The International Journal Of Engineering And Science (IJES)
||Volume||2 ||Issue|| 4 ||Pages|| 18-21 ||2013||
ISSN(e): 2319 – 1813 ISSN(p): 2319 – 1805

Multi-Threaded User Space Network Stack For Intrusion Prevention System

[1,]Amar More, [2,] Narender Singh Chaudhary, [3,] Nayan Shah , [4,] Vivek Jadon, [5,] Satish Goswami
[1,2,3,4,5,]*Department of Computer Science, MIT Aoe(Pune)*

----------**Abstract**----------
Nowadays, internet attacks have increased manifold. As the propagation speed of attacks has increased significantly, waiting for the attacks to happen and look for the diagnosis later on is of no use. Thus a proactive technique for countering attacks called NIPS comes into play. These prevention system can be implemented in both kernel and user space. The performance of kernel space implementations are always better as compared to user space implementations as it comes to processing the packets at higher rate because in user space these systems cannot keep up with speed of the network and starts dropping the packets. On the other hand the effects of system fault in kernel space are much more serious than in user space and thus user space implementation is much more robust. The performance of user space NIPS can be improved by providing a network stack in the user space itself which can overcome the factors responsible for reduced performance of these systems.

Keywords *-Internet attacks, NIPS, kernel space, user space, system fault, robust, Network Stack.*

Date of Submission: 25 March 2013 Date of Publication: 15April 2013

I. INTRODUCTION

A firewall is considered as first line of defense in protecting private information. It is set of filters or rules that are matched against traffic. It can try to detect malicious traffic that tries to enter a computer system but cannot detect anything that has already entered the computer system. For greater security IDS and IPS should be used along the firewall.

IDS (Intrusion Detection System) [1] is considered as passive monitoring system since it just warns you of the suspicious activity taking place. IDS monitors only the copy of the traffic but the problem arises when the real malicious packets have already passed to their intended target. An IPS (Intrusion Prevention System) has all the features of a good IDS but it can also stop malicious traffic from entering the computer system. Unlike IDS, an IPS [2] monitors the real traffic and drops any malicious packets sent over the wire. It can:

[1] Stop the attack by terminating the network connection or user session which is originating the attack.
[2] Blocking access to the target from the user account, IP address, or by blocking all access to the targeted host, service, or application.

In spite of the above advantages that an IPS offers, there are risks associated with it more than that of IDS. The IPS monitors live traffic passing through it. So if IPS fails, or becomes overwhelmed, it will affect your live traffic. Hence IPS influences the network performance as each packet is analyzed in terms of malicious content before being routed to the intended destination. An IPS can be realized in either kernel or user space [3]. Each user space process is regularly interrupted in order to allow pre-emptive multitasking. This is not the case in kernel processes. Further, transferring data from kernel space to user space often requires a set of copy operations which further reduces the execution performance. Efficiency is not the only aspect to be considered in the design, but appropriate attention needs also to be paid to security and robustness. The fact that code which is executed in kernel space cannot be restricted or supervised strongly demands for execution in user space. Process that run in user space can be supervised and restricted.

Finally, regarding robustness, the effects of a system-fault in kernel space are much more serious than in user-space, a fact that also strongly demands for user space implementation of intrusion prevention modules. Thus we are implementing a user space network stack in Linux which will act as a support for the execution of an IPS.

II. MOTIVATION

In our modern network world, the software, protocols and algorithms involved in communication are among some of the critical parts of the operating system. The core communication software in most modern systems is the network stack. There are many overheads in packet processing [4] which results in reduced performance of network stack. So, we need a design which is capable of handling these overheads so that the network stack can work efficiently. The current implementation lacks adaptability and interaction with the applications. Implementing the network stack in user space provides us the opportunities to make modifications in the current implementations so that it can overcome the problems of current implementations. The main components of our user level implementation are as follows:

- Device driver: We need a specialized device driver which can prove us direct NIC access.
- Memory Pool: Pre-allocated memory area divide into 2000 bytes chunks to facilitate faster allocation and de-allocation of memory for outgoing and incoming packets.
- User level network stack: This is the core of our project which will provide us better performance and flexibility in designing and development.

III. RELATED WORK

Various works have been done earlier in user space regarding implementations of network stack.

3.1 Daytona

There are several overheads in the networking stack due to which desirable performance cannot be achieved in higher rate of traffic. Monitoring and profiling of the TCP stack can reveal sources of such overheads. However this becomes a challenge when the TCP implementation resides in the kernel. User Level TCP stack can help us address these diverse challenges. Daytona [5] is a user level TCP implementation, a library available to Linux applications, which works with any network interface and get the same network functionality as provided by the kernel and is also largely independent on kernel version. The kernel is just a communication channel between the library and the network, through the use of raw IP sockets.

3.2 Arsenic

Arsenic [6] was a Linux 2.3.29-based user level TCP implementation. The goal of this implementation was to give user level applications better control for managing bandwidth on a network interface. This implementation was designed to work with a specialized gigabit network interface which provided per connection contexts in interface hardware. Its goal was to extract the best performance and QoS functionality from a specialized network interface. Arsenic's design thus involves a coupling with the interface hardware, and a dependence on the kernel version. Providing zero-copy transfers and connection-specific buffers was a key goal of the project. In contrast to Daytona, our network stack tries to implement zero-copy mechanism [7] where only one copy of the packet is stored in our memory pool and the reference is passed to the protocol layers above the stack. Our implementation works specifically on Ethernet interface because we are getting NIC access from Pfring module which works only for Ethernet interface.

3.3 Fidran and Mipsa

Fidran [3] is a flexible intrusion prevention system for active networks which analyses packets in kernel space. Although it is possible to install user-space modules, a part of the packet processing always takes place in the kernel. Another prototype of an IPS, a modular intrusion prevention system MIPSA [3], which runs completely in user space and is designed to cooperate with active networking environment AMnet. As compared to the kernel space implementation of IPS which required copy operations from kernel space to user space which slows down the execution. In user space implementation of the network stack, both IPS and the memory pool exist in user space itself which will ease the performance a little since there will be no switching from kernel to user space.

IV. ARCHITECTURAL DESIGN

The overall architecture of our approach represents the hardware components, software components and the relationship between them. Lower level design: At lower level we are mainly having the following components - NIC, drivers, and ring buffers. NIC's main task is to receive and transmit the packets to physical network. When it receives the packet it copies it to the ring buffer [8]. In our case the ring buffers is the memory pool in user space and its main task is to hold the packets until they are used by either network stack or NIC. The NIC also informs the device driver about the availability of packets so that the further processing can be performed. Upper level design: At the middle, are our network protocols implemented in the network stack. The network stack completely functions in user space. Its main task is to read to write the packets from memory pool and process them. The processed packets are then given to the intended application using socket interface.

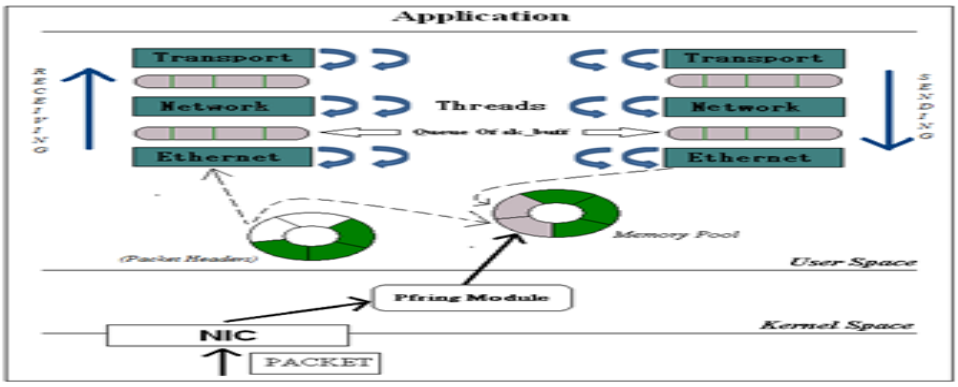

Fig. 1 User Space network stack and its relation with NIC.

The following steps take place in above figure:

- As the network packet arrives at NIC during reception, the installed Pfring module directly stores the packet in our memory pool.
- The multiple threads present at the Ethernet layer fetch the reference of the packets from their stored location in memory pool.
- While one of the threads process a packet at Ethernet layer and passes it to the layer above, another threads continuously fetches packets coming at NIC from the memory pool.
- The thread then passes the processed packet (removing headers) from one layer to the layer above of our TCP/IP stack and the same functionality is carried up the stack till the transport layer and finally to the application layer.Threads from different layers communicate through queues present between two adjacent layers.

V. IMPLEMENTATION DETAILS

The intent of this section is to describe the code structure that implements the various pieces of our user space network stack. Our implementation is best suited for Linux kernel version 2.6 and above.

5.1 Lower Level Processing

At the lower level i.e. the Ethernet layer of our TCP/IP stack, fetching the "up" interfaces takes place and continuous listening is done on those interfaces for availability of packets. Packets which arrive at the NIC are handled and stored in our memory pool of the modified skbuff structure or Packet Headers (Fig. 1). This user space memory pool performs same function as skbuff pool in kernel space.The packet header is then analysed to see whether the packet has to be passed to ARP or IP and then stored in the queue above respectively.

5.2 Protocol Processing at Network Layer

After the packet arrives in network layer, the packet header is checked to see if its IP address matches with the Local IP address and decide whether to forward the packet or send to local network. If there is any problem with any of the fields in packet header then error message is to be sent through ICMP.If the packet at network layer is being handled by IP, then respective options processing, defragmentation (During reception) and fragmentation (During transmission) takes place if needed. If the packet is being handled by ARP during

transmission then first a ARP cache lookup is performed to check for the entry of particular IP address and MAC address is obtained from cache itself else ARP resolution takes place.From the IP layer the packet is then sent to the queue above for processing in the transport layer or ICMP.

5.3 Protocol Processing at the Transport layer

In UDP, again the same functionality is carried on like IP i.e. validating the packet headers and sending it to the layer above which is the socket layer and then eventually to the application for which it is destined.Apart from above functionality, we have maintained a linked list for all the "up" interfaces at the lower level and a linked list for protocol at the transport layer. A hash function is applied to the protocol field in the packet during reception. This obtained hash value is matched against position in the linked list and if no match is found then the protocol is added at the end of the linked list. By this way, we can dynamically add protocols to the list.

VI. CONCLUSION

Intrusion Prevention is a vital dose for rapidly increasing internet attacks. Along with efficiency, appropriate attention needs to be paid on robustness of the system. Since IPS will perform at the user level where our network stack exists, there will be no overhead of copy operations from kernel to user space or vice versa. Our network stack which is already overcoming overheads of the kernel network stack will act as a support for IPS execution.

REFERENCES

[1] Intrusion Detection System" Wikipedia, [Online] Available: http://en.wikipedia.org/wiki/Intrusion_detection_system.
[2] Intrusion Prevention System" Wikipedia, [Online] Available: http://en.wikipedia.org/wiki/Intrusion_prevention_system.
[3] Hess, T. Gingold, S. R. Garzon und G. Schäfer, "Intrusion Prevention with Active Networks: A Performance Comparison between
[4] User and Kernel-Space Implementation".
[5] Matthew Whitworth, "Improving networking by moving the network stack to user space".
[6] Prashant Pradhan, Srikanth Kandula, Wen Xu, Anees Shaikh, Erich Nahum, "Daytona : A User-Level TCP Stack".
[7] Ian Pratt and Keir Fraser. "Arsenic: A User-Accessible Gigabit Ethernet Interface." In Proceedings of IEEE INFOCOM,
[8] Anchorage, Alaska, USA, April 2001.
[9] Ulrich Drepper, "The Need for Asynchronous, Zero-Copy Network I/O Problems and Possible Solutions", Red Hat, Inc.,
[10] drepper@redhat.com.
[11] Srinivas Krishnan, "Improving Memory and Interrupt Processing in FreeBSD", Network Department of Computer Science, University of North Carolina, krishnan@cs.unc.edu.
.