Executive Summary

In the last 10 years network intrusion detection systems (NIDS) have evolved from an obscure area of network security to a billion dollar industry. Many different techniques are used by such systems, and often, different methodologies will be combined together. As a result of the rate at which the industry has grown and the speed at which implementations have evolved, a paper that discusses the different techniques used by NIDS, and which briefly introduces the reader to designing such systems is needed.

Different techniques used in intrusion detection systems are discussed in enough detail to give the reader an understanding of how NIDS systems function. NIDS need to collect and analyze as much data as possible in order to maximize their effectiveness. A distributed approach that incorporates different packet analyzing techniques is advocated for achieving this goal. The collection and storage of information about each packet the NIDS monitors is promoted as a means for accomplishing this. A design that integrates the different types of NIDS discussed is presented at the end of the paper as a way of tying together the topics covered. The design presented is robust enough so that an implementation of it would be practical in a network of any size with little modification.

1 Introduction

As computer break-ins have become more commonplace, there has been a parallel effort to counter this threat. There have been many techniques and tools employed to achieve this, ranging from administrators adopting hacker methodologies, limiting access to the computer networks (both virtually and physically), and attempts to identify compromised machines and illicit behavior.

Many different types of protection are often used to guard a computer network from attack. For instance, a firewall will be used to block outside traffic that meets a certain criteria; an intrusion detection system may monitor the network for out-of-place traffic; logs from each machine might be centrally collected and analyzed; and the system administrator may scan the network for known vulnerabilities as they are reported to the security community. All of these actions combined with a good network security policy can work toward making a network much more secure. Despite the fact that many attempts have been made to integrate all of these different parts into a package that is easy to manage and maintain, it is still difficult to do so. The detection of network attacks will be the area covered in this paper and the other parts of a network security architecture (ie: a firewall) shall be treated as given. The types of attacks that will be discussed are those that take place over a network, as opposed to the attacks that involve physical access or are host based. A set of tools or a tool used to detect a network intrusion is referred to as a ``Network Intrusion Detection System", or NIDS.

Intrusion detection is misleading terminology since it implies the act of attempting to identify an intrusion once it has taken place, while almost all of the intrusion detection systems try to identify possible intrusion attempts. Detecting intrusions and also detecting attempts at intrusions will be treated as the same since virtually all of the literature on the subject makes that generalization. It is worth noting the difference to avoid later confusion. The following sections will show different models for classifying network traffic in the hope of detecting hacker activity. This can be used as a forewarning device ("hacker X is trying to see if a network contains a known vulnerability") and also as a way to detect compromised machines. A discussion is provided at the end of the paper which ties together the different techniques that are discussed into a framework for a comprehensive NIDS system.

The scope of network intrusion detection which will be examined will be limited to working only with network traffic, and at times the alerts generated by a NIDS. Since most computer attacks originate on another portion of the network other than that to which the victim's machine is connected, possibly a LAN or the internet; it is possible to identify the machine an attacker is using. Host based detection, or locally detecting a machine compromise is beyond the scope of this paper.

Modern NIDS have evolved to the point where they are expected to do many different tasks. In some situations they are expected to not only report on attacks but also take preventive action to stop them. NIDS generally never interfere with packet flow, but passively listen and collect data. Some NIDS do pass information to a firewall so that it

will block future traffic flow between a specific machine or network, but generally they take a purely monitoring approach to prevent themselves from becoming part of a denial of service attack. This paper will not discuss the response by either the administrator or the NIDS system unless it aids in detection. Since the action taken owing to an alert is dependant on some kind of security policy which will change depending on the network in question, it is not something that will be discussed here.

The best way to present this topic is to separate it into different categories based on the characteristics of different techniques. The main logical categories are Signature, Anomaly, Active, Passive, and Distributed based techniques. While each of the methods has traits in common, they all have a set of characteristics that make them distinguishable from each other.

2 NIDS Concepts

There are a number of topics in NIDS that must be given consideration whenever an implementation of a NIDS is being considered. What follows is a discussion of several of these topics.

2.1 A Primer on Network Sniffing

In order to examine network traffic, a technique is needed to gather it. This is accomplished with the use of a network sniffer. A sniffer is a program which passively collects network traffic. There are two ways to configure a sniffer. The first allows all of the traffic flowing to and from the computer on which the sniffer is loaded to be visible. The second method makes the traffic that is passing through the hub or switch to which the computer is connected to be visible. This is called listening in promiscuous mode.

In general, information is sent across the network unencrypted. It is because of this that a packet's content can be examined and the made sense of. Even if the traffic was encrypted, the headers which contain the routing information would still have to be readable and unencrypted. It is surprising how many application protocols still use plaintext.

A sniffer will be used as the backbone for all of the packet collection which will be discussed. It should be also noted that hackers use sniffers to read network traffic for a number of reasons. Currently hackers use sniffers as part of more complex attacks such as connection hijacking, active spoofing, and discovering passwords. There have been many cases where reading the plaintext traffic generated as part of a telnet connection has rewarded attackers with passwords. [32] A sniffer must run with root privileges, so if a sniffer can be detected on a network it may be an indication that a machine has been compromised. Detecting sniffers will be discussed later in this paper.

While it is trivial to write a basic sniffer, its success will vary greatly depending on what type of networking hardware is being used. This is due to the different ways in which switches and hubs handle traffic. A hub will broadcast each packet it receives to all of the machines connected to it, allowing any machine running a sniffer to examine each packet. A switch will only pass traffic to a machine if it is destined for it. This is much faster, and it is generally considered to be much more secure. [1]

On the surface it would seem that having a hub is much less secure than a switch, but in practice this is not the case. There have been numerous methods discovered for monitoring traffic on switched networks but they are harder to implement and can cause network problems. Many switches are designed to make it easy for a system administrator to sniff a segment of a network without having to resort to more complex and often flaky methods. Many switches provide a way to configure a span/monitor port which will copy all of the traffic going through the switch to it. [2]

For a general discussion on NIDS techniques the above is about all of the knowledge about network hardware which is needed. It will be assumed that the machine of interest is in a position to view all of the network traffic flowing through its segment (by listening in promiscuous mode), and more complex techniques are not required to acquire the data.

2.2 NIDS Placement

Placement of NIDS systems vary with the needs and requirements of the network administrator. There are several different strategies that are worth noting.

The most obvious place to install a NIDS monitor is in a place where it can monitor the flow of data entering and leaving the network. This is effective when picking up attacks from the internet aimed at the LAN. Complications arise where there is more than one point of entry into the network. For instance, there may be a modem connected to a computer that an employee uses to connect to the internet from home, or maybe the network may be large enough to be on several different high speed connections (eg: UNB is connected to CA*NET and the general internet through different pipes). Another drawback is that a machine is needed which is powerful enough to collect and analyze all of this traffic, which will likely be very large. Also not all network attacks are directed to or from the internet. It is very likely that a hacker will attack other machines on the local network once he has a foothold, and this traffic will not be picked up by the NIDS.

An interesting approach that the author was once told about was to have a NIDS monitor on either side of a firewall. The administrators would then compare the alerts generated by each monitor to see what malicious traffic had gotten past the firewall. While this is an interesting technique, one monitoring node on the network side of a firewall is useful enough.

In order to monitor as much of the network as possible, an NIDS system can be used on each segment of the network (ideally one for each hub/switch). By adding more than one monitor to the network the NIDS coverage of the network is expanded, while decreasing the total number of points of failure. Despite increasing the complexity of an intrusion detection model, the rewards are ultimately worth it. The number of monitoring nodes is not important in the following general discussion of NIDS techniques until distributed NIDS approaches are discussed.

When a number of NIDS systems are being used, there is sometimes a separate network set up just for them. This is done in order to isolate the NIDS machines from the LAN, helping to secure them from attack. For example, a NIDS machine may have two network interface cards (NIC) one for a private LAN consisting of other NIDS nodes, and the other set in promiscuous mode monitoring the LAN's traffic. [3] Generally the transmit cables from that NIC are cut, so the machine can only receive data from the LAN, and not talk to the machines on it. [2] Despite being more expensive to implement, this setup is common in large networks.

2.3 Logging

There is little point in collecting security alerts unless something is going to be done with them. Once a NIDS generates an alert, it can be sent to a logging console somewhere on the network or stored locally. [4] At a logging console the alerts would be collected, possibly analyzed to reduce false positives and saved for future reference. Logging

consoles are not needed if there is just one NIDS node on a network (it can act both as a monitor and a console), but when there is more than one node, it becomes much more practical for an administrator to be able to look at the alerts generated by all of the nodes from a central location.

The logging of alerts will not be discussed in detail in this paper, but it will be assumed that it is occurring when alerts are created by the NIDS.

2.4 Reconstructing TCP Streams

Due to the nature of TCP/IP, and how it conducts conversations between computers, it becomes important that the NIDS has a way of piecing together streams of packets. This allows us to be able to examine the contents of a conversation more effectively. When a NIDS wishes to monitor the data flowing between two hosts, it must take into account that the data may be spread out over more than one packet. [6] So far the only mention of packets has been to say a NIDS monitors them (how NIDS accomplish this will be discussed later), but it is important to realize that since some packets are part of a stream, the contents of many packets can only be understood when examined together.

This is a problem which most NIDS variants have to deal with at some point or another. The easiest way to overcome this obstacle is to create a system that stores each packet after it has been initially processed, then combines and analyze the contents of them. There are two generic methods that can be used to do this. The first is to have the NIDS

keep each packet in memory and analyze the contents as needed. The second approach is to write each packet out in its entirety (headers and data) to disk, and go through them at a later time.

Having a copy of each packet processed by a NIDS requires a large amount of storage space, but it is feasible. This technique allows the NIDS to check each packet at a later time based on a possibly new set of criteria, and also allows it to replay network traffic. This approach unfortunately does not enable alerts to be generated immediately. A possibility of using each technique exists, but it would use excess system resources without giving much additional benefit.

2.5 Reconstructing Fragmented Packets

A problem that arises with IP traffic is that individual packets may be fragmented into smaller ones. Fragmentation occurs when a packet is being passed from one subnet to another, and the maximum transfer unit (MTU) of the new subnet is smaller than the size of the packet. [30] Each new packet will contain an IP header to direct the traffic to the host, while putting other headers (TCP, UDP, etc) into the first of the fragmented packets. The broken up packet will be reassembled by the destination computer.

Fragmented packets require more overhead for a NIDS to work with since in order for it to inspect the contents the packet must be reassembled. Fragmented packets were at one

point used to bypass NIDS, but currently most NIDS implementations reconstruct the packets internally, and then examine their contents, eliminating this threat. [5]

3 Collecting and Dissecting Network Data

Network traffic flows between machines in chunks of data called packets. Each of these packets contains a number of headers that are used to route the packet from the source host to the destination host. For internet (and all TCP/IP) traffic, the headers of a packet are layered, based on the OSI 7 layer model. A basic understanding of the various networking protocols and how they work together is all that is required to understand the rest of this paper. A detailed examination of network protocols is outside the scope of this paper, and knowledge of it is therefore taken for granted.

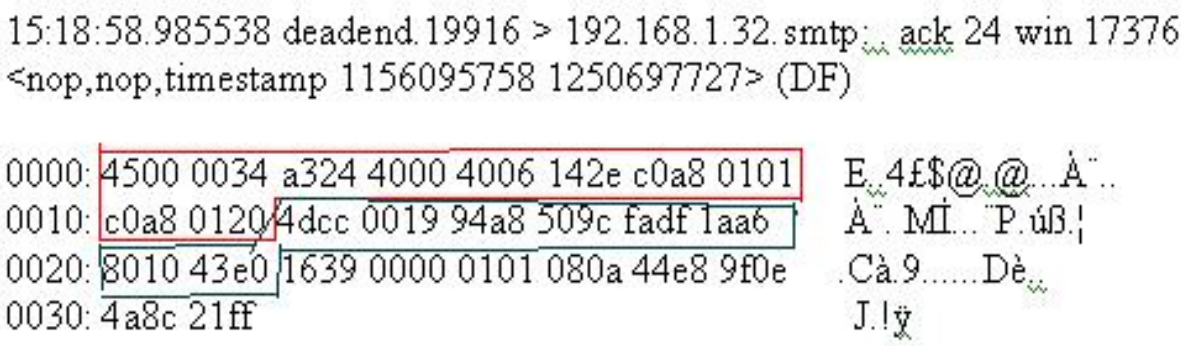
The following section describes the very basics behind traffic analysis, focusing on collecting network traffic in order to make it useful, and also on interpreting this data.

3.1 Introduction to Packet Analysis

A great deal of NIDS work comes from taking a packet and examining its various header fields and contents. This can be done by hand, or by a program. Generally a NIDS will do most of the work in finding interesting packets that could be inspected by hand, based on some criteria which states when alerts should be generated. An example of dissecting

a packet by hand is presented below before discussing areas where programs are expected to do this work in an automated fashion. This information is always useful since packet dumps are often provided alongside many NIDS alerts, and building NIDS signatures often requires being able to read and dissect a packet dump.

Dissecting and analyzing packets by hand is perhaps the oldest method of intrusion detection. Since inspecting every packet by hand is like looking for a needle in a haystack, systems were eventually built to automate the process. Since the ratio of normal to suspect traffic is very large, inspecting traffic by hand was often only done when an administrator thought that a machine or the network was behaving strangely. The approaches and reasons for analyzing the network traffic have always varied. Some administrators may be interested in just knowing what packets are flowing through the network, or they may just be interested in knowing what each packet is attempting to do. In both of these cases different approaches to analyzing the data would be taken. This section provides a brief look at a packet taken from a TCP dump. Methods that can be used to determine if this is a suspicious packet are not developed until later sections.



The output can be split into three individual sections. The top portion of the above log is basic information about the TCP header. The next portion is a dump of the packet in HEX. The output on the right is in emacs-hexl format. The left hand column contains the offset from the start of the packet.

The fields in the top portion comprise of (from left to right), the time the packet was read, the source machine ("deadend" is its DNS name) and source port, destination machine and destination port, flags set, window size, TCP options, and the IP fragment option. In this case, the fragment option set indicates that the packet is not fragmented.[25]

The first 14 bytes of the packet would normally be the Ethernet header, but TCP dump does not print out the link level portion of the packet. The packet shown consists of a 20-byte IP header followed by a 20-byte TCP header. The headers in this case are followed by data. It should be noted that if the option fields of the headers are being used, the size

of the IP and TCP headers can be larger than twenty bytes. If the headers contain additional options, the data size field of the header will indicate this.

Knowing the format of each header, it is possible to convert and interpret the HEX values into data that makes more sense. The Appendix contains the IP and TCP header formats which can be used to figure out what each byte represents in each header. By knowing what the fields are and the offset they are from the start, the above packet can be decoded. [2] There is little need to decode this packet since the information that is most useful is already provided in an easy to read format.

From the above information it can be determined that this is not a fragmented packet since the "DF" flag is set, and it is not trying to establish a TCP connection. Since the response the packet received or what types of packets preceded it are not shown it is impossible to tell if a connection has already been established. The ports used indicate that the packet is being sent to a mail server (on port 25), from a random port (19916). Section 5 introduces a concept of deducing the operating system of a machine by sending a packet (in this case "deadend").

3.2 Introduction to Traffic analysis

Interesting information can be gathered from individual packets, but there is also a lot of benefit in analyzing the conversations which take place between computers. This typically involves looking at what packets are flowing between machines and less on the

data portion or the specific header values of a single packet. There is a class of network attacks which exploit flaws in the IP and TCP protocols. Often these types of attacks can not be detected by looking for a standard feature in one packet, but rather in looking for a combination of packets working together.

A simple example of this is the SYN flood attack. This is a denial of service attack that attempts to fill up the connection queue of the victim machine. The attack consists of sending dozens of TCP packets with the SYN flag set to the target. The victim machine will try to complete the TCP handshake by sending back a SYN/ACK packet. The connection will never be established as the attacker will not send the required ACK packet in response, leaving the victim to keep resending the SYN/ACK packet until it times out. [31]

Since a TCP SYN packet is a valid packet, looking at an individual one will not indicate that an attack is taking place. Detecting this attack is only possible by examining a number of packets. [9]

There are many more network attacks which have the same property and not all of them are denial of service attacks. Connection hijacking, tunneling traffic, and port scanning are a few examples.

3.3 A Framework to Build on

Almost all of the techniques that will be discussed later on require knowledge of the packets which have been sniffed from the network. It is possible to write a program for each technique that would sniff, organize, and sort the data as needed but it is more resource efficient and easier to maintain if only one program was doing this. A basic packet sniffing framework that can be built upon is presented below.

Logging packet headers to the local hard drive is a good way to create an audit trail. An audit trail enables information about specific network connections to be requested at a later time allowing someone who is investigating an intrusion to see what network activity was occurring. It is possible to log the packet header and also the contents of the packet, but owing to hard drive space restrictions, it is reasonable to just log the headers. The important headers to log would typically be network level ones such as ICMP, IP, UDP, and TCP rather than the individual application headers. Each packet header should be recorded and have a time stamp to donate when it was received.

It would also be useful if the application header was logged, but that would require some ingenuity in order to determine the application being used. This of course could be guessed based on the port number (there are assigned ports for each type of application. ie: port 80 is HTTP traffic, port 21 is ftp, etc), but there is no guarantee that the application listening on a given port is following the assigned port rules. It would be practical to separate the data gathered into different files based on the time in which

groups of packets arrive. For instance, every second a new file could be created which will contain all of the headers which passed through the network in that second. Other data can also be recorded besides the header while keeping the log size relatively small, such as the total size of the packet.

Once a large amount of packet data has been collected, it is possible to graph it based on when the packet went through the network and the amount of data sent at any given time. [7] Graphing network flow can be helpful in spotting DDoS attacks and bandwidth hogs, but other than that there is little use for it. It is something to keep in mind when designing a console and would be a nice addition to a NIDS.

3.3.1 Problems with Dynamic IP Addresses

A major problem presented with logging every packet header is that the IP addresses of the machines on the local network may change. Many networks use systems that dynamically assign IP addresses to the computers on the local network using protocols such as DHCP. A way is needed to keep track of the IP address each computer is using in order to make stored logs of traffic flow useful.

The easiest solution is to assign every computer a static IP address. UNB does this on their medium sized campus network and it seems to work well for them.

A more advanced solution would be to rewrite the DHCP daemon and have it keep track of each computer assigned an address based on its MAC address. The time and the length of each DHCP lease would also have to be recorded. From this point it is possible to make more sense of the data that has been gathered since the IP address of every computer at any time would be known. Needless to say, this is a much more complicated route to take and it could affect parts of a NIDS system that use stored logs.

4 Active Network Monitoring

While gathering and analyzing data using a sniffer is a good way to detect hacker behavior, it is not 100% effective. Many types of attacks can be detected via this method, as will be shown later, but some signs of intrusions are only detectable when specially written programs probe a network for them. Active network monitoring is a good name given to this since network traffic must be generated in order for the tests to work.

In order to keep a NIDS monitoring node secure, often the transfer wires from its NIC will be cut, allowing the station to only read packets from the LAN. Obviously this renders active monitoring useless from such machines. However, this type of monitoring can still take place from a different machine on the network. Even if these techniques are not automated to run regularly, they can be used whenever an administrator is concerned about a possible break-in. Combining active monitoring with other techniques of intrusion detection is something which should be thought-out on a case by case basis.

4.1 Sniffer Detection

So far using sniffers to build a NIDS and briefly looking at why a hacker may use a sniffer have been discussed. Since many hackers use sniffers, it is important that an administrator has a way to detect this behavior. The converse of this is that hackers can use the same techniques to detect NIDS systems. If the transfer wires on the network cable from the monitoring node are cut, then none of the methods described below will detect the machine doing the sniffing.

The following tests can be classified based on the flaws they exploit. The two generic flaws are the way a sniffer is written and the way the operating system deals with packets while in promiscuous mode.

4.1.1 Latency Test

When a machine is in promiscuous mode it will be reading a lot of information from the network. On a busy network this will slow down the machine a great deal. It also raises an avenue for detecting a sniffer. This technique is rather straightforward.

The first step is to measure the time it takes to ping the machine that is being tested. Next, ping the machine, and measure how long it takes to receive a reply. Finally create a large amount of network traffic, and then once again ping the machine and measure the response time. Taking this information it is possible to run a number of statistical tests to

determine if the machine is being slowed down as a result of processing packets by a network sniffer. [1] [8]

4.1.2 DNS Lookup Test

As a sniffer collects information from the network it will often resolve the IP addresses into DNS names. This is generally done because the end user of the program will find it easier to visually scan a list against a list of domain names rather than IP addresses. This flaw can be exploited to detect a sniffer by generating packets with a false source address, and seeing if any computer does a DNS lookup on it. By sniffing the DNS traffic it is possible to monitor the DNS lookups to see if this address is being resolved. If a machine does a lookup on our bogus address it is likely that this machine is sniffing the network. [8]

4.1.3 Generating False Data

Hackers often use network sniffers to gather passwords so they can be used to break into more machines. Many computer programs send passwords in plaintext (telnet, the “r” services, pop, imap and ftp to name a few), and hackers in the past have used sniffers to collect lists of passwords that are being sent across the network. This technique has been very valuable to hackers in the past because it has enabled them to easily spread into other computers.

For example, consider network traffic generated that appears to be that of a telnet or ftp connection. If false data such as usernames and passwords for a valid computer system are included, and that username/password combination is tried on the machine at a later time, it is a good indication that a hacker is running a sniffer on the local network. This technique is rather flaky, as there is no guarantee that a hacker will want to break into that specific machine even if he/she has what appears to be a valid password for it. It also assumes that the attacker would want to use a sniffer to look for passwords and not some other piece of information or as part of another attack. If a login using this false information was detected, it is likely that an attacker has compromised a machine on the LAN where the test was performed but there is no way of knowing what machine was compromised. Still, it is a valid test that would be useful in some settings.

4.1.4 The ARP Test

Not all of the known tests are machine independent; some take advantage of the ways in which the operating system treats packets when it is in promiscuous mode. The tests in this section and the next fit into that category.

The ARP test only works for detecting sniffers which are running under the Windows OS (it is known to work against the 95, 98, and NT versions, and possibly also against later releases). The bug this technique takes advantage of, is that when the NIC is in promiscuous mode, only the first octet of a MAC address is checked to see if it is a broadcast address or not. A broadcast MAC address would look like “ff:ff:ff:ff:ff:ff”,

and is a valid part of the ARP. An invalid broadcast MAC address could look like "ff:00:00:00:00:00". An ARP packet with this MAC address and the correct destination IP to the host that is being tested is sent from a machine on the local network. If the host responds to this broken broadcast packet, then it is likely running a sniffer of some kind, since this packet would only be inspected in promiscuous mode. [8] [1]

Alternatively an ARP request with a false destination MAC address but which is otherwise valid could be sent to the machine. If the machine is not in promiscuous mode it will not see the packet, and will not respond to it. If it is, then it may respond to the packet as though it were normal, but since different operating systems each handle these kind of packets differently there is no guarantee that this test will be successful.

4.1.5 Ethernet Ping Test

This next method is known to work on several Linux, BSD, and Windows machines. If a "ping" packet is constructed with the correct IP and ICMP headers, but with a false destination MAC address, the packet should be processed by the machine being tested. On many systems running in promiscuous mode, the packet is still read, processed, and replied to since each packet is being read from the NIC, even the ones with incorrect MAC addresses. If a reply is received from the custom built ping packet, then that host is sniffing traffic in promiscuous mode. [8]

This is very similar to one of the ARP tests mentioned above. A similar flaw is being exploited (a machine responding to a packet which it should not have been able to see), just in a different way. There are likely other variants of this method, but two examples are enough to illustrate the technique.

4.1.6 A Final Note

Many specially written sniffers can beat the detection techniques listed above. Some of the techniques mentioned are also likely to generate false positives. As a result sniffer detection should not be used by itself as the only line of defense against sniffers, let alone hackers. They can play an important role in defending a network but the effectiveness of sniffer detection is greatest when used alongside other NIDS techniques.

It is not possible to spoof MAC addresses unless the spoofing machine is on the same network segment as the machine that is being tested. As a result of this, some of the following tests will not work when tried across a network.

4.2 Probing for New Opened Ports, New Machines, and Changes in Operating Systems

Many networks contain some kind of security policy for the machines running on it. For instance, there may be rules against running ftp or web servers from an office computer. It is also not uncommon for a successful attacker to load a network daemon as part of a backdoor, allowing him/her to return to the machine at a later time with root access. For

these reasons, it becomes of interest to know when new daemons appear on the network. This can be accomplished by port scanning each machine that is being monitored to acquire a list of its open ports at two different points in time, and comparing the results. Needless to say, scanning a large network can be very slow and ports that are in use for just a short time would not likely be picked up by this.

If a workstation is scanned and it appears that it may be running a web server, it might be because the machine's owner is indeed running a web server (with permission or not). It is also possible that it could be a backdoor conveniently placed on a port which many firewalls will allow into their network. Once an administrator knows that the port is open he can investigate further and take additional action as required.

As a side result of scanning the address range of a network, an administrator can keep track of new computers coming online, what OS they are running, and which ports they have open. This allows the administrator to have a better handle of what the current state of the network is at any given time.

While port scanning each machine, it would also be easy to detect the OS of each computer. This can be accomplished by either doing passive network detection (see next section for more on this), or by applying a number of tests to fingerprint the TCP/IP stack of each machine. [10] A technique is later described where both the OS and its list of open ports can be further used by other portions of a NIDS system to minimize the number of false positives. In order to accomplish this, the results must be archived in a

database at the end of each scan (the important information would be the time of scan, IP of each machine, along with its OS and a list of its opened ports).

If a new lpd (Linux printer daemon) exploit is announced on a mailing list, it would only take a minute for an administrator to queue the results of a previous scan for a list of every Linux system running the printer daemon. From this point appropriate action can be taken to see if the machine is vulnerable and update the software if it is.

5 An Introduction to Passive Network Monitoring

In the previous section several techniques were detailed which probed a network to try and find signs of a compromised machine. By passively monitoring a network, it is possible to achieve similar results to scanning each machine. Another reason to passively monitor a network is because it is possible to accomplish results that are not possible with active monitoring.

Passive network monitoring can take many different forms, depending on the techniques used by a NIDS to shift through the collected data. This chapter focuses on using the information taken from packet headers and on knowledge of the frequency of packets in order to generate alerts. Sections 6 and 7 will deal more with specific fingerprints of malicious packets and “out of the ordinary” behavior. Needless to say, these techniques

will complicate things a fair bit, so it is best to start a discussion on passive monitoring with one of the earliest and simplest techniques, which is the use of thresholds.

5.1 Introduction to Thresholds

A threshold is a limit to the number of events which will trigger an alert of some kind when it is reached. For instance, if for every tenth cookie somebody buys they get a free can of Pepsi, then the threshold limit for the event is ten occurrences. This concept can be developed into a means for detecting certain types of denial of service attacks, port scans and at times more obscure security events.

In order to detect a network attack of any kind, the attack must first be defined. For instance, let's define a SYN flood attack as a burst of SYN packets directed to a host which does not result in a TCP connection being established. The result is that the connection queue on the victim machine will fill, and will no longer accept more connections. How would one detect this attack?

A TCP connection is a three part process; an initial SYN packet is sent by the machine initiating the connection, and followed by a SYN/ACK response sent from the destination machine. Finally an ACK packet is sent in response to the SYN/ACK packet, and at this point a TCP connection is established. Knowing this, an administrator can say that if a certain number of SYN packets without a follow up ACK packet are sent to a machine within a certain time window, then they may be experiencing a SYN flood. Once a SYN

flood or another attack has been detected, an alert would be created and the appropriate response taken once the administrator has received it

The above example illustrates the typical approach to thresholds. The number of occurrences of an event is counted, and once it reaches a defined limit, an alarm is sounded.

This framework can be used to detect any attack which has a recurring pattern in it. For example, an administrator can watch for an excessively large number of short telnet or SSH connections within a small time frame, since this could be the result of someone trying to guess a password (generally telnet and SSH close a connection after 3 bad login attempts). Another attack currently in use is the ping flood, where one or more computers repeatedly send ICMP echo (ping traffic) to a host. The purpose of this is to take up the victim's bandwidth and slow the computer down to a crawl. The detection of this attack is straightforward. All a program would have to do is count the number of ping packets destined to a host at any given time, and see if the threshold limit has been exceeded. [9] False positives, or rather, alerts generated in error as a result of normal traffic may appear if the threshold limit is set too low. It is best to experiment and try different threshold limits before deploying these systems.

5.2 Detecting Network and Port Scans

Detecting port and network scans (a repeated probe which occurs across a network), is a bit trickier than detecting DoS attacks. Generally speaking the DoS attacks which are easily detected using thresholds are cases where the network is being flooded with some kind of traffic. If the traffic sent was spread out over a long period of time the thresholds would not be crossed, but the DoS attacks would not take place. [10] Unlike DoS attacks network and port scans do not have to occur within a certain timeframe to be effective.

With port/network scans the attacker has the opportunity to proceed slowly and can be much stealthier in the process. The obvious down side for the attacker is that the scan would take much longer to complete, but the reduced possibility of detection is often worth the wait.

The framework discussed in section 3 for creating an audit trail of packets can also serve as a useful means for detecting drawn out scans. By maintaining a list of every packet that passes through a point on a network, it is reasonable to think that this could be used to determine which machines have been scanning the network. At first, the task appears to be something akin to looking for a needle in a haystack, but it is possible to narrow down the data to improve the results.

One way to accomplish this is by using a list of the listening ports and online machines. Generating a list like this is discussed in section 4.2. Based on this it can be determined

which packets are being sent to a machine that is offline or a port which is not open. A scan is likely to touch both open and closed ports and machines which are online and offline machines since the scanning program obviously does not know the current state of the network. For each packet sent to a closed port, a program could take the source address from the logs of packets that have been collected and look at all of the traffic which this machine is sending to the network. By narrowing down the search of this traffic to this information, it is possible to tell if a machine is suspect or not based on the number of machines that it tries to contact, and the frequency with which it tries closed (and different) ports. A lot of networks have parts of their address range that are not being used, and most machines on a network do not have the same set of open ports, so this method would likely be effective.

It should be noted that the above method fails to spot outgoing scans. Also, distributed inbound scans still have the potential to slip between the cracks. With many different machines probing a small network, it is likely that not all of them will try to access closed ports, and the ones that do may probe too few to trigger a possible threshold on a number of closed port access attempts. Grouping together a lot of small violations that occur in a short time span may be a way of getting around this.

5.3 Passive Operating System Detection

While the protocols which make the internet able to function are standardized, there are certain small discrepancies between the TCP/IP stacks on different operating systems.

These oddities do not cause problems regarding network performance but they can be used to tell which operating systems are talking to each other. By examining the TCP header it is also possible to venture a guess as to the type of OS of the source machine. It is likely that there are other protocols which have similar flaws in them, but since TCP is the most widely used this is the place to start.

Two portions of TCP packets that are dependant on the operating system generating them are the time to live (TTL), and the window size. These values are not required to be set to a predetermined value by the operating system. As a result, different operating systems will set them to different values whenever a packet is sent. For example, Windows 2000 will initially set the TTL to be 128, and Linux will generally set it to 64. [11] The window size of a packet sent from Linux is known to be set to 0x7D78, while the MS Windows size is constantly changing. [12] This being said, it is possible to change these values in many operating systems, and the values listed above are not the same for all versions of an OS. [13] There are also many more portions of the packet which can be analyzed and compared for OS information.

Being able to tell the OS of a machine based on network traffic can find many uses in NIDS systems. Any packet which raises a flag can be passively fingerprinted, letting the network administrator know the OS of the suspicious machine in question. There are both practical and novelty uses in this knowledge. For example, a very general rule of thumb is that penetration attempts coming from a UNIX based machine denote a little more sophistication and should be taken more seriously than those from a computer

running Windows. Also, if a worm targeting Microsoft products is making the rounds on the internet, and then a penetration attempt is seen using the same exploit that the worm is using, coming from a non MS based machine, it can be assumed that this is not worm traffic and should be taken more seriously.

Knowing the operating system of a destination machine in a NIDS alert can also be useful. If an IIS (the Microsoft web server) exploit attempt was detected but the ‘’victim’’ machine was running the OpenBSD operating system, one can assume that this alert is not very serious as the attack will not have succeeded.

5.4 Detecting New Opened Ports

Many remote exploits do not automatically drop a hacker into a shell. Instead they often enable the hacker to gain access to the system by binding a shell to a port. [14] If the exploit is successful, the hacker would connect to the known port and have a certain degree of control over the system. The listening port would generally only be open for, at most, a minute, and would only accept a connection once.

Using the method discussed in section 4.2, a newly opened port that is only available for a short time will not be discovered since a port scan would leave gaps of time in-between scanning each machine. Opened ports that were recently closed can be a sign of an intrusion, so it would be useful if a NIDS could detect them all.

One of the few cases where a client program will bind to a somewhat random port (>1024) and accept connections is during an ftp session. Even in this case the ftp client can be run in a mode which prevents this (passive mode), (many firewalls also prevent connections like this from happening). [26] Dealing with firewalls is beyond the scope of this paper, so this case will be ignored. The fact that the ftp protocol will generate this kind of traffic could be a headache for the administrator since a lot of false positives would be generated every time this happens.

In order to differentiate between connections to recently opened ports and connections to ports which are always open, a list of ports which are left open for each machine would be needed. This is discussed in section 4.2. The next step would be to run a sniffer which will generate an alert once a 3 way handshake is completed with a destination host/port combination which indicates that the port should not be open. This does not taken into account UDP traffic, since the UDP protocol is connectionless.

Instead of writing another sniffer program, it would be reasonable to use the framework discussed in section 3. This basic framework makes it is possible to notice UDP traffic flowing between machines and then check to see if this should be happening. Also, using the framework from section 3, the ftp problem discussed above will not have much of an affect. For each connection to a port numbered above 1024 on machine A from a machine B, before generating an alert the system would check to see if machine A has started any connections to machine B on port 21 (port 21 indicates ftp traffic, and if machine A started the connection, then the program can be confident that it is the client machine). If

this is the case, the NIDS can disregard the traffic as part of an ftp session and not generate an alert for it or generate one but include this information.

6 Signature Based Detection

As previously discussed using passive monitoring to look for patterns in network traffic can detect undesirable behavior. Signature based NIDS is based on a very similar concept. Instead of looking for patterns in the amount and types of packets which are passing through its sensors, it is concerned with the makeup of the packet itself. Each signature gives the NIDS system a bit of knowledge about what traffic should generate alerts.

This method has become a very well known technique for intrusion detection. The popular open source NIDS, Snort has over 1900 signatures and its commercial equivalent, The Dragon NIDS has over 1700. [15]

6.1 Introduction to Signatures

Once a remote exploit has been released into the security community, the packets it generates will be analyzed, and a signature will be constructed based on it. The makeup of a signature is dependant on the NIDS used. A Snort signature will not work on Dragon, but it is generally very easy to port them to and from different systems. [16]

A generic signature will be composed of a list of packet characteristics such as the content of certain header fields, its length, the protocol being used, or a string to look for in the data portion of the packet. A good signature system will allow the writer to specify many different characteristics of a packet to narrow down the possibility of a false positive. For example a NIDS can be told to be interested in all TCP SYN/ACK packets on port 79 with the DF flag set, which contain the string "BAD" in them. This would generate less false positives than a signature which generated an alert for every TCP packet sent to port 79.

As one might imagine, it is very easy for a lot of false positives to be generated if the signature is not well written. Also, it is possible that signature for a dangerous exploit may appear commonly in normal traffic. Despite this it is still a very effective way for detecting known exploits. In order for an NIDS to be as effective as possible, it is best to use this technique alongside others in order to insure that every possible security event is detected.

A benefit for building a basic pattern matching NIDS is that it is rather easy to do. Using common UNIX tools, a basic system can be rigged together using a scripting language like Perl or even shell script. One of the earlier intrusion systems named "Shadow", built by the US government, uses the tcpdump sniffer logs as its backbone and various scripts to add function ability. While shadow does not do signature based detection on the data

portion of a packet it is the great example of putting a NIDS system together using common UNIX utilities and Perl scripts. [17]

The UNIX utility “ngrep” has all of the features needed to create a very basic signature based NIDS. The tool allows you to filter out traffic based on pcap filtering rules (pcap is used by and was developed for tcpdump, as well as finding a home in many NIDS packages, such as snort), and then search for text in each of the packets it picks up. [18] The main difference between this and a regular NIDS system is that a regular NIDS is generally more complex and efficient. Most signature NIDS packages contain features enabling it to piece together fragmented packets, effectively work with streams of packets, and are often optimized to handle the job on a large scale (dealing with hundreds of signatures etc.) which is not practical to implement using a program like ngrep.

6.2 Dissecting a Snort Signature

The most widespread NIDS in use is currently ‘’Snort’’, which is an open source project. Like most NIDS, snort signatures can range from being very simple to being rather complex. Some systems such as Network Flight Recorder use a scripting language as their way of defining a signature, but snort signatures are based around a template format. [19] This means that every signature must have a certain form, but fields in it can be changed and in some cases deleted. One of the advantages to this approach is that creating new signatures is very easy to do, and does not require a great deal of skill.

The easiest way to examine how a snort signature works is by taking one of them apart. Snort signature #301 is for a buffer overflow in the Linux Printer Daemon (LPD is presented below.

alert tcp $EXTERNAL_NET any -> $HOME_NET 515 (msg:"EXPLOIT LPRng overflow"; flow:to_server,established; content: "|43 07 89 5B 08 8D 4B 08 89 43 0C B0 0B CD 80 31 C0 FE C0 CD 80 E8 94 FF FF FF 2F 62 69 6E 2F 73 68 0A|"; reference:cve,CVE-2000-0917; reference:bugtraq,1712; classtype:attempted-admin; sid:301)
[20]

This signature can be split into two parts, one based on content and one based on traffic flow. Below is the traffic related portion of the packet.

alert tcp $EXTERNAL_NET any -> $HOME_NET 515

The format of this is rather intuitive. It tells the NIDS that it is looking for TCP traffic from any external IP range on any port, destined for the local network on port 515. If a packet matching this definition is detected, and this was the only part of the signature, an alert would be generated. The “$EXTERNAL_NET” and “$HOME_NET” parts refer to defined variables which indicate the external and internal IP ranges. A basic signature containing this information alone would not be interested in packets content. Obviously a signature this broad would generate many false positives.

The content portion is much more complicated. There are a lot of optional fields, and some fields have nothing to do with the detection of the packet, but are instead intended to be useful for the administrator once an alert has been generated. Fields are separated from one another by a ";" and a ":" is used to specify an argument to it. The entire content portion is enclosed in circular ("(" and ")") brackets. The fields used would generally depend on the exploit which it is designed to detect.

The complexity of snort signatures has increased as more fields have been introduced with newer releases of the NIDS. A list of the fields from version 1.2.1 is presented below, which provides a basis of what one can expect in a generic signature based NIDS system. Snort has advanced a lot since version 1.2.1 (the current release is 2.1.0), and the signature for the LPR exploit was taken from a more recent version, so there are many fields in it which are not presented below. [19]

1. content: Search the packet payload for the a specified pattern.
2. flags: Test the TCP flags for specified settings.
3. ttl: Check the IP header's time-to-live (TTL) field.
4. itype: Match on the ICMP type field.
5. icode: Match on the ICMP code field.
6. minfrag: Set the threshold value for IP fragment size.
7. id: Test the IP header for the specified value.
8. ack: Look for a specific TCP header acknowledgement number.

9. seq: Log for a specific TCP header sequence number.

10. logto: Log packets matching the rule to the specified filename.

11. dsize: Match on the size of the packet payload.

12. offset: Modifier for the content option, sets the offset into then packet payload to begin the content search.

13. depth: Modifier for the content option, sets the number of bytes from the start position to search through.

14. msg: Sets the message to be sent when a packet generates an event.

[19]

The content portion of the LPR signature is presented below.

(msg:"EXPLOIT LPRng overflow"; flow:to_server,established; content: "|43 07 89 5B 08 8D 4B 08 89 43 0C B0 0B CD 80 31 C0 FE C0 CD 80 E8 94 FF FF FF 2F 62 69 6E 2F 73 68 0A|"; reference:cve,CVE-2000-0917; reference:bugtraq,1712; classtype:attempted-admin; sid:301)

Some of the fields used were available in the 1.2.1 release such as “msg”, “content” and are described above and their meaning has not changed. The reference field relates this signature to two different databases, bugtraq[1] and CVE[2], where the exploit is described in detail. “Classtype” defines what the exploit is designed to accomplish (in this case it is

[1] http://securityfocus.com/bid/bugtraqid/
[2] http://icat.nist.gov/icat.cfm

attempting to gain administrative privileges). The "sid" is the Signature ID, and is used to keep track of the different signatures.

6.3 Pitfalls

For the most part it is very easy to construct a signature based NIDS, but it is difficult getting the process right. In order to be as effective as possible, the NIDS system must be able to reconstruct fragmented packets. [5] This can be very tricky and a memory intensive process. Another downfall is that by splitting a packet in two, a hacker might be able to have the NIDS not notice the signature. For instance, if the system is looking for the string "QWERTYASDFG", all the hacker would have to do is split the packet into two, with the first one ending with QWERTY, and the second beginning in "ASDFG". [6] In order to defend against this the NIDS must be able to work with streams of packets. Both of these topics were briefly discussed in section 2.

Since the patterns which are being searched, are generally well known (there are many sites on the net where this is available[3]) an attacker will have an idea of what traffic will trigger an alert. Based on this, it is possible for an attacker to create thousands of false alarms by sending data specifically meant to trigger the NIDS. There are a number of programs which are designed to do this. [21] Usually these programs are intended to be used by NIDS administrators when testing their systems, but hackers have been known to abuse these tools. By creating hundreds of false positives the attack may be able to

overwhelm the console system in order to have the warnings generated from the real attack be lost in the noise.

The detection of attacks using signatures is based on information about previous known attacks, so it is fair to say that most NIDS systems will not detect new attacks. It is possible that a new attack will trigger an old signature but this is a rare occurrence, and it takes a lot of skill and training to realize that the alert being shown is being generated for an odd reason.

Another major problem is that it is possible to create attack tools for the same bug which create very different traffic. For instance, the Unicode bug in IIS had many different variants since IIS enables the exploit payload to be encoded in many different ways. An interesting avenue is to mutate the exploit payload every time the exploit program is run. This can be accomplished by adding useless chunks of data to the exploit. ADMmutate is a tool which does just that. [22]

7 Anomaly Based Detection

All of the techniques that have been discussed so far require some prior knowledge of what an attack will look like. It has been shown that patterns in traffic and also patterns occurring in individual packets can be an indication that an attack is taking place.

[3] http://www.whitehats.com/ids

Anomaly based detection is a more sophisticated approach that is designed to detect unknown attacks before they have been identified.

The distinctive property of most anomaly detection systems is that it defines what is normal (as opposed to what is abnormal), and uses that to distinguish traffic. [27] One thing to keep in mind is that while previously discussed NIDS designs were given information about what traffic to look for, anomaly detection systems only know what will not generate an alert. For instance, there are signatures for traffic which should not be seen, such as detecting a TCP packet with all of its flags set. Anomaly detection should also detect this attack by noticing that a packet with those characteristics does not appear in “normal” traffic, and the packet is therefore abnormal.

Many of the events which an anomaly system will detect can be detected using other methods. The possibility that unknown exploits and attacks can be detected makes it valuable. [28]

There is a lot of research taking place in this area of intrusion detection at both the network and the host level. What follows is a short introduction to the anomaly detection on networks.

7.1 Defining Normal

Defining normal traffic can be broken down into two categories. The first is to keep track of traffic which is often seen and declare this as being normal. This requires keeping a record of the network traffic and then using data mining and statistical regression techniques on it. This will be discussed in the next section.

The second way is to define what an acceptable use of a protocol is, and anything that is not in that definition is abnormal. This complicates the system since the programmer would need to build into the NIDS a working knowledge of each protocol. Generally only application protocols such as “ftp” or “sendmail” are modeled but it is possible to model network protocols as well.

Each approach is useful, but neither is easy to implement or guaranteed to work all of the time. For instance, there might be a feature in a daemon which can be used by a hacker. An example of this is the “../” bug that in some web servers allows people to run unauthorized commands. Since this is valid web traffic, it would not generally trigger an alert.

7.2 The Data Mining and Statistic Approach

Data mining has become commonplace within large companies who wish to find trends in their customer’s shopping habits. Once these trends have been found it is often

possible for the company to use this knowledge to sell products to their customers more effectively. Needless to say that data mining is a very complex area and is beyond the scope of this paper. Regardless, a very simple introduction to how it can be used for NIDS purposes is presented.

Despite being rather complex most network traffic is predictable and uninteresting. If a large amount of network traffic is collected a program can be designed to find common trends in it. This is possible because most traffic which happens once will happen again at a later time. For example, requests for web pages from a HTTP server will likely occur frequently thus creating a trend in the data. The trend arising from this data may be that the number of web pages requested from a site may not generally exceed 20 or the same directories are continuously accessed. Once a large amount of data has been collected it is possible to have a program spot odd events which do not often occur such as a user requesting the same web page a few hundred times in a short span of time.

Once the NIDS has noticed common trends in the traffic it would be able to classify them as being “normal”. The trends can be classified based on many things. Valid criteria may range from common things such as length of connection, amount of data transferred, and header options, or it might be much more complex.

Once a base model of what is normal has been built the NIDS can examine all new traffic and see if it matches the model. If it does not, it would generate an alert.

One of the most interesting features of this kind of system is how it continues to improve itself as it collects more data. Once the administrator has been alerted to suspicious traffic, often he/she will have the choice of telling the system what to do if it sees the traffic again. For instance if the traffic was harmless the system would be told not to generate alerts if it is seen again. The reverse is also true, so if hacker activity of some kind became a regular occurrence on the network, the system could be tuned to not accept it as being normal.

A serious difficulty anomaly systems face is finding a way to generate enough traffic needed for it to find trends that are free from malicious activity. These systems initially require a lot of time until the system has had time to adapt to the network.

7.3 Modeling Application Protocols

The main applications on the internet use standardized protocols. The specifications of each protocol are available online to anyone who is interested. While it is true that some applications break the protocol specification that they are supposed to use, it can be assumed that a common criterion is being used in almost all cases. [29]

Once a programmer has knowledge of the design of an application protocol, he/she can determine what constitutes valid traffic for each part of it. For instance, when user names or passwords are being transmitted, there is a set definition for the length and which characters can be used in each. By looking at the length and makeup of each request an

NIDS can determine if it is valid traffic. A more relevant example would be to examine the commands which common network daemons are receiving. If command arguments do not pass a basic inspection based on the NIDS interpretation of the protocol, an alert can be generated. The length, character set used, sequence of commands, and volume of commands could be four basic starting points for generating such models.
This process is referred to as modeling a protocol. By modeling different protocols and then inspecting recently received packets to see if they fit into the protocol's model, an NIDS can be build which is able to detect abnormal traffic.

8 Distributed NIDS

Distributed intrusion detection allows alerts and data generated from many different NIDS nodes to be used together. This has the effect of giving an administrator more insight into what is occurring on the network he/she manages, and can be quite useful, especially if a large network is being monitored. A NIDS node will only be able to analyze traffic which is passing through one point (eg: a gateway) of the network. Using a distributed approach a network administrator will use many different nodes in order to monitor different portions of a network. If there is just one monitoring node listening to traffic leaving and entering through a gateway machine, attacks which are occurring between machines on the LAN will not be detected. Not only can distributed nodes be useful for monitoring a LAN, but they can also be used to find attack trends occurring across a wide area network.

The generic distributed approach provides different monitoring nodes logging the different alerts they generate to a central location. This is a reasonable approach for most networks, as it does increase the NIDS's coverage. The concept can be expanded creating applications that better detect events that are occurring on a large scale. The general approach for this is to analyze the alerts gathered on the central logging console, or querying all nodes for information that they may have stored locally.

8.1 Correlating Alerts

If all nodes detect and log attacks to a central server, it is likely that there will be times when some attacks occur more frequently then others. This behavior is generally not very important, as exploits that are new or have a high success rate will be frequently tried most often by hackers. Regardless, there will be times when large spikes of one type of alert may indicate worm activity. This is obviously not always the case, but it is a reasonable assumption to make. It is best to detect worm activity as quickly as possible so the worm's payload can be dissected and understood, and also to find ways to prevent it from spreading. This approach can be taken not just on one network, but across the internet as a whole.

The DShield project is a good example of an alert gathering system being able to pick up on worm activity across the internet.[4] The project collects alerts generated by firewalls (some personal firewalls use NIDS features to block traffic) to determine what attacks are

common and which machines are being extensively used by attackers. Once enough information has been gathered to determine which machines are attacking a large number of people, the maintainers of the site try and have the computer shut down by its access provider.

8.2 Nodes Working Together

Besides generating alerts section 3.2 detailed how individual nodes may collect a great deal of information about the network traffic that it is monitoring. While traffic that is part of an attack will often generate an alert, it is reasonable to assume that there will be traffic related to an attack that is not deemed important enough by itself to raise a red flag. With many different nodes logging the network connections which they are monitoring, the NIDS has at its disposal a pool of information which can be used to tell an administrator more about specific incidents.

To illustrate this idea consider this example. Lets say that an administrator is in charge of managing the security of a mid size company with 3 offices in different parts of a country. Each office consists of a number of people and more importantly, a number of work stations and servers. Each network is monitored by a NIDS system at the gateway with the alerts generated being sent to a central logging server somewhere on the WAN. Each NIDS is also logging information about every packet leaving and entering each office and storing this information locally.

[4] http://www.dshield.org/

If a serious alert such as a buffer overflow attack is generated on a web server hosted at one of the offices, there are a number of approaches that can be taken. After checking to see if the machine was vulnerable to the attack, the administrator could then look at the alert logs and see if the attack is coming from someone who has attacked the network before. If the system was not vulnerable, and this is the only alert generated from that address, it is likely that this is not a serious matter and no more action would need to be taken.

Since each of the monitoring nodes is recording the packet header of each packet that it inspects, the administrators can continue their investigation in order to better determine if this is just a random attack or if this is part of something larger. While the methodology for this would vary, the basic approach is to queue each of the monitoring nodes for a list of activity from the offending IP. Once this data has been put in a central location, a program can then sort through this information and determine more about what the attacker is interested in, such as what network services have been accessed. This data can be broken down by some criteria such as protocol, frequency and volume of traffic in order to get a feel for what the hacker is doing and the level of sophistication that he/she is showing. The process so far can be automated, but it would be best if a real person were to analyze the gathered data since if it is organized well the task can be done quickly and a person is less likely to miss something important.

If the gathered data showed us a pattern of activity that is out of the ordinary, such as probes of different parts of each office network, the administrator can assume that the

network is being targeted, probed, and attacked in a sophisticated manner. For example, imagine that there is a web server on each of the office networks described above and all of them were extensively probed over a long time period. Later the same hacker (or someone from the same machine) later tried an exploit against one of them. This would indicate that the attack was planned out prior to the exploit attempt taking place, and that the hacker was not trying to exploit random machines or networks. These are the kinds of attacks that worry administrators the most.

8.3 Final Notes

It is worth quickly mentioning a couple of things about distributed NIDS which have so far been left out.

The distributed approach unfortunately complicates the work of the network administrators since they must maintain every monitoring node that is added to the network. This means patching and updating the signatures on them and keeping the software compatible between the nodes. The volume of logs stored locally that each machine will generate must also be managed to ensure that each node will not run out of disk space. Old data should not be deleted as it may still be needed. Fortunately a lot of this work can be automated cutting down on time spent on maintenance tasks.

If there are several nodes on a local network, it is possible that alerts may be duplicated. This could be caused by traffic that flows through two listening points, such as one

located on the gateway and another overseeing a cluster of servers. This can be avoided by limiting which traffic each node inspects and logs through a pcap filter. For instance, if the node inspecting traffic on the gateway is also in a position to view traffic flowing between machines on the LAN, it should be instructed not to monitor this traffic. The nodes monitoring the LAN would also have to be instructed to not monitor traffic leaving the network, since it will be monitored by the NIDS node reading the traffic flowing through the gateway. A lot of planning and thought would have to be put into determining the right setup for each network.

9 Putting it All Together

9.1 Approaching NIDS

The previous sections have outlined many techniques and methods that can be used in an intrusion detection system. Throughout the process the possibility of combining different techniques as a means of increasing a system's overall effectiveness has been alluded to.

A design is presented below that attempts to present a NIDS system that offers comprehensive coverage, is scaleable, and is also very effective. A comprehensive NIDS will have the best chance of catching as many attacks as possible. Modules are an important concept for this design as they allow individual nodes to be built that can fulfill

different needs at different times. It also allows for a way of separating different techniques into components that can be run independently of each other.

The scalability of a NIDS is an important design consideration. Being able to add monitoring nodes as needed helps ensure that the implementation is flexible enough to meet the needs of different networks. A NIDS could be designed to offer a very comprehensive coverage of the nodes which it is monitoring, but if it can not work with other nodes on the network its value is diminished.

9.2 NIDS Design

So far a number of different methods used in NIDS have been examined, but little attention has been spent on having them work together. This section is an attempt to do just that, by introducing a design that integrates these different techniques.

The basic building block of this design comes from section 3.3 as it has been a cornerstone for many of the different techniques presented so far. Header data will be stored for every connection that a monitoring node sees in ASCII text files. The log files will be rotated based on the time of day. For instance, the administrator (or programmer) may choose to have each file contain all of the headers for one hour's worth of traffic, or maybe only a minutes worth. This can be written off as an implementation issue that can vary depending on the desire and needs of the administrators. For the moment just assume that the files contain an appropriate timeframe of data that allows us to generate

statistics on network flow which are both accurate and precise. The entire amount of information found in a header is generally not needed. A generic NIDS can do without storing certain pieces of the data such as the header length which will let it save hard drive space. Since this is just a rough outline of a NIDS system, there is no need to worry about this or even the format of the file.

The program that generates the files filled with data about the headers should be the only program that is sniffing the wire and will act as the data gatekeeper for the rest of the suite. This program will be referred to as ``the gatherer''. Once the gatherer has written to the file the information about a packet it has captured, it would then pass it along to another program for further analysis. This is not a requirement as having it simply store the traffic flow information may suit the needs of some systems. There are two points that should be highlighted about passing a packet to another program; the first being that the program is separate from the gatherer; and the second is that the program is not specifically defined, and could be one of several.

Separate programs would be used for each distinct portion of the NIDS and UNIX IPC calls would be used to pass data between them. Using separate programs allows the NIDS to split the use of machine resources. For instance, if the NIDS was running on a computer that has 2 CPU's, then the gatherer program could be used primarily on one, and a signature checking program on the other. Another benefit to this approach is that it is possible to update one of the modules without having to stop the others at the same time. Combining several different programs would allow the programmer to separate the

different functions of the NIDS so that each of them can be used independently of the others.

The individual modules designed to work with the gatherer would depend on what function the monitoring node is to have. Assuming that the NIDS will implement each of the methods discussed in this paper, the obvious approach would be to have different programs which do signature analysis, anomaly detection, and any active scanning of the network.

Implementing the signature portion is rather straightforward. Section 6 discusses the method in detail. The module would need to have a large database of signatures that can either be custom made or taken from open source NIDS (this depends on the license of the NIDS being built, but this is something that would have to be addressed by the programmer). The list must be read in at the start of the program and kept in an efficient data structure for the duration of the program. The searching algorithm used can vary, but the Boyer-Moore algorithm is a good choice. This program can be written as a network daemon or as a command line tool. Having it run as a daemon is appealing since it is possible that restarting the program every time a new signature has to be added can be avoided.

An anomaly detection module is a bit trickier since there is a choice of modeling the traffic or run many statistical tests on it. Anomaly detection will be approached in three parts, with a separate module written for each. The modeling of data would require that

each protocol that the NIDS is dealing with has to have a set definition for what is considered normal. Each definition would have to be loaded at run time similar to the signature based module described above. Writing it as a Daemon would once again mean that one does not have to restart the program if something is added to the definition of a protocol. It would generally not be feasible to have a model for each protocol, as there are hundreds of different applications that can be modeled. Instead it is likely that just a dozen or so of the more popular applications such as http, ftp, DNS, etc. will be modeled. The gatherer should only pass the module packets if the packet can be compared to a preexisting model.

Running anomaly detection on packet data is much trickier. The author does not believe that it is the most productive technique that can be implemented, but it is worth mentioning. The program which does this would accept each packet passed to it by the gatherer, and then would create statistics based upon this information. The types of statistics that it would create would be best if it were related to the protocol being used, and the makeup of the data. For instance, the program calculates the average size of inbound web traffic. If the size of such a packet was too large, it would be out of the ordinary and would raise some kind of alert. The program could be either written as a daemon or as a regular program. An important part of the program would be to store the generated statistics so that it does not have to relearn everything if it is restarted.

Statistical tests can be run on the logs storing each header once the gatherer has created them. This would likely have the affect of detecting odd sets of traffic from specific

hosts and also finding traffic spikes, port scans, and packets with odd header fields. If the link layer portion of the packet is recorded by the gatherer, then it is possible for it to spot some cases of packet spoofing and other network level attacks. Examining link level headers is important in detecting network attacks since the MAC address (recorded in the header) will help a program to determine if packets with the same IP address are being sent from different machines. It is possible to change the MAC address of a spoofed packet, but often this is not done. This program (or set of programs) would best be written as a command line utility and not as a daemon.

Running statistical tests on the data which the gatherer has stored is useful, but it is likely that we can get the same results by writing Perl or shell scripts that look for specific things. For example, the statistics program may be able to spot a port scan but it would be better left to a specific script to search for this. The same applies for spotting packets that have odd header information. Since there is a finite and generally well known list of attacks that generate odd headers tools can be written that look specifically for them. A perfect example is in looking for a TCP packet which has all of its flags set. This type of packet does not occur normally and is a dead giveaway to malicious behavior taking place. While a statistical tool should be able to find that this packet is out of the ordinary, it would be much quicker to just run search for specific patterns from a shell script. Many different command line programs can be written to look for occurrences such as these. It would be best to write a number of scripts that run specific and useful tests on this data.

The next module to implement would be one that examines the network for signs of intrusion. This is discussed at length in section 4 in the discussion on detecting network sniffers. The downside of applying this to the present model is that it requires that the monitoring machine be able to write packets to the machines on the LAN we are monitoring. This has the unfortunate consequence of opening the machine to direct network attacks since the transfer wires on the network cable cannot be cut. A program that does this would not have to work with the gatherer program but it could rely on it for sniffing the replies and is worth a brief mention. A program to detect sniffers can be written as a command line tool and be either run on a regular basis or at the discretion of the administrator.

The other type of active monitoring that has been discussed is the use of a port scanner that records which ports are open and which machines are on the network. This creates the same problem as a module which looks for the presence of a sniffer, in that it requires that the monitoring machine send packets to the local network. It is still useful to have a list of what ports are open on each machine on the network as it can aid an administrator who is tasked with identifying machines that may be at risk of exploitation owing to a newly discovered exploit. Writing a command line utility to do this is rather trivial as it can be run as a script that uses a current port scanner such as nmap.

It is important that there be a way for each of the monitoring nodes to be used together in order to have complete coverage of the network. This leads to the system being able to scale well since a new node for each new segment of the network can easily be added.

The trick to this is having a central console which manages each of the nodes. This involves the creation of a central monitor and also of a process where the monitor can communicate with each node.

The central monitor would be tasked with gathering the alerts generated by each NIDS node and also to be an interface from which administrators can remotely manage both the alerts and the NIDS nodes. Since what is being proposed is essentially a distributed design it is important that the gatherer program on each node be limited to collecting and analyzing packets that the other nodes will not see. This will avoid the duplication of alerts. The console should have a database capability that allows it to store the alerts it receives and also create administrative information about the alerts. The administrative information may include such things as the frequency an alert is triggered, how often specific machines are targeted, and other useful information.

Besides collecting alerts from the nodes it is best if the console has the ability to give instructions to each of the nodes and also query them for information. Each node would have to have a separate managing daemon whose purpose is communicating with the console. For example, when a new signature is created it must be propagated through each of the monitoring stations. The daemon tasked with talking with the console would have to be able to communicate with different modules in order to update this new signature. Besides sending updates commands could be sent to start or stop the services on any node. For example, if a lot of strange activity is found occurring on one segment the administrator could request that sniffer detection software be ran on that segment.

Since a lot of data would be stored on each monitoring node in the form of traffic logs it is important that the console have a way of accessing it. This information is unlike the alerts since the traffic gathered may not be interesting from a security point of view. As a result, this data is best left on the monitoring nodes until the console makes a request to see it. There would be two general types of requests of this nature. The first would be when the administrator through the console would request statistics about the logs that each node has stored. These statistics could be broken down into time periods related to the amount of traffic, frequency and percent of specific protocol traffic, and other broad generalities. These statistics could then be graphed providing interesting pictures of network activity.

The second style of request would be for more specific information. Queries for information on the traffic to and from specific hosts can be made as either a statistic summary or as copies of the stored logs. This information would be very useful to administrators who may need information about a host. This would allow the administrator to quickly gather the traffic history to and from a machine which may have been identified as having been compromised or targeted by a hacker. The traffic relating to a machine used by an attacker could also be requested from the console, allowing the administrator to see all of the activity relating to an attackers IP address.

Since there are many different modules working on the same data set on each node there is the possibility that multiple alerts could be generated as an attack is picked up by each

module. Two approaches can be taken to prevent this. The first is to filter the alerts out either at the individual node or at the console level. The second is to change the NIDS design so that packets flow through the system in such a way that if an alert is generated by one module the packet is not passed onto the next. The author does not like this approach as it means that all of the modules have to be in contact with each other to some degree and not just with the gatherer program (or the data it generated). The design presented above creates a situation where the modules only communicate with the gatherer and not each other, leading to a situation where it would be best to filter out the alerts. A side note to this is that the alerts should indicate which module they came from as well as other relevant data about why the alert was generated in the first place. It should be noted that since we have taken a module approach it is a possibility that we could limit the capabilities of each node only using modules so that multiple alerts would not be generated. This is easily accomplished by not using anomaly detection and having the signature module only use header information to determine which signatures should be tested on a packet, and not to generate alerts. Generating alerts based solely on header

information would be left to scripts examining the traffic logs created by the gatherer.

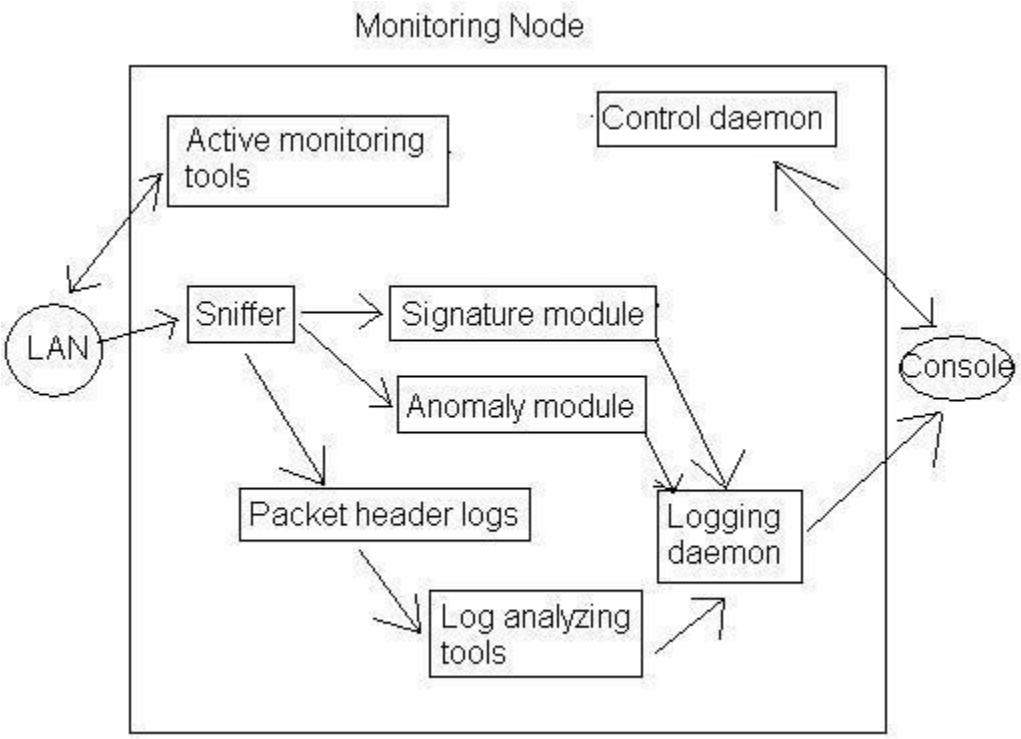


Figure 2.

Figure 2 illustrates the basic design of the NIDS. The arrows indicate the flow of traffic as it is passed through the different NIDS components. To avoid complicating the diagram the arrows from the control daemon have been omitted since it will have to talk to every component.

9.3 Closing Notes

While this design would create a fairly complex system with many different and independently effective parts there is still much room for improvement. Writing such a system is not trivial, and managing it would be an equal challenge. The author believes that the main improvement to the above design would be to increase the integration

between the separate modules. This may include using one module to validate an alert generated by another and specific improvements to the design of each component. While the techniques of each component have been described in the preceding sections the implementation of them has not been discussed in detail. Fortunately there is a wide range of literature and open source software available that will fill in the gap for those who are interested.

There is much room in the area of network intrusion detection for innovation and fresh thought. As computing power increases and computer science techniques such as artificial intelligence, data mining, and cryptology mature, the design and implementation of NIDS will adapt.

10 Recommendations

The complexity of implementing a NIDS system can be enormous. Fortunately effective systems can be built that are both effective and small in size. Examples of this include threshold and signature based detection which have been proven to work in the past, and are also often small in size. Unfortunately these methods do not offer the most extensive coverage of the network possible.

The difficulty of implementing an effective NIDS is providing a system that gives the best coverage possible. In order to currently do this the different approaches which have been discussed must be combined since there does not exist a single superior solution. Snort is an example of a NIDS that takes this approach as it offers a plug-in named spade that allows it to do anomaly as well as signature detection. By combining different detection methods, attacks which can bypass one layer of a system could be detected by another. The numbers of systems that accomplish this are steadily increasing, and this is a trend that needs to continue.

All information of interest should be stored and used by an NIDS system, especially records of network traffic. This can ensure that audit trails exist when cleaning up after a successful attack, and will provide insight into what is occurring on the network. A lot of NIDS systems generate useful alerts, but fail to allow for a more in-depth analysis of network traffic at a later time. This can be corrected by logging the headers of the packets that the NIDS inspects.

Another problem that all NIDS systems experience is that they often generate a large number of false positives. While fine tuning the system will have a desired affect of reducing these inroads can be made by better handling of alerts which are generated by systems that have multiple detection capabilities. Expecting parts of the system to generate false positives and validating or ranking the alerts based on what separate detection techniques that are integrated into the NIDS say about the traffic can be a way of combating this problem. This is an approach which needs to be examined more in-

depth, but sadly the author feels that there are currently not enough reliable techniques available that can be used to identify the same attacks. Currently the techniques that can be used to validate each other would be anomaly and signature based detection but the author is not convinced that they reliable enough to validate each other.

Implementation and the placing of NIDS will vary depending on the network where it will be placed. The author would advise against taking a "one-size fit all" approach to this and instead would encourage final implementations of a system of nodes to be custom made for the network that they will serve. This approach can be simplified by designing a module system of NIDS, parts which can be used independently or work together. Care should also be taken when designing a monitoring node to ensure that it will be able to work with other nodes that may later be added.

Bibliography

[1] Sumit Dhar, "Sniffer Basics and Detection",
http://www.rootshell.be/~dhar/downloads/Sniffers.pdf Jan 14, 2004

[2] Robert Graham, "Sniffing FAQ",
http://www.robertgraham.com/pubs/sniffing-faq.html March 1, 2004

[3] D. Wreski and C. Pallack, "Network Intrusion Detection Using Snort",
http://www.linuxsecurity.com/feature_stories/snort-guide-2.html April 14, 2004

[4] "Snorting the Enterprise",
http://www.securityfocus.com/data/library/Snorting_the_Enterprise.pdf April 14, 2004

[5] Michael Holstein, "Intrusion Detection FAQ",
http://www.sans.org/resources/idfaq/fragroute.php April 14, 2004

[6] Renaud Deraison, "Using Nessus NIDS features",
http://www.nessus.org/doc/nids.html April 14, 2004

[7] "View Based Network Security",
http://www.q1labs.com/products/views.html April 14, 2004

[8] "Bind", "Sentinel",
http://www.mirrors.wiretapped.net/security/network-monitoring/sentinel-readme.txt
April 14, 2004

[9] “hyperion”, “Watcher, NIDS for the masses”,
http://www.phrack.org/show.php?p=53&a=11 Jan 12, 2004

[10] Fyodor, "NMAP Network Security Scanner Man page",
http://www.insecure.org/nmap/data/nmap_manpage.html April 14, 2004

[11] Max Vision, "Whitehats Network Security References",
http://www.whitehats.com/library/passive/ April 14, 2004

[12] Lance Spitzner, "Passive Fingerprinting",
http://www.l0t3k.org/biblio/fingerprinting/english/PassiveFingerprinting.html
April 14, 2004

[13] Lance Spitzner, "Know Your Enemy: Passive Fingerprinting",
http://www.stud.tu-ilmenau.de/~traenk/finger_passive.htm

[14] Big Hawk, "X86 Linux Shellcode",
http://shellcode.org/Shellcode/Linux/shell-bind-shell.html April 14, 2004

[15] Davind LePorte, "Network Intrusion Detection",
www.ccs.neu.edu/groups/acm/lectures/david_laporte/ NetworkIntrusionDetection.ppt
Feb. 14, 2004

[16] Max Vision "Whitehats Network Security Resources",
http://www.whitehats.com/ids/ April 14, 2004

[17] Anonymous, "Shadow",
http://www.nswc.navy.mil/ISSEC/CID/shadow.ppt Feb. 12, 2004

[18] Duane Dunston, "Dsniff 'n the Mirror",
http://www.l0t3k.org/biblio/sniffers/english/dsniff.txt April 14, 2004

[19] Martin Roesch, "Snort - Lightweight Intrusion Detection for Networks",

http://www.usenix.org/publications/library/proceedings/lisa99/full_papers/roesch/roesch.pdf
April 14, 2004

[20] Nigel Houghton, "Snort Signature Database",
http://www.snort.org/snort-db/sid.html?sid=301 April 14, 2004

[21] Herve Security Consultants, "IDSWakeup",
http://www.hsc.fr/ressources/outils/idswakeup/index.html.en April 14, 2004

[22] Ktwo, "Security Info",
http://www.ktwo.ca/security.html April 14, 2004

[23] Anonymous, "TCP Header Format",
http://www.freesoft.org/CIE/Course/Section4/8.htm April 14, 2004

[24] Craig Rowland, "Covert Channels in the TCP/IP Protocol Suite",
http://www.firstmonday.dk/issues/issue2_5/rowland/ Feb. 15, 2004

[25] "Intrusion Signatures and Analysis". Mark Cooper. Que, 2001

[26] Anonymous, "Active FTP vs. Passive FTP, a Definitive Explanation",
http://slacksite.com/other/ftp.html April 14, 2004

[27] Anonymous, "Anomaly Detection for Computer Security",
http://www.cs.unm.edu/~terran/research/security.shtml April 14, 2004

[28] Kyle Haugsness, "What is polymorphic shell code and what can it do?",
http://www.sans.org/resources/idfaq/polymorphic_shell.php April 14, 2004

[29] Symantec, "Intrusion detection systems: Defining protocol anomaly detection", http://www.zdnetindia.com/print.html?iElementId=79203 April 14, 2004

[30] "TCP/IP Illustrated: Volume 1 The Protocols". Stevens, W. Richard. Addison Wesley Publishing Company, Reading MA, 1994

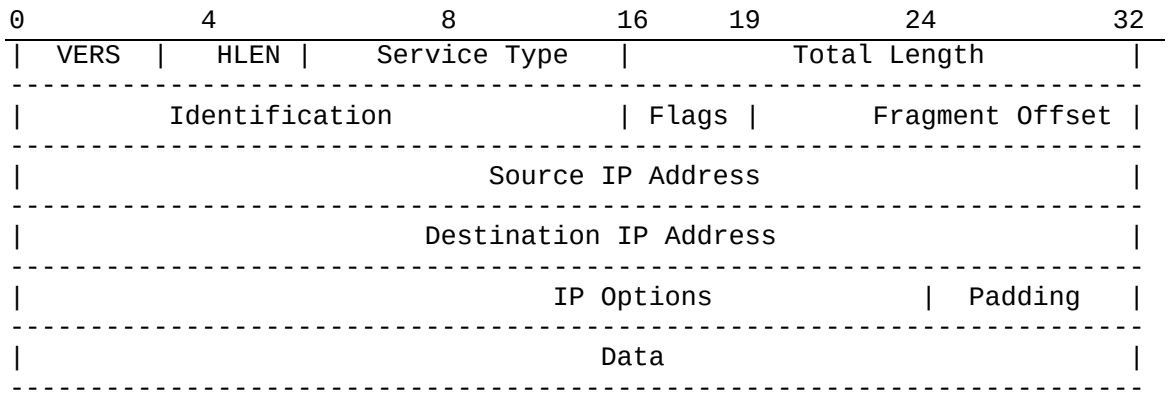
[31] daemon9, route, infinity, "Project Neptune", http://www.phrack.org/phrack/48/P48-13 April 14, 2004

[32] Anonymous, "Password Sniffer", http://www.packet-sniffer.net/password-sniffer.htm April 15, 2004

Appendix I

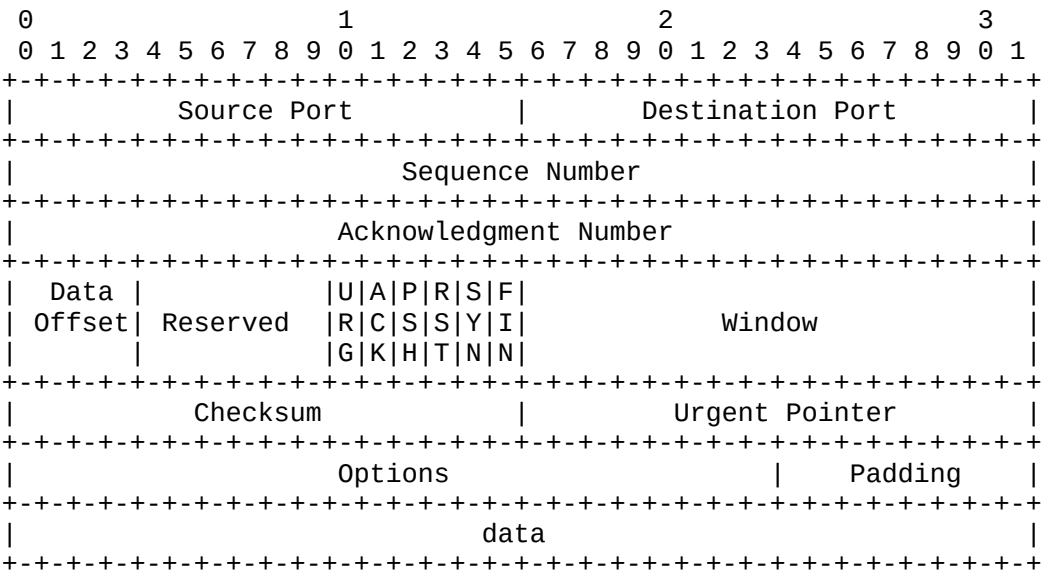
TCP and IP header formats

IP v4 Header Format



[24]

TCP Header Format



[23]