Intrusion Detection System

The packet sniffing and analysis for the intrusion detection system is divided into protocols; the physical layer is analysed, then the internet layer, then the transport layer, then the application layer. This method is based upon the TCP/IP model (Microsoft, 2017). While different packets can be analysed concurrently due to a lack of dependence, an individual packet must be analysed in the aforementioned order. A multithreading approach can be used to improve performance of the system and allow multiple packets to be analysed at any time.

The protocols that this system must be able to analyse are Ethernet, Internet Protocol (IP), Address Resolution Protocol (ARP), Transmission Control Protocol (TCP) and Hypertext Transfer Protocol (HTTP).

Packet Sniffing and Intrusion Detection

Ethernet

The analysis of Ethernet must obtain each piece of data in the header as well as determine the Ethernet type, to know whether to deal with an ARP or IP packet.

Once the packet is passed to the Ethernet analysis method, the first 14 bytes can be read to get all data (IEEE, 2012), with the Ethernet type being one of the pieces of data. The Ethernet type can therefore be compared to known values; IP is 800 while ARP is 806 (IANA, 2017a). These values are stored as ETHERTYPE_PPO, ETHERTYPE_IP and ETHERTYPE_ARP in <netinet/if_ether> respectively. When the Ethernet type is established the packet can be passed on to the relevant method.

Address Resolution Protocol

The analysis of ARP must obtain each piece of data in the message and detect whether there is an ARP cache poisoning attack.

According to the product specification, the detection of any ARP response packet should be considered as a cache poisoning attack. Obtaining the data from the packet is straightforward as the packet will have a fixed structure and size (University of California, 1990). With this data, the ARP packet can be checked to see if it is an ARP response packet by confirming that the operation code is 2.

Internet Protocol

The analysis of IP must obtain each piece of data in the message and determine whether the packet is a TCP packet.

Ignoring the payload, the IP header will have a fixed size and structure for obtaining the contained data (Free Software Foundation, 2016). The piece of data required to determine whether the remaining part of the packet uses the TCP protocol will be in the protocol field. There is a standard list of IP protocol numbers (IANA, 2017b) so if 19 is detected in this field, the packet is a TCP packet. In this case, the packet is passed onto the TCP method.

Transmission Control Protocol

The analysis of TCP must obtain each piece of data in the message, determine whether the packet is a HTTP packet and detect whether there is a Christmas tree attack.

Like the previously mentioned protocols, TCP has a fixed size and structure to make obtaining the data simple (University of California, 1993). Unlike the previous protocols, the TCP header does not indicate the protocol being used on the next layer. Thus there is no easy way to detect whether the packet is a HTTP packet. The well-known port number for HTTP is 80 and therefore a majority of packets going through port 80 will be HTTP packets, and most HTTP packets will go through port 80. The sniffer will therefore assume that anything through port 80 is HTTP, but this method will not be perfect.

The product specification defines a Christmas tree attack as a packet with the FIN, PSH and URG flags set in the TCP header. Since all of the header data can easily be read, the method will detect whether these three flags are set, and if so note that there has been a Christmas tree attack.

Hypertext Transfer Protocol

The analysis of HTTP must detect whether a blacklisted URL is being accessed.

Whenever a blacklisted URL is being accessed, the HTTP request will contain “Host: [URL]”. Assuming that the HTTP request can be read, the method must detect whether the previous string is contained in the HTTP request. The problem occurs in determining whether a HTTP packet is a HTTP request, as the previous string may occur in the payload of the packet for other reasons such as it being in the body of the webpage. To overcome this the system can check if it is a HTTP request by checking the first line of the packet for information that occurs in a HTTP request such as the HTTP method and version (Berners-Lee, 1996). This does not guarantee that a packet is a HTTP request but it increases the chances of falsely identifying a request whilst maintaining that the actual accesses are still detected. Therefore if all of this information is found, the packet denotes a blacklisted URL being accessed. If so, that blacklisted URL is being accessed.

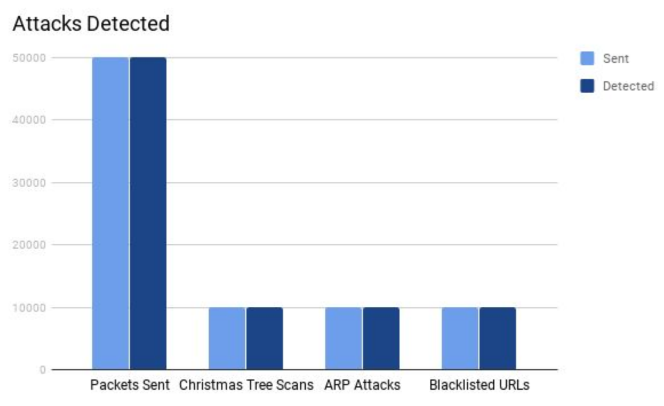
Multithreading

The intrusion detection system must be able to handle a large volume of traffic, hence the need for multithreading. The two main implementation choices are one thread per packet and a thread pool. Creating threads is an expensive process eliminated by the use of a thread pool, and since a system will constantly be receiving packets whilst connecting to a network it seems as though a thread pool is the better choice. One thread per packet may be better when there is little traffic as keeping a thread idle is a waste of CPU resources and inspecting a packet is a short operation, but the system is more likely to deal with many packets coming in quick succession. If the system is expanded to work on a larger network, less programming effort will be required with the thread pool system to ensure efficiency. Therefore this system uses a threadpool for multithreading.

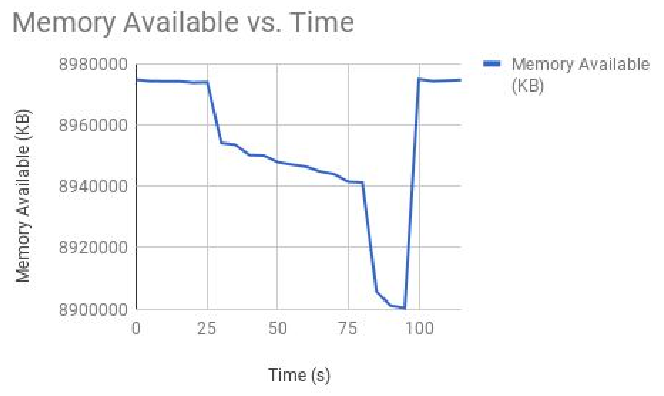
The implementation of the thread pool for this system is kept simple, with eight threads created at the start of the program since the product specification advises one or two threads per processor core. A global queue is maintained for the system which holds the packets to analyse in the order they are received, as well as global variables to count how many times each of the types of attacks occur. Each of these variables, as well as the queue, are mutex locked when being accessed to ensure data integrity when several threads are being accessed.

Testing

The system has been tested using several of the tests given for this project; these tests were combined in a single test script to test how many of each attack the system is able to test. The tests are shown in the all-tests.sh file.



The results show that the system was able to detect 100% of Christmas tree scans, 100% of ARP attacks and 100% of blacklisted URL accesses (Figure 1). Along with this, when packets were sent which did not match any of the criteria for an attack they were not flagged up as an attack by the system. These regular packets included TCP packets with other flags set, ARP response packets and HTTP requests which were not for a blacklisted website.



The system was also tested to see how much memory was used and whether any memory was lost during usage (Figure 2). Packets were first sent at 25 seconds, with all packets sent by 90 seconds. The testing shows that the system was able to free most, if not all, of the runtime memory. Total memory available may fluctuate slightly due to other programs running so it can not be guaranteed that all of the memory was freed, but the amount lost would be minimal.

Overall the tests show that the system is able to perform well based on the required functions given in the specification.

References

Berners-Lee, T et al. 1996. “Hypertext Transfer Protocol -- HTTP/1.0”. Accessed at: http://www.w3.org/Protocols/HTTP/1.0/spec.html#Request-Line (2017-11-17)

Free Software Foundation. 2016. “ip.h source code [netinet/ip.h]”. Accessed at: https://code.woboq.org/gtk/include/netinet/ip.h.html (2017-11-10)

IANA, Internet Assigned Numbers Authority. 2017a. “IEEE 802 Numbers”. Accessed at: http://www.iana.org/assignments/ieee-802-numbers/ieee-802-numbers.xhtml#ieee-802-numbers-1 (2017-11-12)

IANA, Internet Assigned Numbers Authority. 2017b. “Protocol Numbers”. Accessed at: http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml (2017-11-12)

IEEE, Institute of Electrical and Electronics Engineers. 2012. “IEEE Standard for Ethernet 802.3-2012”, p53. Accessed at: http://standards.ieee.org/getieee802/download/802.3-2012_section1.pdf (2017-11-10)

Microsoft. 2017. “TCP/IP Protocol Architecture”. Accessed at: https://technet.microsoft.com/en-gb/library/cc958821.aspx (2017-11-17)

University of California. 1990. “netinet/if_ether.h”. Accesses at: http://unix.superglobalmegacorp.com/Net2/newsrc/netinet/if_ether.h.html (2017-11-08)

University of California. 1993. “tcp.h source code [netinet/tcp.h]”. Accessed at: https://code.woboq.org/gtk/include/netinet/tcp.h.html (2017-11-10)