MIRD: Trigram-Based Malicious URL Detection Implanted with Random Domain Name Recognition

Cuiwen Xiong[1,3], Pengxiao Li[4], Peng Zhang[1,2(✉)], Qingyun Liu[1,2], and Jianlong Tan[1,2]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
{xiongcuiwen, pengzhang, liuqingyun, tanjianlong}@iie.ac.cn
[2] National Engineering Laboratory for Information Security Technologies, Beijing, China
[3] University of Chinese Academy of Sciences, Beijing, China
[4] National Computer Network Emergency Response Technical Team, Beijing, China
lpx@cert.org.cn

Abstract. This paper proposed an approach of malicious URL detection using trigrams-based common pattern of URL, which implanted with random domain recognition, named MIRD. In MIRD the common patterns were composed of three segments common patterns of URL, namely domain segment, path name segment and file name segment. An inverted index based on trigrams was used to improve common pattern extraction of each segment. MIRD used the common patterns based on inverted index to match with the detected URL. Moreover, MIRD implanted with Random Domain Name Recognition Module, named RDM. The RDM identified the length of the domain name and resolved the domain name in iteration to recognize the domain name unresolved, reducing the cumulative error rate of malicious URL detection. Extensive experiments showed that the MIRD is efficient and scalable.

Keywords: Malicious URL detection · Common pattern · Trigram · Inverted index · Radom domain name

1 Introduction

With the development of internet technology, the means of cybercrime is varying, and the form of network threat which promotes cybercrime also becomes more diversity. As a result, the new technology of network threat detecting need to be proposed to cope with the change. The report of Ponemon Institute showed that the occurrence of cyber-attacks on the enterprises or organizations was about 122 every week in 2013 and the financial loss has increased by nearly 78 % in 2013 than the past four years and the State cost more than one million on average to fix up with cyber-attacks [1]. The cybercrime group improved their technology to attack the strengthened intrusion

W. Niu et al. (Eds.): ATIS 2015, CCIS 557, pp. 303–314, 2015.
DOI: 10.1007/978-3-662-48683-2_27

prevention system (IPS), so proposing an efficient technology to detect network threat is necessary. A key feature of network threat is using other protocols and components to access internet while all of the protocols and components use HTTP or HTTPS protocol to transport their pages [2], which makes detecting network threats based on URLs feasible. The research of Le et al. also supported detecting network threats based on URLs is feasible [3]. Otherwise, the attackers hide themselves by some means. For example, Porras et al. randomly generated range from 250 to 50,000 domains using current time and date as seeds [4, 5]. It is more difficult to detect the malicious URLs with random domain. On the other hand, the similarities between malicious URLs and benign URLs mislead users. Users may stray the malicious web site with "login" changed to "log1n" or "index" changed to "1ndex" leading to leak personal information. To prevent privacy disclosure, the malicious URLs must be detected when users click them. A qualified malicious URL detection should meet the followings.

1. Effectiveness, an approach must detect malicious URLs as accurate as possible. The amount of malicious URLs is far less than the benign URLs on the internet. Some malicious URLs are different with benign URLs only in some letter. The two things make it difficult to detect malicious URL accurately.
2. Efficiency, an approach must detect malicious URLs as soon as possible. If a client clicked one URL, the browser sends a request to the server and the server responses the content the client wants. The approach must alert the user that the URL is malicious before the corresponding server responses.
3. Scalability, an approach must detect new malicious URLs as many as possible. The attackers use Domain Generation Algorithm to generate numerous URLs such that avoid be detected. The approach should detect new and uncommon malicious URL.

The paper proposed an approach of trigrams-based malicious URL detection implanted with random domain name recognition, MIRD. MIRD is only based on lexical feature of URL, saving time on extracting extra features, such as WHOIS and geographical location information of each URL. MIRD extracted common patterns of URLs. According to the URL Reference [6], one URL consists of alphanumeric and specific characters such as "?", "=", "–", "_". MIRD created dynamical inverted index with the trigrams of URLs as terms to quickly search URLs which the common pattern can extract from. The inverted index was conducive to avoid the length of common pattern too short. MIRD detected the URL matching up with the malicious common patterns as malicious one. MIRD implanted with random domain name recognition to recognize the random domain name, improving the accuracy of the approach. The paper is organized as follows: Sect. 2 introduces some related works and the contributions. Section 3 introduces the principle of MIRD in details. Section 4 illustrates experiments and analysis. Section 5 makes a conclusion of the research.

2 Related Works

The malicious URL detection can be divided into three categories according to the object been detected, namely the blacklist based approaches, the content based approaches and the URL based approaches.

The blacklist based approaches are simple and accurate in maintaining blacklist, such as Google Safe Browser [7], Netcraft Tool Bar and eBay Tool Bar. Zhang et al. introduced the principle of blacklist in [8]. If the detected URL exists in blacklist, the URL is malicious. That makes it suitable for existed network threats but impossible to detect the new malicious URL.

The content based approaches extract features from the web page. Provos et al. [9], Liu et al. [10] judged the JavaScript exists and whether iFrames is out of range or not to detect malicious URL. Zhang et al. analyzed the TF-IDF of every term in page and then used lexical labels to detect that the website is malicious or not [11]. The approach is appropriate for offline analyze for analyzing content of page inducing significant delay not suit for the high-speed online detection.

The URL based approaches classify URL based on features such as the length of URL and geolocation of URL etc. Garera et al. analyzed the structure of URL of phishing, chose 18 features, classified URL using Logistic regression filter [12]. Ma et al. analyzed lexical feature and host property of suspicious URL, obtained thousands of features using bag of words and proposed the classical algorithm CW [13–15]. These proposed approaches are delay for obtaining features from remote host and cannot process the malicious URL with random domain name.

The MIRD is based on URL, only considers the lexical feature of URL, uses common pattern extracted from URL as lexical feature matching up with detected URL and determines whether the detected URL is malicious or not. The MIRD has the following two contributions. On one hand, using dynamical inverted index accelerates the process and avoids redundancy computation searching common patterns which may match with the detected URL. On the other hand, introducing the random domain name recognition improves the accurate of detection detecting malicious URL with random domain name which cannot be resolved.

3 The Principle of MIRD

3.1 Concepts and Definitions

This section introduces the concepts involved in MIRD. According to the URL Reference [6], URL contains three irrelevant segments with different meanings—domain name, path name, file name. MIRD takes the three segments separately into consideration when computing common patterns. MIRD defines a normal characters set containing the alphanumeric and specific characters such as "?", " = ", "-", "_", "&", and the elements of the normal set is regarded as normal characters except the "/" as segments connector or delimiter of path name and "." as delimiter of domain name and file name. The all three segments are constituted by the characters from the normal set, so MIRD extract separately common patterns of the three segments with same rule. The common pattern of domain name, path name and file name was denoted by s_d, s_p, s_f separately.

Definition 1 (Segment common pattern). A segment common pattern is a string constituted by normal characters, denoted by $s = c_1 \cdots c_1$, where l is the length of segment common pattern, and c_i is a normal character or wildcard meta-symbol "*",

which can match with a string of random length of normal characters except the special character "/" or ".". For any $i(1 \le i < l)$, if c_i = "*", $c_i \neq$ "*".

Rule 1 (Segment matching up with segment common pattern). For a segment common pattern $s = c_1 \cdots c_l$ and segment $u = c'_1 \cdots c'_m$, if there is a function $f : [1, m] \rightarrow [1, l]$ such that: $1, l \le m, 2$, for any $j(1 \le j < m)$, $f(j) \le f(j+1), 3$, for any $i(1 \le i < l)$, if $c_{i+1} \neq$ "*", there are only one $j(1 \le j < m)$, such that $f(j) = i$ and $c'_j = c_i$. The segment u matches up with the common pattern s.

Just as URL common pattern is based on the three segment common patterns, URL matching up with URL common pattern is based on the three segment common patterns.

Definition 2 (URL common pattern). URL common pattern, shorted as common pattern, is consisted of the three segment patterns with the same URL ID, denoted by $p = s_d/s_p/s_f$.

Rule 2 (URL matching up with common pattern). If the three segments of URL match up with the corresponding segment of common pattern respectively, MIRD considers the URL matching up with the common pattern.

3.2 The Procedure of MIRD

The URL data set used by the paper was crawled at specific time by the open source software Larbin [16] from the specified website. The whole procedure of MIRD from obtaining data set to detecting malicious URL is showed in Fig. 1. The MIRD consists of three modules—Data Set Preprocessing, Common Pattern Extraction and URL Determination, implanted with Random Domain Name Recognition. The Data Set Preprocessing segregates URL into three segments—domain name, path name and file name. The Common Pattern Extraction splits segment into trigrams, creates inverted index of segments, extracts common pattern of each segments and connects common patterns to obtain URL common pattern. The URL Determination matches detected URL with common pattern to determinate the URL is malicious or not. Finally, the Random Domain Name Recognition improves the accuracy of detection.

After MIRD obtained URL data set, MIRD invokes the Data Set Preprocessing to segregate the URL into three segments. Then, the Common Pattern Extraction processes each segment in the same step. Taking domain name for example, the Common Pattern Extraction splits domain into trigrams, creates dynamically inverted index with trigrams as term. The reason of choosing trigram as term is that the probabilities of two arbitrary URL shared same bigram and 4-gram are 95.7 % and 33.6 %, while shared same trigram is 75.8 % [17]. To avoid the common pattern too short or computed too frequently, common pattern can be extracted from two segments shared at least one trigram. The Common Pattern Extraction extracts domain segment common pattern based on the domain inverted index, and domain segment common pattern meets the Definition 1 described in Sect. 3.1. The Common Pattern Extraction extracts the common patterns of path name and file name in the same step, then connects segment common patterns sharing same URLIDs to generate common pattern. Finally, the URL

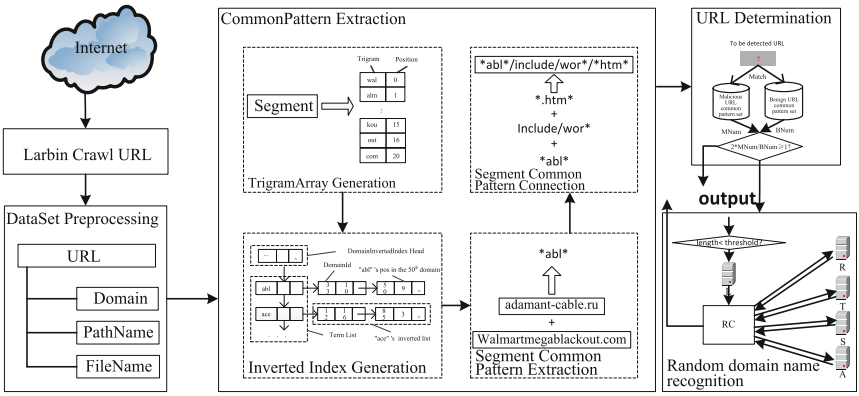

Fig. 1. The principle of the MIRD.

Determination matches the detected URL up with common pattern using the rule 1 and 2, then determines the detected URL and outputs the result. The Random Domain Name Recognition further detects the malicious URL containing with random domain name which cannot be detected by the above step, introduced in detail in Sect. 3.3. The following introduces the three modules in detail.

1. Data Set Preprocessing
   The MIRD invokes open source software Larbin [16] at specific time to crawl URLs from specified website to obtain two kinds of data set with malicious URLs and benign URLs, then process two kinds of data set separately. Considering all processed URLs using http or https protocol, the module delete the "protocol://" from the URL strings. Then, the module segregates the URL into domain name, path name and file name. For example, URL_0 "walmartmegablackout.com/include/wordpress/login.htm" is segregated into $Domain_0$ "walmartmegablackout.com", $Path_0$ "include/wordpress" and $FileName_0$ "login.htm". URL_1 "adamant-cable.ru/include/world/index.html" is segregated into $Domain_1$ "adamant-cable.ru", $Path_1$ "include/world" and $FileName_1$ "index.html". After processed by this module, MIRD gets two kinds of segment sets, one from malicious URL set and another from benign URL set. Malicious segment sets contain domain set MDomain, path set MPath and file name set MFile. Benign segment sets contain domain set BDomain, path set BPath and file name set BFile.
2. Common Pattern Extraction
   The Common Pattern Extraction of MIRD contains the following four steps:
   (1) **Trigram Array Generation:** The Common Pattern Extraction module processes the segment set one by one in the same way, taking domain segment for example. To avoid the common pattern too short and extracted too frequently, this module splits domain segment into trigrams, then creates dynamically inverted index with trigrams as term. This module splits domain into several trigrams using Algorithm 1, where Count is used to store the number of trigrams and its value is not greater than the length of domain minus 2, the

TrigramArray is a structure array to store the trigrams and their position. The algorithm uses the character "." as a delimiter split domain into substrings, which ensures each trigram come from one level of domain, then splits each substring into trigrams and saves into trigram array. The time complexity of the algorithm is linear for just traversing one segment twice. Figure 2(a) shows the TrigramArray of $Domain_0$. This module splits file name segment and path name segment in the same way, but uses the character "/" as a delimiter split path name into substrings.

Algorithm 1. SplitDomainIntoTrigrams

```
01: Input: Domain, Count
02: Output: an array of trigrams of Domain
03: initial TrigramArray← ø
04: CurrentPos = 0
05: Split Domain into SubStrings
06:   while SubString != ø
07:     if strlen(SubString) ≥ 3
08:       Split SubString into successive Trigrams
09:       TrigramArray←Trigram and Position
10:     else
11:       SubString regard as Trigram
12:       TrigramArray←Trigram and Position
13:     end if
14:     Count++
15:   end while
16: return TrigramArray and write back Count
```

(2) **Inverted Index Generation:** When one trigram array was generated, the module adds the domain and its trigrams into inverted index of domain DomainInvertedIndex, whose structure showed in Fig. 2(b). The node of index term contains a Trigram, a pointer to the next node of the term list and a pointer to the inverted list of this term. The node of inverted list contains DomainId,

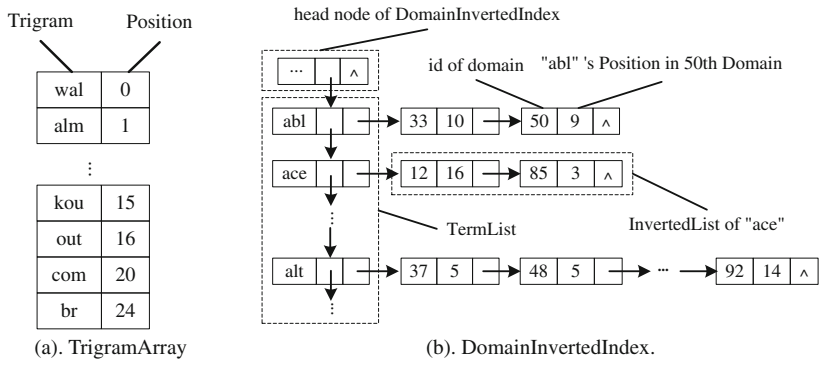

Fig. 2. The primary data structure of MIRD.

Position of the trigram in domain and a pointer to the next node of the inverted list. The inverted list is sorted by ascending order of DomainId. The module also generates dynamically inverted index of path name PathInvertedIndex and inverted index of file name FileInvertedIndex.

Algorithm 2. ExtractDomainCommonPattern

```
01: Input: DomainId_1, DomainId_2
02: Output: the common patter of Domain_1 and Domain_2
03: initial CommonString←ø
04: Traverse TrigramArrays of Domain_1 and Domain_2
05:   if TrigramArray_1[i].Trigram == TrigramArray_2[j].Trigram
06:     if CommonString == ø
07:       Judge( Position_1i, Position_2j)
08:         CommonString= Trigram | ".*"+Trigram | "*"+Trigram
09:     else
10:       Judge( Position_1i, Position_2j)
11:         Append(CommonString, the last letter of Trigram | "*."+Trigram | "*"+Trigram
                  | Trigram)
12:     end if
13:     Position_1 = Position_1i; Position_2 = Position_2j;
14:   end if
15: if any position is not the last Trigram of two Domain
16:   CommonString append "*"
17: end if
18: return CommonString
```

(3) **Segment Common pattern Extraction:** This module traverses the DomainInvertedIndex to search the domains where domain segment common pattern can be extracted from. Segment common pattern can be extracted from segments in the same inverted list for these domains share one trigram at least. Running algorithm 2, this module scans two trigram arrays of two domains to extract segment common pattern. According to the experimental statistic, the probability of more than one segment common pattern between two domains is not exceeding 2 % [17]. So, extracting one segment common pattern from two domains is feasible. The extraction of segment common pattern is computed by the position of trigrams in the two segments. According to the positions, this module appends the trigram or character of trigram or wildcard to the common string. The time complexity of the algorithm is polynomial without beyond the product of the length of two arrays. For example, domain common pattern of $Domain_0$ and $Domain_1$ is "*abl*". At last, all domain segment common patterns are deposited into DomainPatternSet. Using Algorithm 2 process path name and file name obtains path name common pattern set PathPatternSet and file name common pattern set FilePatternSet. Path name common pattern of $Path_0$ and $Path_1$ is "include/wor*", and file name common pattern of $File_0$ and $File_1$ is "*.htm*".

(4) **Segment Common Pattern Connection:** This module connects segment common patterns shared the same URLID using segment connector "/". For example, the common pattern of URL_0 and URL_1 is "*abl*/include/wor*/*.htm*". All common patterns of malicious URLs and benign URLs input into the URL Determination module.

3. URL Determination
   The URL Determination module matches the detected URL up with two kinds of common patterns following the rule 1 and 2 described in Sect. 3.1. The ratio of training set of malicious URL and benign URL is 0.5. The number of the detected URL matching up with malicious common patterns is denoted as MNum, and the number of the detected URL matching up with benign common pattern is denoted as BNum. According to the experimental statistic, the size of the intersection between malicious common pattern set and benign set is less than 1 %. This module uses linear classifier to determinate the detected URL is malicious or not. If $2 * \mathrm{MNum}/\mathrm{BNum} \geq 1$, the detected URL is malicious. To improve the accuracy, if $2 * \mathrm{MNum}/\mathrm{BNum} < 1$, this module outputs the detected URL into the Random Domain Name Recognition Module.

3.3 Random Domain Name Recognition

The cyber-attackers use domain name generation algorithm (DGA) generating a large number of URLs with random domain periodically, which share one server. The random domain is not predefined binary string and its length is often longer than the normal length of domain described in reference [18]. On the other hand, the domain user registers a domain for a period of time, and the time to live of random domain is short. If the time limit was exceeded, it is difficult to re-access the domain name, and the DNS cannot resolve the domain name. Most importantly, the attackers deliberately hide the related information of the domain name.

For the sake of the above three features of random domain name, the Random Domain Name Recognition module (RDM) processes the URL which was considered as benign URL by the URL Determination module as the Fig. 3. After received the URL, RDM checks the length of the domain name. If the length of the domain name is longer than 26, which is the threshold of domain name stipulated in reference [18], RDM outputs the detected URL is malicious. Otherwise, RDM resolves the domain name as iterative, and checks out the benign URL step by step, avoiding redundant resolving and enhancing the whole speed. RDM runs a thread of resolving controller to control the progress of domain name resolving. And the progress of domain name resolving is as follows.

(1) Checking the local domain name server (local DNS). The local DNS cached related information, such as the IP address of the host, after a URL was visited. If the local DNS check out the information about the domain, responds that the corresponding URL is benign. Otherwise, RDM sends the domain name to the Resolving Controller.

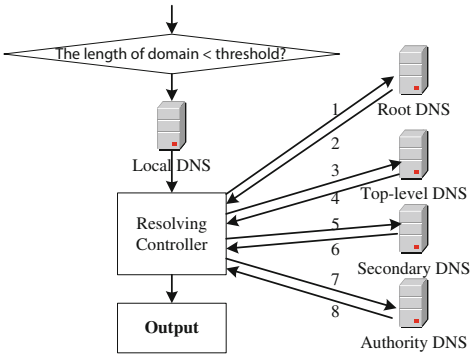

Fig. 3. The progress of domain name resolving.

(2) Checking the root domain name server (root DNS). The Resolving Controller sends a command to the root DNS. The root DNS responds the controller the address information about the top level domain name server.
(3) Checking the top level domain name server (top-level DNS). The Resolving Controller sends the domain name to the top-level DNS. The top-level DNS checks its cache. If the domain name in the cache, responds that the corresponding URL is benign. Otherwise, the top-level DNS responds the address information of the secondary domain name server to the controller.
(4) Checking the secondary domain name server (secondary DNS). The Resolving Controller sends the domain name to the secondary DNS. The secondary DNS checks its cache. If the domain name in the cache, responds that the corresponding URL is benign. Otherwise, the secondary DNS responds the address information of the authority domain name server to the controller.
(5) Checking the authority domain name server (authority DNS). The Resolving Controller sends the domain name to the authority DNS. The authority DNS checks its cache. If the domain name in the cache, responds RDM that the corresponding URL is benign. Otherwise, the authority DNS responds RDM that the corresponding URL is malicious.

4 Experiments and Analysis

There were two kinds of URL data set, namely malicious URL data set and benign URL data set crawled from specific websites by open source software Larbin [16] on October 13th, 2014. The malicious URL data set contained 4,000,000 URLs crawled from Phish Tank [19] and Malware Patrol [20]. The benign URL data set contained 8,000,000 URLs crawled from Google and DMOZ. The source of testing sets was the same with training sets, which contained 800,000 malicious URLs and 1,200,000 benign URLs. For each experiment, MIRD randomly chose malicious and benign URLs in proportion of 1 to 2 in training sets and 2 to 3 in testing set. The experimental

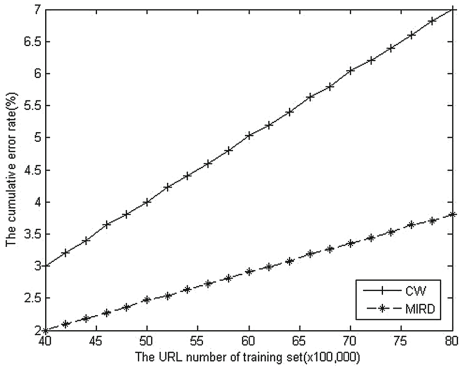

Fig. 4. Comparison of cumulative error rate between MIRD and CW.

training set and testing set was in proportion 5 to 1. The experiments were carried out on computer with 8 cores and 128 GB memory.

To verify the effectiveness of MIRD, the experiments compared the MIRD with the classical URL based approach—CW algorithm proposed by Ma et al. [13–15] in the same environment using the same data sets. The Fig. 4 showed the comparison of cumulative error number of MIRD and CW, and the cumulative error number consisted of the number of malicious URLs mis-detected as benign URL and the number of benign URLs mis-detected as malicious URL. The horizontal axis of Fig. 4 is the URL number of training set in each experiment, hundred thousand, and the vertical axis of Fig. 4 is the cumulative error number, %. The cumulative error number of MIRD was less than CW in each experiment, which showed that MIRD was more effective in detecting malicious URL than CW. The reason is the MIRD implanted with random domain name recognition, which can recognize the URL with random domain name improving the accuracy of detecting malicious URL.

To verify the efficiency of MIRD detecting malicious, Fig. 5 compared the runtime of MIRD with CW in the same environment detecting same number of malicious URLs

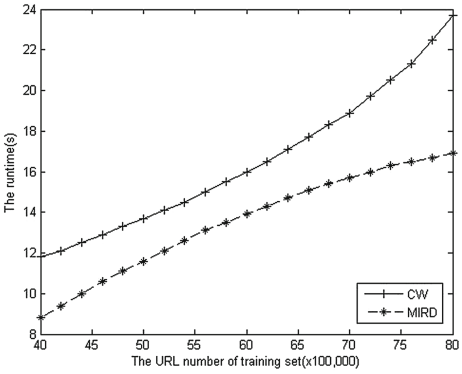

Fig. 5. Comparison of runtime between MIRD and CW.

with the same data sets. In Fig. 5, the horizontal axis is the URL number of training set in each experiment, hundred thousand. The vertical axis is the time of detecting same number of malicious URL, second. The figure showed that the MIRD needs less time than the CW in same data sets and the ratio of increasing is lower than CW. On one hand, the inverted index avoids redundant computation of common patterns and redundant matching. On the other hand, the time complexity of MIRD is linear, for the overhead time of each step in MIRD is only relative to the number of URLs. So the MIRD is more efficiency than CW.

To verify the scalability of MIRD, Fig. 6 compared the number of patterns of MIRD with blacklist of Google Safe Browser [7] to detect same number of malicious URLs. The horizontal axis is the number of detected malicious URLs in each experiment, thousand. The vertical axis is the necessary patterns of detecting same number of malicious URL, thousand. The paper regarded the URL in the blacklist as patterns for convenience. The number of patterns of MIRD is in logarithmic growth, while the number of patterns of blacklist is in linear growth. One common pattern of MIRD may match up with several URLs, so MIRD can detect some unknown malicious URLs. On the other hand, the random domain name recognition module can recognize the random domain which cannot be resolved by domain name servers. So the scalability of MIRD is good.

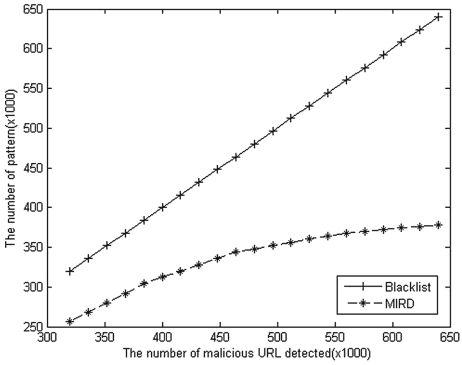

Fig. 6. Comparison of number of pattern between MIRD and blacklist.

5 Conclusion

The paper proposed an approach of malicious URL detection using trigrams-based common pattern of URL, which implanted with random domain recognition, named MIRD. MIRD used common patterns extracted from URLs based on inverted index with trigram as term to match with the detected URL. To fix up the random domain name, MIRD implanted with random domain name recognition module RDM, which identify the length of the domain name and resolve the domain name of the preliminary determined URL to recognize the unresolved domain name. The experimental result proof the MIRD is effectiveness, efficiency and scalability.

Acknowledgments. The research work is supported by Supported by the Strategic Leading Science and Technology Projects of Chinese Academy of Sciences (No. XDA06030200); the National Natural Science Foundation under Grant (No. 61402464).

References

1. [EB/OL] 2015. http://www.hpenterprisesecurity.com/ponemon-2013-cost-of-cyber-crime-study-reports
2. [EB/OL] 2015. http://en.wikipedia.org/wiki/Web_threat
3. Le, A., Markopoulou, A., Faloutsos, M.: Phishdef: url names say it all. In: The 30th IEEE International Conference on Computer Communication, Shanghai, China (2011)
4. Porras, P., Saidi, H., Yegneswaran, V., Conficker, C.: P2P protocol and implementation. SRI International Technical Report (2009)
5. Porras, P., Saidi, H., Yegneswaran, V.: An Analysis of Conficker's Logic and Rendezvous Points (2009)
6. [EB/OL] 2015. https://url.spec.whatwg.org/
7. Likarish, P., Jung, E.: Leveraging google safebrowsing to characterize web-based attacks. Association for Computing Machinery, Chicago, IL, USA (2009)
8. Zhang, J., Porras, P., Ullrich, J.: Highly Predictive Blacklisting. In: Proceedings of the 17th Conference on Security symposium, San Jose, CA (2008)
9. Provos, N., Mavrommatic, P., Rajab, M.A., et al.: All your Iframes point to us. In: 17th Usenix Security Symposium, San Jose, CA (2008)
10. Liu, H., Ma, X., Wang, T., et al.: Modeling the effect of infection time on active worm propagations. In: The 5th Applications and Techniques in Information Security, Melbourne Australia (2014)
11. Zhang, Y., Hong, J., Cranor, L.: CANTINA: a content-based approach to detecting phishing web sites. In: 16th International World Wide Web Conference, Banff, Alberta, Canada (2007)
12. Garera, S., Provos, N., Chew, M.: A framework for detection and measurement of phishing attacks. In: The 5th ACM Workshop on Recurring Malcode, Alexandria, Virginia, USA (2007)
13. Ma, J., Saul, L.K., Savage, S., et al.: Beyond blacklists: learning to detection malicious web sites from suspicious URLs. In: The 15th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Paris, France (2009)
14. Ma, J., Saul, L.K., Savage, S., et al.: Identifying suspicious URLs: an application of large-scale online learning. In: Proceedings of the 26th International Conference on Machine Learning, Montreal, Canada (2009)
15. Thomas, K., Grier, C., Ma, J., et al.: Design and evaluation of a real-time url spam filtering service. In: Proceedings of the 2011 IEEE Symposium on Security and Privacy, San Francisco, CA (2011)
16. [EB/OL] 2014. http://larbin.sourceforge.net/index-eng.html
17. Huang, D., Xu, K., Pei, J.: Malicious URL detection by dynamically mining patterns without pre-defined elements. In: The 22nd World Wide Web Conference, Rio de Janeiro, Brazil (2013)
18. [EB/OL] 2014. http://tools.ietf.org/html/rfc2181
19. [EB/OL] 2014. http://en.wikipedia.org/wiki/Phishtank
20. [EB/OL] 2014. http://www.malware.com.br/