Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 June 2019 doi:10.20944/preprints201906.0144.v1

Article

Applications of Data Mining Algorithms for Network Security

Kai Chain

Department of Computer and Information Science, Republic of China Military Academy, Kaohsiung 83059, Taiwan; chainkai@mail2000.com.tw

* Correspondence: chainkai@mail2000.com.tw; Tel.: +886-7-743-8179; Fax: +886-7-747-9515

Abstract: Typical modern information systems are required to process copious data. Conventional manual approaches can no longer effectively analyze such massive amounts of data, and thus humans resort to smart techniques and tools to complement human effort. Currently, network security events occur frequently, and generate abundant log and alert files. Processing such vast quantities of data particularly requires smart techniques. This study reviewed several crucial developments of existent data mining algorithms, including those that compile alerts generated by heterogeneous IDSs into scenarios and employ various HMMs to detect complex network attacks. Moreover, sequential pattern mining algorithms were examined to develop multi-step intrusion detection. These studies can focus on applying these algorithms in practical settings to effectively reduce the occurrence of false alerts. This article researched the application of data mining algorithms in network security. The academic community has recently generated numerous studies on this topic.

Keywords: data mining; network security; association rules; DDoS

1. Introduction

In short, data mining can be defined as the computational process of discovering meaningful patterns or rules through the semi-automatic or automatic exploration and analysis of copious data. The most commonly used analytical techniques for general data mining algorithms are cluster analysis, classification analysis, link analysis, and sequence analysis. Cluster analysis is the task of grouping data into multiple categories or clusters so that data within the same cluster share a higher similarity with each other than with data in other clusters. Clustering can be used to identify dense and sparse areas in the data space, thereby revealing the global distribution of the data and the relationships between data characteristics. As the most widely applied data mining technique, classification analysis can be used to model the relationships between crucial data or to predict data trends. It involves classifying new data under predefined categories (e.g., normal and invaded). Two of the most common classification methods are decision trees and rules. In 1993, Agrawal et al. at the IBM Almaden Research Center proposed a problem concerning the association rules between itemsets in a customer transaction database and presented the Apriori algorithm [13]. Their study prompted intensive research on the problem of discovering association rules; this further facilitated the application of data mining analysis techniques in various fields, particularly classification analysis. Sequence analysis is a longitudinal analysis technique that identifies data characteristics by employing temporal variables.

Cluster analysis, link analysis, and sequence analysis are the results of theories and research that involved analysis of small-scale databases. Data mining has now extracted or integrated large quantities of data from several large-scale databases. For instance, Wal-Mart utilizes an effective data warehouse system that enables the provision of clear and detailed sales information to clients and product suppliers; this enables presenting data regarding what products are most required by its clients and when the clients need those products. Most importantly, Wal-Mart applied data

© 2019 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 June 2019 doi:10.20944/preprints201906.0144.v1

mining analyses of customer behavior models to increase the effectiveness of inventory control and customer relationship management. The application of data mining for commercial purposes has presently reached a certain level of maturity. In recent years, data mining techniques are often used in research on security [1-12]. One of the most typical examples of a data mining application in network security is an intrusion detection system (IDS). In general, an IDS collects a large quantity of alert information. However, information security personnel cannot identify any associations from large amounts of disorganized alert information. Nevertheless, an IDS that incorporates data mining techniques facilitate identifying accurate and appropriate data characteristics from large quantities of alert information by examining the association between multiple characteristics. The adequacy of the data mining technique applied determines the usefulness of the data classification result, which then affects the accuracy and effectiveness of intrusion detection. Thus, special attention must be paid to the applicability of data mining algorithms under different scenarios; this explains why further analysis should be performed on existing data mining algorithms.

[1-4] utilized an algorithm that augments existing scenarios with alerts generated by heterogeneous IDSs; this algorithm considers each new alert by estimating the probability that this new alert belongs to a given scenario and ascribes each alert to the most likely candidate scenario. This involves real-time data mining on the basis of ideal scenarios. In addition, [1, 4] evaluated the proposed algorithm by comparing it with two other probability estimation techniques; because this method only requires simple data entry, it can serve as a more appropriate probability estimation technique and is thereby more likely to obtain desired results. [5-9] employed hidden Markov models (HMMs) to detect complex network attacks. Although a typical HMM is more effective in detecting such complex intrusion events than are decision trees and neural networks, it has certain drawbacks. Collaborative network attacks are often conducted over a long period (e.g., a few weeks or months). Therefore, developing methods for detecting such slow-paced attacks in advance is one of the primary research topics in this field. These methods must also be capable of detecting attacks that involve highly complex methodologies, and must feature a set of procedures to detect slow-paced attacks, and must be able to counteract attacks that have been detected. Sensor failures or interference in transmission channels (such as open wireless networks), can affect the detection of sequential attacks and cause errors. [10-12] proposed a sequential pattern mining technique that incorporated alert information associated with distributed denial-of-service (DDoS) attacks and reduce the repetition and complexity of abundant alert information received by the IDS, thereby expediting the identification of the DDoS attack types. To identify the alert sequence of DDoS attacks, [10] employed a particular HMM to compare algorithms. The comparison results indicated that sequential pattern mining algorithms are more solid than other algorithms are because they do not result in mutual exclusivity of correlation data during a DDoS attack and can increase the accuracy of multi-step intrusion detection. The application of that HMM in experimental comparison of algorithms verified whether the proposed algorithm exhibited favorable effects for identifying the sequences of DDoS attacks and ensured effective reduction in alert complexity.

This paper is organized as follows. Section 2 elucidates the employed data mining algorithms. Section 3 presents the data mining algorithms deemed as adequate for application to network security. Section 4 presents the conclusions of this paper.

2. Data Mining Algorithms

In general, data mining involves converting collected data into preferred formats for analysis and employing algorithms to derive useful intrusion detection models for identifying known or unknown intrusion types. Data mining studies do not have a long history; they emerged mainly from a study by Agrawal et al. (1993) [13], which stimulated widespread academic enthusiasm in this topic.

Research manuscripts reporting large datasets that are deposited in a publicly available database should specify where the data have been deposited and provide the relevant accession numbers. If the accession numbers have not yet been obtained at the time of submission, please state that they will be provided during review. They must be provided prior to publication.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 June 2019 doi:10.20944/preprints201906.0144.v1

Association rule mining is a procedure used to discover interesting associations or relevant connections between itemsets in abundant data. As increasing amounts of data were collected and stored, database users began to demonstrate interest in discovering association rules in their databases. Discovery of interesting association rules is conductive to formulating commercial policies, such as design categorization and cross selling. The purpose of such discovery (or mining) is to identify any interesting relationships between items within a given data set. Support and confidence are interestingness measures used for evaluating association rules; they reflect the usefulness and certainty of association rules, respectively. Interestingness measures can also be used to assess a model's simplicity, certainty, utility, and novelty. Association rule mining is typically performed with two steps:

(1) Find all frequent itemsets: Frequent itemsets are defined as itemsets with a high probability of appearance in a database. If the obtained itemsets meet this definition, they are considered as frequent itemsets.

(2) Generate strong association rules from qualified frequent itemsets: If the obtained itemsets meet the criteria of minimum support and minimum confidence, they can be defined as strong association rules.

The first step is clarified as follows. The Apriori algorithm is one of the most influential algorithms used for association rule mining; it serves as the basis for comparing various algorithms and is a basic technique for deriving different algorithms. This study designated the Apriori algorithm was as the first algorithm to be thoroughly analyzed because it shares several key terms with the other algorithms discussed herein. The Apriori algorithm requires verification on the basis of the characteristics of frequent itemsets; to meet the criterion of this algorithm, all nonempty subsets in a frequent itemset must be frequent. The Apriori algorithm involves layer-by-layer iteration and mines (k+1)-itemsets by using k-itemsets. This mining process can be illustrated as follows: First, find all frequent 1-itemsets and label them as L_1. Subsequently, L_1 is used to find frequent 2-itemsets (L_2), which are then used to find L_3. This process is repeated until no more frequent k-itemsets can be found. This algorithm scans the database when each search of L_k is completed and generates candidate itemsets from the itemset L obtained in the previous search. The candidate itemset meeting the Apriori criterion is then used as the itemset L for the following search.

To increase the efficiency of layer-by-layer iteration used for mining frequent itemsets, reducing research space is a crucial goal of the Apriori algorithm. Figure 1 presents the pseudocode used in the Apriori algorithm, where *Input* represents the transaction item of D, *min_sup* denotes the minimum support count, and *Output* is the frequent itemset L in the database.

Once the first step has been completed by finding all frequent itemsets from the transaction data in database D, the second step (generate strong association rules) can be performed immediately. The identification of strong association rules requires meeting the criteria of minimum support and minimum confidence. The following equation can be used to calculate confidence:

$$confidence\ (A \Rightarrow B) = P(A|B) = \frac{suooprt\_count\,(A \cup B)}{suooprt\_count\,(A)} \quad (1)$$

where *support_count(A∪B)* denotes the number of transactions containing itemset A∪B and *support_count(A)* denotes the number of transactions containing itemset A.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 June 2019 doi:10.20944/preprints201906.0144.v1


```
Method:
1:   L1=find_frequent_1-itemsets(D);
2:   for (k=2; Lk-1? Ø; k++){
3:       Ck= apriori_gen (Lk-1, min_sup);
4:       for each transaction t? D { //scan D for counts
5:           Ct=subset (Ck, t); //get the subset of t that are candidates
6:           for each candidate c? Ct
7:           c.count++;
8:       }
9:       Lk={c? Ct | c.count = min_sup }
10: }
11: return L=Uk Lk;

Procedure apriori_gen(Lk-1:frequent(k-1)-itemsets; min_sup:minimum support threshold)
1:   for each itemset l1? Lk-1
2:       for each itemset l1? Lk-1
3:           if (l1[1]=l2[1]) ? (l1[2]=l2[2]) ? ...?
               (l1[k-2]=l2[k-2]) ? (l1[k-1] =l2[k-1]) then {
4:           C=l1><l2; //join step: generate candidates
5:           if has_infrequent_subset(c, Lk-1) then
6:             delete c; //prune step: remove unfruitful candidate
7:           else add c to Ck;
8:           }
9:    return Ck;

Procedure has_infrequent_subset(c: candidate k-itemsets; Lk-1: frequent(k-1)-itemsets);
1:   for each (k-1)-subset s of c
2:       if s? Lk-1 then
3:           return TRUE;
4:   return FALSE;
```

Figure 1. Apriori algorithm employed to discover frequent itemsets during Boolean association rule mining

Because all rules are generated from frequent itemsets, each of these rules automatically satisfy the minimum support count. Frequent itemsets and their support levels are included in the list of items to ensure they can be immediately mined. An example of applying the Apriori algorithm to identify frequent itemsets in a database is presented as follows:

TID	List of item_IDs
T100	11,12,15
T200	12,14
T300	12,13
T400	11,12,14
T500	11,13
T600	12,13
T700	11,13
T800	11,12,13,15
T900	11,12,13

Figure 2. Data from the transaction database of Company Branch X

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 June 2019 doi:10.20944/preprints201906.0144.v1

In this example, item IDs 11 and 12 are assumed to represent attacks in the field of network security (they could just as easily be applied to represent other types of data, e.g., diapers and beer in a supermarket). Frequent itemsets were generated from the following steps:

Step 1: In iteration 1, consider every item as a member of the candidate 1-itemset, C_1. Scan all transactions roughly to count all items that appear (Figure 3).

C_1

Itemset	Sup count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

Figure 3. Members of C_1

Step 2: Assume the minimum support count to be 2 (i.e., *min_sup*=2/9=22%) so that the frequent 1-itemset, L_1, consists of candidate 1-itemsets meeting the threshold of the minimum support count.

L_1

Itemset	Sup count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

Figure 4. Members of L_1

Step 3: Search for the frequent 2-itemset L_2. Use the algorithm to generate the set of candidate 2-itemset, C_2^6, where C_2 consists of $C_{L_1}^2$ 2-itemsets (Figure 5).

C_2

Itemset
{I1, I2}
{I1, I3}
{I1, I4}
{I1, I4}
{I2, I3}
{I2, I4}
{I2, I5}
{I3, I4}
{I3, I5}
{I4, I5}

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 June 2019 doi:10.20944/preprints201906.0144.v1

Figure 5. Members of C_2

Step 4: Scan the transactions in database *D*. Calculate the level of support of every candidate itemset in C_2 (Figure 6).

C_2

Itemset	Sup count
{I1, I2}	4
{I1, I3}	4
{I1, I4}	1
{I1, I4}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2
{I3, I4}	0
{I3, I5}	1
{I4, I5}	0

Figure 6. Support count of C_2

Step 5: Confirm the frequent 2-itemset, L_2, which consists of candidate 2-itemsets in C_2 that satisfy the criterion of greater than or equal to minimum support count.

L_2

Itemset	Sup count
{I1, I2}	4
{I1, I3}	4
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2

Figure 7. Set L_2

Step 6: First, let C_2 = *{{I1, I2, I3}, {I1, I2, I5}, {I1, I3, I5}, {I2, I3, I4}, {I2, I3, I5}, {I2, I4, I5}}*. According to the characteristics of the Apriori algorithm, all subsets in frequent itemsets should appear frequently. Because any two items in the last four candidate itemsets do not appear in L_2, these are not frequent items, and thus these four candidate itemsets must be deleted from C_3. Therefore, their count must no longer be calculated when scanning *D* and confirming L_3. Notably, because the Apriori algorithm involves layer-by-layer iteration, once *k* itemsets are given, the mining process starts by only checking whether their (*k*-1) subsets are frequent.

Step 7: Scan the transaction in database *D* to confirm whether L_3 consists of candidate 3-itemsets in C_3 that satisfy the minimum support count (Figure 8).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 June 2019 doi:10.20944/preprints201906.0144.v1

C_3

Itemset
{I1, I2, I3}
{I1, I2, I5}

C_3

Itemset	Sup count
{I1, I2, I3}	2
{I1, I2, I5}	2

Figure 8. Set C_3

Step 8: Use the algorithm to generate the candidate 4-itemset, C_4. Because subset *{I2, I3, I5}* did not appear frequently in the *{{I1, I2, I3, I5}}* itemset, a result of C_4, the subset was removed. Accordingly, C_4 was identified to be equal to an empty set, thereby terminating the algorithm and identified all frequent itemsets in L_3 (Figure 9).

L_3

Itemset	Sup count
{I1, I2, I3}	2
{I1, I2, I5}	2

Figure 9. Set L_3

Now the example database is used to calculate which of the itemsets contain strong association rules. The third data set derived from the example data, including frequent itemset *l={I1, I2, I5}*, indicates that the nonempty subsets of *l* are *{I1, I2}*, *{I1, I5}*, *{I2, I5}*, *{I1}*, *{I2}*, and *{I5}*. The resulting association rules are obtained from the aforementioned calculation method, each listed with its confidence:

I1 ∧ *I2* ⇒ *I5*, *confidence* = 2/4 = 50%
I1 ∧ *I5* ⇒ *I2*, *confidence* = 2/2 = 100%
I2 ∧ *I5* ⇒ *I1*, *confidence* = 2/2 = 100%
I1 ⇒ *I2* ∧ *I5*, *confidence* = 2/6 = 33%
I2 ⇒ *I1* ∧ *I5*, *confidence* = 2/7 = 29%
I5 ⇒ *I1* ∧ *I2*, *confidence* = 2/2 = 100%

If the minimum confidence is 70%, then only the second, third, and sixth rules satisfy the threshold of ≥70%. Thus, only these three items are strong association rules.

In [14, 17, 19], the main drawback of Apriori algorithm is that it must scan the database each time when generating an association file. If the database contains abundant data, this method easily increases the system load. To increase the efficiency of the Apriori algorithm, scholars have modified this algorithm using specific methods. For instance, the hash-based technique can be used to calculate the count of hash itemsets and reduce the size of the candidate *k*-itemset, C_k ($k > 1$). After each transaction in the database is scanned, the frequent 1-itemset L_1 is generated from the candidate 1-itemset in C_1, and subsequently 2-itemsets can be generated from each transaction and then hashed (i.e., mapped) into the buckets in the hash table, after which the corresponding bucket counts can be increased (Figure 10). 2-itemsets with corresponding bucket counts in the hash table that are lower than the support threshold are deemed as nonfrequent and are excluded from the candidate set. The hash-based technique can largely reduce the number of *k*-itemsets examined (particularly when $k = 2$) [19].

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 June 2019 doi:10.20944/preprints201906.0144.v1

bucket address	0	1	2	3	4	5
bucket count	2	2	4	2	2	4
bucket contents	{I1, I4} {I3, I5}	{I1, I5} {I1, I5}	{I2, I3} {I2, I3} {I2, I3} {I2, I3}	{I2, I4} {I2, I4}	{I2, I5} {I2, I5}	{I1, I2} {I1, I2} {I1, I2} {I1, I2}

Figure 10. Hash table, H_2, for candidate 2-itemsets

Note: This table was generated by scanning the transactional database when determining L_1 by C_1. (If the minimum support count is 3, then the itemsets in buckets 0, 1, 3, and 4 are nonfrequent and thus should be removed from C_2).

Transaction reduction can also be used to improve the Apriori algorithm. This method can reduce the number of transactions scanned in future iterations. A transaction without any frequent *k*-itemsets cannot contain any frequent (*k+1*)-itemsets. Thus, such transactions can be labeled or removed; hence, subsequent database scans for *j*-itemsets ($j > k$) will no longer consider such transactions. One variation of the Apriori algorithm involves partitioning the data to find candidate itemsets. The partitioning technique requires only two database scans to mine frequent itemsets (Figure 11). It consists of two phases. In phase I, the algorithm partitions the transactional data of *D* into *n* nonoverlapping partitions. If the minimum support threshold in *D* is set as *min_sup*, then the minimum support count for a partition should be *min_sup* × *the number of transactions in the partition*. For each partition, all local frequent itemsets are found [17, 19].

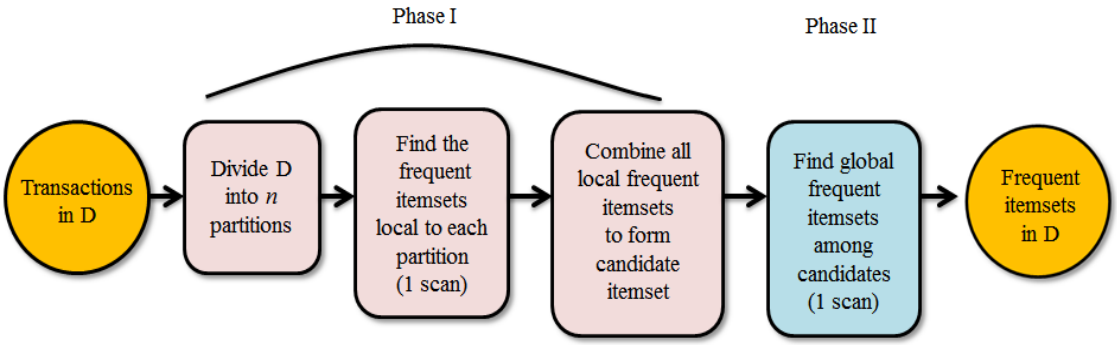

Figure 11. Mining by the partitioning technique

Obtaining all local frequent *k*-itemsets requires only one database scan. All local frequent itemsets can be treated as candidate itemsets for *D*. Therefore, collecting frequent itemsets from all partitions enables generating the global candidate itemsets for *D*. In phase II, a second scan of *D* is performed to evaluate the actual support count of each candidate to identify the global frequent itemsets. This phase determines the partition size and the number of partitions so that each partition can fit into main memory and thus be read only once in each phase [19].

Sampling can also be utilized to mine a subset of the given data. This technique involves selecting a random sample *S* of the given database *D* and searching for frequent itemsets in *S* rather than *D*. This method trades off certain degree of accuracy but increases efficiency. Because the frequent itemsets are searched in *S* instead of *D*, some of the global frequent itemsets might be lost. To reduce that possibility, a support threshold lower than the minimum support is adopted to identify the frequent itemsets local to *S* (denoted L_S). The rest of the database is then used to calculate the actual frequencies of each itemset in L_S to determine whether all the global frequent itemsets are included in L_S. If L_S actually contains all the frequent itemsets in *D*, then only one scan of *D* required. Otherwise, a second scan can be conducted to find any frequent itemsets that might have been missed in the first scan. This sampling technique is particularly adequate when efficiency is prioritized. Scholars have also proposed dynamic itemset counting as a method for enhancing the

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 June 2019 doi:10.20944/preprints201906.0144.v1

Apriori algorithm. This approach adds candidate itemsets at various points during a scan and partitions the database into blocks marked by start points. In contrast to standard Apriori, which confirms new candidate itemsets only before each complete database scan, this variation can add new candidate itemsets at any start point. This technique dynamically evaluates the support count of all itemsets examined. If all subsets in an itemset are confirmed as frequent, then this itemset will be considered as a new candidate itemset. Accordingly, this technique requires fewer scans than standard Apriori does[14, 19].

Under various circumstances, the candidate itemsets generated by Apriori require an examination method that will substantially reduce the size of the candidate itemsets, thereby achieving favorable effectiveness of the Apriori algorithm. However, this algorithm must consider the following two conditions:

(1) It may be required to produce a large number of candidate itemsets. For instance, if there are 10^4 frequent 1-itemsets, the Apriori algorithm will be required to generate more than 10^7 candidate 2-itemsets, during which the frequencies of all the candidate itemsets must be counted and examined. In addition, to find frequent itemsets of length 100, such as *{a1, …, a100}*, a total of $2^{100} \approx 10^{30}$ candidate itemsets must be generated.

(2) It may be required to repeatedly scan the database and examine a large set of candidate itemsets by pattern checking. Thus, a large amount of time is required to scan a database containing abundant data.

[14-17] have improved previous algorithms by proposing an algorithm that mines the complete set of frequent itemsets without requiring generation of candidate itemsets; the improved algorithm involves using frequent pattern growth (FPG) and frequent pattern tree (FP-tree) methods. Because this algorithm (hereinafter referred to as the FPG algorithm) possesses a higher efficiency than conventional algorithms possess, the present study examined it further.

The FPG algorithm employs a divide-and-conquer strategy, illustrated as follows: First, it compresses the database containing frequent itemsets into a FP-tree, which retains the itemset association information. Subsequently, it partitions the compressed database into a set of conditional databases, each of which is a special projected database that is associated with one frequent item. Then it mines each database separately. The same example data were used to illustrate the algorithm proposed by [14-17].

The first database scan of this algorithm is identical to that of the Apriori algorithm, which derives the set of frequent items (1-itemsets) and their support (regarding count and frequency). Let the minimum support count be 2. The set of frequent items is ranked in the order of descending support count and the resulting set or list is denoted by *L*, thereby obtaining *L=[I2:7, I1:6, I3:6, I4:2, I5:2]*. An FP-tree is subsequently constructed. The root of the tree is created and labeled with "null." Perform a second scan on database *D*. The items in each transaction are then arranged in L order (i.e., ranked in the order of descending support count), with a branch created for each individual transaction.

The scan of the first transaction, "*T100:I1, I2, I5*," which contains three items (I2, I1, and I5, sorted in *L* order), leads to the construction of the first branch of the tree with three nodes, (*I2:1*), (*I1:1*), and (*I5:1*), where I2 is linked as a child to the root, I1 is linked to I2, and I5 is linked to I1. The second transaction, "*T200:I2, I4*," which contains the items I2 and I4 in *L* order, leads to a branch where I2 is linked to the root and I4 is linked to I2. However, this branch shares a common prefix, I2, with the existing path for T100. Therefore, the count of node I2 is increased by 1, and a new node, (*I4:1*), is created; it is linked as a child to (*I2:2*). When adding a branch for a transaction, the count of each node along a common prefix is generally increased by 1, and nodes for the items following the prefix are created and linked accordingly.

To facilitate tree traversal, an item header table is constructed so that each item points to nodes with the same ID in the tree through a chain of node-links. The tree obtained after scanning all the transactions is shown in Figure 12, where all the associated node-links are illustrated. Therefore, the problem of mining frequent patterns in databases is transformed into that of FP-tree mining.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 June 2019 doi:10.20944/preprints201906.0144.v1

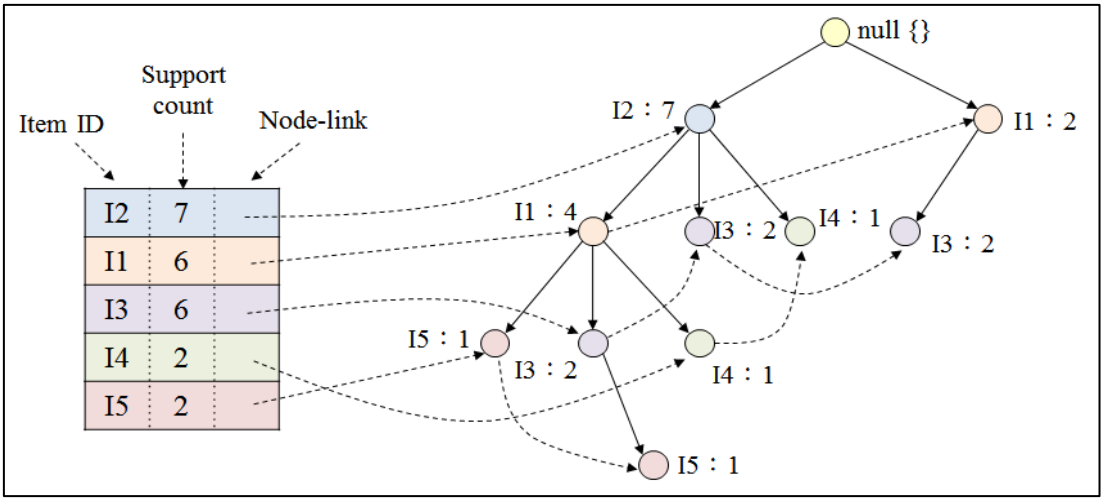

Figure 12. FP-tree that contains compressed and frequent pattern information

Method:

1. The FP-tree is constructed in the following steps.

(a) Scan the transaction database D once. Collect the set of frequent items F and their supports. Sort F in support descending order as L, the list of frequent items.

(b) Cerate the root of an FP-tree, and label it as "null". For each transaction Trans in D do the following.

Select and sort the frequent items in Trans according to the orser of L. Let the sorted frequent item list in Trans be [p|P], where p is the first element and P is the remaining list. Call insert_tree ([p|P], T), which is performed as follows. If T has a child N such that N.item-name = p.item-name, then increment N's count by l; else create a new node N, and let its count be l, its parent link be linked to T, and its node-link to the nodes with the same item-name via the node-link structure. If P is nonempty, call insert_tree (P, N) recursively.

2. Mining of an FP-tree is performed by called FP_growth (FP_tree, null), which is implemented as follows.

procedure FP_growth (Tree, α)

1: *if Tree contains a single path P then*
2: *for each combination (denoted as β) of the nodes in the path P*
3: *generate pattern β ∪ α with support = minimum support of nodes in ;*
4: *else for each a_i in the header of Tree {*
5: *generate pattern β = a_i ∪ α with support= a_i. support;*
6: *consturce β, s conditional pattern base and then β's conditional FP_tree Tree $_β$;*
7: *if Tree β ≠ Ø then*
8: *call F_growth(Tree $_β$, β); }*

Figure 13. Summary of the FPG algorithm, which mines frequent itemsets without generating candidate itemsets

FP-tree mining is performed starting from each frequent length-1 pattern. Construct a conditional pattern base (a sub-database that comprises the set of prefix paths in the FP-tree co-occurring with the suffix pattern) for each pattern. Then, construct its conditional FP-tree, and perform mining recursively on the tree. The pattern growth is attained by connecting the suffix pattern with the frequent patterns generated from a conditional FP-tree.

Take I5 as an example. I5 occurs in two branches of the FP-tree displayed in Fig. 12. The paths formed by these branches are (*I2 I1 I5:1*) and (*I2 I1 I3 I5:1*). Therefore, the two corresponding prefix paths for I5 are (*I2 I1:1*) and (*I2 I1 I3:1*), which form its conditional pattern base. The I5-conditional FP-tree contains only a single path, (*I2:2 I1:2*); I3 is included because its support count of 1 is less than

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 June 2019 doi:10.20944/preprints201906.0144.v1

the minimum support count of 2. This single path generates all the combinations of frequent patterns, which are (I2, I5: 2), (I1, I5: 2), and (I2, I1, I5: 2). The same process repeats when using other items. The FPG algorithm is summarized in Figure 13.

In terms of mining long and short frequent patterns, the FPG algorithm exhibits higher effectiveness, flexibility, and efficiency than the Apriori algorithm does.

3. Data Mining Algorithms Applicable in Network Security

Data related to network security require special attention in certain respects. For instance, they may contain time packets so that the employed data mining algorithm may require adjustment accordingly. Therefore, data mining algorithms applicable to network security were analyzed in the present study to facilitate their subsequent application. Dain and Cunningham [1] proposed an algorithm that organizes alerts generated by heterogeneous IDSs into scenarios; this algorithm computes a new alert by estimating the probability that this new alert belongs to a given scenario and adds alerts to the most likely candidate scenario. In a publication of the IEEE, Ourston et al. [5] proposed a method that utilized HMMs to detect complex network attacks. The HMM is more effective in detecting such complex intrusion events than are decision trees and neural networks.

This section uses [10, 18] to illustrate the application of data mining in network security. These two studies proposed a sequential pattern mining technique that incorporated alert information associated with DDoS attacks and reduced the repetition and complexity of abundant alert information received by the IDS, thereby identifying the type of the DDoS attack. In addition, [10] employed a particular HMM to compare algorithms. This method comprises four algorithms. Algorithm 1 is the main mining loop and the other three algorithms are various extensions of Algorithm 1.

In Algorithm 1, four variables are first inputted. The first variable is set as alertSet, which denotes an alert set consisting of m raw alerts. The second to fourth variables are address prefix length n, time constraint t, and support threshold s, respectively. The output variables are alerts that have undergone correlation analysis and DDoS scenarios. Therefore, alertSet, address prefix length n, and time constraint t are introduced as parameters into genSequence, a function that generates sequential patterns. The output values produced from the calculation using genSequence are then used as the parameters of genSequence and introduced into Algorithm 3, where sequenceSet and the minimum support threshold s are inputted into function minePattern to identify the alert sample set with the longest sequential pattern, labelled lspSet. Subsequently, lspSet and n are introduced into the corrSPattern function to generate the desired DDoS scenarios. This entire process is completed using input parameters and three functions (i.e., the following algorithms to be discussed).

Algorithm 1 corrAlerts
Input: alertSet containing m raw alerts, address prefix
Length n, time constraint t, support threshold s
Output: correlated alerts, DDoS scenarios
1: sequenceSet ← genSequence(alertSet, t, n)
2: lspSet ← minePatten(sequenceSet, s)
3: scenarios ← corrSPattern(lspSet, n)
4: output DDoS scenarios and the alerts after correlation

Figure 14. Algorithm 1

The match (IP1, IP2, l) employed by the following algorithms is used as an example for clarification. Let the addresses of IP1 and IP2 be 135.13.216.191 and 135.13.16.189, respectively. A comparison of these addresses reveals that the third items of the two IP addresses are different. Therefore, this indicates that n is the length of the first two identical IP addresses, namely n = 16.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 June 2019 doi:10.20944/preprints201906.0144.v1


```
Algorithm 2 genSequence
Input: alertSet containing m alerts, t, n
Output: sequenceSet
1: sequence, seqArray ← φ
2: k = 0
3: for all = alert1 to alertm do
4:       if (all is not marked)
5:          for al2 = all to alertm do
6:             if (( al2 is not marked) ^
                    match(all.srcIP, al2.srcIP, n) ^
                    match(all.dstIP, al2.dstIP, n))
7:                curAlert = al2
8:                sequence ← sequence ∪ al2
9:                mark al2
10:               for al3 = al2 + 1 to alertm do
11:                  if ((al3 is not marked) ^
                           (al3.time - curAlert.time = t))
12:                     flag = false
13:                     if (match(al2.srcIP, al3.srcIP, n) ^
                              match(al2.dstIP, al3.dstIP, n) ^
                              al3.stage = = curAlert.stage + 1)
14:                        flag = true
15:                     else if ( curAlert.stage = = 2 ^
                                    al3.stage = = 3)
16:                        flag = true
17:                     end if
18:                     if ( flag )
19:                        curAlert = al3
20:                        sequence ← sequence ∪ al3
21:                        mark al3
22:                     end if
23:               end for
24:               seqArray ← seqArray ∪ sequence
25:               sequence ← φ
26:          end if
27:       sequenceSet[k + +] ← seqArray
28:       seqArray ← φ
29: end for
30: output sequenceSet
```

Figure 15. Algorithm 2

Because real DDoS attacks at the final stage often contain packets generated from address errors, adding alerts to a sequence facilitating counteracting this problem. The objective of Algorithm 2 is to establish alert sequences by inputting variables *m*, *t*, and *n* to generate the desired sequence. This algorithm consists of three loops. The first two loops define an alert group by comparing IP addresses, namely comparing match (IP1, IP2, l) with the host address. Then, the third loop extracts alerts during various attack stages from the previously obtained alert group to construct alert sequences.

According to lines 6 and 7 of the following algorithm, all alerts that have been sorted are marked, starting from alert 2. If an alert is not marked, its srcIP and dstIP will be compared with all other srcIP and dstIP values. After the comparison, the original sequence and the alert will be

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 June 2019 doi:10.20944/preprints201906.0144.v1

redefined into a new sequence, and this alert will be marked. For alerts that have already been marked, alert time is used to determine whether they will be marked as false alerts, as suggested by lines 11 and 12. If the result of subtracting an alert time from the current alert time is ≤ the inputted variable t, then this alert is marked as a false alert. If the alert time meets the aforesaid criterion, then whether the stage of this alert is higher than the current alert stage by one level will be checked, as indicated by lines 13 and 14. If the alert stage is lower than the current stage, the alert is marked as a false alert. The same rule applies to the rest of the alerts.

Algorithm 3 searches the sequential alert patterns from the alert groups created in algorithm 2. Alerts found in each pattern are regarded as correlated. The longest sequential pattern is reserved in the group because normal activities of a legitimate user may sometimes exhibit less frequent patterns that are incorrectly captured into alert sequences. Such alerts are removed from further analysis by this algorithm.

Algorithm 3 is as follows: Create a lspSet variable and set it as an empty set. Create a variable sp and extract a portion of sequence to sp, as indicated in line 3. The sequence of sp is first used to find long alert sequence (labelled lsp). Alert sequences meeting this lsp criterion are defined as srcIP and dstIP. Finally, the identified lsp variable is added into the alert group, which is redefined as a new group. The same process repeats until all lspSet sets found by the algorithm have been outputted.

```
Algorithm 3 minePattern
Input: sequenceSet (p alert groups), support s
Output: longest sequential alert pattern set lspSet
1: lspSet← φ
2: for i = 0 to p do
3:      sp ←SPAM(sequenceSet[i], s)
4:      if (sp = φ)
5:          correlate all alerts covered by sp
6:          lsp = the longest sequential alert pattern in sp
7:          lsp.srcIP ← the srcIP of the first alert in lsp
8:          lsp.dstIP ← the dstIP of the last alert in lsp
9:          lspSet ← lspSet ∪ lsp
10:     end if
11: end for
12: output lspSet
```

Figure 16. Algorithm 3

Algorithm 4 generates DDoS scenarios mainly using the obtained sequential alert pattern sample. The alert sample set with the longest sequential pattern is inputted into the algorithm and loops of sequential patterns are used to generate a scenario catalog. In Algorithm 4, the first lspatternn is required to end the alert at stage2 or stage3 particularly when producing correlation data.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 June 2019 doi:10.20944/preprints201906.0144.v1


```
Algorithm 4 corrSPattern
Input: longest sequential pattern set lspSet (containing n
lspatterns), address prefix length l
Output: DDoS scenarios
1: corrlist, scenariolist ← φ
2: for lsp1 = lspattern1 to lspatternn do
3:       if (lsp1 is not marked )
4:           currentlsp ← lsp1
5:           corrlist.addToTail(lsp1)
6:           marklsp1
7:           for lsp2 = lsp1 to lspatternn do
8:                if (( lsp2 is not maked) ^
                         match(currentlsp.dstIP, lsp2.srcIP, n) ^
                                  currentlsp.lastAlert.stage = 2)
9:                     currentlsp ← lsp2
10:                    corrlist.addToTail(lsp2)
11:                    mark lsp2
12:               end if
13:          end for
14:          scenario ←fuse the lspatterns in corrlist
15:          scenariolist = scenariolist ∪ scenario
16:          corrlist ← φ
17:      end if
18: end for
19: output DDoS scenarios in scenariolist
```

Figure 17. Algorithm 4

The test results confirmed that sequential pattern mining algorithms are more reliable than other types of algorithm because they do not result in mutual exclusivity of correlation data during a DDoS attack and can increase the accuracy of multi-step intrusion detection. Applying HMMs to compare algorithms through comprehensive and experimental methods verifies whether the proposed algorithm exhibits favorable effects for identifying the sequence of DDoS attacks and ensures effective reduction in alert complexity.

We launched attacks with three tools, Nmap, Pizzaviat DDoS, Nikto, and compared them with used algorithms and unused algorithms. Table 1 shows the number of times the alarm was detected. The method of unused algorithms will repeatedly detect alerts, resulting in a much larger number of times. The method of used algorithms will greatly reduce the number of alarms for the same event.

Table 1. The Number of alerts

	Tool	Used Algorithms	Unused Algorithms
1	Nmap	2	6
2	Pizzaviat DDoS	3	15
3	Nikto	5	45

4. Conclusions

The importance of data mining in network security lies in its ability to automatically extract hidden but crucial information from the alert and log data of large systems, thereby providing a helpful tool for information security personnel to make decisions. Therefore, data mining has been gradually introduced to network security studies that aim to investigate methods for detecting

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 June 2019 doi:10.20944/preprints201906.0144.v1

unknown attacks before hazards occur and to minimize potential hazards caused by such attacks. Section 3 indicates that sequential patterns can be used to generate a desired DDoS scenario. If the effectiveness of the algorithm can be enhanced, a future version of the algorithm may be able to construct multi-step attack scenarios; such an improvement will be beneficial to future practice. Accordingly, a number of key algorithms related to this topic were analyzed in this study to facilitate analyzing massive data in the future.

Author Contributions: Kai Chain conceived and designed the experiments; performed the experiments and analyzed the data and wrote the paper.

Acknowledgments: This research was supported by MOST 106-2221-E-145-002.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Dain OM and Cunningham RK. Fusing a heterogeneous alert stream into scenarios. In: Proceedings of the 2001 ACM Workshop on Data Mining for Security Applications, 14th September 2001; pp. 1-13.
2. Eckmann ST, Vigna G and Kemmerer RA. Statl: An attack language for statebased intrusion detection. *J Computer Security* **2002**; 10: 71-103.
3. Totel E, Vivinis B and Me L. A language driven IDS for event and alert correlation. In: SEC 2004; pp. 209-224.
4. Wang L, Ghorbani A and Li Y. Automatic multistep attack pattern discovering. *Int J Network Security*. **2010**; 10: 142-152.
5. Ourston D, Matzner S, Stump W and Hopkins B. Applications of Hidden Markov Models to detecting multi-stage network attacks. In: Proceedings of the 36th Ann. Hawaii International Conference on System Science 2003; pp. 334.
6. Somayaji A and Forrest S. Automated response using system-call delays. In: Proceedings of the 9th USENIX Security Symp, Denver, Colorado, 14 –17 August 2000; pp. 185-198.
7. Lee W, Stolfo SJ and Chan PK. Learning patterns from unix process execution traces for intrusion detection. In: Proceedings of the AAAI97 workshop on AI Approaches to Fraud Detection and Risk Management, AAAI Press, 1997; pp. 50-56.
8. Warrender C, Forrest S and Pearlmutter BA. Detecting intrusions using system calls: Alternative data models. In: Proceedings of the IEEE Symp. on Security and Privacy, 1999; pp. 133-145.
9. Forrest S, Perelson AS, Allen L and Cherukuri R. Self-nonself discrimination in a computer. In: SP'94: Proc. of the 1994 IEEE Computer Society, 1994; pp. 202.
10. Xiang G, Dong X and Yu G. Correlating Alerts with a Data Mining Based Approach. In: Proceedings of the IEEE International Conference on e-Technoligy, e-Commerceande-Service, 29 March-1 April 2005.
11. AL-mamory SO and Zhang HL. A survey on IDS Alerts processing techniques. In: Proceedings of the 6th WSEAS International Conference on Information Security and Privacy, Tenerife, Spain, 14-16 December 2007; pp. 69-78.
12. Cheng B, Liao G and Huang C. A novel probabilistic matching algorithm for multi stage attack forecasts. *IEEE J on Selected Areas in Comm*. **2011**; 29: 1438-1448.
13. Agrawal R, Imielinski T and Swami AN. Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD Intl. Conference on Management of Data, 1993; pp. 207-216.
14. Han J, Pei J and Yin Y. Mining frequent patterns without candidate generation. In: Proceedings of the 2000 ACM SIGMOD international conference on Management of data, 2000; pp. 1-12.
15. Leung CKS, Mateo MAF and Brajczuk DA. A tree-based approach for frequent pattern mining from uncertain data. In: Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2008; pp. 653-661.
16. Zaki MJ, Li F and Yi K. Finding frequent items in probabilistic data. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, 2008; pp. 819-832.
17. Han J and Chang KCC. Data mining for web intelligence. *Computer*. **2002**; 35: 64-70.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 June 2019 doi:10.20944/preprints201906.0144.v1

18. Stroeh K, Madeira ERM and Goldenstein SK. An approach to the correlation of security events based on machine learning techniques. *J Int Services and Applications*. **2013**; 4: 1-16.
19. Han J, Pei J and Kamber M. Data Mining Concepts and Techniques. **2011**.