Global Information Assurance Certification Paper

Using "Wasted" IP Addresses to Amplify Slow Scans

Abstract

There are many different types of scanning used today to find potential vulnerabilities on networks. Many of these scans are very fast and efficient. To slow the fast scan, various methods have been developed[1]. Other scans attempt to use various TCP/IP options to be stealthy[2]. The stealthy scan types are now fairly routine and can be detected by the IDS[3]. Another type of scanning is the "slow scan." These are connection attempts that occur very slowly, such as once per hour, over long periods of time – one or more days. This is done purposely to attempt to avoid detection by the honey pot or IDS. Attempts to detect the slow, random scan with a honey pot or IDS are; advanced reporting from the IDS logs (such as ACID[4]) and that of the Honeypot Farm[5]. Other techniques are also available from commercial software such as Silicon Defense's CounterSleth[6]. This paper proposes yet another scan detection technique: The scan amplification device. The characteristics proposed for this device are:

1. It is lightweight, requiring little hardware and no software purchases.
2. Easily manageable, in that a single machine can amplify whole networks with little to no administrative effort.
3. Effective detection for a variety of slow scans with a single configuration while attempting to fill all unused addresses on a subnet.

[1] Voemel, Chrisof.
[2] Fyodor.
[3] URL: http://www.snort.org/cgi-bin/done.cgi
[4] Danyliw, Roman.
[5] Spitzner, Lance.
[6] "Sentarus. CounterSleuth. Military-grade intruder intelligence."

Concept

Intrusion Detection Systems have become a large part of a well-rounded firewall system[7]. They are useful; not only in indicating attacks, but also in determining which machines might be compromised on one's own network. Running an IDS is a complex, time consuming operation, often with mixed results[8]. There are many methods available to the cracker that can impede an IDS and cause to it miss or even bury the details of an attack or a cracker's reconnaissance effort.

Some of the more popular attack methods which attempt to evade detection by an IDS are[9]:

1. Insertion attacks, where the logs of the IDS are packed with misleading data.
2. Evasion attacks, where the IDS misses an important bit of data.
3. Denial of Service Attack, where the IDS has one or more of its resources depleted.

These attacks are successful because the typical IDS is a passive device, simply watching the network packets go by. It has no concept of the exact disposition of packets at their intended destination.

The three problems listed above have caused some computer security experts to declare that IDS is useless[10]. This has been hotly debated on many Internet forums[11]. While the volume of information that can be generated by an IDS is one issue in and of itself, the more problematic issue is that of the "slow scan."

A slow scan is characterized as a scan occurring over a period of hours to months[12]. It is the slow scan that is often the more dangerous but least detectable threat[13]. Very often these low volume incidents get lost in the other less dangerous but voluminous incidents reported by the IDS[14]. In this paper, a method of amplification and detecting the slow scan using "wasted" IP addresses will be discussed. Examples and tripwires to be used with this method will also be provided. A statistical analysis of the amplification effect, both theoretical and experimental will also be discussed.

The concept of IP address amplification is built on the foundation of the ability to assign multiple "alias" IP addresses to a single physical interface and the fact

[7] Bandy, Phil; Money, Michael; and Worstell, Karen.
[8] "nidsbench. A network inrusion detection system test suite."
[9] Ptacek, Thomas H.
[10] Pescatore, John and Stiennon, Richard.
[11] Judge, Peter.
[12] Chuvakin, Anton.
[13] Ullrich, Johannes.
[14] Hoagland, Jim.

that there are usually many "wasted," unused IP addresses on a network segment[15]. This idea of "IP address aliases" was first invented by several of the Unix vendors in the early 1990's to solve the problem of running multiple web servers on a single system. Today, almost all modern computer operating systems will allow many multiple IP addresses on multiple subnets assigned to a single physical interface. There are many uses of this concept in the world of network security. The use of interest in this article is the enhancement of the detection of slow scans. The many addresses which are unused will be assigned to the network adaptor of the detector. This provides a large pool of addresses which will answer when a cracker scans the network.

The idea of using a "honey pot" or "lightening rod" system to capture cracker activity is well known and documented[16]. These systems are often complex, expensive and hard to manage. They provide an environment for the cracker to attack. The amplification device described in this paper is similar to a honey pot, but lacks the virtual server environment of the honey pot. Its simple purpose is to detect a connection attempt and log that event. Some of the features of a honey pot can be provided by this system, but that is not its primary function. Server connection information can be provided to the cracker with this system. One might wish to do this to try and hide the purpose of the system as an IDS. The goal however, is to provide so many instrumented addresses that by the time the cracker discovers the purpose of the system being connected to, it is to late: They will have already been discovered.

There are three implementation cases which follow the network sizes: Class A, B or C. The size of the network to be instrumented is important and requires consideration. The reason for this is because of some limits found by experimentation in the amplification tool. The tool of choice tested for an amplification device was Red Hat Linux version 9.0. With this version it was discovered that a full class A of IPv4 addresses could be assigned to an interface, but doing so caused the system to become very unstable. A PERL script is provided in the appendix for illustrating the instability for creating aliases for a whole class A (page 27). Considerable memory was used when this was done. It was not a significant enough resource utilization to be too expensive to do. The problem created by assigning the aliases for a whole class A is that the ifconfig command becomes unresponsive. A system reboot or shutdown will hang after creating aliases for the whole class A. Experimentation revealed the upper limit for aliases with this operating system, which can be assigned to a physical adapter without causing instability, is 7777. This limitation means the typical internal class A or B networks require some forethought in laying out the addresses of real devices and amplifiers. For these large networks, more than one amplification device may be necessary to get the desired coverage for amplification.

[15] URL: http://library.mobrien.com/Manuals/MPRM_Group/tcp-ip.htm
[16] Halme, Lawrence R. and Bauer, R. Kenneth

The simplest implementation cases are those of the smaller networks where 7777 aliases are enough to totally fill the addresses of the network. In these networks there are one or several class C addresses on the network segment we wish to instrument with an amplifier. It would be possible to fill ALL unused addresses in these subnets. The addresses not used by servers and other network gear would be assigned to the amplifier providing the added ability to monitor for IP address conflicts. Conflicts found could indicate some unknown, possibly undesired network activity by a cracker not involving the scanning of the network or servers.

In many cases, these small networks would be used on the actual Internet addresses assigned by an ISP. In this setup there is usually significantly less than a class C of addresses provided with probably fewer than half being actually used for server or client addresses. In this case there are a small enough number of addresses to be assigned to the amplifier that a single Linux box could be assigned every single unused address in the range. Planning is still needed to determine how the addresses are distributed between servers, network gear and the amplifier.

For the larger class B network there are possibly many more addresses that need to be instrumented to fill the address space. Since there would possibly be a mixture of client machines and servers on this type of network, one possibility would be to use the client machines as listeners as well as an amplifier for this network. A distributed amplification for the network could be created in this manner. These devices could be arranged to report back to the amplification device. Arrangements such as these are called Meta-Intrusion Detection Systems[17].

In the instance of the large class A network, a single amplifier would be inadequate to provide useful amplification. A system of amplifiers and distributed amplifiers will be necessary to provide instrumentation in this case. For the purpose of this paper to illustrate the concept of filling a network with sensors, the small class C type networks will be examined in the greatest detail.

[17] Ethier, Patrick.

Theoretical Statistical Analysis

The theoretical statistics for using an amplification system are very straight forward: divide the number of addresses used for detection by the total number of addresses on the subnet and you will have the probability of a slow scan being detected for a given connection attempt. This is theoretical because you assume that the addresses that have been instrumented are equally likely and as frequently scanned as those you have not instrumented.

Some scanners may actually listen for traffic (for shared or broadcasted traffic) before deciding what to scan. One type of traffic that a scanner might listen for and use to create a collection of addresses to scan is ARP (Address Resolution Protocol) broadcasts. Scans based on this scheme can be made to be equally likely for any address on the network. Simple pings between machines to addresses will suffice.

Scans based on random addresses would dictate that the distributions of addresses follow the random number generator's distribution for best results. Should the random number generator employed by the scanner be uniform, the best distribution of addresses over the address range would be a uniform and regular mixture of real and instrumented addresses.

Some random scans might be based on either a "regular" normal distribution or one that is skewed. In this case, it would be best to place the server addresses on the tails of the distribution. The amplifier addresses would be in the 80-percentage range under the top of the "bell curve" (if that many instrumented addresses could be spared).

Should the scans be sequential or algorithmic in nature, the best distribution would be to place all of the instrumented addresses where they would be found first. For example, should one observe sequential scans from the bottom of the range up, the best practice is to place all of the servers at the top end of the subnet and the scan detectors at the bottom end of the addresses. If one were to detect scans from the top down, the opposite would be true. If one were to observe that binary scans of the address space were occurring, the pivotal points of the scan would be where the instrumented addresses would need to be placed. The bottom end of the scans would be where the servers would be placed.

In the ideal situation, one could sample and determine the distribution of the scans (slow or otherwise) on the network. The distribution of server addresses verses that of the instrumented addresses would change from time to time to fit the current scanning trends. The various statistical methods of determining the type of distributions of the scan could be used to assist in determining the instrumentation of the network addresses. This could be considered to be an

"adaptive" address distribution. The effort necessary to make such changes would be very great. Some of the necessary processes could be automated making this process at least possible. With the ability to assign IP addresses based on MAC address from today's modern DHCP servers[18], one could change the distribution of addresses if the MAC addresses of all the real machines on the network were known and managed. Additionally, the current version of BIND could then be auto-updated by the requested DHCP address change to reflect the changes in the DNS servers[19]. The only issue remaining would be to automate the process for changing NAT addresses, default routes and/or filter rules in the routes and firewalls for internal networks. This could possibly be automated depending on the integration between the DHCP, BIND, routers and firewall servers, but this is a complex task. The coordinating of an automated change in addressing between these servers is very unlikely to be successful. Even if it were successful, there is still the problem of disciplining the IT staff to use names and not numbers in their connection strings (hard coded in their applications). The issue at hand is how to instrument the amplifier on a network to at least capture the theoretical ratio of real to alias addresses for any of the random or sequential (algorithmic) scanning techniques, all at the same time.

Since the binary search is so ubiquitous to the computing industry[20], it seems logical that this would be a good beginning place for the distribution of server addresses and instrumented addresses for any of the random or sequential type scans. Consider the example of a class C network. A typical cracker using an algorithmic scanning approach would probably try the boundaries of this network first. After probing the beginning and ending addresses, the next logical approach would be to scan the class C network with a binary search. The goal would be to make sure the cracker gets nothing better than using a uniform random distribution of probed addresses.

In the following sections, simulations will be made using the various scanning distributions to experimentally show that a random address instrumentation applied by a binary search tree is a good general starting point for the address distribution. All three scanning distributions not based on ARP (uniform random, normal random, and sequential) will be experimented with. Comparisons of each to the others will be done.

[18] Lemon, Ted.
[19] "BIND 9 Administrator Reference Manual." Internet Software Consortium, 2001. p 23.
[20] Sedgewick, Robert. p 178.

Methods and Limitations

Methods

In the most general sense, the idea is to spread in some distribution a set of aliased, instrumented addresses around in the subnet address space, completely filling all unused, available addresses if possible. These addresses report back to a common log that easily condenses scanning activity. For the amplification platform of choice, one can easily use xinetd to create the logging device for connection attempts without the need of compromising the amplifier or creating a complex honey pot (see example configuration in appendix, page 29). Using xinetd, one can also create an illusion of being many different types of operating systems at the same time by instrumentation of ports not usually found on Linux, but rather found on Windows or Macs. This allows the amplification device to provide the opportunity for a cracker to test any number of OS ports on any of the instrumented addresses. Care must be taken; however, to limit the ports instrumented to just a few. If too many ports are available and logged there is a chance the detector could have a denial of service attack run against it through resource depletion. Fortunately xinetd can help deal with these issues[21].

The real power of using xinetd is that it will not only allow the logging of the connection attempts, but it also allows a dummy script to replace the usual server. These scripts can be as simple as something which just "falls through" taking no action, or as complex as something which tries to determine user name and passwords from the connection attempt. Scripts such as these can also present common headers back to the cracker from a random pool (see examples from example configuration in appendix, page 30). For example, in one instance a connection to determine the type of web server running on an address might provide a certain version of an Apache web server. The next connection attempt might provide an IIS 4.0 header (although it is not recommended to provide any sort of web service on the detection device). The same could be done for sendmail, telnet, ftp and many of the other common services. This is not as powerful as a typical commercial honey pot[22], but can provide some limited bait and switch (see http://violating.us/projects/baitnswitch/ for an example of a real bait and switch honey pot) to the scanner to entice more rapid or at least continued scanning attempts.

While this is a powerful feature for the Linux amplification platform, also very helpful is the fact that xinetd will also run on the client platform of choice: Windows[23]. This allows one to instrument whole ranges of usually unhelpful client address ranges as distributed sensors. Since scripts can be written in place of the actual server that xinetd invokes at the time of a connection, central

[21] Raynal, Frederic.
[22] URL: http://www.keyfocus.net/kfsensor/kfsensoroverview.pdf (30 September 2003).
[23] URL: http://cygwin.com/packages/ (30 September 2003).

logging for the purpose of condensing connection attempts can be made from these otherwise desktop, client machines. Additionally on the large network, the potential cracker will find many, many servers which must be scanned before finding the one which is vulnerable. If each of these clients are also answering with random server headers, a thick cloud of obscurity can be created on the large network.

Limitations

Many of the new worms coming out seem to have an improved scanning logic[24]. Observations have been made that seem to indicate that some of the new worms actually scan based on ARP broadcasts once an infection is made on a local subnet. An experiment was made by purposely placing a worm vulnerable MicroSoft Windows 2000 machine on a shared, super-netted network of two class C networks sub-netted from a class A which are directly on the Internet. Snort was used with the appropriate worm rules installed to watch for infection (see the appendix for the rule listing, page 28).

After about 20 minutes, the target machine became infected. The infection was known because the machine rebooted itself to finish the infecting process. The machine was also scanned with a virus scanner to verify the infection[25]. After the infection was detected, Ethereal was used to monitor all traffic from the infected machine. The machine waited about 15 minutes before trying any scanning on the subnet it was placed on. After that time, it scanned only the existing IP addresses for which ARP broadcasts were made during the time it "rested" after its infection (based on analysis of the Ethereal log).

The infected machine seemed to scan both real and alias addresses with no preference. It even scanned both the real and alias addresses for a single adapter, but the implication is very clear: If a worm were "smart" enough to listen and collect addresses to scan before scanning a network, then so would the slow scanner. The slow scanner, if listening on and scanning from one's own network for ARP broadcasts, could wait, collect and analyze addresses for a very long time. After a suitable collection time, the slow scanner could very efficiently choose the machines on which to try a single connection.

The slow scanning cracker is in a position to devote a great amount of time to the collection of machine addresses to analyze and scan. One process of a careful, detailed analysis might be a simple "Who Has" ARP request on each address collected. A table could be made of MAC addresses for each IP address (only necessary if the network is not shared, that to can be surmised with enough time). For alias IP addresses on the Linux amplification machine, all 7777 addresses would return the same MAC result. An analysis of such a table by the

[24] URL: http://xforce.iss.net/xforce/alerts/id/150
[25] URL: http://vil.nai.com/vil/stinger

cracker would reveal a single machine with 7777 addresses and the amplifier would then be revealed for what it was to the cracker.

It might be argued that the cracker would scan the addresses anyway since it is a common practice to use IP aliases and assign multiple, differently configured services to them[26]. But still, it is unsettling to have such an easy method of detection available to a cracker. A solution to this must be found! On the amplifier platform of choice there is none easily found on the Internet. It seems a really smart cracker will have their way with the amplifier, subverting its effectiveness by only listening to or making ARP requests to one or more of the many (possibly 7777) IP addresses assigned to the adapter of the amplifier.

If the actual MAC address could be set for each IP alias, it would be possible to hide the amplifier addresses with the real ones. There appears to be some limitation in the kernel of Red Hat 9 that will only allow a single MAC to be assigned to the adapter, and not one to the IP alias. Investigations into both the ifconfig and the ip[27] command provided the same result: Only a single MAC address may be assigned to the physical adapter. IP's that are aliases may not be assigned their own MAC address different from the adaptor, seemly due to a limitation in the kernel or design of the kernel.

For the sake of argument, let us suppose one COULD indeed assign a MAC address to each IP alias. A whole set of possibilities open up. Ideally, one would register their very own set of Ethernet MAC addresses with the registry. The registry provides for the registration of "private" addresses, as seen at their web site[28]. This would mean that one would not ONLY manage the IP addresses of their network, but also the MAC addresses of their network. Each device, which could have its address changed in memory or on device, would be assigned a unique MAC address belonging to the owner's MAC address range on the network. There is also the option of just picking a MAC address range for your organization and using that. The cost of the registration could be avoided in this way. Care must be taken to ensure that MAC conflicts will not arise with public MAC addresses that could be purchased on vendor hardware.

If it were possible for the amplifier, each IP address attached to a single physical adapter would also be assigned a unique MAC address from a pool of MAC addresses. The addresses would ideally be assigned in a random fashion from the owner's pool or owner-generated pool. Using these MACs would not indicate the ARP MAC was from an alias IP, or from the same adapter, or from the same machine. Not only would this enhance the effectiveness of the amplifier, it would also obscure the adapters' brand and driver. That can be important since some network cards and their drivers may have vulnerabilities[29]. This would not stop

[26] URL: http://httpd.apache.org/docs-2.0/vhosts/ip-based.html
[27] Kuznetsov, Alexey N.
[28] URL: http://standards.ieee.org/faqs/OUI.html#q1 (01 October 2003).
[29] URL: http://www.secunia.com/advisories/8987/ (01 October 2003).

the cracker from finding low level hardware or network driver exploits, but it would slow them down considerably.

Should this process become popular and shown effective, modified Linux kernels might be written to allow this to be done. If so, another cloud of obscurity could be thrown onto a network to confuse the best of scanners. The only other necessary ingredient would be to have some ARP traffic generated for ALL of the alias addresses. A simple CRON job running somewhere that does a ping would suffice. Should it be impractical or not possible to assign each of the alias IP's its own address, another approach would be to periodically change the assigned MAC address to the lot of the alias IP's with a randomly generated address. This would also best be done using one's own private set of Ethernet MAC addresses. It is only necessary to change the address on any one of the alias IP's (or just on the real adapter) to change them all. A script is provided in the appendix that illustrates this procedure (page 32). Since our amplifier platform of choice is inexpensive and runs well on older, inexpensive hardware[30], another option would be to have many amplifiers with many adapters in each amplifier.

[30] URL: http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/release-notes/x86/

Experimental results and Conclusions

Statistical Simulation Results

To illustrate and test the effectiveness of some of the amplification concepts discussed previously in this paper, statistical simulation was done. The exact parameters for a network will determine how a real instrumentation is implemented. The real results achieved from a real setup will vary directly to its difference from this experimental case. For the tests done, the network size is class C (254 addresses). In this class C, 80% of the addresses are unused and 20% are used for real devices. This means there will be 51 real devices and 203 unused device addresses on the network. The distribution of the real devices will be made to randomly disperse them on a binary search tree of the class C addresses. If this is effective in placing instrumentation around the real addresses, any of the scanner distributions should be no worse than 80% instrumentation hits and 20% real hits.

A PERL script is provided in the appendix (page 19) that creates the distribution given an arbitrary number of real devices for a class C network range. For the purpose of illustrating the results of the script, an encoding is used to show a layout. In the following example string of letters, an "R" is used to denote a real device and an "I" is used to denote an alias, instrumented address. The position in the string indicates the real or instrumented address within the class C, numbering from 1 to 254. The binary search tree[31] is first generated by the script that describes the search for the range of addresses in the class C network. Each level of the search tree is assigned a real or instrumented address based on the probability of an address being real in the class C, for this example 20%. This probability is used to skew the uniform distribution generated by the PERL rand function. The skewed random number is then used to make the assignment randomly based on the order of reading the binary tree from top to bottom and left to right. One other possibility would be to assign the real or instrumented addresses from bottom to the top. Another possibility would be to bounce back and forth between tree limbs on the same level traversing either from top to bottom or bottom to top. All of these methods have the commonality of using a deterministic approach to introduce some complexity into the assignment of the random values of "R" or "I":

```
        1          2          3          4          5          6
IRIIIIIII IRIIIIRIII IIIIRIRIII IIIIIIIRRR RIRIRRIIIR IIIRIIIIIR

         7          8          9         10         11         12
IIIRIIIIII IIIIIIIIII IIIIRIIIII RIIRIIIRII IIIIIIIRII RIIRIRIRII
```

[31] Sedgewick, Robert. p 177.

```
        13         14         15         16         17         18
IIIIIIIIII RIIIIRIIII IIIRIRIIII IIIIIIIRII IIIIIIIIIR IIIRIRIIIR

        19         20         21         22         23         24
RRIIIIIIIR IRIIIRIIII IIIRIIIIRI IIRIIIRIIR IIIRRIIIII RIIIIIIIII

        25    254
IIRIRIRIII IIRI
```

Even though there is an order in which the assignments are made to the addresses in this class C network, the assignment is still random. Since the assignment is random, patterns may appear in the above strings which are just permutations of other patterns in other strings. In the example above, there is nothing to stop a string being generated in which the "I" in column 254 is pushed to the first position and all of the other values moved to the right until the "R" in column 253 takes the current position of the "I" at 254. Since there is no distinction made between any "I" or between any "R", the total number of possible strings that represent the instrumentation of addresses in the class C is calculated as a multinomial coefficient[32]. In this case there are two different objects, an instrumented address ("I") and a real address ("R"). This means the total possible set of address combinations can be calculated as follows:

$$\frac{254!}{51! \bullet 203!} = 1.3x10^{54}$$

This is a very large number of possible network configurations! This is important because it all but eliminates the possibility that any particular network configuration could be guessed. For all practical purposes, it also eliminates the possibility that any particular network configuration, created using the calculations in this paper would be duplicated by a cracker. It is important to remember that the total number of possible network configurations will vary depending on the number of real verses instrumented addresses. The more real addresses there are on the network, the fewer combinations there are and the more likely a random scan will hit a real device.

Next it must be shown that no one of the discussed attack methods is worse than that of any other against a network configuration using this method. That is, any particular instrumentation is statistically no worse than that of any other for any method of scanning. PERL scripts were written to simulate scans using a uniform distribution (page 21), a normal distribution (page 23) and a random sequential run of a fixed size (page 25). The simulated scans were done using no more than 25 scans per day and no more than 30 days of scanning against the total of 51 real addresses within the class C.

[32] Larson, Harold J. p 45.

The null hypothesis will be: there is no difference between any of the proportions of hits against real addresses by any of the attack methods. While these tests cannot be used to indicate the truth of the null hypothesis, they show that there is currently no compelling statistical indication to believe they are not true. The following table shows the raw values from the runs:

Statistic	Uniform attack	Normal attack	Sequential attack
p estimate	0.1853	0.2187	0.196
q estimate	0.8147	0.7813	0.804
Total R's	139	164	147
Trials	750	750	750

The Normal attack seemed to produce the best results. This in spite of the fact, one would hope that the real hits would never exceed 0.20 – the proportion of real addresses. The following table shows the calculated "z" values for the test statistic comparing the ratio of two proportions[33]:

Statistic	U/N	U/S	N/S
z	1.614	0.000	1.054
Alpha	0.1	0.1	0.1
Reject value[34]	1.645	1.645	1.645
Result	No reject	No reject	No reject

The results of the tests indicate there is no apparent reason to believe there is a difference between the attacks given the described method of laying out an instrumented network.

[33] Aczel, Amir. p 320.
[34] Aczel, Amir. p 1018.

Conclusions

To summarize, the objective of this paper was to create an instrumentation of a network to detect slow scans. An amplification device was proposed which is light weight, easy to manage, and effective for a variety of scans. A methodology for the deployment of the amplification device was presented. Key concepts of the methodology were:

1. Assigning an alias address to all or as many as possible of the unused addresses on a network.
2. Using a binary search tree to assign random address type of real or alias to provide some deterministic complexity to the network.
3. The possibility of using desktop machines as distributed sensors on networks larger than 7777 nodes.

The detection software for the amplifier is xinetd. The configuration of xinetd for use as an IDS relies on its ability to throttle, be selectively configured to listen for just a few services, and the replacement of the actual services for which it is listening with simple scripts.

The method described in the paper does not determine if addresses are fake addresses. It simply records what it sees. Filling the address range provides the advantage of looking at the end result of a scan, not just guessing at what the intent of the connection was. It also provides the advantage of alerting on any activity occurring on IP addresses that were thought to be not in use, which is the IP address conflict. Having this information helps better manage the network.

Future research to be done is to actually instrument a real network with an amplifier, an IDS, and a honey pot farm. Controlled scans of this network would be done. The results of each device's detection and analysis could then be compared to indicate the relative success of each against the others. If the results were favorable for the amplifier, the next experiment would be to allow real hostile scans against the devices and compare those results

References

Aczel, Amir. Complete Business Statistics. Richard D. Irwin, Inc., 1989. ISBN 0-256-05716-8.

A.V.E.R.T., Network Associates Technology, Inc. "Stinger." 19 August 2003. URL: **http://vil.nai.com/vil/stinger** (28 August 2003).

Bandy, Phil; Money, Michael; and Worstell, Karen. "Why is intrusion detection required in today's computing environment?" URL: **http://www.sans.org/resources/idfaq/id_required.php** (29 September 2003).

"Bind 9 Administrator Reference Manual." Internet Software Consortium BIND version 9.2, 2001. URL: **http://www.isc.org/products/BIND/** (30 September 2003).

Chuvakin, Anton. "IDS: slow scans?" 12 February 2003. URL: **http://lists.insecure.org/lists/focus-ids/2003/Feb/0065.html** (29 September 2003).

Danyliw, Roman. "Analysis Console for Intrustion Databases." 8 January 2003. URL: **http://www.andrew.cmu.edu/~rdanyliw/snort/snortacid.html** (29 September 2003).

Ethier, Patrick. "What is Meta-Intrusion Detection Systems?" URL: **http://www.sans.org/resources/idfaq/mids.php** (29 September 2003).

Frederic, Raynal. "xinetd." 28 February 2001. URL: **http://www.linuxfocus.org/English/November2000/article175.shtml** (30 September 2003).

Fyodor. "The Art of Port Scanning." 06 September 1997. URL: **http://www.insecure.org/nmap/nmap_doc.html** (29 September 2003).

Halme, Lawrence R. and Bauer, R. Kenneth. "AINT Misbehaving: A Taxonomy of Anti-Intrusion Techniques." URL: **http://www.sans.org/resources/idfaq/aint.php** (29 September 2003).

Hoagland, Jim. "IDS: Re: slow scans?" 12 February 2003. URL: **http://lists.insecure.org/lists/focus-ids/2003/Feb/0074.html** (29 September 2003).

Judge, Peter. "Intruders: Is detection or protection the answer?" 15 April 2003. URL: **http://news.zdnet.co.uk/business/0,39020645,2133452,00.htm** (29 September 2003).

Kuznetsov, Alexey N. "IP Command Reference." 24 April 1999.

Larson, Harold J. Introduction to Probability Theory and Statistical Inference. New York, New York: John Wiley & Sons, 1976. ISBN 0-471-51781-X.

Lemon, Ted. "dhcpd.conf(5)." Internet Software Consortium DHCP version 3.0.1rc9 28 April 2002. URL: **http://www.isc.org/products/DHCP/** (30 September 2003).

"nidsbench. A network inrusion detection system test suite." URL: **http://packetstormsecurity.nl/UNIX/IDS/nidsbench/nidsbench.html** (29 September 2003).

Pescatore, John and Stiennon, Richard. "The Death of IDS." 24 July 2003. URL: **http://www4.gartner.com/teleconferences/attributes/attr_9268_115.ppt** (29 September 2003).

Ptacek, Thomas H. "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection." 16 October 2002. URL: **http://secinf.net/info/ids/idspaper/idspaper.html** (29 September 2003).

Sedgewick, Robert. Algorithms. Reading, Massachusetts: Addison-Wesley Publishing Company, 1984. ISBN 0-201-06672-6.

"Sentarus. CounterSleuth. Military-grade intruder intelligence." URL: **http://www.silicondefense.com/products/sentarusfamily/countersleuth/** (29 September 2003).

Spitzner, Lance. "Honeypot Farms." 13 August 2003. URL: **http://www.securityfocus.com/infocus/1720** (29 September 2003).

Ullrich, Johannes. "IDS: Re: slow scans?" 12 February 2003. URL: **http://lists.insecure.org/lists/focus-ids/2003/Feb/0066.html** (29 September 2003).

URL: **http://httpd.apache.org/docs-2.0/vhosts/ip-based.html** (30 September 2003).

URL: **http://cygwin.com/packages/** (30 September 2003).

URL: **http://standards.ieee.org/faqs/OUI.html#q1** (01 October 2003).

URL**:** **http://xforce.iss.net/xforce/alerts/id/150** 11 August 2003. (1 September 2003).

URL: **http://www.keyfocus.net/kfsensor/kfsensoroverview.pdf** (30 September 2003).

URL: **http://library.mobrien.com/Manuals/MPRM_Group/tcp-ip.htm** (29 September 2003).

URL: **http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/release-notes/x86/** (01 October 2003).

URL: 11 June 2003. **http://www.secunia.com/advisories/8987/** (01 October 2003).

URL: **http://www.snort.org/cgi-bin/done.cgi** (29 September 2003). Rule numbers 621, 623, 624, 625, 629, and 630.

Voemel, Christof. URL: **http://www.giac.org/practical/Christof_Voemel_GCIA.txt** (29 September 2003).

Appendix

PERL Script to Generate Network layout

This script takes as input the number of real devices to randomly place on a class C network. It then places them on a binary search tree from top to bottom, left to right:

```
# ARGV[0] is the number of real devices
$prob = $ARGV[0] / 254;
# create sequence to make binary tree on
for ($i=1,$j=0;$i<512;$i*=2,$j++)
{
 $SEED[$j] = $i;
}
# sequence the distribution
$k = $SEED[$#SEED];
for ($j=0;$j<$#SEED;$j++)
{
 $level = $#SEED - $j -1;
 for ($i=$SEED[$j],$l=0;$i<$k;$i+=$SEED[$j+1],$l++)
 {
  $DIST[$level][$l] = $i;
 }
}

# make the I or the R in the sequence created
$r_count = 0;
$rep_count = 0;
while ($r_count < $ARGV[0])
{
 for ($j=0;$j<$#SEED;$j++)
 {
  for ($i=0;$i<=($#{$DIST[$j]});$i++)
   {
   if ($r_count < $ARGV[0])
   {
     $TYPE_IP[$DIST[$j][$i]] = &I_or_R();
    if ($TYPE_IP[$DIST[$j][$i]] eq "R") {$r_count++;}
   }
   else
   {
    if ($rep_count > 0) {last;}
    $TYPE_IP[$DIST[$j][$i]] = "I";
   }
  }
  if (($rep_count >  0) && ($r_count >= $ARGV[0])) {last;}
 }
 $rep_count++;
}
# output the I or R in groups of 10
for ($i=0;$i<254;$i++)
{
 if ((($i % 10) == 0)&&($i != 0)) {print(" ");}
```

```
 print("$TYPE_IP[$i]");
}
print("\n");
# sub to get a uniform random distribution of I or R in the proportion
# of real to instrumented addresses

sub I_or_R()
{
 $a_random = rand(1.0);
 if ($a_random < $prob)
 {
  return("R");
 }
 else
 {
  return("I");
 }
}
```

PERL script to Scan with a Uniform Random Distribution

```
# number of real nodes $ARGV[0]
# number of connection attempts made per scan $ARGV[1]
# number of networks to test on

# calculate the probablity where there are $ARGV[0] real network nodes
$prob = $ARGV[0] / 254;

$instrument_counter = $real_counter = 0;

###### Create Binary Search Tree
for ($i=1,$j=0;$i<512;$i*=2,$j++) {  $SEED[$j] = $i; }

$k = $SEED[$#SEED];
for ($j=0;$j<$#SEED;$j++)
{
 $level = $#SEED - $j -1;
 for ($i=$SEED[$j],$l=0;$i<$k;$i+=$SEED[$j+1],$l++)  {   $DIST[$level][$l] = $i;  }
}

###### Generate the instrumented network, run the simulated scan, do it $ARGV[2] times
for ($run_nums=0;$run_nums<$ARGV[2];$run_nums++)
{
######### Generate the next net, make sure it is unique to this run
 $HIST[$run_nums] = "";
 while ($HIST[$run_nums] eq "")
 {
  $matched = 0;
  &Instrument_Class_C;
  for ($im=0;$im<=$#HIST;$im++)
  {
   if ($S eq $HIST[$im])
   {
    $matched = 1;
    break;
   }
  }
  if (!$matched) { $HIST[$run_nums] = $S;}
 }
######### Run the simulated scan against our network, scan $ARGV[1] times against the net
 for ($i=0;$i<$ARGV[1];$i++)
 {
  $j = &Class_C_ran;
  if ($TYPE_IP[$j] eq "R") {$real_counter++;} else {$instrument_counter++;}
 }
}

######  Calculate the answers
$act_prob = $real_counter / ($real_counter + $instrument_counter);
print("Out of $ARGV[2] runs of $ARGV[1] scan sizes\n");
print("   hit $real_counter real and $instrument_counter instrumented\n");
$q = 1.0 - $act_prob;
print("   p estimated is $act_prob, q estimated is $q\n");
```

```
sub Instrument_Class_C
{
 $r_count = 0;
 $rep_count = 0;
 $S = "";
 while ($r_count < $ARGV[0])
 {
  for ($j=0;$j<$#SEED;$j++)
  {
   for ($i=0;$i<=($#{$DIST[$j]});$i++)
    {
    if ($r_count < $ARGV[0])
    {
      $TYPE_IP[$DIST[$j][$i]] = &I_or_R;
      $S .= $TYPE_IP[$DIST[$j][$i]];
     if ($TYPE_IP[$DIST[$j][$i]] eq "R") {$r_count++;}
    }
    else
    {
     if ($rep_count > 0) {last;}
     $TYPE_IP[$DIST[$j][$i]] = "I";
     $S .= $TYPE_IP[$DIST[$j][$i]];
    }
   }
   if (($rep_count >  0) && ($r_count >= $ARGV[0])) {last;}
  }
  $rep_count++;
 }
}


sub Class_C_ran
{
 return(int(rand(254))+1);
}


sub I_or_R
{
 $a_random = rand(1.0);
 if ($a_random < $prob)
 {
  return("R");
 }
 else
 {
  return("I");
 }
}
```

PERL script to Scan with a Normal Random Distribution

```
use Math::Random;
# calculate the probablity where there are $ARGV[0] real network nodes
$prob = $ARGV[0] / 254;$instrument_counter = $real_counter = 0;
###### Create Binary Search Tree
for ($i=1,$j=0;$i<512;$i*=2,$j++) {  $SEED[$j] = $i; }

$k = $SEED[$#SEED];
for ($j=0;$j<$#SEED;$j++)
{
 $level = $#SEED - $j -1;
 for ($i=$SEED[$j],$l=0;$i<$k;$i+=$SEED[$j+1],$l++)  {   $DIST[$level][$l] = $i;  }
}

###### Generate the instrumented network, run the simulated scan, do it $ARGV[2] times
for ($run_nums=0;$run_nums<$ARGV[2];$run_nums++)
{
######### Generate the next net, make sure it is unique to this run
 $HIST[$run_nums] = "";
 while ($HIST[$run_nums] eq "")
 {
  $matched = 0;
  &Instrument_Class_C;
  for ($im=0;$im<=$#HIST;$im++)
  {
   if ($S eq $HIST[$im])
   {
    $matched = 1;
    break;
   }
  }
  if (!$matched) { $HIST[$run_nums] = $S;}
 }
######### Run the simulated scan against our network, scan $ARGV[1] times against the net
 @R = random_normal($ARGV[1],127,50);
 for ($i=0;$i<$ARGV[1];$i++)
 {
  $R[$i] = int($R[$i]);
  if ($R[$i] < 1) {$R[$i] = 1;}
  else
  {
   if ($R[$i] > 254) {$R[$i] = 254;}
  }
 }
 for ($i=0;$i<$ARGV[1];$i++)
 {
  $j = $R[$i];
  if ($TYPE_IP[$j] eq "R") {$real_counter++;} else {$instrument_counter++;}
 }
}
######  Calculate the answers
$act_prob = $real_counter / ($real_counter + $instrument_counter);
print("Out of $ARGV[2] runs of $ARGV[1] scan sizes\n");
print("   hit $real_counter real and $instrument_counter instrumented\n");
```

```
$q = 1.0 - $act_prob;
print("   p estimated is $act_prob, q estimated is $q\n");

sub Instrument_Class_C
{
 $r_count = 0;
 $rep_count = 0;
 $S = "";
 while ($r_count < $ARGV[0])
 {
  for ($j=0;$j<$#SEED;$j++)
  {
   for ($i=0;$i<=($#{$DIST[$j]});$i++)
    {
    if ($r_count < $ARGV[0])
    {
      $TYPE_IP[$DIST[$j][$i]] = &I_or_R;
      $S .= $TYPE_IP[$DIST[$j][$i]];
     if ($TYPE_IP[$DIST[$j][$i]] eq "R") {$r_count++;}
    }
    else
    {
     if ($rep_count > 0) {last;}
     $TYPE_IP[$DIST[$j][$i]] = "I";
     $S .= $TYPE_IP[$DIST[$j][$i]];
    }
   }
   if (($rep_count >  0) && ($r_count >= $ARGV[0])) {last;}
  }
  $rep_count++;
 }
}
sub Class_C_ran
{
 return(int(rand(254))+1);
}
sub I_or_R

{
 $a_random = rand(1.0);
 if ($a_random < $prob)
 {
  return("R");
 }
 else
 {
  return("I");
 }
}
```

PERL script to Scan with a Sequential Run

```
# calculate the probablity where there are $ARGV[0] real network nodes
$prob = $ARGV[0] / 254;

$instrument_counter = $real_counter = 0;

###### Create Binary Search Tree
for ($i=1,$j=0;$i<512;$i*=2,$j++) {  $SEED[$j] = $i; }

$k = $SEED[$#SEED];
for ($j=0;$j<$#SEED;$j++)
{
 $level = $#SEED - $j -1;
 for ($i=$SEED[$j],$l=0;$i<$k;$i+=$SEED[$j+1],$l++)  {   $DIST[$level][$l] = $i;  }
}

###### Generate the instrumented network, run the simulated scan, do it $ARGV[2] times
for ($run_nums=0;$run_nums<$ARGV[2];$run_nums++)
{
######### Generate the next net, make sure it is unique to this run
 $HIST[$run_nums] = "";
 while ($HIST[$run_nums] eq "")
 {
  $matched = 0;
  &Instrument_Class_C;
  for ($im=0;$im<=$#HIST;$im++)
  {
   if ($S eq $HIST[$im])
   {
    $matched = 1;
    break;
   }
  }
  if (!$matched) { $HIST[$run_nums] = $S;}
 }
######### Run the simulated scan against our network, scan $ARGV[1] times against the net
 $r = &Class_C_seq_scan_ran;
 for ($i=0;$i<$ARGV[1];$i++)
 {
  $j = $r + $i;
  if ($TYPE_IP[$j] eq "R") {$real_counter++;} else {$instrument_counter++;}
 }
}

######  Calculate the answers
$act_prob = $real_counter / ($real_counter + $instrument_counter);
print("Out of $ARGV[2] runs of $ARGV[1] scan sizes\n");
print("   hit $real_counter real and $instrument_counter instrumented\n");
$q = 1.0 - $act_prob;
print("   p estimated is $act_prob, q estimated is $q\n");

sub Instrument_Class_C
{
 $r_count = 0;
```

```
 $rep_count = 0;
 $S = "";
 while ($r_count < $ARGV[0])
 {
  for ($j=0;$j<$#SEED;$j++)
  {
   for ($i=0;$i<=($#{$DIST[$j]});$i++)
    {
    if ($r_count < $ARGV[0])
    {
      $TYPE_IP[$DIST[$j][$i]] = &I_or_R;
      $S .= $TYPE_IP[$DIST[$j][$i]];
     if ($TYPE_IP[$DIST[$j][$i]] eq "R") {$r_count++;}
    }
    else
    {
     if ($rep_count > 0) {last;}
     $TYPE_IP[$DIST[$j][$i]] = "I";
     $S .= $TYPE_IP[$DIST[$j][$i]];
    }
   }
   if (($rep_count >  0) && ($r_count >= $ARGV[0])) {last;}
  }
  $rep_count++;
 }

}

sub Class_C_seq_scan_ran
{
 return(int(rand(254-$ARGV[1]))+1);
}

sub Class_C_ran
{
 return(int(rand(254))+1);
}

sub I_or_R
{
 $a_random = rand(1.0);
 if ($a_random < $prob)
 {
  return("R");
 }
 else
 {
  return("I");
 }
}
```

Script to assign an alias of a class A to an adapter

```
#!/usr/bin/perl
$l = 0;
for ($i=0;$i<256;$i++)
{
 for ($j=0;$j<256;$j++)
 {
  for ($k=1;$k<256;$k++)
  {
   $l++;
   system("ifconfig eth0:$l 10.$i.$j.$k up");
  }
 }
}
```

Snort Rule for Detection of Worm Infection

alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP PING CyberKit 2.2 Windows"; content:"|aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa|"; itype: 8; dsize:64;)

xinetd setup

The amplification device uses the TCP Wrapper functions available to xinetd. The following example of a hosts.allow file shows how the TCP Wrapper is used to log connection attempts and break them out into separate logs. The hosts.deny file shows how all remaining, instrumented services are blocked and their logging combined into a single file.

There are two sample server replacement scripts provided. There is one for telnet that shows how the returns from telnet connection attempts can be randomized. Another script for ftp shows how a simple empty script can be used to log connection attempts and do nothing else.

hosts.allow

```
#
# hosts.allow   This file describes the names of the hosts which are
#               allowed to use the local INET services, as decided
#               by the '/usr/sbin/tcpd' server.
#
# ALL: ALL
 in.telnetd : ALL: SPAWN ( echo "`/bin/date` Break in attempt from %u %d-%c" | /bin/cat >>
/var/log/telnet.log )
 in.ftpd : ALL: SPAWN ( echo "`/bin/date` Break in attempt from %u %d-%c" | /bin/cat >>
/var/log/ftp.log )
```

hosts.deny

```
#
# hosts.deny    This file describes the names of the hosts which are
#               *not* allowed to use the local INET services, as decided
#               by the '/usr/sbin/tcpd' server.
#

 ALL: ALL: SPAWN ( echo "`/bin/date` Break in attempt from %u %d-%h" | /bin/cat >>
/var/log/break.ins )
```

/dummys/in.ftpd

```
# just an empty file
```

/dummys/in.telnetd

```
#!/usr/bin/perl
@CNTS = (
"This is a SUN box\nCome on in: ",
"This is a AIX box\nCome on in: ",
```

```
"This is a MAC box\nCome on in: ",
"This is a LINUX box\nCome on in: ",
"This is a Microsoft box\nCome on in: ");
$i = int (rand($#CNTS+1.0));
# print ("$i\n");
syswrite STDOUT,$CNTS[$i],length($CNTS[$i]);
$l = <>;
open(FO,">> /var/log/telnet.log");
print(FO "index $i, Telnet tried with $l\n");
close(FO);
```

Example Random Headers

```
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
This is a AIX box
Come on in: asdf
Connection closed by foreign host.
[root@localhost log]# telnet 127.0.0.1
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
This is a Microsoft box
Come on in: asdf
Connection closed by foreign host.
[root@localhost log]# telnet 127.0.0.1
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
This is a LINUX box
Come on in: asdf
Connection closed by foreign host.
[root@localhost log]# telnet 127.0.0.1
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
This is a LINUX box
Come on in: asdf
Connection closed by foreign host.
[root@localhost log]# telnet 127.0.0.1
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
This is a MAC box
Come on in: asdf
Connection closed by foreign host.
```

Sample log for telnet connections

```
Sat Oct  4 16:49:33 CDT 2003 Break in attempt from unknown in.telnetd-127.0.0.1
index 2, Telnet tried with  & &          ! " '     #asf

Sat Oct  4 16:55:28 CDT 2003 Break in attempt from unknown in.telnetd-127.0.0.1
index 2, Telnet tried with  & &          ! " '     #asdf

Sat Oct  4 16:58:56 CDT 2003 Break in attempt from unknown in.telnetd-127.0.0.1
index 0, Telnet tried with  & &          ! " '     #Trying 127.0.0.1...
  Connected to localhost.localdomain (127.0.0.1).
  Escape character is '^]'.
  Connection closed by foreign host.
  [root@localhost etc]#

  asdf
```

Sample log for ftp connections

```
Sat Oct  4 16:59:06 CDT 2003 Break in attempt from unknown in.ftpd-127.0.0.1
Sat Oct  4 16:59:42 CDT 2003 Break in attempt from unknown in.ftpd-127.0.0.1
```

Sample log for other connection attempts

```
Sat Oct  4 17:09:50 CDT 2003 Break in attempt from unknown in.rlogind-127.0.0.1
Mon Oct  6 08:52:30 CDT 2003 Break in attempt from unknown in.tftpd-127.0.0.1
Mon Oct  6 08:52:33 CDT 2003 Break in attempt from unknown in.tftpd-127.0.0.1
Mon Oct  6 08:52:34 CDT 2003 Break in attempt from unknown in.tftpd-127.0.0.1
Mon Oct  6 08:52:37 CDT 2003 Break in attempt from unknown in.tftpd-127.0.0.1
Mon Oct  6 08:52:39 CDT 2003 Break in attempt from unknown in.tftpd-127.0.0.1
Mon Oct  6 08:52:42 CDT 2003 Break in attempt from unknown in.tftpd-127.0.0.1
Mon Oct  6 08:52:45 CDT 2003 Break in attempt from unknown in.tftpd-127.0.0.1
Mon Oct  6 08:52:48 CDT 2003 Break in attempt from unknown in.tftpd-127.0.0.1
Mon Oct  6 08:52:49 CDT 2003 Break in attempt from unknown in.tftpd-127.0.0.1
Mon Oct  6 08:52:52 CDT 2003 Break in attempt from unknown in.tftpd-127.0.0.1
```

Example Script to randomly assign a MAC address

```
#!/usr/bin/perl
$macbeg = "66:66:00:";
$macend1 = int(rand(100.0));
$macend2 = int(rand(100.0));
$macend3 = int(rand(100.0));
$newmac = "$macbeg$macend1:$macend2:$macend3";
system("ifconfig eth0 down");
system("ifconfig eth0 hw ether $newmac");
system("ifconfig eth0 up");
```