Extracting Feature Sequences in Software Vulnerabilities Based on Closed Sequential Pattern Mining

Qunhui Wu
State Key Lab of Software Development Environment, Beijing University of Aeronautics and Astronautics, Beijing, P.R.China
Email: wuqunhui126com@126.com

Shilong Ma
State Key Lab of Software Development Environment, Beijing University of Aeronautics and Astronautics, Beijing, P.R.China

Hao Wang
NARI Group Corporation, State Grid Electric Power Research Institute, Beijing, P.R.China

Abstract—**Feature Extraction is significant for determining security vulnerabilities in software. Mining closed sequential patterns provides complete and condensed information for non-redundant frequent sequences generation. In this paper, we discuss the feature interaction problem and propose an efficient algorithm to extract features in vulnerability sequences. Each closed sequential pattern represents a feature in software vulnerabilities. We explore how to efficiently maintain closed sequential patterns in vulnerability sequences. A compact structure *WClosedTree* is designed to keep closed sequential patterns, and its nice properties are carefully studied. Two main pruning strategies, backwards super pattern condition and equivalent position information condition, are developed to remove frequent but non-closed sequential patterns in *WClosedTree*. During the process of maintaining *WClosedTree*, the weight metric of each feature sequence is calculated to better meet the needs of decision makers. Thus, the proposed algorithm can efficiently extract features from vulnerability sequences. The experimental results show that the proposed algorithm significantly improves the runtime efficiency for mining closed sequential patterns, and the feature interaction framework implements feature extraction in software vulnerabilities.**

Index Terms—**feature extraction, closed sequential pattern, vulnerability sequence, software vulnerability**

I. Introduction

With the ever-increasing number of hacker attacks and the spread of the worms on the Internet, information security now becomes the hot topic within both research and industry fields. The software vulnerabilities existing in the computer systems are the most important core issue of information security, which a malicious attacker can use to enhance competence, visit unauthorized resources, or even destroy sensitive data. So analyzing software vulnerabilities is critical for information security. Otherwise, feature extraction is an important element when analyzing software vulnerabilities [1, 2, 3, 4].

In order to make the complexity of modern software systems manageable their functionality is increasingly being decomposed into features. Feature extraction is an effective method for fault diagnosis. M. Pistoia [5] described static analysis methods to identify security vulnerabilities in software systems, which summarized security features, libraries and interfaces among various typical systems. A. Nhlabatsi [6] concluded the feature interaction problem and its possible implications for security requirements. In process planning (*MASCAPP*) systems [7], A. Nassehi discussed some simple types of feature interactions, though *STEP-NC* data are not considered in its final process plan. Moreover, feature interactions are considered in two different categories, geometric and technological interactions in Liu and Wang's work [8]. Otherwise, the feature interaction method was presented in both 3D and feature interaction graph format in a developed software system [9]. G. C. Rui [10] generated hypothetical features for feature interaction detection, which is supported by sets of all possible events, predicates and inconsistent behaviors.

Vulnerability is the weakness in a software system that can be exploited to compromise system's security. The mechanism of extracting the feature sequence of the software vulnerability can be described as a process. First, vulnerability models are loaded into memory and the data structures of these models are established. Second, operation sequences of vulnerability models are extracted through abstracting software vulnerabilities and filtering independent operations. Finally, feature sequences are extracted according to the information of operation sequences. In this paper, when extracting operation sequences of vulnerability models, each program module is defined as fundamental function units. Consequently, the operation sequences of program modules are considered as the process of invoking software vulnerabilities. An operation sequence is denoted formally as a sequence of strings.

According to the above presentation, we introduce the method of mining closed sequential pattern to extract features of software vulnerabilities. So mining closed sequential patterns becomes a crucial task in feature sequences extraction. There exist many algorithms for mining closed sequential pattern, such as *CloSpan* [11], *BIDE* [12], *CMP-Miner* [13], *BIDE-Margin* [14], *Stream_FCI*[15], $IWFP_{WA}$ and $IWFP_{FD}$ [16] and so on. In *CloSpan*, it first generates a set of closed candidates stored in a hash indexed tree and then performs post-pruning on it. Because *CloSpan* needs to maintain the set of closed candidates, it will consume a large amount of memory for closure checking. In *BIDE*, this mines closed sequential patterns without maintaining candidates, which adopts forward and backward directional extension checking to perform closure checking and to prune the redundant patterns in the mining process. The *CMP-Miner* enumerates the frequent patterns by a frequent pattern tree in a depth-first search manner. In addition, *CMP-Miner* adopts closure checking and pruning strategies to accelerate the mining closed patterns process in a multi-sequence time-series database. In *BIDE-Margin*, a new constraint is presented for reducing the output of sequential pattern mining, the changes are required to enforce margin-closeness in *BIDE*, the flag margin-check is used in back-scan function instead of closed-check. *Stream_FCI* detects the frequent closed itemsets in each sliding window using a DFP-tree with a head table, and adopts a table to store the frequent closed itemsets so as to avoid the time-consuming. $IWFP_{WA}$ and $IWFP_{FD}$ are the incremental and interactive *WFP* mining using a single database scan. When a database is updated or a minimum support threshold is changed, $IWFP_{WA}$ and $IWFP_{FD}$ are effective for incremental and interactive mining to utilize the current tree structure and to use the previous mining results. To our knowledge, this paper is the first one to apply the closed sequential pattern mining theory to feature extraction in software vulnerabilities

In this paper, we propose an efficient algorithm called *WCSMining* (Weighted Closed Sequential pattern Mining) to extract features in vulnerability sequences where each closed sequential pattern represents a feature. To maintain closed sequential patterns incrementally, *WClosedTree* is proposed to keep the closed sequential patterns and other auxiliary information, and the nice properties of *WClosedTree* are investigated. Moreover, we apply two closure checking strategies to update node information during the mining process, and the weight metric of each feature sequence is calculated to meet the needs of decision makers more closely.

The remaining of the paper is organized as follows. Section II describes the problem definitions. The feature extraction model and our proposed algorithm are discussed in detail in Section III. Section IV illustrates experimental results and performance analysis. Finally conclusion will be given in Section V.

II. PROBLEM DEFINITION

In order to better extract feature sequence, the software vulnerabilities are denoted formally as sequences of strings as follows: Let $L = \{l_1, l_2, ..., l_m\}$ be a set of literals called items. A *sequence* $S = <a_1, a_2, ..., a_n>$ is an orderly list of items, such that each item $a_i \in L$. Let the *sequence*s *database SDB* be a set of operation sequences.

In a *sequence S*, each item a_i represents a fundamental function operation of software vulnerability. *Sequence S* stands for an operation sequence of vulnerability model, and the *sequence*s *database SDB* = $\{S_1, S_2, ..., S_m\}$ is a set of all operation sequences. Each operation sequence in *sequence*s *database SDB* includes a sequence identifier (*Sid*) and a vulnerability sequence.

We say that the *sequence* $S_1 = <a_1, a_2, ..., a_n>$ supports the *sequence* $S_2 = <b_1, b_2, ..., b_m>$ if there exist integers $1=i_1<i_2<…<i_n=m$, such that $a_1 = b_{i1}$, $a_2 = b_{i2}$, …, $a_n = b_{in}$, then S_1 is called a *sub-sequence* of the *sequence* S_2 or S_2 is *super-sequence* of *sequence* S_1. The support of *sequence* S_i is denoted by $sup(S_i)$, it is the number of sequence in *SDB* that contain S_i. Given a support threshold *min-sup*, a *sequence* S_i is a frequent sequence in *SDB*, then $sup(S_i) \geq min\text{-}sup$. If a *sequence* S_i is frequent and there exists no proper *super-sequence* with the same support, we call it a frequent closed sequence.

Definition 1. Item position (*IPos*). The positional information of each item a_i in vulnerability sequence $S= <a_1, a_2, ..., a_n>$ is denoted as *IPos* (*Sid*, *itemID*), where *Sid* is the sequence identifier, *itemID* is the position identifier.

Definition 2. Backwards super pattern. Given two *sequence*s S_1 and S_2, if S_2 is a *super-sequence* of S_1, i.e. S_1 is generated before S_2, otherwise S_1 and S_2 have the same hash key and the last item, then S_2 is backwards super pattern of S_1, where hash key is composed of the support, the total number of *Sid*s and the total number of *itemID*s.

Definition 3. Equivalent position information. Given two *sequence*s S_1 and S_2, except for the same hash key and the last item, and there exists no inclusion relation between S_1 and S_2, then S_1 and S_2 have the equivalent position information.

Definition 4. The weight for *item* a_i is defined as

$$weight(a_i) = -\sum_{i=1}^{Sid_{total}} \left[\frac{weight(S_i)*|a_i|}{\sum_{j=1}^{i}\left(weight(S_i)*|a_i|\right)} * log \frac{weight(S_i)*|a_i|}{\sum_{j=1}^{i}\left(weight(S_i)*|a_i|\right)} \right].$$

The details for these entries are as follows:

$|a_i|$ is the number of *item* a_i in *sequences* S_i. $\{a_1, a_2, ..., a_m\}$ is a set of items, each *item* a_i of *sequences* $S_i \in \{a_1, a_2, ..., a_m\}$.

$weight(S_i)$ is the weight of *sequence* S_i, where $weight(S_i) = \sum_{a_i \in S_i} weight(a_i)$. The greater the value of $weight(S_i)$ is, the greater the representation degree is for S_i.

Sid_{total} is the total number of *sequences* S_i in *sequence*s *database SDB*.

TABLE I.
SEQUENCE DATABASE *SDB* OF SOFTWARE VULNERABILITIES

Sid	vulnerability sequence	weight of sequence S_{Sid}
S_1	*C A A B C D*	0.5
S_2	*A B C B D*	0.2
S_3	*C A B C*	0.2
S_4	*A B B C A*	0.1

Definition 5. Each node of multi-branches tree *WClosedTree* is defined as a tuple, *Tnode* = [a_i, *sup*(S_i), *weight*(S_i)]. The details for these entries are as follows:

a_i is an item which constitutes prefix of closed sequence.

sup(S_i) is denoted as the corresponding closed sequence in the frequency of *sequences database SDB*.

weight(S_i) is the weight of closed sequential pattern, where S_i is denoted as the path from root node to the current node, if *sup*(S_i) is greater than 0, S_i is a weighted closed sequential pattern.

TABLE II.
POSITION INFORMATION OF ITEMS IN SEQUENCE DATABASE *SDB*

item	position information	weight of item
<*A*>	(1,1) (1,2) (2,0) (3,1) (4,0) (4,4)	0.467
<*B*>	(1,3) (2,1) (2,3) (3,2) (4,1) (4,2)	0.559
<*C*>	(1,0) (1,4) (2,2) (3,0) (3,3) (4,3)	0.465
<*D*>	(1,5) (2,4)	0.259

Example 1. Assume that software vulnerabilities have been denoted formally as sequences of strings, given sequence identifier *Sid* to each vulnerability sequence, these vulnerability sequences are deposited in the *sequences database SDB*, as shown in Table I. The position information *IPos* of each item in *sequences database SDB* are as shown in Table II. In *sequence* S_1, *item C* appears at position 0, so its position information is (1, 0), position information for other items can be obtained in the same way. Otherwise, according to definition 4, the corresponding weights for items are shown in the third column of Table II.

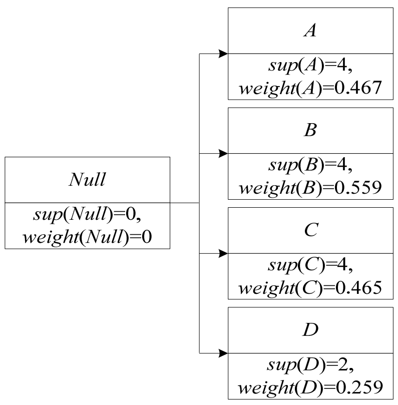

A
sup(A)=4, weight(A)=0.467
B
sup(B)=4, weight(B)=0.559
Null
sup(Null)=0, weight(Null)=0
C
sup(C)=4, weight(C)=0.465
D
sup(D)=2, weight(D)=0.259

Figure 1. *WClosedTree* with 1-*fresequences*.

Frequent sequences of length one (1-*fresequences*) are inserted into *WClosedTree* as shown in Fig. 1. 1-*fresequences* can be calculated simply by comparing the support threshold *min-sup* with the number of position information for sequences of length one (1-*sequence*), where *min-sup* is set to 2, *min-sup* is a user specified support threshold.

III. DESIGN OF WEIGHTED CLOSED SEQUENTIAL PATTERN MINING ALGORITHM

In this section, we first discuss the method *WCSMining* (Weighted Closed Sequential pattern Mining) which is used to extract features in vulnerability sequences. Then an example is given to demonstrate the execution process of our method.

A. The Description of Algorithm

For the process of mining weighed closed sequential patterns in our method, firstly, the weighted measure factor and the positional information are introduced to calculate each item in vulnerability sequences. After obtaining the frequent sequences of length one (1-*fresequences*), frequent sequences of length two (2-*fresequences*) and corresponding position information can be generated simply through matching the position information of 1-*fresequences* with each other. Then 2-*fresequences* are implemented to closure-check whether conditions for pruning strategies are not met. Finally, we utilize the above procedure to generate weighted closed sequences, whose lengths are gradually increased.

```
Algorithm: WCSMining
Input: SDS = {S_1, S_2, ..., S_m}: the vulnerability sequences database
       weight (S_i): weight of sequence S_i
       min-sup: the threshold for the minimal support
       WClosedTree: the null tree for weighted closed sequential
                    tree
Output: the weighted closed sequential patterns
  1:  scan sequences database SDS once, calculate the weight of
      each item a_i in SDS according to definition 4;
  2:  calculate the position information IPos of each item a_i;
  3:  obtain frequent sequences of length one 1-fresequences, and
      insert 1-fresequences into WClosedTree;
  4:  for (each 1-fresequence <a_i> of 1-fresequences) {
  5:     match IPo of <a_i> with each IPos of 1-fresequences, and
         generate 2-fresequences and the corresponding IPos;
  6:     perform the closure check, and insert 2-fresequences into
         WClosedTree;
  7:  }
  8:  set null sets P and P';
  9:  for (each 2-fresequence <a_i, a_j> of 2-fresequences) {
  10:    for (each 1-fresequence <a_k> of 2-fresequence <a_i, a_j>) {
  11:       if (exist backwards super pattern || exist same positional
            information))
  12:            set sup(<a_k>) = 0 in WClosedTree;
  13:       else
  14:            add <a_i, a_j> to set P;
  15:    }
  16: }
  17: if (P = null)
  18:     Exit;
  19: else {
  20:     for (each sequence S_j of P) {
  21:        match S_j with each of the 1-fresequences, and generate
             sequence set P_S;
  22:        for (each sequence S_k of P_S) {
  23:            if (S_k is frequent)
24:         insert $S_k$ into WClosedTree;
25:       else
26:         delete $S_k$ from $P_S$;
27:       if (exist backwards super pattern || exist same positi-
          onal information))
28:         update node information and the connections;
29:         add $S_k$ to P';
30:     }
31:     P' = P, P' = ∅ ;
32:   }
33: }
34: implement depth first search WClosedTree, identify
    sequences that supports are not 0
35: Exit.
```

Figure 2. The Weighted Closed Sequential pattern Mining method.

Fig. 2 shows the algorithm to compute the complete set of weighted closed sequences. In our algorithm, the sequence information is completely reserved by position information. Sequence connection $IPo_i \Diamond IPo_j$ is proposed to match the position information between IPo_i and IPo_j, i.e., if $Sid_i = Sid_j$ and $itemID_i \leq itemID_j$, then the position information of $IPo_i \Diamond IPo_j = (Sid_j, itemID_j)$, where $IPo_i = (Sid_i, itemID_i)$ and $IPo_j = (Sid_j, itemID_j)$ denotes as position information of *item* a_i and a_j respectively. For example, position information of *item A* is (1, 1), position information of *item B* is (1, 3), then position information of sequence S = <*A*, *B*> is $IPo_S = IPo_A \Diamond IPo_B$ = (1, 3). The process of performing the sequence connects in such a way that the position information for sequences is simply calculated. Backwards super pattern and same positional information are utilized to implement the closure check, if two sequences have the same positional information, they also have the same projection database. We adopt the *WClosedTree* structure to store closed sequences discovered so far, and detection of backwards super pattern technology to accelerate the closure check. Otherwise, the measurement of weight for each closed sequence ensures that feature sequences with comparatively great values are more valuable to future speculation.

B. The Example of Algorithm

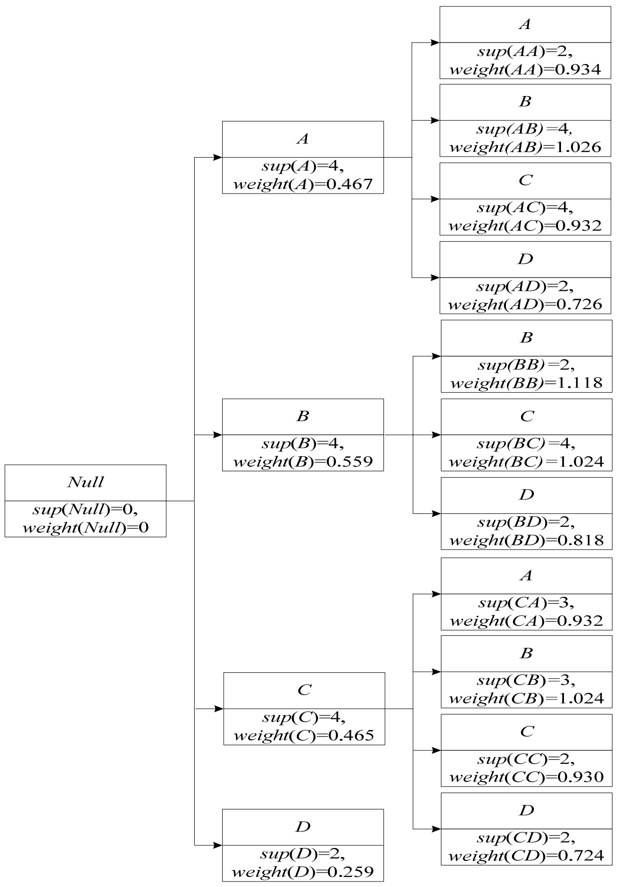
Given an example to illustrate the algorithm *WCSMining*, sequences and their corresponding identifiers *Sid*s in *sequence*s *database SDB* are shown in Table I. *SDB* has totally 4 unique items. The position information *IPos* of each item in *sequence*s *database SDB* are as shown in Table II. Suppose that *min-sup* is set to 2, the whole set of 1-*fresequence*s and their weights are {<*A*>: (4, 0.467); <*B*>: (4, 0.559); <*C*>: (4, 0.465); <*D*>: (4, 0.259)}, which are inserted into *WClosedTree* as shown in Fig. 1.

By employing the sequence connection, position information is matched between 1-*fresequence*s. So 2-*fresequence*s, positional information and their corresponding weights are generated and shown in Table III. The whole set of 2-*fresequence*s consist of eleven sequences, that is {<*A*, *A* >: (2, 0.934); <*A*, *B*>: (4, 1.026); <*A*, *C*>: (4, 0.932); <*A*, *D*>: (2, 0.726); <*B*, *B*>: (2, 1.118); <*B*, *C*>: (4, 1.024); <*B*, *D*>: (2, 0.818); <*C*, *A*>: (3, 0.932);

TABLE III.
POSITION INFORMATION OF 2-*FRESEQUENCES*

2-*fresequence*	position information	weight of 2-*fresequence*
<*A*, *A*>	(1,2) (4,4)	0.934
<*A*, *B*>	(1,3) (2,3) (3,2) (4,2)	1.026
<*A*, *C*>	(1,4) (2,2) (3,3) (4,3)	0.932
<*A*, *D*>	(1,5) (2,4)	0.726
<*B*, *B*>	(2,3) (4,2)	1.118
<*B*, *C*>	(1,4) (2,2) (3,3) (4,3)	1.024
<*B*, *D*>	(1,5) (2,4)	0.818
<*C*, *A*>	(1,2) (3,1) (4,4)	0.932
<*C*, *B*>	(1,3) (2,3) (3,2)	1.024
<*C*, *C*>	(1,4) (3,3)	0.930
<*C*, *D*>	(1,5) (2,4)	0.724

<*C*, *B*>: (3, 1.024); <*C*, *C*>: (2, 0.930); <*C*, *D*>: (2, 0.724)}. Through referring to Table III, performing the closure check, we insert the closed 2-*fresequence*s, their support and their corresponding weight into *WClosedTree* structure as nodes in Fig. 3. For each 2-*fresequence* in *WClosedTree*, we implement the backwards super pattern and same positional information checks. If there exists any pruning strategies, the node information is updated, i.e. *sequences* {<*A*>: (4, 0.467); <*B*>: (4, 0.559); <*C*>: (4,



Null
sup(Null)=0, weight(Null)=0
A
sup(A)=4, weight(A)=0.467
B
sup(B)=4, weight(B)=0.559
C
sup(C)=4, weight(C)=0.465
D
sup(D)=2, weight(D)=0.259
A
sup(AA)=2, weight(AA)=0.934
B
sup(AB)=4, weight(AB)=1.026
C
sup(AC)=4, weight(AC)=0.932
D
sup(AD)=2, weight(AD)=0.726
B
sup(BB)=2, weight(BB)=1.118
C
sup(BC)=4, weight(BC)=1.024
D
sup(BD)=2, weight(BD)=0.818
A
sup(CA)=3, weight(CA)=0.932
B
sup(CB)=3, weight(CB)=1.024
C
sup(CC)=2, weight(CC)=0.930
D
sup(CD)=2, weight(CD)=0.724

Figure 3. *WClosedTree* with 2-*fresequences*.

0.465)} in *WClosedTree* are updated with set support to 0, which is denoted as the corresponding sequences are not closed, as shown in Fig. 4.

Fig. 5 shows the final *WClosedTree* structure, the complete set of weighted closed sequential patterns is obtained, that is {<*A*, *A* >: (2, 0.934); <*A*, *B*, *B*>: (2, 1.585); <*A*, *B*, *C*>: (4, 1.491); <*A*, *B*, *C*, *D*>: (2, 1.750); <*C*, *A*>: (3, 0.932); <*C*, *A*, *B*, *C*>: (2, 1.956); <*C*, *B*>: (3, 1.024)} through implementing depth first search *WClosedTree* structure and identifying the sequences whose supports are not 0.

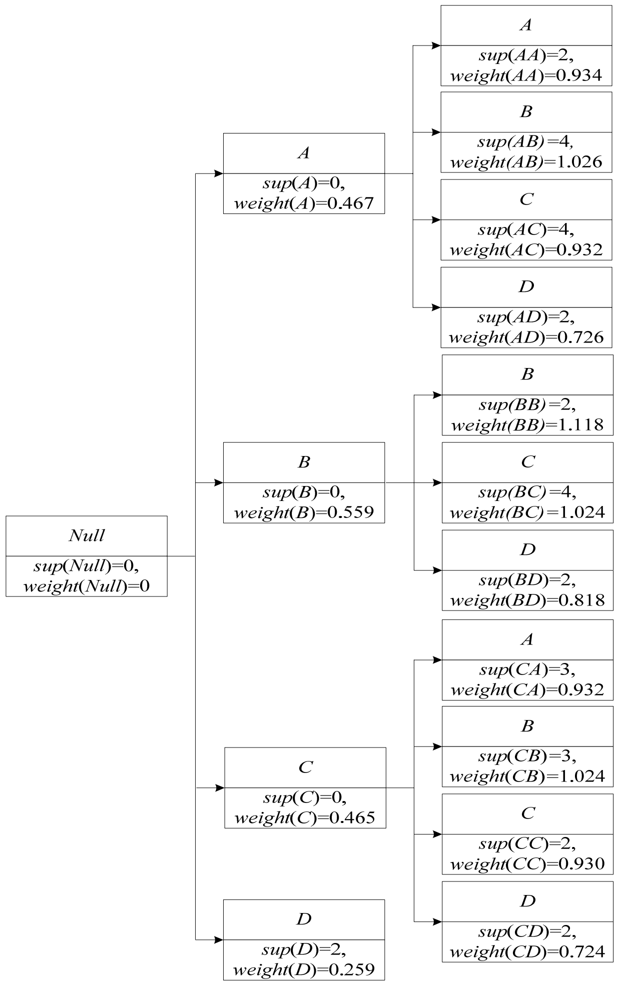

Null
sup(Null)=0, weight(Null)=0
A
sup(A)=0, weight(A)=0.467
B
sup(B)=0, weight(B)=0.559
C
sup(C)=0, weight(C)=0.465
D
sup(D)=2, weight(D)=0.259
A
sup(AA)=2, weight(AA)=0.934
B
sup(AB)=4, weight(AB)=1.026
C
sup(AC)=4, weight(AC)=0.932
D
sup(AD)=2, weight(AD)=0.726
B
sup(BB)=2, weight(BB)=1.118
C
sup(BC)=4, weight(BC)=1.024
D
sup(BD)=2, weight(BD)=0.818
A
sup(CA)=3, weight(CA)=0.932
B
sup(CB)=3, weight(CB)=1.024
C
sup(CC)=2, weight(CC)=0.930
D
sup(CD)=2, weight(CD)=0.724

Figure 4. *WClosedTree* with 2-*fresequences* after updating operation.

C. The Process to Extract Feature Sequences in Software Vunerabilities

The process of extracting feature sequences in software vulnerabilities is shown in Fig.6. The function of each module is explained as follows:

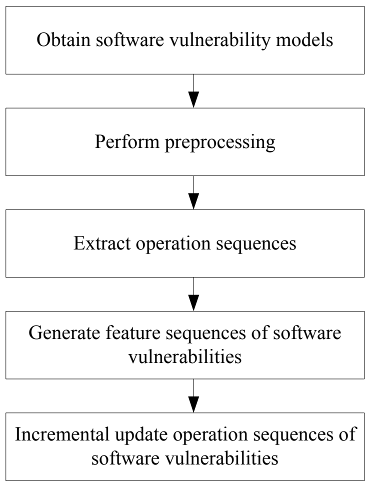

Obtain software vulnerability models
Perform preprocessing
Extract operation sequences
Generate feature sequences of software vulnerabilities
Incremental update operation sequences of software vulnerabilities

Figure 6. *WClosedTree* with final weighted closed sequences.

a. Obtain software vulnerability models

The existing software vulnerability models are collected and loaded in memory, which sent to next step for further treatment.

b. Perform preprocessing

The irrespective functions and program sentences are filtrated, e.g., the path of database, the list of the calling functions, a variety of string information and so on.

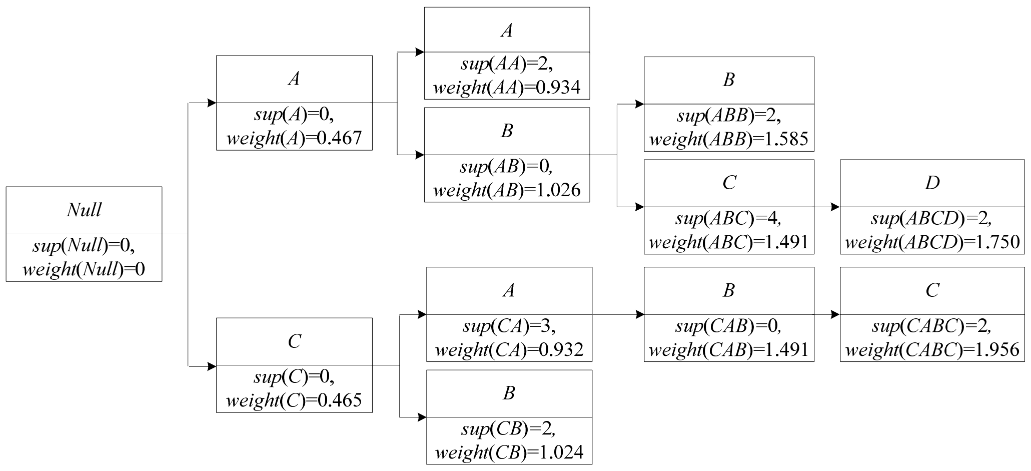

Null
sup(Null)=0, weight(Null)=0
A
sup(A)=0, weight(A)=0.467
C
sup(C)=0, weight(C)=0.465
A
sup(AA)=2, weight(AA)=0.934
B
sup(AB)=0, weight(AB)=1.026
B
sup(ABB)=2, weight(ABB)=1.585
C
sup(ABC)=4, weight(ABC)=1.491
D
sup(ABCD)=2, weight(ABCD)=1.750
A
sup(CA)=3, weight(CA)=0.932
B
sup(CAB)=0, weight(CAB)=1.491
C
sup(CABC)=2, weight(CABC)=1.956
B
sup(CB)=2, weight(CB)=1.024

Figure 5. *WClosedTree* with final weighted closed sequences.

c. Extract operation sequences

After pretreatment processing, the data structures for vulnerability models are established. We utilized the control flow extraction algorithm to process the above models, and the corresponding control flow graphs are converted. Moreover, we extract function relationships in creating medium file, the call graphs can be constructed for vulnerability models. The operation sequences are obtained by carrying on the control flow graphs and calling graphs depth first path traversal. Otherwise, each operation sequence is denoted formally as sequence of strings, which is given a unique identification *Sid* in *sequences database SDB.*

d. Generate feature sequences of software vulnerabilities

We adopt *WCSMining* algorithm proposed in this paper to mine the closed sequential patterns, and each closed sequence as a feature sequence of software vulnerability. In addition, the frequencies for vulnerability models are diverse, i.e. some of software vulnerabilities appear frequently and some appear rarely, we introduce the weight to meet the needs of decision makers more closely.

e. Incremental update operation sequences of software vulnerabilities

When the new operation sequences of software vulnerabilities generate, the new operation sequences are combined to incremental update *sequences database SDB*, based on the discussion of existing operation sequences.

IV. Empirical Results

In this section, we perform a thorough evaluation of *WCSMining* on various kinds of datasets, compared with two closed sequence mining algorithms *CloSpan* and *BIDE*. *CloSpan* and *BIDE* were provided as source code. We perform all the experiments on a 2.8GHz processor computer with 2GB memory and running on Windows XP professional. In addition, all the proposed techniques are implemented in Java.

A. Datasets

Synthetic datasets, the datasets were produced from the well-known IBM synthetic dataset generator (http://www.almaden.ibm.com/cs/quest), which has been used in the evaluation of almost all sequence mining algorithms in data mining literature. The generator simulates a set of user product buying sequences. A user sequence is a list of transactions in the purchase order. The parameters and their value ranges are shown in Table IV. For example, D100C10T5N20 means there are 100K sequences, each sequence has about 10 transactions, each transaction has about five items and the dataset has 20K distinct items.

TABLE IV.
PARAMETERS OF THE DATA GENERATOR

parameter	description	value range
D	Number of sequences in the datasets	10-1000 (*1000)
C	Average number of transactions per customer	5-20
T	Average number of items per transaction	1-10
N	Number of distinct items in the datasets	10-100(*1000)
Other	Other parameters	Default

Gazelle is a sparse dataset, but it contains some very long frequent closed sequences with low support threshold. This dataset was originally provided by Blue Martini Company. It contains totally 29369 customers' Web click-stream data. For each customer there is a corresponding series of page views, and we treat each page view as an event. This dataset contains 29369 sequences (i.e., customers), 87546 events (i.e., page views), and 1423 distinct items (i.e., web pages). The average length and the maximal length of this dataset is 3 and 651, respectively.

In order to test all algorithms, two datasets are employed. One is the *Synthetic datasets*; the other is the *Gazelle* dataset. The sequences in *Synthetic datasets* are generated by the input parameters, and *Gazelle* dataset is randomly selected. When testing *WCSMining*, we use D10C10T2.5N10, D10C15T2.5N10, D10C5-20T2.5N10 and D10C6T1-10N10 as the *Synthetic databases*.

B. Comparison of WCSMining with other Algorithms

Fig. 7 shows the running time of *WCSMining*, *BIDE* and *CloSpan* when *min-sup* is varied from 0.02% to 0.1% on D10C10T2.5N10.

On average, closed sequence mining algorithms are faster than algorithms which mine the complete set of frequent sequences from scratch. In comparison with the closed sequence mining algorithms *BIDE* and *CloSpan*, *WCSMining* is about 4 or more times faster than others. When *min-sup* is low, the gap between *WCSMining* and other algorithms is much more obvious. For instance, with *min-sup* = 0.02%, *WCSMining* completes in 9.27s, while *BIDE* and *CloSpan* completes in 44.29s and 102.55s, respectively. It is because at the extremely low support, there are too many non-closed patterns generated; *WCSMining* can successfully prune the non-closed sequences. Another main reason is that *WCSMining* starts its performance from calculating position information, using the nice properties of *WClosedTree* to extend the nodes and to update the support states of only a few nodes in the *WClosedTree*.

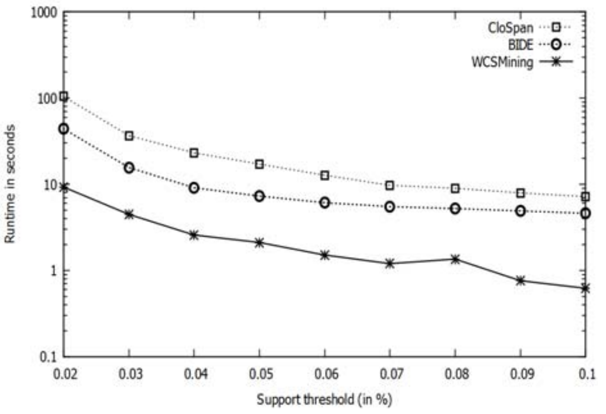


Figure 7. Running time on D10C10T2.5N10.

To test whether the algorithms perform unstably in different runs, we implemented the three algorithms 15 times, and the average running time is illustrated in Fig. 8. It can be shown that the running time of the three algorithms displays little variations in different runs.

The running time of all the algorithms on dataset D10C15T2.5N10 is given in Fig. 9. Fig.9 displays the same trend as Fig.7. The only difference between dataset D10C15T2.5N10 from D10C10T2.5N10 is the average number of transactions per customer. For this reason, the running time of each algorithm in Fig.9 is slightly higher than the running time of each algorithm in Fig.7.

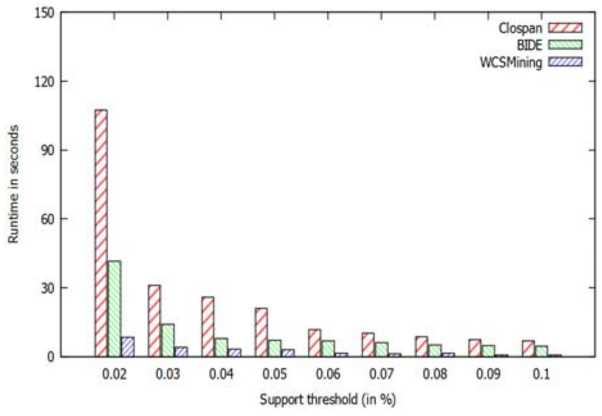


Figure 8. Mean running time on D10C10T2.5N10.

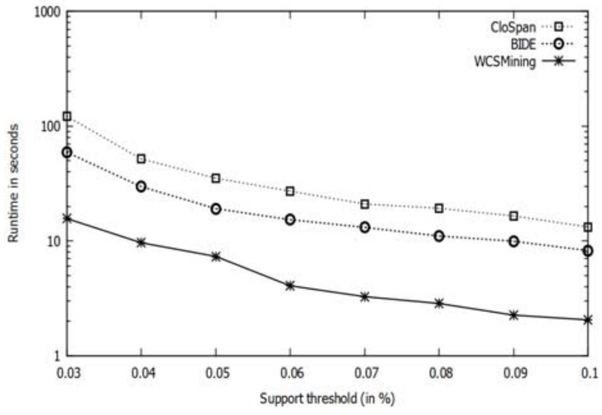


Figure 9. Running time on D10C15T2.5N10.

Fig. 10 and Fig. 11 depict the comparison results among *CloSpan*, *BIDE* and *WCSMining* for *Synthetic datasets* D10C5-20T2.5N10 (the number of transactions per customer is increased from 5 to 20) and D10C6T1-10N10 (the number of items per transaction is increased from 1 to 10), where *min-sup* is fixed at 0.05%. When the number of transactions per customer or the number of items per transaction is increased, the average length of sequences is also increased. Thus, the datasets become much denser and there are more closed sequential patterns. This results in the significant increase of the running time of all the three algorithms. We can see from the figures that *WCSMining* still outperforms *BIDE* and *CloSpan* by a large margin.

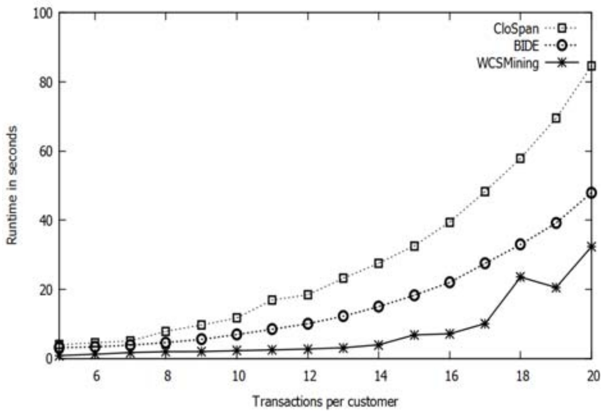


Figure 10. Varying the number of transactions per customer.

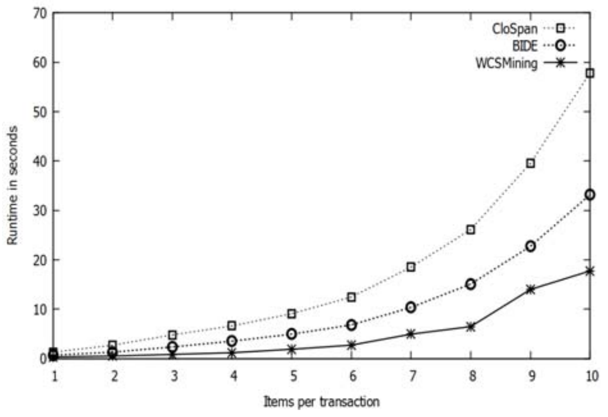


Figure 11. Varying the number of items per transaction.

In Fig. 12, the running time of the three algorithms on the real dataset *Gazelle* is illustrated. The support threshold is varying from 0.04% to 0.02%. The average length of this dataset is shorter than that of *Synthetic datasets*, so it is sparser than *Synthetic datasets*. Form Fig.12 we can see that when the support is greater than 0.025%, *WCSMining* show better performance than *BIDE* and *CloSpan*, especially when we continue to lower the support threshold, *WCSMining* algorithms will outperform a lot the two algorithms. On this sparse dataset, an explosive number of non-closed sequences are generated, *BIDE* and *CloSpan* cannot prune the redundancy sequences in a relatively short time.

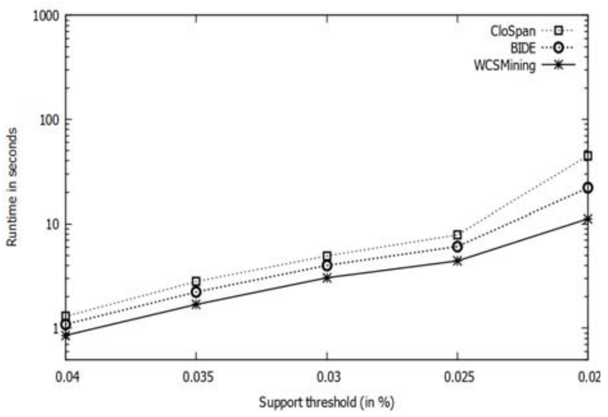


Figure 12. Running time on *Gazelle*.

Overall, the *WCSMining* algorithm runs faster than *BIDE* and *CloSpan* algorithms in five different types of experimental datasets, and there are different levels of improving performance. In other words, the overall performance of *WCSMining* algorithm is significantly reduces the time consumption.

C. Representing Feature Extraction in Software Vulnerabilities

A feature extraction system has been developed based on the above mentioned methodology. Java is the main program development environment. The existing software vulnerability models have been used as the underlying modeling kernel. The filtrate task is considered indispensable as the pretreatment processing to be carried out. Operation sequences are obtained from the data structure for vulnerability models. Alongside the complete closed sequential patterns as feature sequences generated, *WCSMining* also offers the weight of each feature sequence for displaying weightiness. In a word, the method of mining closed sequential patterns is introduced to extract feature sequences in software vulnerabilities is unprecedented and effective.

V. CONCLUSION

The rapid development of Internet and the extensive applications of software in many key domains bring a new challenge in software security, so feature extraction of software vulnerability has aroused a deep concern increasingly. In this paper, closed sequences mining technology is introduced to extract feature of software vulnerability. Aiming at the problem of varying frequencies for software vulnerabilities, we present an effective algorithm *WCSMining* for mining weighted closed sequences as feature sequences. *WCSMining* utilizes position data to reserve sequence information in order to reduce the search space and degraded the time complexity. Moreover, *WClosedTree* is defined to keep the closed sequential patterns and other auxiliary information, and two main pruning strategies are developed to remove frequent but non-closed sequential patterns in *WClosedTree*. In addition, *WCSMining* accelerates the process of extracting features; this method overcomes the disadvantages of including a great deal of repetition works, and improves the efficiency of the whole extractive process.

ACKNOWLEDGMENT

The authors thank Jiadong Ren, who is a PhD candidate supervisor of Yanshan University for his careful review and detailed guidance. The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

[1] R. E. Crossler, A. C. Johnston, P. B. Lowry, "Future directions for behavioral information security research," *Computers & Security*, vol. 32, 2013, pp. 90-101.

[2] N. C. Patterson, M. Hobbs, "Virtual World Security Inspection," *Journal of Networks*, vol. 7, no. 6, 2012, pp. 895-907.

[3] S. Sun, Y. Wang, "Research and application of an improved support vector clustering algorithm on anomaly detection," *Journal of Software*, vol. 5, no. 3, 2010, pp. 328-335.

[4] G. X. Yao, Q. L. Guan, K. B. Ni, "Test Model for Security Vulnerability in Web Controls based on Fuzzing," *Journal of Software*, vol. 7, no. 4, 2012, pp. 773-778.

[5] M. Pistoia, S. Chandra, S. J. Fink, E. Yahav, "A survey of static analysis methods for identifying security vulnerabilities in software systems," *IBM Systems Journal*, vol. 46, no. 2, 2007, pp. 265-288.

[6] A. Nhlabatsi, R. Laney, B. Nuseibeh, "Feature interaction: the security threat from within software systems," *Special issue: The future of software engineering for security and privacy*, Progress in information, no. 5, 2008, pp. 75-89.

[7] A. Nassehi, R. Liu, S. T. Newman. "A new software platform to support feature-based process planning for interoperable STEP-NC," *International Journal of Computer Integrated Manufacturing*, vol. 20, no. 7, 2007, pp. 669-683.

[8] Z. Liu, L. Wang. "Sequencing of interacting prismatic machining features for process planning," *Computers in Industry*, vol. 58, no. 4, 2007, pp.295-303.

[9] D. Tobias, X. Xun, K. Peter, "Defining, recognizing and representing feature interactions in a feature-based data model," *Robotics and Computer-Integrated Manufacturing*, vol. 21, Issue 1, 2011, pp. 101-114.

[10] R. G. Crespo, "Predicting feature interactions by using inconsistency models," *Computer Networks*, vol. 54 Issue 3, 2010, pp. 416-427.

[11] X. Yan, J. Han, R. Afshar, "CloSpan: mining closed sequential patterns in large databases," *In: Proceedings of the SIAM International Conference on Data Mining*, San Francisco, CA, 2003, pp. 166–177.

[12] J. Wang, J. Han, "BIDE: efficient mining of frequent closed sequences," *In: Proceedings of the 20th International Conference on Data Engineering*, 2004, pp. 79–90.

[13] A. J. T. Lee, H. W. Wu, T. Y. Lee, Y. H. Liu, K. T. Chen, "Mining closed patterns in multi-sequence time-series databases," *Data & Knowledge Engineering*, vol. 68, Issue 10, 2009, pp. 1071-1090.

[14] D. Fradkin, F. Moerchen, "Margin-Closed Frequent Sequential Pattern Mining," *In: Proceedings of the ACM SIGKDD Workshop on Useful Patterns*, New York, NY, 2010, pp. 45-54.

[15] K. Tang, C. Dai, L. Chen, "A Novel Strategy for Mining Frequent Closed Itemsets in Data Streams," *Journal of Computers*, vol. 7, no. 7, 2012, pp. 1564-1573.

[16] C. F. Ahmed, S. K. Tanbeer, B. Jeong, "Single-pass incremental and interactive mining for weighted frequent patterns," *Expert Systems with Applications*, vol. 39, Issue 9, 2012, pp. 7976-7994.

Qunhui Wu She is a doctor candidate at state key lab of software development environment, school of computer science and engineering, Beijing University of Aeronautics and Astronautics. She was born in 1984. Her main research interests include data mining and software engineering.

Shilong Ma He is a doctor supervisor at school of computer science and engineering, Beijing University of Aeronautics and Astronautics. He was born in 1953. His main research interests include calculating model on the network environment, dynamic statistic behavior of logic and computation, calculating model of massive data process, grid computing technology and application.

Hao Wang He is a master, who now works at NARI group corporation, State Grid Electric Power Research Institute. He was born in 1978. His main research interests include information security and software engineering.