Mining Regular Expression Rules based on q−grams

Inbok Lee

Abstract

Signature−based intrusion systems use intrusion detection rules for detecting intrusion. However, writing intrusion detection rules is difficult and requires considerable knowledge of various fields. Attackers may modify previous attempts to escape intrusion detection rules. In this paper, we deal with the problem of detecting modified attacks based on previous intrusion detection rules. We show a simple method of reporting approximate occurrences of at least one of the network intrusion detection rules, based on q−grams and the longest increasing subsequences. Experimental results showed that our approach could detect modified attacks, modeled with edit operations.

Keywords : regular expression | q−gram | longest increasing subsequences | intrusion detection system

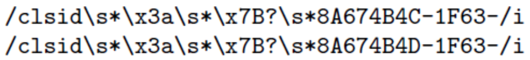
I. INTRODUCTION

Network security is a major issue. New malwares are emerging everywhere, and their economic cost is non−negligible. Also, cyberspace is considered as a future theater of war and network intrusion will be its weapon. For example, The United States Department of Defense considers the Internet both as threats and platform for attack [1].

Hackers begin with finding vulnerabilities in operating systems and applications. Then such vulnerabilities are exploited. An intrusion detection system (IDS) reads incoming and outgoing packets. If they contain suspicious contents exploiting such vulnerabilities, they are blocked, or reported for further consideration.

Once it was enough to check just headers of packets. By checking IP addresses and ports of source and destination, one could tell whether these packets are from dubious origins or targeting services with vulnerabilities. Well−known tools include `ipchains` [3] and `iptables` [4]. Nowadays checking headers is not enough: with deep−packet inspection, contents in payloads are also checked. IDS will report warnings if they contain signatures, strings or regular expressions obtained from analysis of such attacks.

We will focus on deep−packet inspection here. We need one or more signatures to detect one attack. Figure 1 shows a simple example of signatures.



```
/clsid\s*\x3a\s*\x7B?\s*8A674B4C-1F63-/i
/clsid\s*\x3a\s*\x7B?\s*8A674B4D-1F63-/i
```

Fig. 1. An example of intrusion−detection rules

Informally, rules in Figure 1 say that if there exists a substring which begins with clsid, followed by spaces (`\s*`), colon (`\x3a`), spaces, { (`\x7b`), spaces, and either `8A674B4C-1F63` or `8A674B4D-1F63`, it will report that there is an attack. From this example we can see that those rules are not easy to read and to write.

There is another approach: instead of checking incoming and outgoing network traffics against intrusion detection rules, the intrusion detection systems monitor system activity and learn what is normal and what is anomalous. We call them as anomaly−based intrusion detection systems [4, 5]. Artificial intelligence techniques are applied in learning normal and anomalous states. If these states are correctly recognized, it will beat

*Member, Department of Software, Korea Aerospace University

Manuscript: 2019. 09. 05

Confirmation of Publication : 2019. 09. 23

Corresponding Author : Inbok Lee,
e−mail : inboklee@kau.ac.kr

signature−based systems as it can handle zero−day attacks and advanced persistent threats.

However, so far as we know, signature−based systems are widely in use now: learning states is hard as there are many factors to be considered (network traffics, processes and threads, memory usage patterns, and so on). At this moment, signature−based intrusion detection systems are the de facto standard.

The main drawback of signature−based intrusion detection system is that it can only detect attacks described in intrusion detection rules. There is no rule detecting a new attack targeting a vulnerability which has not been exploited yet. IDS is off−guard until someone has dissected the attack and writes a new rule. However, isolating the corrupted network packets, understanding the unknown attack, and writing signatures requires extensive knowledge and can be done by a few experts.

There is an asymmetry between writing a new attack and protecting systems from it. Intrusion detection rules are freely available to system administrators and hackers too. From this information one can easily create a new attack by rewriting parts described in its detection rule. Statistics on the numbers of malwares support this assumption [6].

There can be several approaches to handle this situation. One is to devise methods for finding approximate occurrences of signatures. The other is to devise methods for writing a new intrusion detection rules from network traffics. We will consider a mix of these two approaches.

The main problem with automatically generated rules is that it is not easy to maintain these rules. Intrusion detection rules are hard to read and understand. Automatically generated rules are harder to read and understand, as we do not know why these rules were made. Therefore, it would be better if automatically generated rules are similar to preexisting ones and if it is hard to tell them apart. In our assumption, new attacks are just simple variation of old ones obtained by changing words in signatures. Then their signatures will be very close to those for old ones. Here we need a simpler method for finding approximate occurrences of regular expressions. From these occurrences, we can create a new intrusion detection rule for those variations.

II. RELATED WORK

Vulnerability analysis and network intrusion detection has been an important topic. Recent works covering these topics include [7, 8, 9].

Examples of network intrusion detection systems include Snort [10], Bro [11], and Suricata [12]. Snort is the de facto standard of intrusion detection system: it means that other intrusion detection systems use similar settings and signatures. Currently most intrusion detection rules are written in the Snort format and are distributed.

Approximate matching on strings and regular expression has been a major topic in algorithm research. For simple strings, it can be done by a simple dynamic programming [13]. Approximate regular expression matching was first solved by Myers in [14], but the algorithm is complicated and impractical. Another algorithm in [15] is simpler, but still it is not practical.

Approaches for automatic intrusion detection rules include [16, 17, 18, 19]. Details on how to extract rules from log data is not clearly presented.

III. PROPOSED METHOD

Here we will solve the problem based on techniques in string algorithms. Our key observation is that modified malwares are more or less the same, except that they are modified to escape intrusion detection rules. Also, we will assume that old signatures can still detect them partially: we can't find their occurrences as there is no perfect match with the current intrusion detection rules.

Instead of solving approximate regular expression matching directly, we will make the problem simpler and easier to handle. One key idea is that we will consider regular expressions as simple strings. We will consider only solid characters, that is, skipping operators and related parts. We also consider their relative order. Then the regular expression is converted into a series of subpatterns. We label each of them with increasing numbers. For packets containing modified attacks, some subpatterns may not appear, but there remains matching subpatterns. Furthermore, their labels will form an increasing sequence.

This idea can transform our problem into finding the longest increasing subsequence [20], defined as follows.

Definition 1. Given a sequence, a subsequence is the sequence obtained by deleting zero or more numbers. An increasing subsequence is a subsequence whose elements are in ascending order, and the longest increasing subsequence (LIS) is the one with the maximum length. For example, if there is a sequence 1, 5, 3, 4, and 7, its longest increasing subsequence is 1, 3, 4, and 7.

Figure 2 shows how to compute the longest common subsequence S out of the input sequence X. The algorithm runs in $O(N \log N)$ time.

```
$P$ is an array of length $N$.
$M$ is an array of length $N+1$.
$L = 0$
for $i \in 0...N-1$ do
    $lo = 1$
    $hi = L$
    while $lo \leq hi$ do
        $mid = \lceil \frac{lo+hi}{2} \rceil$
    end while
    if $X[M[mid]] < X[i]$ then
        $lo = mid + 1$
    else
        $hi = mid - 1$
    end if
    $newL = lo$
    $P[i] = M[newL - 1]$
    $M[newL] = i$
    if $newL > L$ then
        $L = newL$
    end if
end for
$S$ is an array of length $L$.
$k = M[L]$
for $i \in L-1..0$ do
    $S[i] = X[k]$
    $k = P[k]$
end for
    return $S$
```

Fig. 2. An algorithm for finding LIS in X

Definition 2. A q−gram in text T is a substring of T whose length is q. For example, when T = ABCDEF and q = 3, we have four q−grams of T, ABC, BCD, CDE, and DEF.

Before going any further, we discuss the relation between approximate pattern matching and the longest increasing subsequence. Now we analyze the effect of one edit operation. Consider the example in Definition 2 again.

– If we insert a character G after C, we get ABCGDEF. The q−grams are ABC, BCG, CGD, GDE, and DEF. Note that three q−grams were affected (as the inserted character can be located one of q positions) but the first and the last ones were not affected. It tells that there was one insertion between `ABC` and `DEF`.
– If we delete a character `D`, we get `ABCEF`. Then we have `ABC`, `BCE`, and `CEF`. Still we can see that `ABC` remains in T.
– If we replace `D` with `G`, we get `ABCGEF`. Then we, have `ABC`, `BCG`, `CGE`, and `GEF`. Again, we can see that `ABC` remains in T.

Our algorithm consists of several steps, summarized in Figure 3. We will assume that we know L, the maximum length of a substring in T matching the signature.

```
Split the input stream $T$ into substrings of length $2L$, such that two neighbouring substring will share a common suffix/prefix of length $L$.
for each substring of length $L$ do
    for each rule in the intrusion detection rule set do
        Create the $q$-grams of the rule.
        Label $q$-grams with increasing numbers.
        Create the $q$-grams of the substring.
        for each $q$-gram of the substring do
            if it is contained in the $q$-grams of the rule then
                Replace it with its label.
            else
                Replace it with zero.
            end if
        end for
        Compute the longest increasing subsequence.
        Compute the similarity score. If there are $k$ $q$-grams in the rule and $l$ is the length of the longest increasing subsequence, the score is $k/l$.
    end for
    Sort the similarity scores.
    Report rules with high similarity scores.
end for
```

Fig. 3. Outline of the proposed algorithm

Step 1: Split the network stream into overlapping strings of length $2L$. That is, if we represent the network stream as T, we will create strings $T[0 : 2L-1]$, $T[L : 3L-1]$, $\mathrm{T}[2L : 4L-1]$, and so on. It is evident that any occurrence of the signature will be contained in one of these strings.

Step 2: We pick one rule from the rule sets. Then create q−grams of the chosen rule. For example, if the rule is 8A674B4C and q = 4, then the q−grams are 8A67, A674, 674B, 74B4, and 4B4C. We will also assign them numbers between one and five in that order. That is, *label*(8A67)=1, *label*(A674)=2, *label*(674B)=3, *label*(74B4)=4, and *label*(4B4C)=5.

Step 3: We will create q−grams from the network stream and check whether there are common q−grams found in Step 2. A naive approach will take $O(qL)$ time, but it can be done in $O(L)$ time using the technique in [18]. For each q−gram, we check whether it is also contained in those of the chosen rule. If so, we represent it with its label obtained in the previous step. Otherwise, we represent it with zero. For example, assume that the network stream was `98A6754B4C`. With q = 4, the q−grams are `98A6`, `8A67`, `A675`, `6754`, `754B`, and `54B4`. We can see that `8A67` and `54B4` are contained in the q−grams of the chosen rule and their labels were 1 and 5, respectively. Then the output is 0, 1, 0, 0, 0, 5. Note that we create q−grams for words in the rule only: wildcards or special symbols are ignored. A complicated procedure can handle them, but we did not see a noticeable enhancement in our experiments, so we choose to ignore them.

Step 4: Find the LIS in the sequence obtained in Step 3. Note that we will ignore zeros as it doesn't mean one match of a common q−gram between the rule and the stream. In our example, the longest increasing subsequence is 0, 1, 5. After removing the first zero, we obtain that 1, 5. It means that the network stream of length L contained a substring which begin with `8A67`, followed by `54B4`. As the length of the sequence is $O(L)$, the time complexity is $O(L \log L)$.

Step 5: Compute the similarity score. If there were k q−grams from the rule, and the length of the LIS is l, the similarity score is l/k. The reason why we divide l with k, that is, normalize the score is that a long rule can contain many q−grams and the chance of unintended matches is high. For each signature, we store its similarity score.

Step 6: Repeat Steps 2~5 for each signature in the intrusion detection rules. Sort them by the similarity scores and report ones with highest scores. It is easy to show that the time complexity is $O(mL \log L)$, where m is the number of intrusion detection rules.

IV. EXPERIMENTAL RESULTS

We used the snort−snapshot−2983 rule sets for the experiment. There were 8,472 rules and 1,586 distinct regular expressions were contained. The experiment was done on an iMac, running macOS Sierra 10.12.5. The scripts were written in Python 3.4.1.

The experiment is composed with 13 rounds, in which we randomly picked 200 intrusion detection rules, and created random packets containing the signature using Sniffles [22, 23]. It reads one intrusion detection rule in Snort format and then creates traffics in PCAP format [24]. To simulate the modified attacks, we randomly picked at most two positions in the rule and modified it with edit operations randomly chosen. Then we generated packets with the modified rules. The results were packets with at most two edit operations, insertions, deletions, and substitutions.

Table 1 shows the summary of experimental results. Out of 200 modified rules, our algorithm was able to find the original intrusion detection rule even with the modified packets. We used q = 3 and q = 4, respectively.

One may wonder why the original rule may not have the highest score. After a closer look, we found that one edit operation could make q q−grams mismatches, and another longer rule might have more matching q−grams and having higher score than the original one. We tried other scoring schemes, but with them sometimes we were not able to find the original rule within Top 10. Surely, our approach favors longer rules as they have more matching q−grams regardless of its meaning. We believe that with more experiments we can design a better scoring function.

Larger q will make the algorithm run faster as it will make fewer q−grams, but it may miss the answer. In our experiment when q > 4, there were cases when the original one was not included in Top 10.

Table 1. Experimental Results

	q=3		q=4	
Round	Within Top 10	Not within Top 10	Within Top 10	Not within Top 10
1	200	0	200	0
2	200	0	200	0
3	200	0	200	0
4	200	0	200	0
5	200	0	200	0
6	200	0	200	0
7	200	0	200	0
8	200	0	200	0
9	200	0	200	0
10	200	0	200	0
11	200	0	200	0
12	200	0	200	0
13	200	0	200	0

V. CONCLUSION

We showed that mining a new rule for a modified attack can be done easily with a combination of q−grams and LIS. It is based on well−known techniques of string algorithm, which is simple and easy to understand and to implement. Also, it can be turned into a parallel one by assigning each processor with different packets.

Experimental results show that our approach can detect the modified attacks with the original intrusion detection rules. From this information one can find clues to write a new intrusion detection rule.

Future works include designing an efficient scoring function and another method of mining a new intrusion detection rule based on the current rules.

REFERENCES

[1] *US Department of Defense Cyber Strategy*, US Department of Defense, pp. 2-8, 2015.

[2] Linux IP Firewalling Chains. http://people.netfilter.org/rusty/ipchains (accessed Sept., 25, 2019).

[3] Netfilter: firewalling, NAT, and packet mangling for Linux. http://www.netfilter.org (accessed Sept., 25, 2019).

[4] K. Wang, "Anomalous Payload -Based Network Intrusion Detection," Recent Advances in Intrusion Detection. Springer Berlin. doi:10.1007/978-3-540-30143-1_11.

[5] R. Perdisci, D. Ariu, P. Fogla, G. Giacinto, and W. Lee, "McPAD : A Multiple Classification System for Accurate Payload-based Anomaly Detection," *Computer Networks, Special Issue on Traffic Classification and Its Applications to Modern Networks*, vol. 5, no. 6, pp. 864-881, 2009.

[6] AV-TEST: Malware statistics. http://www.av-test.org/en/statistics/malware (accessed Sept., 25, 2019).

[7] K.H. Lee and G.S. Ryu, "Research for improving vulnerability of unmanned aerial vehicles," *Smart Media Journal*, vol. 7, no. 3, pp. 64-71, 2018

[8] W.J. Joe, H.J. Shin, and H.S. Kim, "A log visualization method for network security monitoring," *Smart Media Journal*, vol. 7, no. 4, pp. 70-78, 2018

[9] S.I. Bae and E.G. Im, "Unpacking Technique for In-memory malware injection technique," *Smart Media Journal*, vol. 8, no. 1, pp. 19-26, 2019

[10] Snort: Network intrusion detection and prevention system. http://www.snort.org (accessed Sept., 25, 2019).

[11] The Bro Network Security Monitor. https://www.bro.org (accessed Sept., 25, 2019).

[12] Suricata: Open IDS / IPS / NSM engine. https://suricata-ids.org (accessed Sept., 25, 2019).

[13] G. Navarro, "A guided tour to approximate string matching," *ACM Computing Surveys*, vol. 33, no. 1, pp. 31-88, 2001

[14] E.W. Myers, "A Four Russians Algorithm for Regular Expression Pattern Matching," *Journal of ACM*, vol. 39, no. 2, pp. 430-448, 1992

[15] D. Belazzougui and M. Raffinot, "Approximate regular expression matching with multi-strings," *Journal of Discrete Algorithms*, vol. 18, pp. 14-21, 2013

[16] H. Altwaijry and K. Shahbar, "Automatic SNORT Signatures Generation by using Honeypot," *Journal of Computers*, vol. 8, no. 12, pp. 3280-3286, 2013

[17] B. Rice, "Automated Snort Signature Generation", *Masters Theses, James Madison University*, 2014

[18] S. Ashfaq, M.U. Farooq, and A. Karim, "Efficient rule generation for cost-sensitive misuse detection using genetic algorithms," *Proc. of CIS*, pp. 282-285, 2006

[19] H.A. Kim and B. Karp, "Autograph: Toward automated, distributed worm signature detection," *USENIX Security Symposium*, pp. 271-286, 2004

[20] C. Schensted, "Longest increasing and decreasing subsequences," *Canadian*

Journal of Mathematics, vol. 13, pp. 179-191, 1961

[21] R.M. Karp and M.O. Rabin, "Efficient randomized pattern-matching algorithms," *IBM Journal of Research and Development*, vol.31, no. 2, pp. 249-260, 1987

[22] Sniffles: Capture Generator for IDS and Regular Expression Evaluation. https://github.com/petabi/sniffles (accessed Sept., 25, 2019).

[23] M. Shao, M.S. Kim, V.C. Valgenti, and J. Park, "Grammar-Driven Workload Generation for Efficient Evaluation of Signature-Based Network Intrusion Detection Systems," *IEICE Transactions on Information and Systems*, vol. 99-D, no. 8, pp. 2090-2099, 2016

[24] tcpdump and libpcap. http://www.tcpdump.org (accessed Sept., 25, 2019).

Authors

Inbok Lee

He received his B.E., M.S., and Ph.D degree in Computer Engineering from Seoul National University in 1997, 1999, and 2004, respectively. Since 2006, he has been an associate professor in Department of Software, Korea Aerospace University, Korea. His research interests are design and analysis of algorithms, competitive programming, and intrusion detection system.