Volume 2, Issue 4, April 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Signature Based Rule Matching Technique in Network Intrusion Detection System

Mr. Chandrapal U. Chauhan
Master of Technology Scholar, G.H.R.C.E, Nagpur
chanderpaul1@gmail.com

Mrs. V.A.Gulhane
Assistant Professor, G.H.R.C.E, Nagpur
vinagulhane@gmail.com

***Abstract*— Signature is the pattern that you look for inside a data packet. A signature is used to detect one or multiple types of attacks. Signatures may be present in different parts of a data packet depending upon the nature of the attack. We can find signatures in the IP header, transport layer header (TCP or UDP header) and application layer header or payload. Usually IDS depends upon signatures to find out about intruder activity. With the increased amount of data transferred by computer networks, the amount of the malicious traffic also increases and therefore it is necessary to protect the network by security system such as firewalls and the Intrusion Detection System. Pattern matching is the time critical operation of current Intrusion Detection System. In this project this pattern matching is based on the regular expression where as these pattern of known attack are stored in the database of Intrusion Detection System. Regular Expressions are often used to describe malicious network pattern.**

***Keywords*- Regular Expression, Intrusion Detection System, Rule set, KMP algorithm, SNORT IDS, Malicious data.**

I. INTRODUCTION

In this paper, we are concentrating on two areas like Computer Network Security and Automata Theory. In the recent years, Internet has become a very popular method to connect computers all over the world. While the availability of continuous communication has created many new opportunities, it has also brought new possibilities for malicious users. The Importance of network Security is therefore growing; one of the ways of malicious activity detection on a network is by using Intrusion Detection System.

Most modern Intrusion Detection System relies on a set of rules that are applied to each input packet in order to define suspicious activities. The simplest rules are described by packet header field content and pattern of data in the packet payload. Detecting such patterns is the core operation of an Intrusion Detection System.

Network Intrusion Detection System uses a collection of signatures of known security threats and viruses to scan each packet's payload. These signatures are used to scan the data streams of various flows traversing through the network link, when the flow matches the signature, appropriate action will be taken. Traditionally, these signatures have been specified as string based exact match, however as the complexity of rule set increases, traditional string matching techniques are being replaced by more sophisticated regular expression matching technique.

II. REVIEW OF LITERATURE

Many string matching algorithms and architectures have been proposed in recent years for the network security. The architecture of regular expression matching has been studied by Floyd and Ullman[3]. They have implemented NFA in hardware, proposing a hierarchical implementation based on Automata Theory. Wie Zhang has proposed the syntax of regular expression in PCRE (Perl Compatible Regular Expression) is the most popular definition which is used in deep packet inspection [1].The authors in [2] have proposed the novel method to reduce the memory requirement and still provide worst case speed guarantees called state merging DFA. They have implemented all state merging algorithm using C++.In this paper, Authors [12] propose how the Intrusion Detection System (IDS) has been adopted into the SSFNet and describe the implementation based on IDS. They extend the SSF'Net to be used as a framework which plays a role of defense system against the network intrusion. IDS is a software tool to detect and report the patterns of network intrusion on the internet. It can be classified into two categories: intrusion detection system based on hosts and that on networks. They propose a rule based system with misuse detect in the category of IDS on networks. Marly Roesch who is a developer of the Snort systems [6] defines a given network at "lightweight network intrusion detection system" when the network traffic and the packets on the IP network can be analyzed and logged in real-time. Like a network snipper who is based on the network packet collecting system library called "libpcap". Snort is a too1 to audit record and alert to the network traffic which allows to easily changing the rule sets to cope with various intrusion detections. It can work on many functions including protocol analysis search of text, rule matching, and detection of various intrusion attempts such as overflow. The rules of intrusion detection on the

Snort system can continually be updated by security community via the internet so that the information on new patterns of network intrusions can be reflected.

2.1 Summary of Literature Survey

Three algorithms are widely used to perform the pattern matching are as follows:

2.1.1. Kunth Morris Prat Algorithm

Knuth have proposed a string matching algorithm that turns the search string into a finite state machine, and then runs the machine with the string to be searched as the input string.

2.1.2. *Karp-Rabin Algorithm*

Karp and Rabin [6] have proposed a string matching algorithm that searches a pattern within a text using hashing. The main idea is to use a hash function to convert every substring in the text to a numeric value (hash value). The algorithm exploits the fact that if two strings are equal, their hash values are also equal.

2.1.3. Boyer-Moore Algorithm

The detection engine in Snort IDS depends on Boyer- Moore string matching algorithm. The Boyer-Moore algorithm is one of the early algorithms and is the most widely used algorithm for string matching. It is based on two heuristics: bad character heuristic and good suffix heuristic.

As mention above, out of three pattern matching algorithm KMP algorithm is perform the pattern matching on the rule set based on the regular expression.

2.2 KNUTH MORRIS PRAT ALGORITHM

Knuth, Morris and Pratt proposed a linear time algorithm for the string matching problem. A matching time of O(n) is achieved by avoiding comparisons with elements of 'S' that have previously been involved in comparison with some element of the pattern 'p' to be matched. i.e., backtracking on the string 'S' never occurs [09].

Components of KMP algorithm:

- The prefix function, Π

The prefix function,Π for a pattern encapsulates knowledge about how the pattern matches against shifts of itself. This information can be used to avoid useless shifts of the pattern 'p'. In other words, this enables avoiding backtracking on the string 'S'.

- The KMP Matcher

With string 'S', pattern 'p' and prefix function 'Π' as inputs, finds the occurrence of 'p' in 'S' and returns the number of shifts of 'p' after which occurrence is found.

The prefix function, Π

Following pseudocode computes the prefix fucnction, Π:

```
Compute-Prefix-Function (p)
m ← length[p]            //'p' pattern to be matched
 Π[1] ← 0
k ← 0
for q ← 2 to m
                do while k > 0 and p[k+1] != p[q]
               do k ← Π[k]
                  If p[k+1] = p[q]
                    then k ← k +1
                  Π[q] ← k
   return Π
```

The KMP Matcher:

The KMP Matcher, with pattern 'p', string 'S' and prefix function 'Π' as input, finds a match of p in S.Following pseudocode computes the matching component of KMP algorithm:

```
KMP-Matcher(S,p)
1 n ← length[S]
2 m ← length[p]
3 Π ← Compute-Prefix-Function(p)
4 q ← 0                    //number of characters matched
5 for i ← 1 to n           //scan S from left to right
6   do while  q > 0 and p[q+1] != S[i]
7        do  q ← Π[q]      //next character does not match
8             if p[q+1] = S[i]
9       then q ← q + 1     //next character matches
10   if q = m              //is all of p matched?
11       then print "Pattern occurs with shift" i – m
12            q ← Π[ q]    // look for the next match
```

Algorithm 1: KMP algorithm.

III. ARCHITECTURE

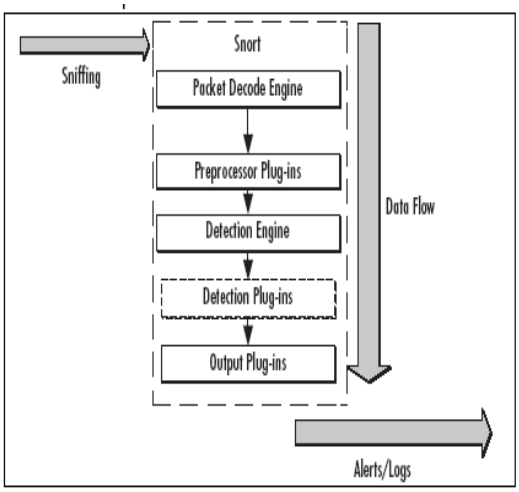

Fig. 3.1 Data flow diagram of IDS

IV. RESEARCH DESIGN

4.1 Method of Data Collection

Marly Roesch who is a developer of the Snort systems [6] defines a given network at "lightweight network intrusion detection system" when the network traffic and the packets on the IP network can be analyzed and logged in real-time. Like a network snipper who is based on the network packet collecting system library called "libpcap". Since the libpcap will be log into a database for my project and then with the help of pattern matching algorithm compare the content of the packet with the rule set present in the database.

4.2 Source of Data Collection

Snort is a Free and open source NIDS. It utilizes the rule driven language to perform the pattern matching. Since the detection process of Snort is heavily depend on the rule set present in the database.

4.3 Structure of IDS Rule

All IDS rules have two logical parts: rule *header* and rule *options*. This is shown in Figure 3-1.The rule header contains information about what action a rule takes. It also contains criteria for matching a rule against data packets. The options part usually contains an alert message and information about which part of the packet should be used to generate the alert message. The options part contains additional criteria for matching a rule against data packets. A rule may detect one type or multiple types of intrusion activity. Intelligent rules should be able to apply to multiple intrusion signatures.

Figure 4.3-1 Basic structure of IDS rules.

The general structure of rule header is shown in Figure 3-2. The *action* part of the rule determines the type of action taken when criteria are met and a rule is exactly matched against a data packet. Typical actions are generating an alert or log message or invoking another rule.

Figure 4.3-2 Structure of IDS rule header.

The *protocol* part is used to apply the rule on packets for a particular protocol only. This is the first criterion mentioned in the rule. Some examples of protocols used are IP, ICMP, UDP etc. The *address* parts define source and destination addresses. Addresses may be a single host, multiple hosts or network addresses. We can also use these parts to exclude some addresses from a complete network. There are two address fields in the rule. Source and destination addresses are determined based on direction field. As an example, if the direction field is "->", the Address on the left side is source and the Address on the right side is destination. In case of TCP or UDP protocol, the *port* parts determine the source and destination ports of a packet on which the rule is applied. In case of network layer protocols like IP and ICMP, port numbers have no significance. The *direction* part of the rule actually determines which address and port number is used as source and which as destination. For example, consider the following rule that generates an alert message whenever it detects an ICMP1 ping packet (ICMP ECHO REQUEST) with TTL equal to 100, as follow:

alert icmp any any -> any any (msg: "Ping with TTL=100"; \ ttl: 100;)

The part of the rule before the starting parenthesis is called the rule header. The part of the rule that is enclosed by the parentheses is the options part. The header contains the following parts, in order:

• A rule action: In this rule the action is "alert", which means that an alert will be generated when conditions are met. The packets are logged by default when an alert is generated. Depending on the action field, the rule options part may contain additional criteria for the rules.

• Protocol: In this rule the protocol is ICMP, which means that the rule will be applied only on ICMP-type packets. In the IDS detection engine, if the protocol of a packet is not ICMP, the rest of the rule is not considered in order to save CPU time. The protocol part plays an important role when you want to apply IDS rules only to packets of a particular type.

• Source address and source port. In this example both of them are set to "any", which means that the rule will be applied on all packets coming from any source. Of course port numbers have no relevance to ICMP packets. Port numbers are relevant only when protocol is either TCP or UDP.

• Direction. In this case the direction is set from left to right using the -> symbol. This shows that the address and port number on the left hand side of the symbol are source and those on the right hand side are destination. It also means that the rule will be applied on packets traveling from source to destination. You can also use a <- symbol to reverse the meaning of source and destination address of the packet. Note that a symbol <> can also be used to apply the rule on packets going in either direction.

• Destination address and port address. In this example both are set to "any", meaning the rule will be applied to all packets irrespective of their destination address. The direction in this rule does not play any role because the rule is applied to all ICMP packets moving in either direction, due to the use of the
keyword "any" in both source and destination address parts. The options part enclosed in parentheses shows that an alert message will be generated containing the text string "Ping with TTL=100" whenever the condition of TTL=100 is met. Note that TTL or *Time To Live* is a field in the IP packet header.

V. IMPLEMENTATION DETAILS

5.1 Winpcap

Libpcap and WinPcap provide the packet-capture and filtering engines of many open source and commercial network tools, including protocol analyzers (packet sniffers), network monitors, network intrusion detection systems, traffic-generators and network-testers.

Libpcap and WinPcap also support saving captured packets to a file, and reading files containing saved packets; applications can be written, using libpcap or WinPcap, to be able to capture network traffic and analyze it, or to read a saved capture and analyze it, using the same analysis code. A capture file saved in the format that libpcap and WinPcap use can be read by applications that understand that format, such as tcpdump, Wireshark, CA NetMaster, or Microsoft Network Monitor 3.x.

The MIME type for the file format created and read by libpcap and WinPcap is application/vnd.tcpdump.pcap. The typical file extension is .pcap, although .cap and .dmp are also in common use. [2]

5.2 *Snapshot*

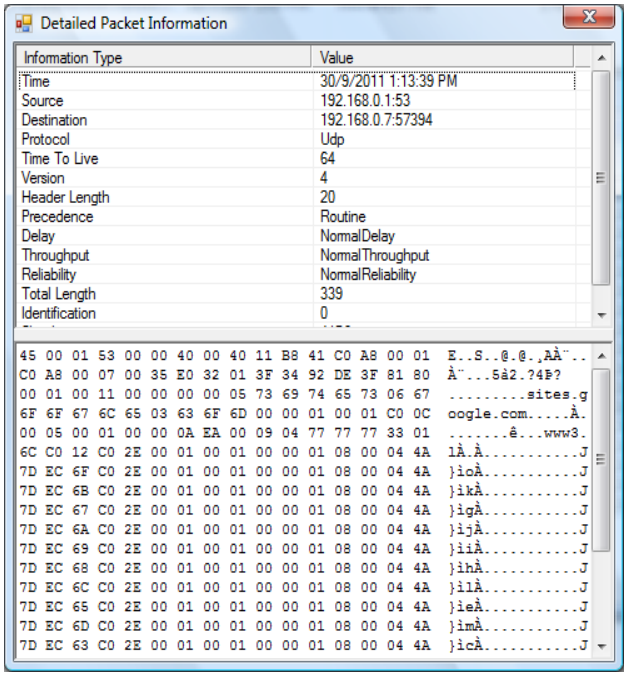

Fig 5.1 Packet Details

As soon as we start the internet, the host systems on which we access this module start capturing the packets. It shows the data in the decimal format. The details of the captured packets are shown in the snapshot. The function I used to capture and monitor the packet is as follows:

Getting IP address to Keep watch / monitor

m_Monitor = new Socket (AddressFamily.InterNetwork, SocketType.Raw, ProtocolType.IP);

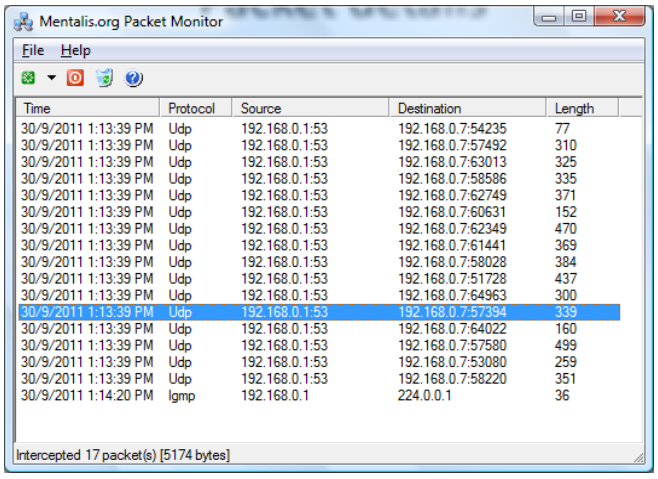

Fig 5.2 Packet Information and Hex Data

Once we select any packet by double click on it that is shown in the first snapshot, we are able to see the details of the packet i.e. the header field and the payload. The header part is consist of source IP address and destination IP address, name of the protocol, Time to live field, version of a protocol, Header length, various type of services and the total length field. The data of the header field is shown in the decimal form whereas the data of the payload is display in the hexadecimal form. The function which is used to get the details of the packet which is shown in the snapshot is as follow:

Collecting data in HEX or raw format

m_Monitor.IOControl (SIO_RCVALL, BitConverter.GetBytes ((int) 1), null);

Start Packet Receiving and execution of user defined function on receive

m_Monitor.BeginReceive(m_Buffer, 0, m_Buffer.Length, SocketFlags.None, new AsyncCallback(this.OnReceive), null)

VI. POSSIBLE CONTRIBUTION OF THE STUDY

Memory requires to store the rule base will be small, so memory optimization can be achieved. Improvised version of pattern matching technique will be implemented. Implement the technique to improve the structure of rule based to perform the fast and efficient pattern matching.

VII. FUTURE WORK

Store the self defined rules in our database using rules we will match the incoming packet with the known signature Use regular expression as a part of our rule set to minimise the matching time.

VII. CONCLUSION

With Time New Malicious data with New Pattern may exist, Update can be done in this Project. Other Algorithm can be used as per Requirement. New Rules can be added later on.Improvement of Rules Matching by Dynamic Adjustment Algorithm

IX. REFERENCES

[1] Wei Zhang, Yibo Xue, Dongsheng Wang, and Tian Song, "A Multiple Simple Regular Expression Matching Architecture and Coprocessor for Deep Packet Inspection" IEEE INFOCOM 2008, Phoenix USA.

[2] Jiekun Zhang, Dafang Zhang and Kun Huang, " A Regular Expression Matching Algorithm using Transition Merging" IEEE, 2009.

[3] Perl Compatible Regular Expression, http ://www.pcre.org

[4] M. Becchi and S. Cadambi, "Memory efficient regular expression search using state merging," In Proc. of IEEE Infocom 2007, May 2007.

[5] J. van Lunteren, "High-Performance pattern-matching for intrusion detection," in 25th Conference of IEEE Infocom, Apr.2006.

[6] S. Kumar, B. Chandrasekaran, J. Turner, and G. Varghese. "Curing regular expressions matching algorithms from insomnia, amnesia, and acalculia," In ANCS 2007, pages 155-164.

[7] S. Kumar, S. Dharmapurikar, and F. Yu. "Algorithms to accelerate multiple regular expression matching for deep packet inspection," In Proc. of SIGCOMM'06, pages 339-350.ACM.

[8] S. Kumar, J. Turner, and J. Williams. "Advanced algorithms for fast and scalable deep packet inspection," In Proc. of ANCS'06, pages 81-92. ACM.

[9] F. Yu, Z. Chen, Y. Diao, T. V. Lakshman, and Randy H. Katz. "Fast and memory efficient regular expression matching for deep packet inspection," In Proc. Of ANCS'06, pages 93-102.

[10] D. Ficara, S. Giordano, G. Procissi, et al. "An improved DFA for fast regular expression matching," ACM SIGCOMM Computer Communications Review, 38, 2008.

[11] Jill Hyuk Kim, Seung Kyu Park Jung kuk Sea Joo Beom Yun, Dae Sik Choi. "Implementation of IDS for Network Intrusion simulation based on SSFNet", 10th Asis~Pacific. Conference on Communications and 5th International Symposium on Multi-Dimensional Mobile Communications 09.

[12] T. Johnson, S. Muthukrishnan, and I. Rozenbaum, "Monitoring regular expressions on out-of-order streams," in *Data Engineering, 2007. ICDE2007. IEEE 23rd International Conference on,* April 2007, pp. 1315–1319.

