerabilities in software application in NVD. We use N−1 vulnerabilities in an application to determine the Nth vulnerability from the historical collection of N-grams. For example, to determine the second vulnerability in an application, we use its first vulnerability to search the first vulnerability in 2-grams. If a match is found, we predict the next vulnerability from the N-gram as the expected vulnerability in that application. Similarly, to determine the third vulnerability in an application, we search the first two vulnerabilities in 3-grams to predict the third vulnerability.

In order to evaluate our approach, we use hold out validation technique. This is a commonly applied technique in machine learning and data mining communities to evaluate the accuracy of a technique. It consists of splitting the available data into training and testing set (Witten and Frank, 2005). The training set is used to compute the N-gram models of vulnerabilities while the test set is only used to evaluate the precision and recall values of each model. We divide the data in a ratio of approximately 60/40 (train/test), that is the vulnerabilities of first few years (e.g., 2009–2012) in NVD are used as a training set to extract historical N-grams and the remaining years (e.g., 2013–2014) are used as the test set. This is further discussed in Section 4.4. The evaluation measures that we use are precision and recall. Precision is the fraction of patterns that are retrieved from the training set of N-grams and are relevant to the patterns in the test set. Precision is measured using Eq. (3). Recall is the fraction of relevant patterns in the test set that are actually retrieved from the training set. It is measured using Eq. (4).

$$\text{Precision} = \left\{ \frac{\begin{array}{c}\text{Number of patterns retrieved}\\ \text{from the training set}\\ \text{that also exist in the test set}\end{array}}{\begin{array}{c}\text{Total patterns retrieved from the}\\ \text{training set}\end{array}} \right\} \tag{3}$$

$$\text{Recall} = \left\{ \frac{\begin{array}{c}\text{Number of pattterns retrieved}\\ \text{from the training set}\\ \text{that also exist in the test set}\end{array}}{\begin{array}{c}\text{Total relevant patterns}\\ \text{in the test set}\end{array}} \right\} \tag{4}$$

4. Results

This section answers the research questions that we identified earlier. In our experiments, we used vulnerability identifier (CVE), vulnerability type, vulnerability release date, and vulnerable software applications from NVD repository. Our dataset consists of 31,993 vulnerabilities of 15,076 software applications. There were 6078 vulnerabilities whose vulnerability types did not exist. We did not consider those vulnerabilities in our analysis, bringing the total number of vulnerabilities to 25,915.

4.1. RQ1: how significantly is the trend of software vulnerabilities increasing or decreasing over a period of time?

Key result: *There is not enough evidence that the trend of software vulnerabilities has changed significantly with the passage of time despite their rise and fall in each year.*

We expect that different types of vulnerabilities in software applications will show increasing or decreasing trends with the emergence of new technologies and software applications. In this section, we are going to determine whether the software vulnerabilities really exhibit such a trend or not.

NVD classifies software vulnerabilities into twenty nine different types depending on the nature of exploits. In NVD, each vulnerability type is given a unique identifier called Common Weakness Enumeration (CWE) identifier. The CWE identifier and the description of each of the twenty nine vulnerabilities are shown in Table 3. It is worth noting that ten of these vulnerability types have appeared only in 2013 and 2014 because of the inclusion of the mobile applications in NVD repository.

We analyze the likelihood of these vulnerabilities from 2009 to 2014 according to Eq. (1). The result is shown in Table 4. Table 4

Table 3
Vulnerability type (Common Weakness Enumeration: CWE) and its description.

CWE id	Vulnerability description	CWE id	Vulnerability description
94	Code injection	189	Numeric errors
89	SQL injection	16	Configuration
79	Cross site scripting	134	String format vulnerability
78	Command injection	119	Buffer errors
59	Link following	+77	Command injection
399	Resource management error	17	Code, specification and design errors
362	Race conditions	74	Injection
352	Cross site request forgery	345	Insufficient verification of data authenticity
310	Cryptographic issues	284	Improper access control
287	Authentication issues	254	Security features
264	Privilege and access control	21	Path equivalence
255	Credential managements	199	Information management errors
22	Path traversal	19	Data handling errors
200	Information leak	18	Source code errors
20	Input validation		

Table 4
Likelihood of vulnerabilities (in percentage) from 2009 to 2014 (CWE is the vulnerability identifier described in Table 3).

CWE	2009	2010	2011	2012	2013	2014
94	6.4508	6.4408	2.8539	3.1963	3.6207	2.5607
89	16.26	15.0802	4.4208	5.2765	3.7791	4.3347
79	17.7579	15.0802	12.8707	18.6707	15.4334	14.4435
78	0.2416	0.3104	0.3917	0.7357	0.6563	0.6192
59	0.7248	0.7243	0.9233	0.4059	0.6336	0.6527
399	5.726	7.0357	11.052	6.5703	6.8341	3.9665
362	0.7973	1.8107	0.6995	0.6849	1.3578	0.6527
352	2.0295	1.8365	1.9586	3.6783	3.7791	3.3808
310	2.0778	1.6813	1.9866	2.7651	3.3492	25.6402
287	3.6241	1.4485	1.9306	2.5622	2.874	1.7238
264	7.9971	8.8722	9.9888	14.7641	14.0756	9.6569
255	1.3771	1.4227	1.1192	1.4967	2.014	1.523
22	5.9435	6.8288	3.0218	2.8412	2.8061	2.6611
200	3.4791	4.3197	8.8696	5.5302	5.6121	5.272
20	6.0643	7.9928	12.2832	9.2593	11.7221	7.431
189	3.9865	4.061	4.113	3.45	3.3265	1.6736
16	1.1839	0.5432	1.1192	0.6849	0.5205	0.1674
134	0.5799	0.3104	0.2518	0.279	0.2489	0.0669
119	13.699	14.2007	20.1455	17.1487	17.2663	11.8159
77	NA	NA	NA	NA	0.0453	0.2176
17	NA	NA	NA	NA	0.0453	0.4351
74	NA	NA	NA	NA	NA	0.0335
345	NA	NA	NA	NA	NA	0.0502
284	NA	NA	NA	NA	NA	0.4017
254	NA	NA	NA	NA	NA	0.1339
21	NA	NA	NA	NA	NA	0.0167
199	NA	NA	NA	NA	NA	0.0335
19	NA	NA	NA	NA	NA	0.4184
18	NA	NA	NA	NA	NA	0.0167

also shows the trend of theses vulnerabilities over years. For example, buffer error vulnerability (CWE id = 119) have seen an increase from 2009 until 2011 and then it started decreasing. SQL injection vulnerability (CWE id = 89) and code injection vulnerability (CWE id = 94) have seen an important decrease from 2009 to 2014. However, the cryptographic vulnerabilities (CWE id = 310) have suddenly increased in 2014 to 25.64% from 3% in earlier years. Other vulnerabilities, such as input validation errors (CWE id = 20) show a mix trend from 2009 to 2014. We have also highlighted these vulnerabilities in Table 4. In Fig. 2, we show the trends of some selected vulnerabilities from Table 4 for better visualization. The figure can be used by analysts to quickly pinpoint vulnerabilities that show constant increase or decrease.

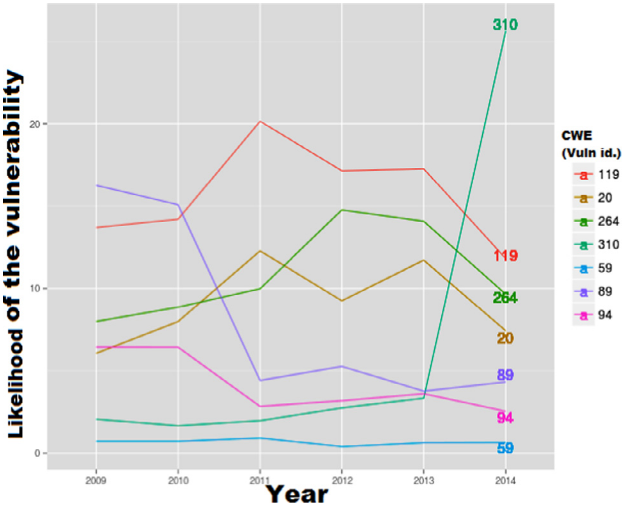

Fig. 2. Selected vulnerabilities and their likelihood over a period of time.

We can see in Fig. 2 that the cryptographic vulnerabilities (CWE id =310) in 2014 have seen an astounding increase. When we further examined the NVD repository, we found that many vulnerabilities that have been reported in 2014 are related to Android applications, which were not present in previous years. The rise of cryptographic vulnerabilities is mostly in mobile (Android) based software applications. To find the actual reason, detailed analysis of Android applications is required. However, Rozenfeld (Rozenfeld, 2015) reported that there are more than one million mobile applications and equally the same number of developers, compared to a handful of companies that develop laptop and desktop applications. Many of the mobile application developers are novice designers with little knowledge of security and data privacy concepts (Rozenfeld, 2015). We shall further investigate this by individually analyzing applications' vulnerability in the next section.

We continue our investigation of trends of vulnerabilities further by using the Cox Stuart trend test (Cox and Stuart, 1955). We use the Cox Stuart test to understand whether there is a significant difference in the impact of vulnerabilities across years or not. We state the following null hypothesis: there is no significant difference in the trend of vulnerabilities across different years. We evaluate this null hypothesis for every vulnerability type shown in Table 4. We use 95% confidence level to accept or reject the null hypothesis.

The Cox Stuart trend test could not reject the null hypothesis for all the vulnerability types. This means that there is not enough evidence to determine the statistically significant difference in the trend of vulnerabilities across years. This also implies that the rise and fall in likelihood of vulnerabilities across different years is just by chance and the vulnerabilities in software systems are continued to be exploited in a similar likelihood.

We conclude in this section that there is not enough evidence that the likelihood of software vulnerabilities has changed significantly with the passage of time despite their rise and fall in each year. This answers our research question (RQ1).

4.2. RQ2: what is the trend of vulnerabilities over time within a software application?

Key result: *There is not enough evidence to conclude that the trend in vulnerabilities within every individual application is really significantly increasing or decreasing.*

Table 5
Likelihood (in percentage) of top 20 attacked software applications sorted from 2014 to 2009 (NA means that no information was found in NVD).

Application	2009	2010	2011	2012	2013	2014
Microsoft internet explorer	0.45	0.014	NA	0.20	1.23	2.42
Apple Mac OS X	1.13	1.40	1.22	0.22	0.60	1.50
Apple IPhone OS	0.38	0.46	0.70	1.09	0.87	1.38
Google chrome	0.43	2.20	5.70	1.49	1.78	1.24
Linux kernel	1.53	2.06	2.09	0.76	1.52	1.20
Mozilla firefox	1.78	1.49	1.47	1.69	1.43	1.17
Oracle JDK	NA	0.01	0.04	0.55	1.83	1.04
Oracle JRE	NA	0.01	0.06	0.66	1.83	1.04
Sun JDK	0.47	1.07	0.54	0.42	1.23	NA
Sun JRE	0.56	1.07	0.54	0.52	1.23	NA
Apple TV	NA	NA	0.06	NA	0.04	0.97
Adobe flash player	0.29	0.84	0.93	0.70	0.53	0.76
Apple safari	0.99	1.77	0.70	0.91	0.16	0.71
Mozilla thunderbird	0.56	0.87	0.99	1.53	1.09	0.66
Mozilla sea monkey	0.79	1.13	0.97	1.53	0.99	0.63
Mozilla firefox ESR	NA	NA	NA	1.23	0.95	0.61
Oracle fusion middleware	NA	0.46	0.40	0.66	0.41	0.59
Oracle MySQL	NA	NA	0.01	0.78	0.60	0.58
Moodle	0.19	0.16	0.70	0.64	0.30	0.47
Adobe air	0.10	0.10	0.40	0.58	0.46	0.45

In the previous section, the likelihood of software vulnerabilities in each year gave us useful information about the trend of software vulnerabilities across different years. However, it did not reveal the likelihood of exploited vulnerabilities within individual software applications. In this section, we analyze the likelihood of software vulnerabilities within each software application in the NVD repository. The likelihood is measured using Eq. (2). This section also analyzes whether some software applications could have more software vulnerabilities than other software applications.

The NVD repository contains approximately 15,076 distinct software applications in total. In Table 5, we present the top 20 most frequently exploited software applications by attackers in the period of 2009–2014. Each cell in the table represents the likelihood of vulnerabilities in an application in the mentioned year. We have also highlighted the most ecploited software application for each year. It could be observed that Microsoft Internet Explorer was the highly vulnerable software application in 2014. Similarly, Oracle JDK and Oracle JRE were the most vulnerable software applications in 2013, Mozilla Thunderbird was the most vulnerable in 2012, Google Chrome took the lead in vulnerabilities in 2011 and 2010, and Mozilla Firefox was the most vulnerable application in 2009. It can be inferred from Table 5 that there is no one single software application that has been mostly exploited by software attackers.

Table 5 also shows the trend of vulnerabilities in each software application. For example, vulnerabilities in Google Chrome peaked in 2011 since the tool's launch in September 2008. In fact, Google Chrome had the highest number of vulnerabilities in 2011 than any other application across all six years. However, in the years following 2011, vulnerabilities have reduced substantially in Google Chrome.

In Section 4.1, we note that collective (cryptographic) vulnerabilities in Android applications are higher than all other software applications. However, individually Android-based applications do not have as many vulnerabilities as traditional software applications as we see in this section. The reason could be the large code base of traditional software applications compared to the small code base of Android applications, due to the processing and storage limitations of mobile platforms. This also implies that large vulnerabilities are not necessarily due to the novice developers of some mobile applications than the professional developers of desktop applications. There are other reasons too, which need to be explored, such as large code bases and popularity of the applications, etc.

In order to ascertain that there is a significant increase or decrease in software vulnerabilities of each application, we employed the Cox Stuart trend test at the 95% confidence level on each individual application separately. Our null hypothesis is that no significant difference exists in the trend of vulnerabilities in a software application across different years. The results of the Cox Stuart trend test on each application showed that we cannot reject the null hypothesis for any application. In other words, there is not enough evidence to conclude that the trend in vulnerabilities of every individual application is really significantly increasing or decreasing.

Thus, the increase and decrease in vulnerabilities of an application is just showing numerical characteristics of rise and fall of vulnerabilities but the numerical evidence is not sufficient to conclude that the trend is increasing or decreasing. This answers our research question (RQ2).

4.3. What are the common patterns of software vulnerabilities across different software applications?

Key result: *The most common patterns of vulnerabilities are the same type of vulnerabilities occurring multiple times in the same application and at the same point of time.*

Vulnerabilities in the NVD repository can be chronologically ordered by their release dates and we can get the sequences of vulnerabilities for any application in NVD. For example, for Apple Safari web browser, a sample of CWE identifiers (vulnerability types), from the year 2014, in a chronological order are 119, 119, 20, 399, and 399. This sequence can be read as two buffer error vulnerabilities (119) are followed by an input validation vulnerability (20) and two resource management error vulnerabilities (399) during the year 2014.

To automatically extract sequential pattern of vulnerabilities, recall from Section 3.3, that we employ N-gram pattern extraction algorithm to extract vulnerabilities of length 2, 3, 4 and 5. We extract different length N-grams of vulnerabilities for each application during each year. We also count the frequency of occurrence of each N-gram. The same N-gram of vulnerabilities can occur in different applications and across different years too. We determine the frequency of N-grams across all applications and years by summing their frequency of occurrences in individual applications.

For example, Table 6 shows the list of top 20 frequently occurring 5-grams of vulnerabilities in the period of 2009–2014. The top 20 frequent 5-grams are determined by ordering the 5-grams in the descending order of their maximum occurrence in an application. Consider the 5-gram pattern "CWE-119, CWE-119, CWE-119, CWE-119, CWE-119"of 5 buffer error vulnerabilities in Table 6. It has occurred 1138 times in all the applications and during all the years. This 5-gram of buffer error vulnerability occurred only once at the minimum in one application and 148 times at the maximum in another application during one year. Table 6 also shows many other interesting patterns, such as the race condition (CWE-362) vulnerability can follow each other up to 33 (28+5) times in an application, information leak vulnerability (CWE-200) can occur as many as 29 (23+5) times in an application, and so on.

We found similar patterns of vulnerabilities in the case of 2-grams, 3-grams and 4-grams as for 5-grams. For example, patterns of buffer error vulnerability, race condition and information leak vulnerability remained the most frequent patterns with different lengths of N-gram. Thus, we showed here only 5-grams of

Table 6
Frequent N-grams (5-grams) across all software applications in NVD from 2009–2014.

N-gram pattern of length 5	Occurrences		
	Min	Max	Total
"CWE-119" "CWE-119" "CWE-119" "CWE-119" "CWE-119"	1	148	1138
"CWE-362" "CWE-362" "CWE-362" "CWE-362" "CWE-362"	28	28	140
"CWE-200" "CWE-200" "CWE-200" "CWE-200" "CWE-200"	3	23	30
"CWE-399" "CWE-399" "CWE-399" "CWE-399" "CWE-399"	1	21	260
"CWE-264" "CWE-264" "CWE-264" "CWE-264" "CWE-264"	1	18	270
"CWE-94" "CWE-94" "CWE-94" "CWE-94" "CWE-94"	1	14	80
"CWE-79" "CWE-79" "CWE-79" "CWE-79" "CWE-79"	1	13	41
"CWE-20" "CWE-20" "CWE-20" "CWE-20" "CWE-20"	1	8	134
"CWE-264" "CWE-119" "CWE-119" "CWE-119" "CWE-119"	1	4	51
"CWE-119" "CWE-189" "CWE-119" "CWE-119" "CWE-119"	1	4	24
"CWE-189" "CWE-119" "CWE-119" "CWE-119" "CWE-119"	1	4	38
"CWE-119" "CWE-119" "CWE-119" "CWE-119" "CWE-20"	1	4	46
"CWE-20" "CWE-399" "CWE-20" "CWE-20" "CWE-20"	1	4	7
"CWE-399" "CWE-399" "CWE-399" "CWE-119" "CWE-399"	1	4	27
"CWE-399" "CWE-399" "CWE-399" "CWE-399" "CWE-119"	1	4	25
"CWE-119" "CWE-399" "CWE-399" "CWE-399" "CWE-399"	1	4	24
"CWE-119" "CWE-119" "CWE-119" "CWE-119" "CWE-399"	1	3	40
"CWE-119" "CWE-264" "CWE-119" "CWE-119" "CWE-119"	1	3	28
"CWE-119" "CWE-119" "CWE-189" "CWE-119" "CWE-119"	1	3	27

patterns to avoid cluttering and 5-gram patterns also show patterns of smaller lengths.

We further explored the NVD repository to examine if the sequential occurrences of the same vulnerabilities (e.g., "CWE-119, CWE-119, CWE-119, CWE-119, CWE-119") are due to the duplicate entry in the database or they are really due to the occurrence of a different vulnerability. We found that, in NVD, there are vulnerabilities with different CVE identifiers but with the same vulnerability type (CWE identifier), release date, and description. We contacted the CVE editor, the author of 2007 technical report (Christey and Martin, 2007), to validate the duplicate CVE identifiers. He explained that there are no CVE identifiers that are duplicates. If two CVE identifiers occur at the same time and have the same description but no distinguishing information, then the vendors have not provided distinguishing information other than that the vulnerabilities are different. This could be due to the ripple effect of interconnected components in large software applications. Thus, the patterns of occurrences of similar vulnerabilities occurring multiple times are perfectly valid.

In this section, we conclude that the most common patterns of vulnerabilities are the same type of vulnerabilities occurring multiple times in the same application and at the same point of time. This may be due to the large number of components in an application: when a given vulnerability occurs in one component, it ripples through the whole source code. This answers our research question (RQ3).

Table 7
An excerpt of different length vulnerability patterns (2-grams, 3-grams, 4-grams and 5-grams).

Different N-gram patterns
"CWE-189" "CWE-399"
"CWE-189" "CWE-119"
"CWE-20" "CWE-119" "CWE-264"
"CWE-20" "CWE-119" "CWE-399"
"CWE-119" "CWE-119" "CWE-189" "CWE-94"
"CWE-119" "CWE-119" "CWE-189" "CWE-119"
"CWE-119" "CWE-119" "CWE-20" "CWE-119" "CWE-399"
"CWE-119" "CWE-119" "CWE-20" "CWE-119" "CWE-264"

4.4. How can we predict the type of vulnerability in a software application?

Key result: *The 2-gram patterns of historical vulnerabilities in different software applications can be used to predict the next vulnerability in a new application with approximately 90% precision and approximately 80% recall.*

In this section, we discuss the use of patterns (N-gram) of historical vulnerabilities in software applications to predict the next vulnerability that can occur in any application. For example, suppose that we have a software application, a buffer error vulnerability has recently been discovered in that application, and we also have a historical collection of 2-grams of vulnerabilities (i.e., patterns of 2 consecutive vulnerabilities) found in NVD. We can search for all those 2-grams whose first vulnerability is the buffer error vulnerability and use the second (following) vulnerability to predict the next possible vulnerability. Similarly, if we know the first two vulnerabilities in an application, then we can search 3-grams of vulnerabilities to predict the third vulnerability in an application. We can continue this trend for higher N-grams too by finding the match of N−1 vulnerabilities in a collection of N-gram patterns to predict the nth vulnerability.

In Table 7, we show an excerpt of 2-grams, 3-grams, 4-grams, and 5-grams, when the first N−1 vulnerabilities are similar, and the Nth vulnerability is different. For example, in the case of 2-grams, it is shown that Numeric errors (CWE -189) is followed by two buffer error vulnerabilities are often followed by resource management error (CWE-399) and the buffer error vulnerability (CWE-119). Similarly, in the case of 5-grams, it is shown that two buffer error vulnerabilities (CWE-119), input validation error (CWE-20), and a buffer error vulnerability (CWE-119) are followed by resource management error (CWE-399), and privilege and access control error (CWE-264).

In order to evaluate the predictive capability of N-grams to predict the next vulnerability, we perform a simple information retrieval experiment. We divide our collection of N-gram vulnerabilities into training set and test set, as explained in Section 3.4. We performed two experiments. First, we use the N-gram vulnerabilities from 2009 to 2012 for training and N-gram vulnerabilities from 2013 to 2014 (2 years) for testing. Second, we use the N-gram vulnerabilities from 2009 to 2011 for training and N-gram vulnerabilities from 2012 to 2014 (3 years) for testing; we use only data of 2009–2011 for training because we wanted to test the prediction power with less and older data. We perform these two experiments for four types of N-grams: 2-gram, 3-grams, 4-grams and 5-grams.

Our criterion for evaluation is simple. We use the N−1 vulnerability sequences from N-grams in the test set to search for the next (Nth) probable vulnerability from the N-grams in the training

Table 8
Using initial vulnerabilities to predict the next vulnerability when training set spans 2009–2012 years and test set spans 2013–2014 years.

N-grams type	Vulnerability search (N-1)	Recall	Precision
2-grams	1	0.78	0.90
3-grams	2	0.60	0.59
4-grams	3	0.34	0.39
5-grams	4	0.16	0.32

Table 9
Using initial vulnerabilities to predict the next vulnerability when training set spans "2009–2011" years and test set spans "2012–2014" years.

N-grams type	Initial vulnerability search	Recall	Precision
2-grams	1	0.77	0.92
3-grams	2	0.50	0.67
4-grams	3	0.33	0.43
5-grams	4	0.15	0.36

set. We then determine how many N-gram patterns that start with N−1 vulnerabilities in the test set are actually present in the retrieved list of patterns in the training set, and quantify it by using the precision and recall measures (see Section 3.4). Tables 8 and 9 show the results of our experiments for two different combinations of the training set and the test set. The first column in the tables show the N-gram type, the second column shows the N-1 grams of vulnerabilities used to search the Nth vulnerability, the third column shows the recall and the fourth column shows the precision. For example, the first row in Table 8 shows that when the first vulnerability is known, then the following (second) vulnerability can be predicted with 78% recall and 90% precision. Similarly, if first two vulnerabilities are known then the third vulnerability can be predicted with 60% recall and 59% precision.

It can be observed from Table 8 and Table 9 that the recall and precision are high when we are searching for the next vulnerability based on a single vulnerability (i.e., using 2-grams). However, the recall and precision measures decrease when higher N-grams are used to search for the next vulnerability. This is because the number of ordered (sequential) combinations of vulnerabilities increases in higher N-grams and there are more chances of not finding an existing pattern of vulnerabilities similar to the pattern searched. Nonetheless, the recall of 77–78% and precision of 90–92% shows that the following vulnerability based on an existing vulnerability can be predicted efficiently by using the 2-grams patterns of vulnerabilities.

Tables 6 and 7 show that many vulnerabilities reoccur after their first occurrences. One may argue that high precision and recall in the case of 2-grams could simply be due to the contiguous repetition of the same vulnerability. To ascertain the results of our experiments, we repeated our experiments by removing the contiguous repetitions of vulnerabilities—i.e., if a vulnerability occurs more than once contiguously, we just consider that it occurred once. Some examples of the vulnerability patterns after removing contiguous repetitions are shown in Table 10.

Table 10
Different N-gram patterns of vulnerabilities without contiguous repetitions.

N-grams
"CWE-20" "CWE-362"
"CWE-20" "CWE-399"
"CWE-119" "CWE-189" "CWE-399"
"CWE-399" "CWE-119" "CWE-399" "CWE-119"
"CWE-362" "CWE-20" "CWE-399" "CWE-20" "CWE-399"

We again performed the same two experiments on N-grams (2-grams to 5-grams) without contiguous repetitions of vulnerabilities. In the first experiment, training was performed on vulnerabilities from 2009 to 2012 and testing was performed on vulnerabilities from 2013 to 2014. In the second experiment, training was done on vulnerabilities from 2009 to 2011 and testing was done on vulnerabilities from 2013 to 2014. The results are shown in Tables 11 and 12.

Table 11
Using initial vulnerabilities to predict the next vulnerability when training set spans "2009–2012" years, test set spans "2013–2014" years and contiguous repetitions of vulnerabilities are removed.

N-grams type	Initial vulnerability search	Recall	Precision
2-grams	1	0.83	0.88
3-grams	2	0.57	0.55
4-grams	3	0.30	0.36
5-grams	4	0.11	0.27

Table 12
Using initial vulnerabilities to predict the next vulnerability when training set spans "2009–2011" years, test set spans "2012–2014" years and contiguous repetitions of vulnerabilities are removed.

N-grams type	Initial vulnerability search	Recall	Precision
2-grams	1	0.76	0.91
3-grams	2	0.46	0.62
4-grams	3	0.28	0.40
5-grams	4	0.10	0.32

It is clear from Tables 11 and 12 that the precision and recall are similar to the precision and recall of Table 8 and Table 9. This means that contiguous repetition of vulnerabilities is not the cause of high precision and recall with 2-grams and there are indeed similar patterns of vulnerabilities across software applications and they can be used to predict vulnerabilities. Also, training on older data have reduced the recall of predicting new patterns (can't predict the vulnerabilities that were not known before) but can still predict the occurrence of known vulnerabilities with high precision.

In Table 13, we show the precision and recall for prediction of vulnerabilities in Linux and Windows operating systems and their components. We predict vulnerabilities in one operating system (e.g., Linux or Windows) and components related to that operating system by using the prior vulnerabilities in the same operating system and associated components. We again performed two experiments by using the vulnerability patterns from 2013 to 2014 as a test-set with vulnerability patterns from 2009 to 2012 as training-set and patterns from 2012 to 2014 as a test-set and patterns from 2009 to 2011 as a training-set for both Windows and Linux separately. It can be seen that the trend of precision and recall is similar to the trend shown earlier for vulnerabilities of all types of software application—i.e., 2-grams are the best predictor of next vulnerability. In a similar manner, we also show the prediction of vulnerabilities in individual applications by using the prior vulnerability patterns in exactly the same application in Table 14. The precision and recall in this case is low implying that the same applications mostly do not experience the same vulnerability pattern again.

The results from Table 8 to Table 14 show that including more historical events in the N-gram model (e.g., increasing the N value), not only did not improve the prediction power of the model, but resulted in a significant decrease in prediction, as shown by precision and recall for $N = 3$, 4, and 5 in Table 13. One of the most interesting finding is that 2-grams yield the best prediction results in all the cases. This shows that the prediction of the next vulnerability only depends on the previous vulnerability and not

Table 13
Predicting next vulnerabilities for only Linux and Windows OS and their components, when test-set spans "2013–2014" years and "2012–2014" years.

N-grams type	Initial vulnerability search	Linux (2013–2014)		Windows (2013–2014)		Linux (2012–2014)		Windows (2012–2014)	
		Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision
2-grams	1	0.80	0.70	0.85	0.45	0.78	0.80	0.84	0.52
3-grams	2	0.56	0.48	0.73	0.34	0.48	0.54	0.73	0.37
4-grams	3	0.16	0.31	0.29	0.23	0.13	0.33	0.31	0.26
5-grams	4	0.04	0.27	0.09	0.24	0.05	0.30	0.09	0.23

Table 14
Predicting next vulnerabilities by using the training-set and test-set of patterns of the same product.

N-grams type	Initial vulnerability search	Test-set duration: 2013–2014		Test-set duration: 2012–2014	
		Recall	Precision	Recall	Precision
2-grams	1	0.51	0.65	0.45	0.73
3-grams	2	0.34	0.55	0.27	0.60
4-grams	3	0.15	0.44	0.11	0.47
5-grams	4	0.05	0.41	0.04	0.43

on a sequence of historical vulnerabilities, confirming that the sequential pattern of vulnerability events follows a first order Markov property. A first order Markov chain is a stochastic model for a sequence of events in which the probability of the next event is dependent only on the present event (Stroock, 2005). It seems that the temporal correlation between vulnerabilities diminishes quickly over time, which may be explained by the high speed at which the attacks continue to evolve. These findings may have important practical implications because by only monitoring the recent history of vulnerabilities, we should be able to predict future vulnerabilities with high precision. This would help mitigate future threats and zero-day attacks.

Another interesting finding is that the best precision and recall is obtained in the cases of Tables 8–12, when prior vulnerability patterns from all the applications are used to predict unknown vulnerabilities in an application. The reason lies in the vulnerability types (CWE), which are general and mostly applicable to all the software applications (Neuhaus and Zimmermann, 2010), making it possible to predict the next vulnerability accurately in an application.

In this section, we conclude that the sequential pattern of vulnerability events possess a first order Markov property; i.e., 2-gram patterns of historical vulnerabilities can be used to predict the next vulnerability in an application with approximately 90% precision and approximately 80% recall. There will be vulnerability patterns that may not be predicted because of their absence in the historical set of 2-grams due to which the recall and precision may not be 100%. This answers our research question (RQ4).

5. Threats to validity

In this section, we outline the threats to the validity of our analysis in the form of four categories: construct validity, conclusion validity, internal validity and external validity (Wohlin et al., 2002).

A threat to construct validity may exist due to the choice of attributes from NVD for our analysis. This paper looks at software vulnerability types and software applications to address the specific research questions discussed in the introductory section. The paper ignores other attributes present in the vulnerability information. It is possible that other information attributes, such as vulnerability severity and CVSS score, may change our findings of vulnerability trends over a period of time. However, we have used all the valid information in NVD for our analysis and we consider using the remaining information as part of our future work.

A threat to conclusion validity may exist because we assumed that every vulnerability corresponds to a possible attack via an exploit. Though, there exists a theoretical possibility that attacks exist for every such vulnerability reported in NVD, the actual exploits might not exist for some of these vulnerabilities. This threat is mitigated by the fact that the majority of vulnerabilities have possible exploits.

A threat to internal validity may arise due to the lack of available information about vulnerabilities. Some vulnerabilities (CVE identifiers) lack information about the vulnerability type (CWE identifier). In these cases, we have ignored such vulnerabilities. This may have skewed statistics of our empirical analysis. However, such cases have been rare and do not affect the overall conclusions. Additionally, vulnerability types that have directly been used from NVD are large in number, error free and adequately informative across all six years.

The number of representative vulnerabilities that have been selected may be a threat of external validity. We selected a total of 25,915 vulnerabilities over six years for our analysis. While this forms a large representative subset of the total number of vulnerabilities over the past six years, the numbers and percentages of vulnerabilities may have minor distortions as vulnerabilities from earlier years beyond the last six years.

6. Conclusion and future work

Everyday new software attacks are emerging and they continue to threaten the security of software systems, despite the development of new techniques and security mechanisms. In this paper, we mine the patterns and trends of vulnerabilities in an attempt to facilitate vendors in making proactive decisions about the occurrence of vulnerabilities in software applications. In particular, we addressed the following novel research questions. (RQ1) How significantly is the trend of software vulnerabilities increasing or decreasing over time? (RQ2) What is the trend of vulnerabilities over time within a software application? (RQ3) What are the common patterns of software vulnerabilities across different software applications? (RQ4) How can we predict the type of vulnerability in a software application?

We use National Vulnerability Database (NVD), supported by US government, to mine the last six years of software vulnerabilities from 2009 to 2014. The most interesting findings of our analysis is that the sequential patterns of vulnerability events follow the first order Markov property; i.e., the next vulnerability can be best predicted by the previous vulnerability and the next vulnerability is not dependent on historical sequences of vulnerabilities. We found that the next vulnerability can be predicted with approximately 90% precision and 80% recall by using the previous vulnerability. Another interesting finding is that collectively mobile applications have higher vulnerabilities than traditional software applications but individually traditional software applications have higher vulnerabilities than the mobile applications in a year. This implies that lesser professional developers for many mobile applications compared against the developers of large firms for traditional mature applications are not the only cause of more vulnerabilities in mobile applications. The vulnerabilities in an application may be due to its large code base, its popularity and other reasons that we have yet to uncover.

Our other results show that no significant difference exists in trend of vulnerabilities overall and in individual applications. The trend of any type of vulnerability in software applications cannot be considered significantly increasing or significantly decreasing despite the rise and fall of trend of vulnerabilities in different years. Our results also show that sequences of similar vulnerability patterns of buffer errors can occur more than 150 times in a software product.

Using the answers of our research question, software vendor can improve the security of their systems. For example, they can focus on employing additional measures for those vulnerabilities that are expected to occur because of frequent pattern of vulnerabilities. If the trend of vulnerabilities is not significantly changing in vendors' applications, then they need to further enhance the software quality (e.g., testing, code review, etc.) process for specific vulnerabilities.

NVD contains a rich repository of information about software vulnerabilities in most publicly known software applications. We did not consider all the fields present in the repository, such as description of vulnerability, vulnerability metrics, vulnerability score, etc. In future, we plan to utilize the additional characteristics of vulnerabilities in our analysis of vulnerabilities and their patterns.

References

Barlowe, B., Blackbird, J., et al. The Evolution of Malware and the Threat Landscape—a 10 Year Review, Microsoft Security Intelligence Report: Special Edition, 2012. Available online: download.microsoft.com.

Barua, A., Stephen, W.T., 2014. A Hassan, Ahmed E., H. What are developers talking about? An analysis of topics and trends in Stack Overflow. Journal of Empirical Softw. Eng. 9 (13), 1382–3256.

Blei, D., Ng, A., Jordan, M., 2003. Latent dirichlet allocation. J. Mach. Learn. Res. 3, 993–1022.

Check Point Ltd., 2015 Security Report, Check Point Technologies Ltd., June 2015 [Online]: https://www.checkpoint.com/resources/2015securityreport/.

Christey, S. Martin, R. A 2007 report on vulnerability type distribution in CVE, 2007, Available online at: http://cwe.mitre.org/documents/vuln-trends.html.

Cox, D.R., Stuart, A., 1955. Some quick sign tests for trend in location and dispersion. Biometrika 42 (1–2), 80–95.

Frei, S., May, M., Fiedler, U., Plattner, B., 2006. Large-scale vulnerability analysis. In: Proceedings of the 2006 SIGCOMM Workshop on Large-scale attack defense (LSAD), pp. 131–138.

Houmb, S.H., Nunes Leal Franqueira, V., 2009. Estimating ToE risk level using CVSS. In: Proceedings of the 4th International Conference on Availability, Reliability and Security, pp. 718–725.

Khreich, W., Granger, E., Miri, A., Sabourin, R., 2010. Iterative Boolean combination of classifiers in the ROC space: an application to anomaly detection with HMMs. Pattern Recogn. J. 43 (8), 2732–2752.

Murtaza, S.S., Hamou-Lhadj, A., Couture, M., Khreich, W., 2014. TotalADS: automated software anomaly detection system. In: Proceedings of the 14th International Working Conference on Source Code Analysis and Manipulatino, 2014, pp. 83–88.

Murtaza, S.S., Khreich, W., Hamou-Lhadj, A., Couture, M., 2013. A host-based anomaly detection approach by representing system calls as states of kernel modules. In: Proceedings of 24th International Symposium on Software Reliability Engineering (ISSRE), pp. 431–440.

Neuhaus, S., Zimmermann, T., 2010. Security trend analysis with CVE topic models. In: Proceedings of the 21st International Symposium on Software Reliability Engineering (ISSRE), 111, p. 120.

Rozenfeld, M., 2015. Special Report Cyber Security Thwarting Attacks, 39. The Institute Magazine, pp. 7–9.

Stroock, W.D., 2005. An Introduction to Markov Processes. Springer, Berlin Heidelberg.

Symantec 2013 trends, April 2014. Internet Security Threat Report, 19.

Wijayasekara, D., Manic, M., McQueen, M., 2013. Information gain based dimensionality selection for classifying text documents. In: Proceedings of IEEE Congress on Evolutionary Computation, pp. 440–445.

Wijayasekara, D., Manic, M., McQueen, M., 2014. Vulnerability identification and classification via text mining bug databases. In: Proceedings of 40th Annual Conference of the IEEE Industrial Electronics Society, pp. 3612–3618.

Wijayasekara, D., Manic, M., Wright, J.L., McQueen, M., 2012. Mining bug databases for unidentified software vulnerabilities. In: Proceedings of the 5th International. Conference on Human System Interaction, 2012.

Witten, I.H., Frank, E., 2005. Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann, San Francisco, USA.

Wohlin, C., Runeson, P., Host, M., Ohlsson, M.C., Regnell, B., Wesslen, A., 2002. Experimentation in Software Engineering: An Introduction. Springer, Berlin.

Zaman, S., Adams, B., Hassan, A.E., 2011. Security versus performance bugs: a case study on Firefox. In: Proceedings of the 8th Working Conference on Mining Software Repositories, pp. 93–102.

Dr. Syed Shariyar Murtaza received his Ph.D. from the University of Western Ontario, Canada, in 2011. He contributed this work as a postdoctoral researcher at Concordia University and Ryerson University. In the past, he has received Ontario Graduate Scholarship and worked with several known research and industrial organizations. He is currently associated with software security industry in Canada and is a member of Canada's big data consortium. He is also fond of teaching and teaches big data analytics at Ryerson University. He specializes in the applications of statistics and machine learning in software engineering, software security and healthcare.

Dr. Wael Khreich received his PhD in Engineering from the École de technologie supérieure (ÉTS), Montreal, in 2011. He contributed to this work while he was postdoctoral fellow at the department of Computer and Electrical Engineering at Concordia University, Montreal, working on adaptive anomaly detection systems in collaboration with Defense R & D Canada (DRDC) Valcartier. He also received an Industrial R&D Fellowship (IRDF) from the Natural Sciences and Engineering Research Council of Canada (NSERC) for 2011–2013. His research interests include computer security, machine learning and natural language processing.

Dr. Abdelwahab Hamou-Lhadj is an associate professor in the Department of Electrical and Computer Engineering at Concordia University, Montreal. His research interests are software engineering, software logging and tracing, big data analytics of machine data, software maintenance and evolution, and anomaly detection systems. Hamou-Lhadj received a PhD in computer science from Ottawa University. Contact him at abdelw@ece.concordia.ca.

Dr. Ayse Basar Bener is a professor and the director of Data Science Laboratory (DSL) in the Department of Mechanical and Industrial Engineering, Ryerson University. She is the director of Big Data in the Office of Provost and Vice President Academic at Ryerson University. She is a faculty research fellow of IBM Toronto Labs Centre for Advance Studies, and affiliate research scientist in St. Michael's Hospital in Toronto. Her current research focus is big data applications to tackle the problem of decision-making under uncertainty by using machine learning methods and graph theory to analyze complex structures in big data to build recommender systems and predictive models. She is a member of AAAI, INFORMS, AIS, and senior member of IEEE.