International Journal of Computer Trends and Technology
ISSN: 2231-2803 / https://doi.org/10.14445/22312803/IJCTT-V67I4P115
Volume 67 Issue 4, 67-73, April 2019

Original Article

Real-Time Packet in Network Intrusion Detection System Filtering Module

Kamaljeet Singh[1], Umesh Sehgal[2]

Assistant Professor, Faculty of Computational Science & GNA University , GNA University, Phagwara-Punjab India

Abstract - *Computer networks bring us the benefits, such as more computing power and better performance for a given price, and some challenges and risks, especially in system security. During the past two decades, significant effort has been put into network security research, and several techniques have been developed for building secure networks. Packet filtering plays an important role in many security-related techniques, such as intrusion detection, access control, and a firewall. A packet-filtering system constitutes the first line of defense in a computer network environment. The key issues in the packet-filtering technique are efficiency and flexibility. Efficiency refers to the ability of a filter to quickly capture network packets of interest, while flexibility means the filter can be customized easily for different packet patterns.*

This paper presents a real-time packet-filtering module, which can be integrated into a large-scale network intrusion detection system. The core of this packet-filtering module is a rule-based specification language, ASL (Auditing Specification Language), used to describe the packet patterns and reactions for a network intrusion detection system. The important features of ASL that are not provided by other packet-filtering systems are protocol independence and type safety. ASL provides several new features that distinguish it from other languages used for intrusion detection and packet filterings, such as packet structure description and protocol constraint checking.

We develop the algorithms and heuristics for constructing a fast packet filter from ASL specifications. Our algorithms improve upon existing techniques in that the performance of the generated filters is insensitive to the number of rules. We discuss the implementation of these algorithms and present experimental results.

Keywords - *Sensor Sniffing Tools, NF2 with MATLAB filtering.*

I. INTRODUCTION

Computation models have experienced a significant change since the emergence of computer networks, which allow heterogeneous computers to communicate with each other. During the past two decades, several interconnected computers have replaced most centralized systems. This Factor has led to more computing power, increased flexibility, and better performance/price ratio.

However, at the same time, we also face many new challenges and risks with networked computing, such as lack of communication reliability, coordination, resource management, and so on. As more and more computer networks are brought into electronic commerce, transaction management, and even national defense, people begin to pay increasing attention to system security.

II. NETWORK SECURITY AND POTENTIAL THREATS

There are a number of security issues for a computer network environment [1]:

- *Availability:* The system must be functional and correctly provide services.
- *Confidentiality:* The data transmitted from one system must be accessible only to the authorized parties.
- *Authentication:* The identity associated with the data must be correct. The identity can apply to a user, host, or software component.
- *Integrity:* The processed or transmitted data can be modified only by the authorized parties.
- *Non-repudiation:* Neither the sender nor the receiver of data can deny the fact of data transmission.

.

A. *Intrusion Detection*

As defined by Heady et al. [2], an intrusion is any set of actions that attempt to comprise the integrity, confidentiality, or availability of a resource. Intrusion leads to violations of the security policies of a computer system, such as unauthorized access to private information, malicious break-in into a computer system, or rendering a system unreliable or unusable.

A full-blown network security system should include the following subsystems:

- *Intrusion Detection Subsystem:* Distinguishes a potential intrusion from a valid network operation.
- *Protection Subsystem:* Protects the network and security system from being compromised by network intrusions.

- *Reaction Subsystem:* This part traces an intrusion's origin or fights back against the hackers.

This thesis focuses on the intrusion detection subsystem, which constitutes the first line of defense for a computer network system. There are a number of approaches in this field. They fall into three primary categories: anomaly detection, misuse detection, and hybrid schemes.

The anomaly detection approach is based on a model of normal activities in the system. This model can either be predefined or established through techniques such as machine learning. An anomaly will be reported once there is a significant deviation from this model. By contrast, a misuse detection approach defines specific user actions that constitute misuse and uses rules for encoding and detecting known intrusions [3]. The hybrid detection approach uses a combination of anomaly and misuse detection techniques.

III. KEY CONTRIBUTIONS

Packet filtering is a critical technique in network management, firewall strategy, and intrusion detection. However, the existing packet filtering systems have a number of limitations in system efficiency, flexibility, and scalability. For instance, a packet filter for one protocol suite cannot easily be changed to fit another. In addition, most packet filters suffer from significant performance degradation as the number of packet patterns increases.

In this thesis, we present a novel approach for constructing a real-time packet-filtering module that can be used for network intrusion detection purposes. One of the main contributions of our approach is a specification language designed for describing intrusion patterns and reactions. This language provides a number of features that distinguish it from other specification languages used for intrusion detection or packet filterings, such as protocol independence and type safety. Another important focus of our work is the development of fast pattern-matching algorithms (for packet filters) that are insensitive to the number of patterns.

A. Synopsis Organization

We briefly review the TCP/IP (Transmission Control Protocol/Internet Protocol) protocol suite and several security holes in the design and implementation of TCP/IP. Chapter 3 surveys some existing techniques in building a secure computer network system. We also discuss some general issues on packet filtering, which is one of the main techniques in network intrusion detection. Chapter 4 gives a detailed description of our specification language and its application to intrusion detection. Chapter 5 discusses the issues in designing and implementing our packet-filtering module. The primary concern is to reduce the processing time of a packet filter. In the last chapter, we provide some experimental results from our packet filter performance testing and summarize our work.

IP is the workhorse protocol of the TCP/IP protocol suite. It provides an unreliable, connectionless datagram delivery service. All the TCP, UDP (User Datagram Protocol).

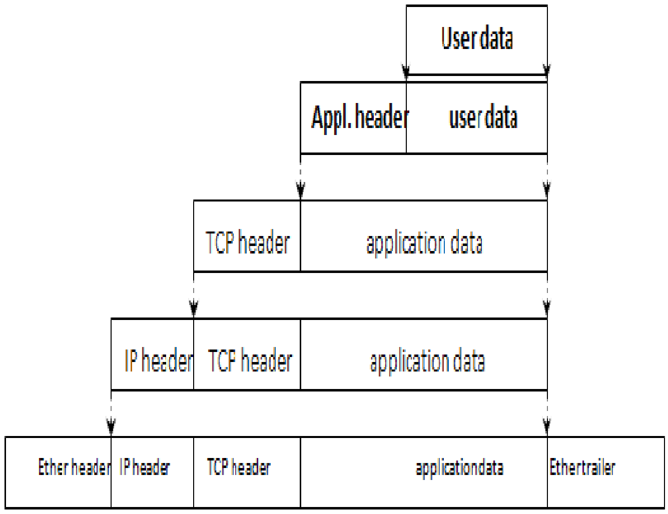

Fig. 1.1 TCP/IP Protocol Hierarchy

ICMP (Internet Control Message Protocol) and IGMP (Internet Group Management Protocol) data are transmitted as IP datagrams [4].

An IP header has information like source IP address and destination IP address, which plays an important role in routing a packet around the networks. A detailed description of the IP header can be found in [4]. Figure 2.2 shows the structure of a normal IP header.

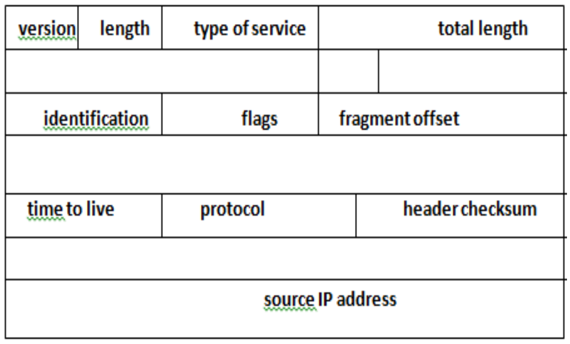

Fig. 1.2 IP Header

Delivering a packet to the correct destination is non-trivial, especially in a large-scale network. Each intermediate routing device makes its best effort to deliver the IP packet, but there is no guarantee that it will reach its destination finally. So, a packet can be lost, duplicated, or delivered out of order [4]. It is the task of higher-layer protocols to correct such errors.

UDP is a transport layer protocol, but it does not offer much functionality over and above IP. The port numbers in the UDP header identify the sending process and the receiving process [4], while the checksum provides the limited ability for error detection (Figure 2.3).

source port number	destination port number
UDP length	UDP checksum

Fig. 1.3 UDP Header

However, due to its simplicity and low overhead compared to connection-oriented protocols, UDP is suitable for designing simple request/reply application protocols, such as DNS (Domain Name System), SNMP (Simple Network Management Protocol), and so on.

B. TCP

TCP is built on top of the IP layer, which is unreliable and connectionless. But TCP provides the higher layer application a reliable connection-oriented service. Each TCP connection requires an established procedure and a termination step between communication peers as the tradeoff. TCP also provides sequencing and flow control.

A TCP header occupies 20 bytes without any option, as shown in Figure 2.4. The sequence number is essential in keeping the sending and receiving datagram in proper order.

Source port number	Destination port number
Sequence number	
Acknowledge Number	
Header Reserved URG,ACK,PSH,RST,SYN,FIN	Window Size

Fig. 1.4 TCP Header

There are six flag bits within a TCP header, namely URG, ACK, PSH, RST, SYN, and FIN, each of which has a special meaning in connection establishment, connection termination, or other control phases. Window size, which specifies how many bytes of data can be accepted each time by the receiving side, is advertised between the two communication peers for flow control.

TCP establishes a connection in three steps, commonly known as a three-way handshake. Figure 2.5 shows a typical three-way handshake procedure between a source host S and a destination host D.

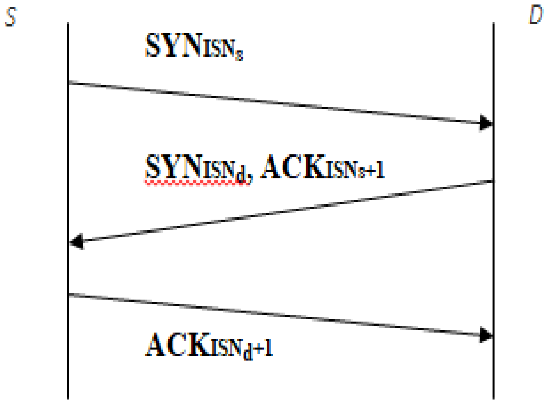

Fig. 1.5 Three-Way Handshake

First, S sends an SYN packet to D to establish a connection. Meanwhile, S sets its own ISN (Initial Sequence Number) in the sequence number field of the packet. Upon receiving the request packet, D sends back an SYN_ACK packet as the acknowledgment, including its ISN and the incremented ISN from S. As the acknowledgment packet reaches the source host S, S immediately transmits an ACK packet back to D. In the last ACK packet, S needs to include the incremented ISN of D as the confirmation of the connection. At this point, the connection has been set up. One extra point is to suppose that host S does not send any SYN packet but receives an SYN_ACK packet from host D; it will then send back an RST packet to reset the connection.

IV. CONCLUSION

APPENDIX A PACKET DATA STRUCTURES FOR ASL

```
Ethernet Header:
------------------------------------
----------------------------------
#define ETHER_LEN        6

struct ether_hdr
{
      byte
      e_dst[ETHER_LEN];
      byte
      e_src[ETHER_LEN];
      short              e_type;
}

ARP:
------------------------------------
----------------------------------
#define ETHER_IP         0x0800
#define ETHER_ARP        0x0806

struct arp_hdr : struct ether_hdr
with e_type == ETHER_ARP
{
      short ar_hrd;      /* Format of
hardware address */
```

```
      short ar_pro;      /* Format of
protocol address */
      byte  ar_hln;      /* Length of
hardware address */
      byte  ar_pln;      /* Length of
protocol address */
      short ar_op;       /* ARP
opcode (command). */
}

/* ARP protocol HARDWARE identifiers
*/
#define ARPHRD_ETHER    1
      /* Ethernet 10Mbps */
/* ARP protocol PROTOCOL identifiers
*/
#define ARPPRO_IP       0x0800
      /* IP */
/* ARP protocol opcodes */
#define ARPOP_REQUEST   1
      /* ARP request */
#define ARPOP_REPLY     2
      /* ARP reply */
#define ARPOP_RREQUEST  3
      /* RARP request */
#define ARPOP_RREPLY    4
      /* RARP reply */

struct ether_ip_arp : struct arp_hdr
with
      (ar_hrd == ARPHRD_ETHER) &&
(ar_pro == ARPPRO_IP)
{
      byte
      arp_sha[ETHER_LEN];/* sender
hardware address */
      int         arp_spa;
      /* sender protocol address */
      byte
      arp_tha[ETHER_LEN];/* target
hardware address */
      int         arp_tpa;
      /* target protocol address */
}
```

IP:

```
struct ip_hdr : struct ether_hdr
with e_type == ETHER_IP && ihl == 5
{
      bit         version[4];
      /* ip version */
      bit         ihl[4];
      /* header length */
      byte        tos;
      /* type of service */
      short       tot_len;
      /* total length */
      short       id;
      /* identification */
      bit         flag[3];
      /* flags */
      bit         frag_off[13];
      /* fragment offset */
      byte        ttl;
      /* time to live */
      byte        protocol;
      /* protocol */
      short       check_sum;
      /* header checksum */
      ip_addr     s_addr;
      /* source ip address */
      ip_addr     d_addr;
      /* destination address */
}

struct ip_pkt : struct ip_hdr
{
      byte  ip_data[tot_len - ihl];
}
```

ICMP:

```
/* IP protocol PROTOCOL identifiers.
*/
#define IP_ICMP         0x0001
      /* ICMP */
#define IP_IGMP         0x0002
      /* IGMP */
#define IP_TCP          0x0006
      /* TCP */
#define IP_UDP          0x0011
      /* UDP */

struct icmp_hdr : struct ip_hdr with
protocol == IP_ICMP
{
      byte        icmp_type;
      /* icmp message type */
      byte        icmp_code;
      /* icmp message code */
      short       icmp_csum;
      /* checksum for entire message
*/
}

struct icmp_pkt : struct icmp_hdr
{
      byte  icmp_data[tot_len - ihl
- sizeof(icmp_hdr)];
}

#define ICMP_ECHO_TYPE_REQUEST
      8
#define ICMP_ECHO_TYPE_REPLY
      0
#define ICMP_ECHO_CODE
      0
```

```
struct icmp_echo_request : struct
icmp_hdr with
      (icmp_type ==
ICMP_ECHO_TYPE_REQUEST)
      && (icmp_code ==
ICMP_ECHO_CODE)
{
      byte        icmp_echoid;
      /* identifier */
      byte        icmp_echoseq;
      /* sequence number */
      byte
      icmp_echodata[tot_len - ihl -
sizeof(icmp_hdr) - 2];
}

struct icmp_echo_reply : struct
icmp_hdr with
      (icmp_type ==
ICMP_ECHO_TYPE_REPLY) && (icm_code
== ICMP_ECHO_CODE)
{
      byte        icmp_echoid;
      /* identifier */
      byte        icmp_echoseq;
      /* sequence number */
      byte
      icmp_echodata[tot_len - ihl -
sizeof(icmp_hdr) - 2];
}

#define ICMP_DESUNREA_TYPE
      3
struct icmp_unreach : struct
icmp_hdr with
      icmp_type ==
      ICMP_DESUNREA_TYPE
{
      short       icmp_reserved;
      ip_hdr      icmp_iphdr;
      byte        icmp_data[8];
}

#define ICMP_SRCQUEN_TYPE     4
#define ICMP_SRCQUEN_CODE     0
struct icmp_squench : struct
icmp_hdr with
      icmp_type == ICMP_SRCQUEN_TYPE
{
      short       icmp_reserved;
      ip_hdr      icmp_iphdr;
      byte        icmp_data[8];
}

UDP:
------------------------------------
----------------------------------
struct udp_hdr : struct ip_hdr with
protocol == IP_UDP
{
      byte        udp_sport;  /*
source port number */
      byte        udp_dport;  /*
destination port number */
      byte        udp_len;    /*
header + data length */
      byte        udp_csum;   /*
checksum for header & data */
}

struct udp_pkt : struct udp_hdr
{
      byte  udp_data[udp_len -
sizeof(udp_hdr)];/* data */
}


TCP:
------------------------------------
----------------------------------
struct tcp_hdr : struct ip_hdr with
protocol == IP_TCP
{
      short       tcp_sport;
      /* source port number */
      short       tcp_dport;
      /* destination port number */
      int         tcp_seq;
      /* sequence number */
      int         tcp_ack;
      /* acknowledge number */
      bit         tcp_hlen[4];
      /* header length */
      bit         tcp_reserved[6];
      /* reserved */
      bit         tcp_urg;
      /* flags */
      bit         tcp_ack;
      bit         tcp_psh;
      bit         tcp_rst;
      bit         tcp_syn;
      bit         tcp_fin;
      short       tcp_win;
      /* window size */
      short       tcp_csum;
      /* checksum for header & data
*/
      short       tcp_urp;
      /* urgent pointer */
}

struct tcp_pkt : struct tcp_hdr
{
      byte  tcp_data[tot_len - ihl -
tcphlen];
}

DNS:
------------------------------------
----------------------------------
#define DNS_PORT        53
```

```
struct dns_hdr: struct udp_hdr with
      (udp_sport == DNS_PORT) ||
(udp_dport == DNS_PORT)
      /* either to a dns port or
from dns port */
{
      short       dns_id;
      /* identifier */
      short       dns_flags;
      /* flags */
      short       dns_nques;
      /* No. of questions */
      short       dns_nans;
      /* No. of answers RR */
      short       dns_nauth;
      /* No. of authority RRs */
      short       dns_nadd;
      /* No. of additional RRs */
}

struct dns_ques
{
      string      dns_qname;
      /* query name */
      short       dns_qtype;
      /* query type */
      short       dns_qclass;
      /* query class */
}

struct dns_rr_hdr
{
      string      dname;
      /* domain name */
      short       type;
      /* RR type */
      short       class;
      /* RR class */
      int         ttl;
      /* time to live */
}

#define DNS_QUERY_A           1
struct dns_rr_A : struct dns_rr_hdr
rrhdr with rrhdr.type == DNS_QUERY_A
{
      short rdlen;            /*
resource data length */
      ip_addr rdata[rdlen];
}

struct dns_pkt_A: struct dns_hdr
dnshdr
{
      struct dns_ques
      dques[nques];     /* dns
questions */
      struct dns_rr_A   dans[nans];
      /* dns answer RRs */
      struct dns_rr_A
      dauth[nauth];     /* dns
authority RRs */
      struct dns_rr_A   dadd[nadd];
      /* dns additional RRs */
}

RIP:
-----------------------------------
----------------------------------
#define RIP_PORT        520

struct rip_hdr: struct udp_hdr with
      (udp_sport == RIP_PORT) ||
(udp_dport == RIP_PORT)
      /* either to a rip port or
from rip port */
{
      byte        rip_command;
      /* rip command */
      byte        rip_version;
      /* rip version */
      short       rip_zero;
      /* must be zero */
}

struct rip_rec
{
      short       rip_afid;
      /* address family identifier
*/
      short       rip_zero;
      /* must be zero */
      int         rip_ipaddr;
      /* ip address */
      int         rip_zero[2];
      /* must be zero */
      int         rip_metric;
      /* metric */
}

strcut rip_pkt : struct rip_hdr
{
      rip_rec riprec[(udp_len -
sizeof(struct rip_hdr))
            / sizeof(struct
rip_rec)];
}
```

REFERENCES

[1] Larry J. Hughes, Jr. Useful Internet Security Techniques, New Riders Publishing, Indianapolis, IN, 1995.

[2] R.Heady, G. Luger, A. Maccabe, and B. Mukherjee. A Method To Detect Intrusive Activity in a Networked Environment. In Proceedings of the 14th National Computer Security Conference, pages 362-371, October 1991.

[3] Abdelaziz Monnji. Languages and Tools for Rule-Based Distributed Intrusion Detection, Ph.D. thesis, Faculties Universitaires, Notre-Dame de la Paix, Belgium, September 1997.

[4] W. R. Stevens. TCP/IP Illustrated Vol. 1 – The Protocols, Addison-Wesley Publishing Company, Inc. Reading, MA, 1994.

[5] S.M.Bellovin. Security Problems in the TCP/IP Protocol Suite, Computer Communications Review, Vol. 19, No. 2, pp. 32-48, April 1989.
[6] Morris R. A Weakness in the 4.2 BSD UNIX TCP/IP Software, Computer Science Technical Report No 117, AT&T Bell Laboratories, Murray Hill, NJ, 1985.
[7] CERT. TCP SYN Flooding and IP Spoofing Attacks, Carnegie Mellon University, Pittsburgh, PA, September 1996
[8] C.Cobb and S. Cobb. Denial of Service, Secure Computing, pp.58-60, July 1997.
[9] C.L.Schuba, I.V. Krsul, Makus G. Kuhn, E.H. Spafford, A. Sundaram, D. Zamboni. Analysis of a Denial of Service Attack on TCP, Purdue University, West Lafayette, 1996.
[10] S.Dash. Integration of DNSSEC (key-server) with Ssh Application, MS thesis, Iowa State University, Ames, IA, 1997.
[11] W.R.Stevens. UNIX Network Programming Vol. 1 – Network APIs: Sockets and XTI, Second Edition, Prentice Hall PTR, Upper Saddle River, NJ, 1998.
[12] Vern Paxson. Bro: A System for Detecting Network Intruders in Real-Time, Lawrence Berkeley National Laboratory, Berkeley, CA, 1998.
[13] R.C.Sekar, R. Ramesh, I. V. Ramakrishnan. Adaptive Pattern Matching, Bellcore, Morristown, NJ, 1993.
[14] Steven McCanne, Van Jacobson. The BSD Packet Filter: A New Architecture for User-level Packet Capture, Lawrence Berkeley Laboratory, Berkeley, CA, 1992.
[15] Biswanath Mukherjee, L. Todd Heberlein, Karl N. Levitt. Network Intrusion Detection, IEEE Network, pp.26-41, May/June 1994.
[16] Frederick B. Cohen. A Node on Distributed Coordinated Attacks, Computer & Security, pp.103-121, v15, 1996.
[17] Steven Cheung, Karl N. Levitt. Protecting Routing Infrastructures from Denial of Service Using Cooperative Intrusion Detection, University of California, Davis, CA, 1997.
[18] Christoph L. Schuba. Addressing Weakness in the Domain Name System Protocol, COAST Laboratory, Purdue University, West Lafayette, IN, 1993
[19] Eugene H. Spafford. The Internet Worm Incident, Technical Report CSD-TR-993, Purdue University, West Lafayette, IN, September 19, 1991