Coursework 2018-2019
CS 241 Operating Systems and Computer Networks
Report

Warwick ID number: 1713804

Introduction

The purpose of this report is to provide an extensive analysis of the design, implementation and testing for a basic intrusion detection system that could deal with a high packet throughput. The systems firstly analyses the packets that come through the network and it provides warnings when any malicious activity is detected: an ARP reply, an Xmas Tree Scan or a request from a blacklisted URL. The multi-threading strategy used is implemented in C and based on the skeleton file provided in the specifications, composed of three files that work together to manage the network packet flow.

Design

The three files the project is based on are sniff, dispatch and analysis and are modified in order to complete three tasks: parsing packets, analysing their nature and dispatching them using multi-threading. The skeleton program was modified to use pcap's loop (pcap_loop) to deal with packets and to end the program with CTRL+C. The analysis file used in part one of the coursework parses the layers of the OSI model in order to access the Ethernet header (link layer), the IP or ARP headers (network layer - the type depends on the packet) and the TCP header (transport layer) by casting structs to the raw data. The Ethernet header has a fixed size, while the IP and TCP headers may include optional information that causes the size to vary after each packet, therefore these sizes had to be recalculated after each packet. For part 2, these headers are used to test for malicious activities in the network such as a Christmas Tree Packet , an ARP cache Poisoning or a Blacklisted URL. A Christmas Tree Packet is a TCP packet with the FIN, PSH and URG flags all set and can be used to fingerprint the target system, as different systems react differently to such unusual situations. ARP Cache Poisoning is a rare strategy of exploiting sensitive data by session sniffing or denial of service, due to the fact that the ARP protocol doesn't require validation and the Blacklisted URL is a virus trying to connect to a control server and requires an intrusion detection system. After the program stops, it has to print the number of attacks using signal processing and displaying the report. The third part of the assignment incorporates a thread-pool method that deals better with high-traffic scenarios by limiting the number of threads active at any given time. This approach is preferable to the One Thread per X Model that creates a separate thread for each packet that travels through the network. The X Model is problematic in context switching as it might cause the program to freeze or the high number of threads could cause the system to run out of memory for the next thread. In the thread-pool method a fixed number of threads is allocated and every packet that arrives is added to a queue. The components of this queue are dequeued and sent to the analyse function in analysis.c. When the queue is empty, it blocks the threads and waits.

Implementation

When the first packet is dispatched, 8 threads were created as the specifications mentioned there should typically be a maximum of two per processor core (using a quad-core processor). In a node structure, the information about each packet is stored: its header, data, the verbose and a pointer that will point to the next element in the waiting queue where the packets will be stored. The packets are enqueued and added to the tail of the queue and a signal is sent if the queue is nonempty. If there is at least one element in the queue, it stores the packet at the head of the queue in order to be analysed as part of the same thread. The packed at the head of the queue is replaced with the first one after it if such a packet exists. The mutex lock is used to enforce a mutual exclusion control policy: each thread is processed only after acquiring the lock before accessing the corresponding data. To analyse packets simultaneously, the mutex is unlocked. Because the packet at the head of the queue is stored separately, the first packet points to the beginning of the ethernet header and the sizes of each header have to be sequentially added in order to obtain the next header: firstly the ethernet header to obtain the IP header, than the IP header to obtain the TCP header. Everything that can be converted into an ASCII character is displayed. This will be helpful in the eventuality of an HTTP payload, as it helps find the host website. The payload needs to be an unsigned char pointer that will eventually store the sum of the ether header, ip header and tcp header. If the packet on the TCP header is not a TCP/IP packet then it should be processed as an ARP packet. The mutex locks are used inside the functions that count the number of malicious packets detected because this way it avoids the possibility of multiple threads detecting the same type of packet simultaneously. At the end of the program, memory leaks are prevented by freeing any allocated memory and closing the pcap handler. The program generates and prints a final report before it ends.

Testing

In part one, every item of the headers was printed in order to form a better understanding of how each layer operates and the sign_handle() function was printing the number of malicious packages detected for the second part with the source and destination host, exiting with 0. In part 2, each of the functions that were catching malicious packets was tested as they were developed. To ensure that the queue is fully functional, each thread was printing the thread that was processing a given packet. When simulating a large amount of packages coming simultaneously, the thread wouldn't be able to clear the queue in time before the new packets arrive. The Phyton script from the test folder was used to test how packets are analysed by sending a series of malicious packets. To test for memory leaks when the program stops, valgrind was used.

Conclusions

The intrusion detection system sniffs the packages sent into the network that are analysed by a thread-pool model. The model addresses the layers of the OSI Model while a large number of packages are being sent.

References

1.Carstens, T. (2002). *Programming with pcap*. [online] TCPDUMP / LIBPCAP public repository. Available at: http://www.tcpdump.org/pcap.html [Accessed 20 Nov. 2016].

2. The Department of Computer Science website : https://warwick.ac.uk/fac/sci/dcs/teaching/material/cs241/coursework18-19/