2016

An Improved String-Searching Algorithm and Its Application in Component Security Testing

Jinfu Chen
School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang 212013, China.

Saihua Cai
School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang 212013, China.

Lili Zhu
School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang 212013, China.

Yuchi Guo
School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang 212013, China.

Rubing Huang
School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang 212013, China.

Follow this and additional works at: https://tsinghuauniversitypress.researchcommons.org/tsinghua-
See next page for additional authors

Part of the Computer Sciences Commons, and the Electrical and Computer Engineering Commons

Recommended Citation
Jinfu Chen, Saihua Cai, Lili Zhu et al. An Improved String-Searching Algorithm and Its Application in Component Security Testing. Tsinghua Science and Technology 2016, 21(3): 281-294.
An Improved String-Searching Algorithm and Its Application in Component Security Testing

Authors
Jinfu Chen, Saihua Cai, Lili Zhu, Yuchi Guo, Rubing Huang, Xiaolei Zhao, and Yunqi Sheng
An Improved String-Searching Algorithm and Its Application in Component Security Testing

Jinfu Chen, Saihua Cai, Lili Zhu, Yuchi Guo, Rubing Huang*, Xiaolei Zhao, and Yunqi Sheng

Abstract: Mass monitor logs are produced during the process of component security testing. In order to mine the explicit and implicit security exception information of the tested component, the log should be searched for keyword strings. However, existing string-searching algorithms are not very efficient or appropriate for the operation of searching monitor logs during component security testing. For mining abnormal information effectively in monitor logs, an improved string-searching algorithm is proposed. The main idea of this algorithm is to search for the first occurrence of a character in the main string. The character should be different and farther from the last character in the pattern string. With this algorithm, the backward moving distance of the pattern string will be increased and the matching time will be optimized. In the end, we conduct an experimental study based on our approach, the results of which show that the proposed algorithm finds strings in monitor logs 11.5% more efficiently than existing approaches.

Key words: component testing; security vulnerabilities detection; monitor log; abnormal information; string-searching

1 Introduction

With the development of informatization infrastructure, the security issues that exist in information systems have become more and more serious, and security problems are now a major problem for enterprises and individuals. Due to the rapid development of component technology, more and more commercial software vendors buy and use third-party component products[1], including some key programs that have high requirements for security, such as military software, medical software, banking software, railway software, and finance software. Component-Based Software Engineering (CBSE) has become a research focus in the field of software engineering due to its characteristics of reuse and “plug and play”, which improves software development efficiency and reduces the cost of development and maintenance. However, many security problems that perplex component developers and users have not been resolved[2, 3].

Attributes including credibility, confidentiality, availability, integrity, and reliability are the criteria for judging whether a component is secure. Component vulnerability means that there are security defects, including factors that may threaten or damage the security of computer systems of which the components are part[4]. The main source of component security vulnerabilities is that the running states of components’ methods violate component security specifications. The current approach to testing component security is to generate mutation test cases and execute test sequences of component interface methods[5, 6]. This method produces a large number of monitor logs. Hence, digging out components’ explicit and implicit security exception information from mass monitor logs has become the main way to determine whether a
component has security vulnerabilities. Mining components’ explicit and implicit security exception information in monitor logs involves finding keywords, usually in the form of strings. Therefore, string-searching technology is used for keyword matching in monitor logs. In this technology, the monitor log is the main string, and the keyword, which is also called the “insecure execution sequence”, is the pattern string. A string matching technique is an operation of finding a pattern string in the main string. If matching succeeds, it will return a successful match location, which means the component has a security vulnerability. If not, it means that the component has no security vulnerability.

The existing log matching methods, based on BF algorithm, are straightforward. However, BF algorithm is not suitable for components. This is because the efficiency of BF algorithm is low, and the components will produce a large number of monitoring log sets during the running process. Despite the fact that Knuth-Morris-Pratt (KMP) algorithm, BM algorithm, and Sunday algorithm have improved BF algorithm, they still do not apply to a large set of monitoring logs. Hence, we need to find a new string matching algorithm that is suitable for large main strings.

2 Research Background

The main content studied in this paper is a part of a previous project (third-party component security testing based on data mining). The main modules of this project include specification mining, method execution sequence mining, and component security testing\cite{5}. The principle of the specification mining module is to use pattern recognition technology and frequent-items mining technology, which are both data mining techniques, to exploit security requirement specifications of third-party components based on component interface information, component descriptions, and component interface definitions\cite{6–8}. The principle of method execution sequence mining is to use frequent-items and sequential patterns mining algorithms\cite{9} to exploit method execution sequences based on component specifications, interface method information, and monitor logs, and generate component method execution sequences. The principle of component security testing is to use data mining techniques such as data classification, frequent-item, and sequence pattern mining algorithms to mine security association rules, abnormal methods, and sequences based on monitor logs produced by dynamically executing components, and insecure method execution sequences produced by component mutation testing\cite{10, 11}.

Component security testing is the research focus of this paper, and the specific test framework is shown in Fig. 1. First, monitor log data should be preprocessed to form a sequence database. Then the sequence database should be handled by using three data mining techniques such as data classification, frequent-item, and sequence pattern mining algorithms to get component method execution sequences. Next, unsafe method sequences and all execution sequences in the monitor logs should be matched by using a sequence matching algorithm. If the matching is successful, these method execution sequences have insecure sequences, which means that these method execution sequences have security vulnerability. Otherwise, the method execution sequences do not have insecure sequences. Finally, a vulnerability testing report is generated, based on the matching results.

3 String-Searching Algorithms

Some string-searching algorithms in current use include BF algorithm, KMP algorithm\cite{12, 13}, BM algorithm\cite{14–17}, and Sunday string-searching algorithm. Among them, BF algorithm is considered...
simple, but its matching efficiency is not high. KMP algorithm is more efficient than BF algorithm. BM algorithm determines the distance of the pattern string moving to the right by using three efficient approaches: a "scanning from right to left" rule, a "bad character" rule, and a "good suffix" rule, to reduce the number of matching attempts. Sunday string-searching algorithm improves searching efficiency by moving as many characters as possible to the right once matching fails during the matching process.

Notations: The main string is denoted by \(S[1, 2, \ldots, n]\), and \(n\) is its length; the pattern string is denoted by \(T[1, 2, \ldots, m]\), and \(m\) is its length; \(X\) denotes the right matching location in the main string.

### 3.1 BF algorithm

BF algorithm was the first string-searching algorithm to be proposed; however, it has the worst performance among the current algorithms. The core idea of BF algorithm is as follows. First, the first characters of the pattern string and the main string are aligned. They are named \(T[1]\) and \(S[1]\), respectively. If \(T[1]\) and \(S[1]\) are matched successfully, \(T[2]\) and \(S[2]\) are compared. This process is repeated until all the characters of the pattern string have been matched. If \(T[1]\) and \(S[1]\) have different values, the pattern string will move a character backward. Then \(T[1]\) and \(S[2]\) are aligned. The process will be repeated until matching succeeds or the end of the main string is reached. In a word, assume that there exists a character \(X\) that is greater than or equal to 1 and less than or equal to \(n\), and if \(S[X + 1, X + 2, \ldots, X + m]\) is equal to \(T[1, 2, \ldots, m]\), then the matching is successful. If the hypothesis is not established, we say that the matching is not successful. The time complexity of BF algorithm is \(O(mn)\).

### 3.2 KMP algorithm

KMP algorithm\(^{12, 13}\) was proposed by Knuth et al. in 1977. The core idea of KMP algorithm is as follows. If matching fails, the main string character need not be traced back; instead, the pattern string is moved as far to the right as possible. Two points about KMP algorithm need to be stated: (1) The distance that the pattern string moves back is not just a single character length. (2) It is not necessary to begin with a new match from the first character of the pattern string after moving back; that is, pattern string can move backward a distance greater than a single character, and matching can start from any character of the pattern string. \(\text{Next}[j]\) is an important value in KMP algorithm. Suppose there exist two characters \(i\) and \(j\), where \(i\) and \(j\) are greater than or equal to 1, \(i\) is less than or equal to \(n\), and \(j\) is less than or equal to \(m\). If \(S[i]\) is not equal to \(T[j]\), the matching failed; then \(S[i]\) and \(T[\text{Next}[j]]\) should be aligned when the next matching is started. The calculation method of \(\text{Next}[j]\) is shown in Eq. (1).

\[
\text{Next}[j] = \begin{cases} 0, & j = 1; \\ \max\{x | 1 < x < j, T[1, 2, \ldots, x - 1] = T[j - x + 1, j - x + 2, \ldots, j - 1]\}, & \text{other situations} \\
\end{cases}
\]

For example: The value of pattern string \(T\) is "abcab"; the values of \(\text{Next}[j]\) are shown in Table 1.

KMP algorithm is efficient, but it is hard to understand. The time complexity of KMP algorithm is \(O(m + n)\).

### 3.3 BM algorithm

In 1977, Boyer and Moore proposed a new pattern searching algorithm (BM algorithm)\(^{14-17}\). BM algorithm is a very efficient string-searching algorithm, and its searching rate is three to five times as fast as that of KMP algorithm.

BM algorithm uses three ways to improve string-searching efficiency: (1) a "scanning from right to left" rule, (2) a "bad character" rule, and (3) a "good suffix" rule.

Scanning from right to left means that matching starts from the last character of the pattern string and moves forward each time to execute the next matching.

The "bad character" rule is as follows. During the process of scanning from right to left, suppose there exist two characters, \(i\) and \(j\), where \(i\) and \(j\) are greater than or equal to 1, \(i\) is less than or equal to \(n\), and \(j\) is less than or equal to \(m\). If \(S[i]\) is not equal to \(T[j]\), there will be two cases. (1) As shown in Fig. 2, if \(T[k]\) is equal to \(S[i]\), where \(k\) is greater than 1 and less than \(i\), the pattern string is moved to the right to keep \(T[k]\) and \(S[i]\) aligned, and then matching begins from right to left. (2) As shown in Fig. 3, if \(T[k]\) is not equal

| \(j\) | \(\text{Next}[j]\) | \(j\) | \(\text{Next}[j]\) | \(j\) | \(\text{Next}[j]\) |
|---|---|---|---|---|---|
| 1 | 0 | 3 | 1 | 5 | 2 |
| 2 | 1 | 4 | 1 | 6 | 3 |
to $S[i]$ all the time, the pattern string will be shifted to the right to keep $T[1]$ and $S[i+1]$ aligned, and then matching begins from right to left.

The “good suffix” rule: During the process of scanning from right to left, suppose that there exist two characters, $i$ and $j$, where $i$ and $j$ are greater than or equal to 1, $i$ is less than or equal to $n$, and $j$ is less than or equal to $m$. If $S[i]$ is not equal to $T[j]$, then search for the part that matched successfully and name it $A$ in $S$. These same strings with $A$ in $T$ are marked as $A'$. Then there will be two cases: (1) As shown in Fig. 4, if the number of $A'$ is greater than 1 and the previous character of the first $A'$ is different from the previous character of the next $A'$, then the pattern string should be moved to the right to make $A'$ and $A$ aligned. (2) As shown in Fig. 5, if the number of $A'$ is just equal to 1, a part named $X$, which has the same longest suffix with $A$, will be searched. Then $T$ is moved to the right to make $X$ and the same suffix of $A$ aligned.

### 3.4 Sunday string-searching algorithm

Sunday string-searching algorithm\(^{[18-22]}\), whose search speed is faster than that of BM algorithm, was proposed by Sunday in 1990. The core idea of this algorithm is as follows. Pattern string $T$ can be matched either from left to right or from right to left during the matching process. And when matching fails, Sunday string-searching algorithm shifts backward further to improve the matching efficiency.

Sunday string-searching algorithm is similar to BM algorithm. When matching fails, the character which is the next one after the character $S$ should be taken into consideration. $S$ is the character that aligns with $T[m]$. Meanwhile, $T[m]$ is the last figure of the pattern string. This means $T[1, 2, \ldots, m]$ need to be matched with $S[i, i+1, \ldots, i+m]$.

More precisely, once matching fails, the distance of moving backwards for $T[1, 2, \ldots, m]$ is determined by $S[i+m+1]$. The moving distance can be divided into two situations: (1) If $S[i+m+1]$ does not appear in pattern string $T$, $S[i+m+1]$ will be skipped. And the moving distance is equal to $m+1$. (2) If $S[i+m+1]$ appears in the pattern string $T$, the moving distance is equal to the distance between the character that is same as $S[i+m+1]$ at the bottom right of the pattern string and the last character $T[m]$ in pattern string $T$ plus 1. The distance that Sunday moves to the right is denoted as “move”, and how “move” is calculated is shown in Eq. (2); the matching process is shown in Fig. 6.

![Fig. 2 BM algorithm: Bad character rule shift 1.](image)

![Fig. 3 BM algorithm: Bad character rule shift 2.](image)

![Fig. 4 BM algorithm: Good suffix rule shift 1.](image)

![Fig. 5 BM algorithm: Good suffix rule shift 2.](image)

\[
\text{move} = \begin{cases} 
  m + 1, & S[i+m+1] \neq T[j], \\
  1 \leq j \leq m; & \min\{j | 1 \leq j \leq m\}, S[i+m+1] = T[j] \text{ and } S[i+m+1] = S[m-j] \end{cases} 
\]

(2)

### 4 Improved String-Searching Algorithm Based on the Monitor Log

The longest distance of existing Sunday string-searching algorithm moving backwards is one plus the length of the pattern string, which is obviously inadequate for handling the long monitor log. So a new algorithm is needed to handle the monitor log.

An improved string-searching algorithm based on the monitor log is proposed in this paper. We call it ML-Sunday string-searching algorithm. First we define the relevant variables, shown in Table 2, and explain their meanings.

The basic steps of ML-Sunday string-searching algorithm are as follows:

1. $TS$ is matched with ML from back to front in proper order. This means $TS_{m-1}$, $TS_{m-2}$, $TS_{m-3}$, $$, $TS_0$ will be matched with $ML_{pos+m-1}$, $ML_{pos+m-2}$, $ML_{pos+m-3}$, $$, $ML_{pos}$ in turn.

2. If the matching is successful, the matching process is over, and the location that matched success is
Table 2 Variable names and relevant meanings.

| Variable names | Meanings |
|----------------|----------|
| n              | Counts of monitor log dataset which is produced after running component test |
| m              | Counts of unsafe component test sequence set |
| j              | Subscript of character in Monitor Log (ML); initial subscript is 0 (0 ≤ j < n) |
| i              | Subscript of character in Test Sequence (TS); initial subscript is 0 (0 ≤ i < m) |
| move           | Figures by which TS is moved backwards when next match is executed; initial initial value is 0 |
| pos            | Subscript of the first character in ML when current match is being executed, pos = pos + move; initial value is 0 |
| next           | Subscript of ML_{pos+m-1}’s next character when current match is being executed, next = pos + m; initial value is 0 |
| TS_d           | Character that appears for the first time and is dissimilar to and furthest from TS_{m-1} |
| ML_d           | Character that is equal to TS_d |

(3) If TS_i is not equal to ML_j, the matching fails, and searching proceeds from TS_{m-1} to the front, to find the character that matches ML_{pos+m}.

(4) If TS_i equals ML_{pos+m}, then the value of “move” is equal to “move” plus m and minus i. If TS_i is not equal to ML_{pos+m}, then the value of “move” is equal to “move” plus m and plus 1.

(5) Search for TS_d from TS.

(6) Search for the next ML_d from the character that aligned with TS_d in ML to the back. Count the number of characters in the backward move, denoted by “move”.

(7) If “move” is greater than “move’”, then “pos” is equal to “pos” plus “move’”; If “move” is less than or equal to “move’”, then “pos” is equal to “pos” plus “move’”.

(8) The matching process repeats steps (1)–(7). If the result of “pos” plus m and minus 1 is greater than n, matching is unsuccessful; otherwise, the process returns to step (2).

In ML-Sunday string-searching algorithm, the distance to move backward is determined by the comparison result of “move” and “move’”. The matching number can be decreased by the moving-back distance. As the moving-back distance in ML-Sunday string-searching algorithm is longer than that in other string-searching algorithms, ML-Sunday is more suitable for large main strings. So, ML-Sunday is particularly appropriate for use with monitor logs.

The time complexity of ML-Sunday string-searching algorithm is the same as that of Sunday string-searching algorithm, $O(mn)$. The specific algorithm is shown in Algorithm 1.

5 Experiment Analysis

To verify the feasibility and effectiveness of ML-Sunday string-searching algorithm, matching experiments with monitor logs and security method sequences that have different counts of characters are conducted. All tests were run on a machine having an Intel dual core i3-2120 2.93 GHz processor, 2 GB of RAM, a 500 GB hard drive, and running Windows 7. The development environment is Visual Studio 2010. Each group of experiments is run 100 times, and the average number of matches is calculated.

Three candidate character sets with different sizes are listed in Table 3. The monitor log consists of the characters that belong to these three candidate character sets. These three sets are Long-can, Medium-can, and Short-can. Characters in Long-can stand for all keywords that appear in the monitor log.
Algorithm 1 ML-Sunday string-searching algorithm

1: **Input:** monitoring log set ML, unsafe component test sequence set TS
2: **Output:** pos
3: pos = 0;
4: while pos + m < n do
5:   flag = 1;
6:   for i = 0 to m do
7:     if TS_i \notin ML_pos+i then
8:       flag = 0;
9:     end if
10:   end for
11:   if flag = 1 then
12:     return pos;
13:   else
14:     forward scan the same character as ML_pos+m from TS_{m-1};
15:     if TS_i = ML_pos+m then
16:       move = move + m - i;
17:     else
18:       move = move + m + 1;
19:     end if
20:     find next ML_d from ML and the number of pattern string’s movement is move’;
21:     if move > move’ then
22:       pos = pos + move;
23:     else
24:       pos = pos + move’;
25:     end if
26:   end if
27: end while
28: return fail

Keywords can contain lowercase letters, uppercase letters, numbers, and special symbols. Characters in Medium-can represent method names in the monitor log. Method names can be combinations of uppercase letters and numbers. Characters in Short-can represent code segments that may exist in the monitor log. Code segments can comprise numbers and special symbols. In this experiment, the character counts (denoted by \( n \)) of the monitor log datasets are 2000, 5000, 10 000, 20 000, 50 000, and 100 000, respectively.

5.1 Long-can experiment data analysis

Strings in the monitor log datasets consist of random characters from Long-can. The character counts of the method sequence sets are 20, 30, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600, 650, 700, 750, and 800, respectively. The monitor log data is the main string and the method sequence set is the pattern string.

1) The average matching times of Sunday string-searching algorithm and ML-Sunday string-searching algorithm are shown in Figs. 7–12.

From Figs. 7–12, we can see that, with the increasing of the character count of pattern string, the overall average match times of Sunday string-searching algorithm and ML-Sunday string-searching algorithm are decreasing. When the character count of the pattern string is lower, with ML-Sunday

![Fig. 7 n=2000 for candidate set Long-can.](image)

![Fig. 8 n=5000 for candidate set Long-can.](image)
An Improved String-Searching Algorithm and Its Application in Component Security Testing

Jinfu Chen et al.

Fig. 9 \( n = 10,000 \) for candidate set Long-can.

Fig. 10 \( n = 20,000 \) for candidate set Long-can.

Fig. 11 \( n = 50,000 \) for candidate set Long-can.

Fig. 12 \( n = 100,000 \) for candidate set Long-can.

Fig. 13 Average matching time for Long-can.

Fig. 14 Average matching time for Long-can.

String-searching algorithm, the reduced matching time is relatively more, compared with Sunday string-searching algorithm. As the increasing of the character count of pattern string, the reduced time of matching is reducing. When the character count of pattern string reaches 650, the average matching times of ML-Sunday string-searching algorithm and Sunday string-searching algorithm are almost the same. What’s more, with the character count of pattern string continuously increasing, the average matching times of ML-Sunday string-searching algorithm and Sunday string-searching algorithm remain the same.

(2) The average matching times of ML-Sunday string-searching algorithm are shown in Fig. 13. Compared with Sunday string-searching algorithm, the rates of average matching times improved by ML-Sunday string-searching algorithm are shown in Fig. 14.

From Fig. 13, we can conclude that when the character count of a pattern string is constant, with the increase of the character count of a monitor log dataset, the average matching times of ML-Sunday string-searching algorithm grow. From Fig. 14 we can see that when the size of the monitor log dataset is constant,
Fig. 14 Improved efficiencies of ML-Sunday algorithm compared with Sunday algorithm for Long-can.

With the increase of the character count of the pattern string, the rate by which efficiency is improved by ML-Sunday string-searching algorithm decreases. When the character count of the pattern string is 650, the rate by which efficiency is improved by ML-Sunday string-searching algorithm is zero. And with the increase of the character count of the pattern string, the rate remains at zero. But when the character count of the pattern string is fixed, with the growth of the character count of the monitor log dataset, the rate of improvement provided by ML-Sunday string-searching algorithm is almost the same.

5.2 Medium-can experiment data analysis

Strings in monitor log datasets are made up of random characters from Medium-can. The character counts of the method sequence sets are 20, 30, 50, 100, 150, 200, 250, 300, 350, 400, 450, and 500, respectively. The monitor log data is the main string and the method sequence set is the pattern string.

(1) The average matching times of Sunday string-searching algorithm and ML-Sunday string-searching algorithm are shown in Figs. 15–20.

From Figs. 15–20, we can see that, compared with Sunday string-searching algorithm, when the character count of the pattern string is lower, with ML-Sunday string-searching algorithm, the matching time is lower. As the character count of the pattern string increases, matching time improves more. When the character count of the pattern string reaches 350, the average matching times of ML-Sunday string-searching algorithm and Sunday string-searching algorithm are almost the same. What’s more, with the character count of the pattern string increasing continuously, the average matching times of ML-Sunday string-searching algorithm and Sunday string-searching algorithm remain the same.
(2) The average matching times of ML-Sunday string-searching algorithm are shown in Fig. 21. Compared with Sunday string-searching algorithm, the rates of average matching times improved by ML-Sunday string-searching algorithm are shown in Fig. 22.

From Fig. 21, we can conclude that when the character count of a pattern string is constant, with the increase of the character count of the monitor log dataset, the average matching times of ML-Sunday string-searching algorithm grow. From Fig. 22 we can see that when the character count of the monitor log dataset is constant, with the increase of the character count of the pattern string, the rate by which efficiency is improved by ML-Sunday string-searching algorithm decreases. When the character count of the pattern string is 350, the rate by which efficiency is improved by ML-Sunday string-searching algorithm is zero. And with the growth of the character count of the pattern string, the rate remains at zero. But when the character count of the pattern string is constant, with the growth of the character count of the monitor log dataset, the rate of improvement brought by ML-Sunday string-searching algorithm is almost the same.
5.3 Short-can experiment data analysis

Strings in the monitor log datasets consist of random characters from Short-can. The character counts of the method sequence sets are 20, 30, 50, 100, 150, 200, 250, 300, 350, and 400, respectively. The monitor log data is the main string and the method sequence set is the pattern string.

(1) The average matching times of Sunday string-searching algorithm and ML-Sunday string-searching algorithm are shown in Figs. 23–28.

From Figs. 23–28, we can see that, when the character count of the pattern string is lower, with ML-Sunday string-searching algorithm, the improvement in performance compared with Sunday string-searching algorithm is greater. As the character count of the pattern string increases, the efficiency advantage of ML-Sunday string-searching algorithm drops. When the character count of the pattern string reaches 250, the average matching times of ML-Sunday string-searching algorithm and Sunday string-searching
algorithm are almost the same. What’s more, with the character count of the pattern string continuously increasing, the average matching times of ML-Sunday string-searching algorithm and Sunday string-searching algorithm remain the same.

(2) The average matching times of ML-Sunday string-searching algorithm are shown in Fig. 29. A comparison of Sunday string-searching algorithm results with the rates of ML-Sunday string-searching algorithm is shown in Fig. 30.

From Fig. 29, we can conclude that when the character count of the pattern string is constant, the average matching time of ML-Sunday string-searching algorithm increases with the growth of the character count of the monitor log dataset. From Fig. 30, we can see that when the character count of the monitor log dataset is constant, the performance improvement brought by ML-Sunday string-searching algorithm decreases with the increase of the character count of the pattern string. When the character count of the pattern string is 250, the rate of efficiency improvement coming from ML-Sunday string-searching algorithm is zero. And with the growth of the character count of the pattern string, the rate remains zero. But when the character count of the pattern string is constant, with the increase of the character count of the monitor log dataset, the rate of improvement resulting from ML-Sunday string-searching algorithm is almost the same.

5.4 Overall data analysis

The characters in the monitor log dataset are chosen from Long-can, Medium-can, and Short-can, and the strings are formed with 2000, 5000, 10000, 20000, 50000, and 100000 counts. The character counts of the insecure method sequence (the pattern string) are 20, 30, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600, 650, and 700, respectively. Compared with Sunday string-searching algorithm, the improved efficiencies of ML-Sunday searching algorithm are shown in Table 4.

From Table 4, we can see that, when the character counts of monitor log dataset and pattern string are both constant, the count of the character candidate set is bigger, the improved efficiency of ML-Sunday string-searching algorithm is greater, and the improved interval of efficiency is greater.

5.5 Time analysis of ML-Sunday string-searching algorithm

Overall, ML-Sunday string-searching algorithm spends more time than Sunday string-searching algorithm takes in the process of matching, because ML-Sunday string-searching algorithm takes more time to find the next
ML<sub>d</sub>. Times spent by ML-Sunday string-searching algorithm when character candidate sets are Long-can, Medium-can, and Short-can are shown in Tables 5–7. The character counts of the insecure method sequence we take are 20, 30, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600, 650, and 700, respectively. The unit of time in these tables is milliseconds.

From Tables 5–7, we can see that when the character count of the monitor log dataset is constant, the time spent by ML-Sunday string-searching algorithm grows with the increase of the character count of the pattern string. And when the character count of the pattern string is constant, the time spent by ML-Sunday string-searching algorithm grows as well, with the growth of the character count of the monitor log dataset. What’s more, when the character count of the monitor log dataset and the pattern string are both constant, the bigger the character candidate set, the more time ML-Sunday string-searching algorithm takes.

### Table 5 Time spent by ML-Sunday string-searching algorithm when character candidate sets are Long-can.

| Candidates | 20 | 30 | 50 | 100 | 150 | 200 | 250 | 300 | 350 | 400 | 450 | 500 | 550 | 600 | 650 | 700 | Ave |
|------------|----|----|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Long-can   | 62.9 | 53.7 | 42.4 | 30.5 | 25.8 | 22.7 | 19.8 | 16.2 | 12.8 | 10.2 | 7.6 | 4.8 | 3.5 | 2.2 | 0.2 | 0 | 19.7 |
| Medium-can | 44.6 | 36.7 | 27.9 | 18.4 | 12.2 | 6.9 | 4.3 | 1.9 | 0.1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 9.6 |
| Short-can  | 30.4 | 24.4 | 17.7 | 11.4 | 5.4 | 2.5 | 1.6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5.1 |
| Ave        | 46.0 | 38.3 | 29.3 | 17.7 | 13.5 | 10.4 | 8.0 | 6.0 | 4.3 | 3.4 | 2.6 | 1.6 | 1.2 | 0.7 | 0.1 | 0 | 11.5 |

### Table 6 Time required by ML-Sunday string-searching algorithm when character candidate sets are Medium-can.

| m | 2000 | 5000 | 10000 | 20000 | 50000 | 100000 |
|---|------|------|-------|-------|-------|--------|
| 20 | 0.0587 | 0.2062 | 0.6858 | 2.2285 | 13.2315 | 39.6851 |
| 30 | 0.0633 | 0.2121 | 0.7011 | 2.3006 | 13.5926 | 40.8810 |
| 50 | 0.0677 | 0.2256 | 0.7325 | 2.3262 | 14.1772 | 42.6470 |
| 100 | 0.1011 | 0.2811 | 0.8193 | 2.4948 | 16.7481 | 46.6477 |
| 150 | 0.1827 | 0.4575 | 0.9830 | 3.1808 | 18.2871 | 52.2292 |
| 200 | 0.2863 | 0.7341 | 1.8709 | 4.6133 | 21.6525 | 58.9644 |
| 250 | 0.5527 | 1.0870 | 3.4764 | 6.3894 | 27.1318 | 63.0525 |
| 300 | 0.7477 | 2.5755 | 5.1023 | 11.4015 | 38.0451 | 91.3404 |
| 350 | 1.2581 | 4.2115 | 6.7885 | 15.553 | 60.0951 | 108.9429 |
| 400 | 2.2530 | 5.7501 | 11.4254 | 25.2706 | 75.2369 | 150.3701 |
| 450 | 3.2941 | 9.0589 | 20.3124 | 37.4217 | 101.1515 | 204.3588 |
| 500 | 3.9434 | 11.1645 | 23.8968 | 46.533 | 154.8158 | 263.9857 |
| 550 | 5.0163 | 13.7981 | 31.7572 | 65.9019 | 162.3982 | 350.7341 |
| 600 | 7.0098 | 17.4997 | 41.0245 | 80.6399 | 228.4169 | 407.6928 |
| 650 | 7.7608 | 23.2089 | 43.6171 | 102.0929 | 269.7719 | 468.1501 |
| 700 | 8.0294 | 25.4761 | 57.5669 | 112.449 | 326.4544 | 572.0296 |

### Table 7 Time required by ML-Sunday string-searching algorithm when character candidate sets are Short-can.

| m | 2000 | 5000 | 10000 | 20000 | 50000 | 100000 |
|---|------|------|-------|-------|-------|--------|
| 20 | 0.1602 | 0.6796 | 2.0006 | 7.3084 | 52.8332 | 269.1415 |
| 30 | 0.2114 | 0.7535 | 2.2713 | 8.0466 | 58.9347 | 285.7428 |
| 50 | 0.4428 | 1.2317 | 3.1391 | 9.7617 | 70.4583 | 335.6923 |
| 100 | 1.5327 | 4.2110 | 9.3823 | 21.4595 | 104.1241 | 433.3322 |
| 150 | 3.4911 | 9.3142 | 20.905 | 45.6053 | 162.7953 | 567.0276 |
| 200 | 5.2956 | 14.7592 | 34.1536 | 67.8998 | 227.8529 | 718.3699 |
| 250 | 8.0313 | 21.5599 | 45.2144 | 99.2325 | 278.3657 | 855.4266 |
| 300 | 9.9704 | 28.1402 | 56.6563 | 123.1153 | 382.7558 | 993.5048 |
| 350 | 12.1509 | 35.5100 | 73.6702 | 143.2670 | 458.2050 | 1181.9049 |
| 400 | 14.0836 | 40.3026 | 86.8805 | 177.0644 | 505.3039 | 1392.5375 |
| 450 | 15.7534 | 47.5116 | 103.9259 | 209.0146 | 593.5890 | 1587.5654 |
| 500 | 18.1744 | 54.6923 | 117.2287 | 247.6013 | 708.0994 | 1718.9824 |
| 550 | 20.3775 | 61.3908 | 131.1796 | 276.8190 | 771.9475 | 1852.9698 |
| 600 | 23.0666 | 71.0783 | 147.2037 | 305.7423 | 877.4052 | 2128.0236 |
| 650 | 24.4680 | 73.7036 | 166.6765 | 350.8208 | 967.4448 | 2385.6909 |
| 700 | 25.5388 | 85.2832 | 187.1652 | 389.4950 | 1085.8076 | 2501.2327 |
6 Conclusion

Generally speaking, it is hard to obtain the source code and the specifications of third-party components. This brings great challenges to testing the security of third-party components. Pattern sequence matching modules, studied in this paper, are an important part of component security testing. During the matching process, the monitor log is set as the main string, and the keyword, also called the “insecure execution sequence”, is seen as the pattern string. Then ML-Sunday string-searching algorithm, proposed in this paper, is used to effect pattern sequence matching. If the matching is successful, the location matched successfully is returned. As a result, these method execution sequences are shown to have insecure sequences, and thus, security vulnerability. If the matching is unsuccessful, then ML-Sunday string-searching algorithm will locate the character which occurs for the first time in the pattern string and differs from the last character in the pattern string, and the distance between this character and the last character in pattern string is the furthest. ML-Sunday string-searching algorithm will then search for the next character, the same as the character that was found in the pattern string, from the character that aligned with this character in the main string. Then the distance will be noted, denoted as “move’”. If “move’” is greater than “move”, the distance of moving backwards for the pattern string is “move’”. Otherwise, the distance of moving backwards for the pattern string is “move”. The efficiency of component security testing is improved by using ML-Sunday string-searching algorithm through increasing the distance of moving backwards, and the matching efficiency is thereby increased by 11.5% on average.

In the case that the length of the main string is relatively small but the length of the pattern string is relatively big, the improved efficiency of ML-Sunday string-searching algorithm is limited. In the future, we will improve the proposed algorithm to facilitate component security testing.

Acknowledgment

This work was partly supported by the National Natural Science Foundation of China (Nos. 61202110 and 61502205), the Postdoctoral Science Foundation of China (Nos. 2015MS1687 and 2015MS1739), and the Graduate Research Innovation Project of Jiangsu Province (No. KYLX15_1079).

References

[1] J. F. Chen, Y. S. Lu, and H. H. Wang, Component security testing approach based on extended chemical abstract machine, International Journal of Software Engineering and Knowledge Engineering, vol. 22, no. 1, pp. 59–83, 2012.
[2] L. C. Briand, Y. Labiche, and M. M. Sówka, Automated, contract-based user testing of commercial-off-the-shelf components, in Proc. 28th Int. Software Engineering, Shanghai, China, 2006, pp. 92–101.
[3] J. F. Chen, H. H. Wang, D. Towey, C. Y. Mao, R. H. Huang, and Y. Z. Zhan, Worst-input mutation approach to web services vulnerability testing based on SOAP messages, Tsinghua Science and Technology, vol. 19, no. 5, pp. 429–441, 2014.
[4] C. Y. Mao and Y. S. Lu, Research progress in testing techniques of component-based software, Journal of Computer Research and Development, vol. 43, no. 8, pp. 1375–1382, 2006.
[5] J. F. Chen, Y. S. Lu, and X. D. Xie, Testing approach of component security based on fault injection, in Proc. Int. Computational Intelligence and Security, Harbin, China, 2007, pp. 763–767.
[6] J. F. Chen, Y. S. Lu, and X. D. Xie, A fault injection model of component security testing, Journal of Research and Development, vol. 46, no. 7, pp. 1127–1135, 2009.
[7] B. Potter, G. M. cryaw, and B. Allen, Software security testing, IEEE Security & Privacy, vol. 2, no. 5, pp. 81–85, 2004.
[8] D. R. Kuhn and A. M. Gallo, Software fault interactions and implications for software testing, IEEE Transactions on Software Engineering, vol. 30, no. 6, pp. 418–421, 2004.
[9] M. G. Wang, J. J. Jie, T. X. Shi, and X. Fang, An agent-based autonomous component model for internetware, in Int. Web Information Systems and Mining, Nanjing, China, 2010, pp. 348–351.
[10] P. Inverardi and A. L. Wolf, Formal specifications and analysis of software architectures using the chemical abstract machine model, IEEE Transactions on Software Engineering, vol. 21, no. 4, pp. 373–386, 1995.
[11] A. Ju and A. Wang, Security testing in software engineering courses, in Proc. 34th ASEE/IEEE Frontiers in Education Conference, California, USA, 2004, pp. 13–18.
[12] D. E. Knuth, J. H. Morris, and V. R. Pratt, Fast pattern matching in strings, SIAM Journal on Computing, vol. 6, no. 2, pp. 323–350, 1977.
[13] B. Durian, J. Holub, H. Peltoila, and J. Tarhi, Improving practical exact string matching, Information Processing Letters, vol. 110, no. 4, pp. 148–152, 2010.
[14] R. S. Boyer and J. S. Moore, A fast string-searching algorithm, Communications of the ACM, vol. 20, no. 10, pp. 762–772, 1977.
[15] S. Tevatia, R. Prasad, and D. Rai, An offensive algorithm for multi-pattern parameterized string matching, in Int. Control, Computing, Communication and Materials, 2013, pp. 1–5.
[16] S. Faro and M. O. Külekci, Fast multiple string matching using streaming SIMD extensions technology, in String Processing and Information Retrieval, Springer-Verlag Berlin Heidelberg, 2012.

[17] T. Lecroq, Fast exact string matching algorithms, Information Processing Letters, vol. 102, no. 6, pp. 229–235, 2007.

[18] D. M. Sunday, A very fast substring search algorithm, Communications of the ACM, vol. 33, no. 8, pp. 132–142, 1990.

[19] S. Faro and T. Lecroq, Efficient pattern matching on binary strings, in Proc. 35th Int. Current Trends in Theory and Practice of Computer Science, Spindleruv Mlyn, Czech Republic, 2009.

[20] S. Faro and M. O. Külekci, Fast packed string matching for short patterns, in Proc. 15th Meeting on Algorithm Engineering and Experiments, New Orleans, USA, 2013, pp. 113–121.

[21] S. Faro and T. Lecroq, The exact online string matching problem: A review of the most recent results, ACM Computing Surveys, vol. 45, no. 2, p. 13, 2013.

[22] H. Le and V. K. Prasanna, A memory-efficient and modular approach for large-scale string pattern matching, Proceedings of IEEE Conference on Computers, vol. 62, no. 5, pp. 844–857, 2013.

Jinfu Chen is an associate professor in the School of Computer Science and Communication Engineering, Jiangsu University, China. He received the PhD degree in computer science and technology from Huazhong University of Science and Technology, China, in 2009. His major research interests include software testing, software analysis, and trusted software. He is a member of the ACM, IEEE CS, and China Computer Federation.

Saihua Cai is a master student in the School of Computer Science and Communication Engineering, Jiangsu University, China. He received the BE degree from Huaiyin Normal University, China, in 2013. His research interests include software testing and service computing.

Lili Zhu is a master student in the School of Computer Science and Communication Engineering, Jiangsu University, China. She received the BE degree from Jiangsu University, China, in 2014. Her research interests include software testing and service computing.

Yuchi Guo is a master student in the School of Computer Science and Communication Engineering, Jiangsu University, China. She received the BE degree from Huaiyin Institute of Technology, China, in 2013. Her research interests include software testing and service computing.

Rubing Huang is an assistant professor in the Department of Software Engineering, School of Computer Science and Communication Engineering, Jiangsu University, China. He received the PhD degree in computer science and technology from Huazhong University of Science and Technology, China, in 2013. His current research interests include software testing and software maintenance, especially combinatorial interaction testing, random testing, adaptive random testing, and test case prioritization. He is a member of the IEEE, the ACM, and the China Computer Federation.

Xiaolei Zhao is a master student in the School of Computer Science and Communication Engineering, Jiangsu University, China. He received the BE degree from Jiangsu University, China, in 2013. His research interests include software testing and service computing.

Yunqi Sheng is an undergraduate student in the School of Computer Science and Communication Engineering, Jiangsu University, China. Her research interests include software testing and service computing.