Reducing Payload Scans for Attack Signature Matching Using Rule Classification

Sunghyun Kim and Heejo Lee*

Korea University, Seoul 136-713, South Korea
{afshkim,heejo}@korea.ac.kr

Abstract. Network intrusion detection systems rely on a signature-based detection engine. When under attack or during heavy traffic, the detection engines need to make fast decision whether a packet or a sequence of packets is normal or malicious. However, if packets have a heavy payload or the system has a great deal of attack patterns, the high cost of payload inspection severely diminishes the detection performance. Therefore, it would be better to avoid unnecessary payload scans by checking the protocol fields in the packet header first, before executing their heavy operations of payload inspection. Furthermore, when payload inspection is necessary, it is better to compare attack patterns as few as possible. In this paper, we propose a method which reduces payload scans by an integration of processing protocol fields and classifying payload signatures. While performance improvements are dependent on a given networking environment, the experimental results with the DARPA data set show that the proposed method outperforms the latest Snort over 6.5% for web traffic.

1 Introduction

Intrusion detection is a set of techniques and methods that are used to detect suspicious activities both at the network and host level. The process of intrusion detection aims to find data packets that contain any known intrusion-related signatures or anomalies related to the Internet protocols. Intrusion detection methods fall into two basic categories: signature-based intrusion detection and anomaly-based detection.

Signature-based detection is used to compare against activity in the network or host with predefined signatures which are produced by an analysis of an attack or malicious packets. This method relies on a database of attack signatures. Therefore, it is only as effective as its database. Most signatures have patterns to search known attacks. Anomaly-based intrusion detection, by contrast, utilizes a more generalized approach when searching for and detecting threats in a network. A rule of normal behavior is developed and when an event falls outside that norm, it is detected and logged. The behavior is a characterization of the state of the protected system, which is a reflective of the system health and is sensitive to attacks. In this context, an anomaly-based method of intrusion detection has the potential to detect new or unknown attacks. In a manner to similar to the signature-based method, anomaly-based intrusion detection relies on information that signifies what is normal and what is an anomaly.

* Corresponding author.

Y. Mu, W. Susilo, and J. Seberry (Eds.): ACISP 2008, LNCS 5107, pp. 350–360, 2008.

Network Intrusion Detection System(NIDS) captures data from the network and applies rules to that data or detects anomalies. NIDS detects malicious activities such as denial of service attacks, port scans or even attempts to hack into computers by monitoring network traffic. Based on a set of signatures and rules, after a match is found, the detection system takes some actions, such as logging the event or sending an alarm to a management console. Numerous studies about NIDS have attempted to rapidly decide which packets are malicious.

Recently, a large number of signatures are associated with well known ports, such as HTTP and SMTP. Also, because volume of multi-media data, such as video, file download services of web sites and P2P services, is increasing at an amazing rate, the cost of pattern matching in the packet payload is increasing. Therefore, to reduce the cost of payload scanning, it is reasonable to check the protocol fields before searching the payload to compare patterns. This is why we proposed the method whereby the protocol fields have a great priority than the packet payload for signature matching. The proposed method is similar to research on rule classification by protocol fields such as a decision tree [8] or an evaluation tree [9]. However, our method calculates all possible results, based on expected values of the protocol fields and makes small rule groups. Thus, the payload inspection of the packets is performed only when it is necessary. Our method has some advantages compared with previous methods, which are as follows:

- It processes the rules without considering the values of protocol fields.
- It is a flexible structure such that we can change the examining sequence of protocol fields and add or remove some of the protocol fields.
- It is more effective to handle the complex rules.

The contribution of this study is to propose a new method of rule set classification and the integrated processing of protocol signatures. In spite of additional overhead, it can yield small rule groups and provide fast detection. In the remainder of this paper, §2 briefly provides related works. §3 describes the proposed method. §4 analyzes the performance. §5 presents experimental results. In §6, we summarize our experience.

2 Related Works

Just as a network packet consists of the header and the payload, the research about signature matching can be classified into two categories. One is a pattern matching for a packet data, which consists mainly of string matching. The other is the classification of a rule set by the protocol fields. The former focuses on reducing the number of rules to be searched by grouping, in other words classification or clustering. The latter mainly focuses on the means to rapidly certain strings. We will briefly discuss some of the methods for signature matching and explore Snort's internal.

For the payload matching, several pattern matching algorithms have been proposed. Among the single pattern matching, a well-known algorithm is the Boyer and More algorithm [2]. It preprocesses the target string that is being searched for to generate a table of mismatch skip values based on the pattern position involved in the mismatch. Another well-known algorithm, Knuth-Morris-Pratt(KMP) [3] also preprocessed patterns, to generate a look-up table that indicates how many positions the pattern can be shifted to the right based on the position in the pattern where a mismatch occurs.

The multi-pattern matching method searches a text string for the occurrence of any pattern in a set of patterns, using only a single iteration. A well known algorithm is the Aho-Corasick(AC) algorithm [4] which preprocesses the set of patterns, to construct a pattern matching machine based on a deterministic finite automaton (DFA). The matching procedure works by reading successive characters from the input string, making state transitions based on each character, and producing output after a complete pattern is matched. The Wu-Manber algorithm [5], is based on the bad character heuristic, which is similar to Boyer-Moore, but uses a one or two-byte bad shift table constructed by re-processing all the patterns, instead of only one. Also it uses the hashing table to index the patterns in the actual matching phase, thus saving a great deal of time. Another method, Exclusion-based signature Matching(E^2xB) [6] is designed to provide rapid negatives, when the search string does not exist in the packet payload.

At well as this, research about the classification of rules has progressed. Kruegel and Toth proposed a decision tree method to improve signature-based intrusion detection [8]. In order to create an optimized decision tree, which is used to find malicious events, using a minimum of redundant comparisons, this method uses a well-known clustering algorithm which is applied in machine learning. The algorithm builds a decision tree from a classified set of data items with different features using the notion of information gain. Sinha et al [9] proposed an evaluation tree which determines which rule groups are maintained in memory by choosing protocol fields and values recursively. Initially, the method selects the protocol field that is most effective in rejecting the rules, and then separates those groups by values of the chosen protocol field. After forming groups for each of these values, the algorithm recursively splits the groups by other protocol fields that reject at least a threshold number of rules, producing smaller groups. By this means, it generates a hierarchy of protocol fields and values, for which groups are maintained. As discussed above, these methods are used for IDSes' detection engines.

Among several NIDSes, Snort [1,10] is an open source network intrusion prevention and detection system utilizing a rule-driven language, which combines the benefits of protocol and payload signature. Snort is commonly used to actively block or passively detect a variety of attacks and probes performing protocol analysis and content matching. Snort considers rules to be composed of 2 components, a rule header and a rule option. The rule header has predicates of the protocol fields. The rule option mainly has strings for pattern matching, and other predicates. After parsing the rules, based on the port, Snort makes 3 port groups, which consist of; a destination port group for rules having a unique destination port, a source port group for rules having a unique source port and a generic port group for rules without a unique destination port and source port. Each port in a port group has two multi-pattern matchers. One is for content, which is a keyword that searches for specific content in the packet payload. The other is for uricontent, which is a keyword that searches the normalized request URI field. Also each port has rule chains for rules without content or uricontent. A generic port group is copied to other port groups for efficiency. Snort provides the Wo-manber and the Aho-corasick pattern matching algorithm. When packets are going through, based on port number, multi-pattern matchers of the corresponding port group are called. In the worst case, a packet is scanned three times, once for the destination port group, once for the source

port group and once for the generic rule group. In the best case, a packet is only scanned once for one of the three groups. Snort only uses one protocol field for grouping rules. Under the condition of a heavy payload or a large number of patterns, a great deal of time is required for inspecting the payload. Our method integrates rules' predicates into each protocol field and pre-calculates all possible results. Because we inspect the protocol fields first, we avoid unnecessary payload scanning. In what follows, Our method is discussed in detail.

3 Detection and Classification by Grouping Predicates

NIDS has been deployed behind a firewall which inspects network traffic passing through it and denies or permits passage based on policy. Thus, a large number of signatures are associated with specific ports. In table 1, based on our analysis of snort's rules[1] (VRT Certified Rules for Snort v2.7), among 6985 default rules related to TCP, 4935 (70%) rules are associated with 3 destination ports(80,445,139) and 1 source port(80).

Table 1. Top 5 lists of Snort's TCP rules classified by port

Destination Port	Rules	Source Port	Rules
80	1570	80	743
445	1359	1024	42
139	1263	23	13
1521	291	666	8
>1024	147	5050	6

Our method integrates each predicate of protocol field used in the rule set into a single data structure, we call a protocol filter, and calculates all possible results based on the values of the protocol field in advance. After pre-calculating the results, it makes small rule groups by the combination of the pre-calculated results. When a packet is moving through the system, it searches each result based on value of a packet's protocol fields. Combining these results, it identifies a single pre-calculated rule group. Only checking this rule group, our method can reduce the chance of payload scanning and alleviates the load of pattern matching. Figure 1 shows our method briefly. We shall explain this in more detail.

3.1 Formal Description

We will restrict our description to relationships of protocol fields. Let R be the set of n rules, i.e., $R = \{r_1, r_2, ...r_n\}$. Let $F = \{f_1, f_2, ..., f_m\}$ denote the set of m protocol fields present in the rule set. Let $P_{f_i} = \{p^1_{f_i}, p^2_{f_i}, ..., p^k_{f_i}\}$ denote the set of the predicates associated with a protocol field f_i in the rule set. Also let $V_{f_i} = \{v^1_{f_i}, v^2_{f_i}, ..., v^j_{f_i}\}$ denote the set of unique values which are extracted from a protocol field f_i's predicates used in R and sorted in ascending order. A rule in R can be described as the relationship of predicates of protocol fields like $r_t = p^j_{f_1} \bigwedge p^j_{f_2} \bigwedge ....p^k_{f_m}$. And a predicate

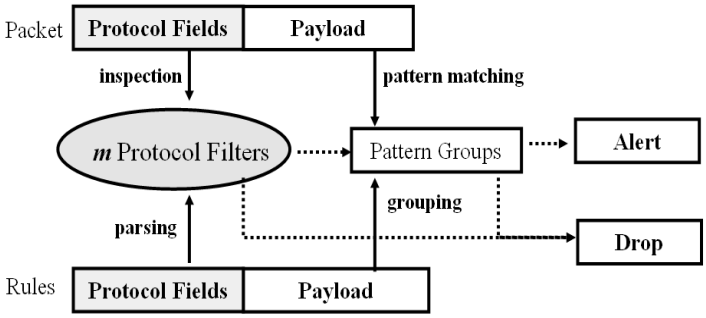

Fig. 1. Detecting and grouping by protocol filters

used in R can be presented like $p^j_{f_i} = f_i \bigodot v^k_{f_i}$ where $\bigodot$ is an operator used in the predicate(=,≤,≥,etc.). Based on values of f_i, each predicate in P_{f_i} can be true or false. Likewise, based on each predicate results of P_{f_i}, each rule in R can be false or true. Therefore, Based on values of f_i, each rule in R has various result. Let $s^j_{f_i}$ denote all rules' results depending on an element of V_{f_i}. Let $dom(f_i)$ denote f_i's domain.

Like Figure 2, $dom(f_i)$ can be divided into $(2k+1)$'s sub range or sub domain, where $k = |V_{f_i}|$. Let $s^j_{f_i}$ denote rule set's result in the j^{th} subrange of $dom(f_i)$. And we describe $S_{f_i} = \{s^1_{f_i}, s^2_{f_i}, ...s^{2k+1}_{f_i}\}$ to present the set of all possible results which depend on all values of f_i. We can find the all results of the rule set by each value of the protocol field f_i used in the rule set in advance. Based on values of f_i, P_{f_i} is decided and then depending on P_{f_i}, which rules among $r_1, r_2, ...r_n$ is matched or not is determined. $s^{2j}_{f_i}$ of S_{f_i} has the results of $r_1, r_2, ...r_n$ regarding to $v^j_{f_i}$ of V_{f_i}. Likewise, $s^{2j+1}_{f_i}$ of S_{f_i} has the results of $r_1, r_2, ...r_n$ depending on all values between $v^j_{f_i}$ and $v^{j+1}_{f_i}$ of V_{f_i}. Because the rule set includes $v^j_{f_i}$, we can calculate $s^j_{f_i}$. Also, we can compute $s^{2j+1}_{f_i}$ if we generate any value which is included in between $v^j_{f_i}$ and $v^{j+1}_{f_i}$. The proposed method is to pre-calculate possible results, i.e., S_{f_i} and stores them for decision about rules' matching.

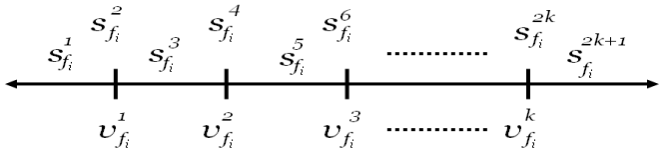

Fig. 2. Rules' results based on values of the protocol field f_i

For example, if r_1 has a predicate for destination port, like $p^1_{dport} = \{dport, >, 3\}$, r_1 obtains 3 results which is one for values of less than 3, one for a value of 3, and one for values of greater than 3. If r_2 having a predicate for destination port, like $p^2_{dport} = \{dport, =, 6\}$, is added, r_1 and r_2 can obtain 5 results which is one for values of less than 3, one for a value of 3, one for values of greater than 3 and less than 6, one for

values of 6, and one for values greater than 6. The number of results of the rule set including a predicate of protocol field f_i is 5 because of $2 * n|V_{f_i}| + 1$.

3.2 Detection by Protocol Filters

As has demonstrated, we pre-calculate all the rules' results and save them into an array data structure, so called a protocol filter. Figure 3 shows the proposed method. When a packet is reached, based on the value of the packet's protocol field, we search the results of the rule set in protocol filters. Based on the combined results of the rule set, we identify whether the packet need a payload scan or not. For example, we have two rules similar to Snort's rule such as the following.

r_1: alert tcp 10.1.1.1 25 –>!$HOME_NET 80 . . . ;content:login;. . .
r_2: alert tcp 20.1.1.1 1024:2024 –>$HOME_NET !143 . . . ;content:root;. . .

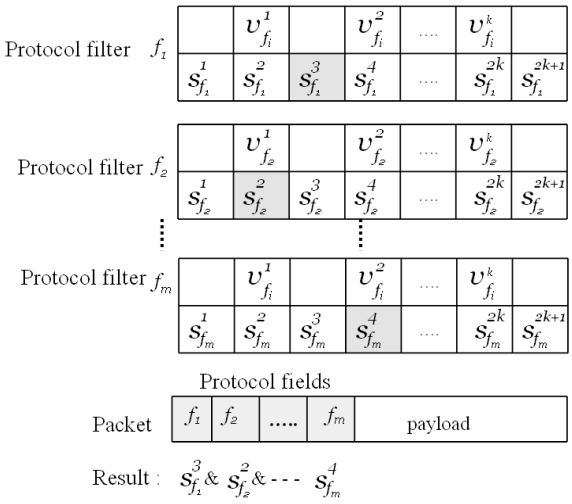

Fig. 3. Detection using protocol filters

Figure 4 shows in detail the proposed method. Let's make the protocol filter for destination port with r_1 and r_2. If a packet's destination port is less than 80, greater than 80 and less than 143, or greater than 143, only r_2 is matched. In case of the destination port 80, both rules are matched. However, for port 143, both rules are not matched. We represent the rules' result as bit strings. We call this structure "protocol filter". We made the protocol filter for the source port in the same way. In this case, we only used two protocol fields and made two protocol filters. When a packet P_1 is moving through, we search for the results of rules in protocol filter based on protocol field's value. If we obtain all corresponding result bit strings of the protocol filters, the final result is decided by 'AND' bit operation to each result bit strings. The other predicates including pattern matching are indexed by the position of bit strings. Based on value of bits, we filter out unnecessary rules.

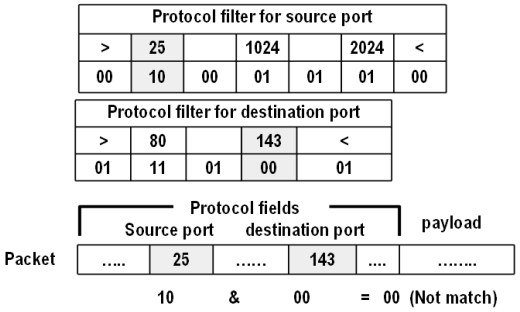

Fig. 4. Example of protocol filters

As well as reducing the cost of payload searching, the proposed method has the advantage that upon execution. It only performs search operation, irrespective of the rule set's predicates. Therefore, the greater the complexity of rules' predicates, the better the performance. Also, because protocol filters do not have a fixed order, we can change the search order. For example, if we can obtain information that a certain protocol field can drop early normal packets, we can change the checking order of protocol filters.

3.3 Grouping by Protocol Filters

As noted previously, a protocol filter for f_i provides the integrated processing of predicates and result of the rule set related to f_i. Taking this idea further, if we calculate all combinations of protocol filters for $f_1, f_2, \ldots f_m$ in advance, we obtain all possible results of the rule set, i.e., $S_{f_1}, S_{f_2}, ...S_{f_m}$. The maximum number of these results is 2^n if there are n rules.

In Figure 4, when we have two rules, we can make the maximum four distinct results of the rule set, because of 2^2. However, if we calculate the combination of 2 protocol filters' result bit strings, we can obtain 2 result bit strings such as '01','10'. Based on the number of result bit strings, we can classify rules into 2 small groups. We ignore result '00', which means all rules are not matched and '11', which is impossible results bit strings. The reason we do not make rule groups as much as 2^n, is that we can remove unnecessary rule groups by a combination of protocol filters. The greater the number of results, the smaller number of rules in a group. The rule grouping by protocol filters can easily tune the degree of grouping. Whereas smaller rule groups are made when many protocol filters are used, larger rule groups are made when few protocol filters are used. By contrast, the overhead of protocol fields matching is proportional to the number of protocol filters.

4 Performance Analysis

The proposed method is based on the fact that, in general, pattern matching of payload needs more processing time than protocol fields matching. In the case of TCP, while the packet header has 20 bytes without option, the packet data can have 1460 bytes,

considering MTU(Maximum Transmission Unit). The size of the packet data and values of the protocol field depend on network environment. Therefore, if a packet has little or no data, checking the payload first can yield inferior results. Also the number of patterns which affects performance can be scattered in proportional to the number of protocol fields used in grouping. We analyze performance these two aspects.

We compare a single protocol field grouping with multiple protocol fields grouping. To simplify our analysis, we assume that rule set, R, has n rules and that each rule has a single predicate of $f_1, f_2, ..., f_m$. So, every rule has m predicates. As noted above, V_{f_i} is the set of unique values which are extracted from a protocol field f_i's predicates used in R and P_{f_i} is the set of the predicates associated with a protocol field f_i used in R. In addition, let D_{f_i} denote f_i's domain and let A_{f_i} denote all f_i's value ranges used in P_{f_i}. If f_i' value is within A_{f_i}, at least one among P_{f_i} satisfy the predicate. In the case of a single protocol field grouping, the average rule count of subgroups is $\frac{n}{|V_{f_i}|}$ and probability of payload scanning is $\frac{A_{f_i}}{D_{f_i}}$. However, if we make rule groups with multiple protocol fields, such as $f_1, f_2, \ldots f_m$, the average rule count of subgroups is $\frac{n}{\prod_{i=1}^{k}|V_{f_k}|}$. In a manner similar to the average rule count of subgroups, the probability of payload scanning is $\prod_{i=1}^{k}(\frac{A_{f_i}}{D_{f_i}})$. Clearly, the more protocol fields are used, the few rules are included in a subgroup and the lower the probability of payload scanning is. Small rule groups mean few patterns to search in the payload, in other words, the pattern match engines have a light work load.

For example, if rules are grouped only by the destination port, n rules can be scattered with 2^{16} because the destination port has 2 bytes. However, if rules are grouped by a combination of the destination port, source port, destination IP and source IP, which consist of 12 bytes, n rules can be scattered with 2^{96}. Also, the probability of payload scanning for subgroups is lowered in the same manner. However, the performance of the proposed method is strongly dependent on the environment of network and distribution of rules.

5 Experiments

To implement the proposed method, we have modified Snort version 2.7.0.1. The experimental platform is a personal computer with a Pentium 4 Core 2 6,400 CPU and 3 Gbytes RAM. We used the Linux operating system, Fedora 6.

The rule set used for the experiments consisted of only the TCP rules among the VRT Certified Rules for Snort version 2.7. We used well-known data sets, the DARPA Intrusion Detection Evaluation Data Set from MIT Lincoln Lab [11]. We analyzed three types of the DARPA data files and selected ports which have a large number of rules and frequently appeared in data files at the same time. Table 2 shows the average payload size and the percentage of several destination ports in data files.

We made 4 protocol filters using destination port, source port, destination IP, and source IP. Table 3 shows how the rule set can be grouped by 4 protocol fields, compare to Snort's a single protocol field grouping. In the case of the destination port 80, whereas Snort makes one port group with 2026 rules, the proposed method make 4 small groups.

Table 2. Destination port's characteristics of DARPA data set

Destination Port		1999 week1	2000 LLDOS1.0	2000 window NT
80	avg. payload bytes	261	259	409
	% of total traffic	5.7	2.47	6.02
25	avg. payload bytes	393	623	491
	% of total traffic	13.9	10.25	15.94
23	avg. payload bytes	113	114	97
	% of total traffic	0.27	0.12	0.34
139	avg. payload bytes	-	126	192
	% of total traffic	-	0.17	0.16

Table 3. Rule classification by protocol fields

Classification	80	25	23	139
	rule/group	rule/group	rule/group	rule/group
Destination port or source port (Snort)	2026/1	147/2	25/1	1263/1
4 Protocol Fields	3866/4	215/3	25/1	1263/1

Table 4. CPU times of DARPA data set

Dest Port	method	1999 week1	2000 LLDOS1.0	2000 window NT
80	Snort	26.3	12.4	29.5
	Prot. Filter	24.6	11.5	28.1
Improvement		93.6%	93.5%	93.5%
25	Snort	0.8	0.52	0.87
	Prot. Filter	0.73	0.5	0.82
Improvement		91.2%	96.1%	94.2%
23	Snort	0.038	0.026	0.035
	Prot. Filter	0.019	0.014	0.02
Improvement		50%	54%	57%
139	Snort	-	0.45	1.19
	Prot. Filter	-	0.0049	0.92
Improvement		-	1%	77%

We used the default IP address setting, consisting of only a home net and an external net. Therefore, the effect of the grouping rules was tiny.

After the method was executed 10 times, we recorded the average time of detection. We only evaluated the packets over 20 bytes which is the TCP header size, considering overhead. In Table 4, the proposed method improved the processing time to various

degrees, compared with Snort. In the case of port 80 and port 25, improved performance results from the port's small rule groups, because the proportion of packets that skipped payload inspection is below 0.01%. In the other cases, improvement of port 23 and port 139 results from skipped payload inspection, because the proportion of skipped packets is between 37% and 100%. In the case of data file LLDOS1.0 and 139 port, while all packets have the external network addresses in the source IP field, all rules have the internal network addresses in the source IP field. Thus the protocol filters dropped all packets without payload inspection. Clearly, grouping by multiple protocol fields can improve performance in that the amount of pattern matching and the probability of payload scanning are reduced. Because of a packet's payload size and the distribution of the protocol fields' values, there will be a variety of results. If we use the more protocol fields to classify a rule set, we can make the smaller rule groups and avoid a great number of payload scanning. Also if the packets have a heavy payload, the performance will be much better.

6 Conclusions

In this paper, we proposed the method to reduce the cost of payload matching. The proposed method involves integrated detection of protocol fields and the separation of a large signature group into several small signature groups, by multiple protocol fields. The effect of the proposed method can be various depending on rules and packets. However, the proposed method can reduce the payload scanning for patterns matching and reduce the number of patterns for packets to check, because packets which do not match protocol fields can be dropped before payload scanning. In addition, the proposed method is independent of a predicate's operand, because of the pre-calculation of all the predicates. Also this allows a detailed rule description, which enables us to easily represent complex and complicated predicates. Unfortunately, it suffers from rule replication and bit operation overhead. However, the memory requirement can be tolerated by system, if we use only some overloaded rule groups. For a bits operation, if we adapt the proposed method to only heavy payload packets, the advantage generally more than compensates for the overhead. In the future, we intend to include some protocol fields which are frequently used in rules and can reject packet early. Also we will evaluate our method in real network environments.

Acknowledgments

This work was supported in part by the ITRC program of the Korea Ministry of Knowledge Economy. Additionally supported by a Korea University Grant.

References

1. Snort:Open source Network Intrusion Detection System, `http://www.snort.org`
2. Boyer, R.S., Moore, J.S.: A fast string searching algorithm. Communications of the ACM 20, 762–772 (1977)

3. Knuth, D.E., Morris, J.H., Pratt, V.R.: Fast pattern matching in strings. SIAM Journal of Computing 6, 323–350 (1977)
4. Aho, A.V., Corasick, M.J.: Fast pattern matching: an aid to bibliographic search. Communications of the ACM 18, 333–340 (1975)
5. Wu, S., Manber, U.: A Fast Algorithm for Multi-Pattern Seaching, Technical Report TR-94-17, Department of Computer Science. University of Arizona (May 1994)
6. Wang, X., Li, H.: Improvement and Implementation of Network Intrusion Detection System. Journal of Communication and Computer, 49–52 (January 2006)
7. Fisk, M., Varghese, G.: Fast Content-Based Packet Handling for Intrusion Detection, UCSD Technical Report CS2001-0670. University of California, San Diego (May 2001)
8. Kruegel, C., Toth, T.: Using Decision Trees to Improve Signature-based Intrusion Detection. In: Vigna, G., Krügel, C., Jonsson, E. (eds.) RAID 2003. LNCS, vol. 2820, pp. 173–191. Springer, Heidelberg (2003)
9. Sinha, S., Jahanian, F., Patel, J.M.: WIND:Workload-Aware INtrusion Detection Recent Advances in Intrusion Detection. In: Zamboni, D., Krügel, C. (eds.) RAID 2006. LNCS, vol. 4219, pp. 290–390. Springer, Heidelberg (2006)
10. Roesch, M.: Snort: Lightweight Intrusion Detection for Networks. In: Proc. of the USENIX LISA 1999 Conference, pp. 229–238 (November 1999)
11. McHugh, J.: Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Lab. ACM Trans. Information and Systems Security(TISSEC) 3(4), 262–294 (2000)
12. Commentz-Walter, B.: String Matching Algorithm Fast on the Average. In: Proc. of the 6th International Colloquium on Automata, Languages, and Programming, pp. 118–132 (1979)
13. Paxson, V.: Bro: A System for Detecting Network Intruders in Real-Time. Computer Networks 31(23-24), 2435–2463 (1999)
14. Sommer, R., Paxson, V.: Enhancing byte-level network intrusion detection signatures with context. In: Proc. of the 10th ACM Conference on Computer and Communication Security (CCS 2003), pp. 262–271 (October 2003)
15. Kruegel, C., Toth, T.: Automatic rule clustering for improved signature-based intrusion detection, Technical report, Distributed systems group:Technical Univ. Vienna, Austria (2002)
16. Dreger, H., Feldmann, A., Mai, M., Paxson, V., Sommer, R.: Dynamic Application-Layer Protocol Analysis for Network Intrusion Detection. In: Proc. of the 15th USENIX Security Symposium, pp. 257–272 (July 2006)
17. Allen, W.H.: Mixing Wheat with the Chaff: Creating Useful Test Data for IDS Evaluation. IEEE Security & Privacy, 65–67 (July 2007)
18. Antonatos, S., Anagnostakis, K.G., Polychronakis, M., Markatos, E.P.: Performance analysis of content matching intrusion detection systems. In: Proc. of the 4th IEEE/IPSJ SAINT (January 2004)
19. Mell, P., Hu, V., Lippmann, R.: An overview of issues in testing intrusion detection systems (June 2003),
`http://csrcnist.gov/publications/nistir/nistir-7007.pdf`