Experiences Benchmarking Intrusion Detection Systems

Marcus J. Ranum
<mjr@nfr.com>

Chief Technology Officer,
NFR Security, Inc.

http://www.nfr.com

Abstract

Intrusion Detection Systems *(hereafter abbreviated as "IDS") are a topic that has recently garnered much interest in the computer security community. As interest has grown, the topic of testing and benchmarking IDS has also received a great deal of attention. It has not, however, received a great deal of* ***thought****, since an embarrassingly large number of IDS "benchmarks" have proven to be so fundamentally flawed that they actually provide misleading information rather than useful results. In this paper, we discuss the topic of IDS benchmarking, and present a few examples of poor benchmarks and how they can be fixed. We also present some guidelines on how to design and test IDS effectively.*

Introduction: Benchmarks and Tests

Constructing good benchmarks is difficult regardless of what they propose to measure. In order to accurately measure something complex, you often need to expend considerable effort in designing tests, to make sure that the tests aren't inherently biased or inaccurate. This is especially difficult when you're measuring something complex like an IDS that is highly dependent on its operating environment. Generally, when designing a test, you should determine first off whether you want to:

a) Quantitatively measure varying systems against a predictable baseline

– or –

b) Comparatively measure systems against each other

This may seem like a subtle difference, but it's critical since a) is harder to implement and many organizations desiring benchmarks really want to use the data from a) to make a choice between product offerings. For that purpose b) makes more sense, but it seems less scientific. Comparative measures lack one of the important properties of a) – namely *repeatability* – an important property of the scientific method. To illustrate by analogy, consider attempting to measure the performance of automobiles: if you're trying to choose between two cars you simply take them to the drag strip and buy the one that wins. If you're trying to choose between 300 cars, you need a more abstract measuring-stick – perhaps 0-60 times and top quarter-mile speeds. There's another important way this analogy applies to benchmarking IDS: if you take 2 cars to the drag strip *you* run the test and *you* see the results but if you're comparing 300 cars you're more likely to *rely* on *someone else's* published 0-60 times. That's when benchmarking gets interesting: you need to know how each of those 0-60 times was measured and whether their measurement techniques were balanced and not prone to skewing by accident or on purpose. Unfortunately for someone measuring IDS, it's actually harder to do a reliable tamper-free baseline measurement of IDS than the straight-line acceleration of cars.

The history of benchmarking and testing is rife with examples of deliberately and accidentally flawed testing methodologies. The computer industry has some outstanding examples of creative benchmarking[1] but many of the flawed tests that are published result from oversight rather than deliberate misrepresentation. Software systems are often extremely complex and their inner workings are "invisible" which means that the testers tend to rely on external behaviors – not understanding that it's quite possible for a complex system to sometimes yield the right result for the wrong reason. As a thought experiment, consider an IDS that does nothing more than beep whenever it detects a packet crossing a network. Such an IDS will score 100% in terms of detecting attacks – nothing will get past it. Of course, it will generate a lot of

[1] I lived through the furor surrounding the SPEC 1991 benchmark and one vendor's employing a technique called "register strip mining" (a highly specialized compiler optimization specific to the benchmark) which gave them a 1400% improvement in a particular FORTRAN operation that would actually not benefit any real customer applications. As a consequence I am slightly wary of benchmarks and have devoted considerable time, with mixed success, to trying to keep vendors honest in my field.

false alarms! This imaginary IDS would perform quite well on an isolated test network but would prove to be miserable if tested against realistic traffic loads. So, before we start to think of a good benchmark, we need to figure out what it is, exactly, we are trying to measure.

What are We Measuring?

Since we're measuring "Intrusion Detection Systems" let's take for given that *a good measuring criterion would be* ***how well they detect intrusions***. It's not the only one, however, as we saw above. Other good criteria might be accuracy (false positives[2] versus false negatives) as well as operational impact – if the IDS slows down the network or systems it interacts with. Another important property is how clearly and accurately they diagnose an attack – unfortunately this is such a subjective measurement criterion that nobody has attempted it to date. The impact of an IDS' accuracy is going to be organization specific. Some organizations may have high tolerance of false positives because they have staff and time to investigate them, while other organizations would rather have a system that misses attacks as long as it doesn't raise false alarms. Consequently, it would be doing the customer a disservice if a test somehow combined the false positive/false negative information into a single synthetic value. Unfortunately, many of the published reviews of IDS products have completely ignored false positives and focused on a simpler value of false negatives versus number of attacks simulated. Remember our fictional IDS from our first thought experiment? If false positives are not considered, an IDS that simply beeps whenever it sees a packet will get a perfect score even though it would be utterly useless in a fielded situation.

Determining what measures *don't* matter is also an important part of designing a good benchmark. Knowing what *not* to measure is sometimes a harder problem than knowing what to measure. Let's consider a possible value to measure for IDS: *packets per second*. Measuring packets/second for IDS is like measuring miles/hour for food: some foods are moving and others aren't but it's almost always not coupled to the caloric content or nutritional value of the food. Packets/second is an important operational consideration but it doesn't apply equally to all IDS therefore it needs to be applied with caution. What does packets/second mean to host-based IDS? Since the HIDS only needs to deal with packets directed at the individual host, it's going to have completely different packets/second properties than network-based IDS.

So, let's look at a reasonable measurement for a simple IDS test. Imagine that we have set up a test network/system where we install various IDS. We'll imagine we have a set of attack tools (or simulations thereof – more on this later) that we can use to launch a pre-determined number of attacks against a target. When we've launched the attacks, we will examine the results from the IDS and will publish the total number of attacks launched, the number of attacks detected, and the number of false alarms generated.[3] It might be wise to put a decent time interval between attacks so as not to confuse false alarms with detections and misdiagnoses of actual attacks. The author was involved with reviewing an IDS test that made exactly this mistake: the tester counted one IDS as having much better performance than it actually had because it "correctly" detected "fragmented packets" as attacks when a variety of real attacks were launched over the top of fragmented traffic. Fragments aren't necessarily always an attack, and the tester was disappointed to discover that the IDS continued to report "fragmented packet attacks" when legitimate web traffic was transmitted over legitimate, fragmented, packets. Technically, this result is actually a false positive, since the IDS is raising an alert but it is not diagnosing an attack – merely a normal feature of IP network traffic that is sometimes found during the course of attacks. Unfortunately, this example underscores another problem with IDS testing: considerable expertise may sometimes be required to meaningfully interpret test results for accuracy. One vendor once cheated on an IDS "shootout" at a major conference by field-

[2] A "*false positive*" is when an IDS alerts on an event that is not an attack. A "*false negative*" is when an IDS fails to detect a real attack.

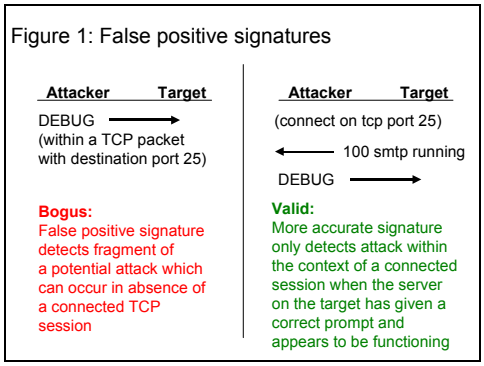
[3] False negatives can be computed by subtracting the number of accurately detected attacks from the total number of attacks sent. False positives can be computed by subtracting the number of accurately detected attacks from the total number of alerts raised.

programming their IDS to generate an alert whenever *any* NFS traffic was seen because they knew the benchmark included NFS attacks but no normal NFS traffic. The product received high marks for detection because the test organizers were not sophisticated IDS practitioners and didn't realize they had been fooled.

The Real Versus The Semi-Real

Most intrusion detection systems rely on some kind of pattern-matching algorithm in order to identify and categorize attacks. Understanding how pattern-matching is used is critical in designing a useful benchmark. For an IDS to diagnose an attack *quickly* it is easiest to announce its diagnosis *as soon as something resembles the beginning of the attack*. Unfortunately, attacks that begin do not necessarily end, nor are they always followed to completion. For example, suppose an IDS was looking for specific http-based attacks against a web server and the signature matched for a partial string indicating that a particular vulnerable CGI-script was being invoked. In the simplest form, the IDS could search for that CGI-script string in TCP packets aimed at port 80. Suppose the benchmarker decided to test IDS by simply replaying the offending packet containing the CGI-script URI – an IDS that didn't track session or TCP states would generate an alarm but an IDS that tracked session states wouldn't. In terms of a benchmark, this makes the IDS that is more susceptible to false positives look better than it really is! Triggering on a packet fragment, rather than the complete stateful transaction, is not really intrusion detection – it's deliberately accepting a false positive. An attacker wishing to spoof such an IDS could easily flood it by simply sending the fragments necessary to cause it to trigger without an attack taking place.

Figure 1: False positive signatures



Recently, NFR Security was involved in an IDS test wherein the original benchmark suffered from the false-positive-test problem. When our system was initially deployed on the test network, and the attack load was begun, the tester was very concerned because our system apparently didn't detect anything. That's because, in fact, there wasn't anything to detect! Our NIDS does complete session state tracking of ***both*** *client and server side* traffic – what the tester had done was set up a target, run attacks against it, and captured the packets from the attacking system *only*. This, the tester believed, represented a decent attack simulation. In testing, many other IDS products "detected" attacks in the captured packets when they were replayed, which further reinforced this view. In fact, since there were no valid TCP sessions taking place during the test, *there* ***were*** *no attacks to detect*, and the other products were merely triggering on partial false positives. To make the benchmark valid, it was necessary to modify the tests so that actual attacks over valid traffic were taking place, at which time the NFR NID began detecting the attacks once they were actually happening.

This is a subtle issue and is raised primarily to show the importance of understanding *what* is being tested and how it is supposed to behave. Unfortunately, many of the IDS tests that are currently being published ignore the fact that their tests are based on *synthetic* data – which behaves differently from real traffic. We'll examine some of the problems presented by simulated traffic in greater detail later in this paper.

Host, Network, Network Node

Measuring qualitative differences between Host IDS (HIDS) and Network IDS (NIDS) will most likely prove nearly impossible except on the basic criterion of attacks detected versus attacks launched. Since a HIDS operates "above" the network layer, there are a number of attacks that will be undetectable to the HIDS. For example, a HIDS will not have access to information pertaining to overlapping fragments, unless the underlying operating system records that information and presents it to the IDS software running above the networking stack. Benchmarking a HIDS against a NIDS will require that the attacks used for baseline measurement are all attacks that would actually affect the host. Indeed, to be detected, some of them would need to be carried through to completion (e.g.: a stack smashing attempt against web server software would have to be executed to test whether the web server was vulnerable). The author believes that the differences between HIDS and NIDS are so great – and complimentary – that a direct comparison is infeasible.[4] Network Node IDS (NNIDS) will have properties that are similar to those of a NIDS except for when it comes to performance. Since an NNIDS operates only on traffic directed toward the individual system, it does not need to concern itself with high-speed packet capture: the NNIDS layer deals with one system's worth of traffic.

Due to the different performance properties of NIDS, HIDS, and NNIDS, the author believes that, as IDS capabilities become more closely coupled, benchmarking will shift towards measuring the performance of an aggregate of IDS that may include multiple types of IDS, possibly even sourced from different vendors. At that point, IDS testing will either become primarily comparative and subjective, or will focus on counting attacks seen versus attacks sent, and the quality of the overall diagnostics returned in the IDS alerts.

Reordering, Reassembling, Defragmenting, State Tracking

Most NIDS vendors now claim that their NIDS perform "reassembly" – a convenient term for…. What? It turns out that there are 3 important reassembly-related activities that a NIDS could perform. To define these properties:

- **Defragmenting**: the process of combining multiple IP packet fragments into a single packet so that it can be checked for attacks.
- **Reordering**: the process of rearranging multiple IP packet fragments or TCP segments so that they are in the correct sequence.
- **Stream Reassembly**: the process of combining multiple TCP segments so that they represent a complete stream of data as the target system received it. Note that to perform reassembly the NIDS must perform defragmenting and reordering as well.
- **State Tracking**: the process of tracking TCP sequence numbers and TCP states so that the NIDS understands what traffic the target machine is treating as valid data.

It turns out that when many vendors claim they do "reassembly" what they mean is that they reassemble IP fragments, not that they perform complete TCP stream reassembly. In fact, it's only recently that many vendors have added even rudimentary defragmenting to their NIDS. Until just last year, two well-known NIDS did not perform fragment reassembly – so an attacker could use a simple tool like *fragrouter*[5] to pass an attack right under the nose of a watching NIDS without it seeing anything. Today, most commercial NIDS perform stream reassembly, but it is difficult to extract details from the vendors pertaining to the criteria used in reassembling TCP streams. What does the NIDS do if it receives packets in sequence 1,7,6,5,4,3,2 ? Will it keep

[4] See NFR IDS whitepaper "Coverage in Intrusion Detection Systems" by Marcus J. Ranum for an assessment of the overlap between HIDS and NIDS.

[5] Fragrouter takes IP packets and artificially breaks them into small fragments which are then shuffled out of order and transmitted. The receiving system's network stack correctly marshals the packet fragments and reassembles them into complete packets that contain the attack.

packet ‘1’ in memory long enough to reassemble them in the correct sequence, or will it receive packet ‘1’, followed by packet ‘7’ and discard packet ‘1’ because ‘7’ is too far out of sequence? If an attacker knows how large a reordering chain the NIDS maintains, he can fool the NIDS by sending traffic that is sufficiently out of order that the NIDS cannot reorder it into anything useful.

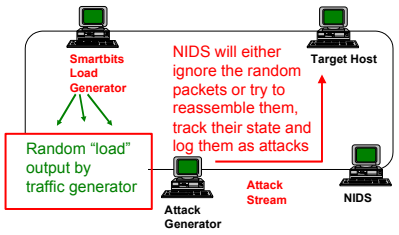
State tracking is also critical since it works effectively to reduce false positives. A NIDS that tracks TCP states will “know” what sessions the target “sees” and will not raise alerts on traffic that the target would discard as invalid. This is extremely useful in hardening the NIDS against storms of random traffic intended to confuse it. It also means that the NIDS will be able to keep accurate information of TCP session start-up times, client/server relationship, and amounts of data transferred in either direction. One early anti-IDS attack was to send a single spoofed FIN packet with a faked sequence number that was outside of the current stream’s TCP window. The target machine would receive the spoofed packet and discard it as spurious. If a watching NIDS saw the FIN packet and did not correctly track state it might conclude that the TCP stream had been shut down and might cease monitoring it, allowing an attacker to ghost past the IDS’ no longer watchful eye.

In one early IDS benchmark, NFR Security submitted an NIDS for comparison against a number of other systems. It turned out that the benchmark was focusing on packets/second performance and the designer of the benchmark had chosen to generate a load by replaying packet traffic that had been captured to a hard disk from a test network. When the NFR system was tested, it initially generated a large number of alerts that the other NIDS had failed to generate, and the benchmarker complained about the “large number of false positives.” It turned out that the benchmarker was replaying the same packet traffic repeatedly, some of it at the same time. The NFR NIDS TCP state tracking module was detecting the duplicated packets, which were outside of a valid TCP window, and generating alerts since the traffic appeared to be a replay attack or denial of service attack. This was actually correct behavior, but the fact that none of the other NIDS performed even rudimentary state tracking caused the NFR NID’s reaction to surprise the reviewer.

Simulated Traffic and the Load Generator Problem

Perhaps the worst mistake that can be made when benchmarking a NIDS is to use a synthetic load generator like a SmartBits. Tools like the SmartBits were designed for load-testing switches and routers – devices that don’t typically examine packet payloads. Any IDS that is worth the name will examine packet payloads as a matter of course. So, what happens when you connect a NIDS to a network with a SmartBits and tell the SmartBits to generate 100Mbit/sec of traffic? The answer is: “it depends on the NIDS.” Some NIDS only look at traffic that is destined for specific TCP ports. If the SmartBits is configured to generate pseudorandom TCP frames, a NIDS that is only looking for traffic on a handful of ports will see virtually no load whatsoever. It will, in fact, perform very well because it’s hardly looking at any of the traffic. If the NIDS is an NFR NIDS, which performs stream reassembly and state tracking, the NIDS will very quickly begin to generate a great deal of alerts, identifying the traffic from the load generator as a potential denial of service attack. It’s important to remember that in a normal network, pseudorandom traffic is never seen – hosts are not subjected to valid frames that do not belong to established TCP sessions unless they are being brought under a denial of service attack. For a benchmark to generate “background traffic” that is a denial of service attack is going to actually penalize the NIDS that “understand” the traffic they are monitoring and will make the NIDS that are less rigorous appear to perform better.

Figure 2: A load generator confuses the picture



As a rule, any NIDS test that uses a pseudorandom load generator is flawed.

Real networks do not have lots of random traffic on them, unless they are under attack or are severely broken.

Newer generation load generators are intended to support load-testing of web servers. These load generators can be programmed to connect to a web server and request URIs (random or pre-programmed) over and over to generate a specific load. This is somewhat more useful than the pseudorandom load generators but the way it will impact a NIDS will depend on the types of URIs that are being requested as well as the types of data returned from the server. Some NIDS actually parse the HTTP transaction and headers (which the load generator may or may not correctly generate) while others simply look for a pattern to match in the client side payload. A NIDS that is parsing the HTTP transaction and headers, a more rigorous approach, will actually be punished more severely than the NIDS that is performing simple pattern matching. Place the same 2 NIDS in a live network with real traffic and the NIDS that is doing full protocol parsing will generate far fewer false positives and will have less difficulty dealing with the non-synthetic load. The author has seen a number of NIDS that perform very well on networks with a synthetic load generator, but which fail miserably when confronted with massive amounts of traffic containing a mixture of real web traffic and real Email traffic. How can this be? The most likely reason is that those NIDS must be simply discarding traffic that does not look real – so they perform splendidly as long as they don't have to actually look at anything.

At NFR Security, we initially[6] tested our NIDS using packets captured from the "Capture the Flag" network at the DEFCON hackers' conference.[7] This traffic is, in terms of normal networks, highly unusual. It contains large amounts of vicious denial of service and scanning traffic. During one year, the DEFCON packet capture was heavily loaded with ARP spoofing packets – this is traffic that would virtually never be seen on a production network since even basic router filtering would eliminate it entirely. While it represents an interesting data-set to use as a torture test, it would be a bad idea to evaluate a product based on how it reacted to something so unusual.

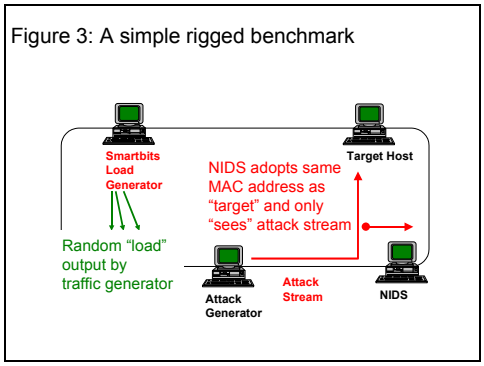
Another important factor to consider in a NIDS test is the presence or absence of firewalls and routers. In a real production network, there will almost certainly be a router between the "bad guys" and the segment where the NIDS resides. The presence of the router will become a wild card factor in a production network, especially if the router is being used to perform even basic filtering. Firewalls have an even greater "focusing" effect on a traffic mixture. A network that is not firewalled may carry a mixture of http, smtp, POP, smb, DCOM, napster, etc. traffic. A network that is heavily firewalled may transmit little more than http, ssh, and smtp traffic. If simulating a load for a NIDS, it is important to understand the application mix that the NIDS is most likely to have to deal with.

[6] Actually, we still do; it's a good test.
[7] The CTF network is an open network where hackers compete to break into and/or screw up each other's systems and selected sacrificial victims.

Doing it Utterly Incorrectly

The author was recently involved in analyzing some benchmark results from a NIDS product that had apparently achieved remarkable measures (packets/second) in detecting intrusions. There were many aspects of the test that were flawed, but most noteworthy was the fact that the IDS in question was apparently able to approach gigabit speeds. It did so when presented with random data because as part of its implementation it discarded data that was not directed toward a TCP/UDP port that the IDS tracks for signature checks. In other words, the benchmark measured *that the IDS was apparently able to **throw away** traffic at gigabit speeds*. "Apparently" is the correct word, here, because, since the IDS wasn't checking signatures, it's impossible to tell if it actually even collected the packets before discarding them. Because of how the benchmark test network was constructed, the IDS discarded the vast majority of the traffic it collected, and analyzed the small amount that remained for attacks – which proved fruitful since the only traffic that remained was the simulated attack stream! Unfortunately, the product was given an excellent review because the reviewers operating the benchmark were not skilled IDS practitioners.

Figure 3: A simple rigged benchmark



As a thought experiment, let's construct an even more egregious example of such a "benchmark" and how it might be implemented. Suppose we have a naïve simulation network where the target machine's hardware MAC address is easily discovered. Imagine that a network IDS is being tested, and, to crack the benchmark, we *simply tell the IDS that its packet collection interface has no IP address but has the **same** MAC address as the target machine*. Now, the IDS' packet collection interface card will automatically winnow out all packets that are not aimed at the target. Since most network interface cards don't even interrupt the host processor when they discard a packet, the benchmark network could be loaded down with a huge amount of randomized network traffic with no effect on the IDS whatsoever. In fact, this IDS could be implemented as the IDS from our first thought experiment – it simply beeps whenever it gets a packet of any type whatsoever. Fortunately, nobody has yet stooped so low as to pull a trick like this on an IDS benchmark, but it would be quite easy.[8]

Benchmarking it Right

So what is the right way to benchmark an IDS? If you're trying to publish results and want to have a repeatable result, be prepared to make a significant investment in infrastructure and time. The best approach would be to do a parallel-simultaneous test involving various functioning

[8] The most egregious benchmarking cheat I ever heard of was a compiler writer in 1986 that wrote a recognizer into the compiler/optimizer to detect a benchmark that was popular (sieve of eratosthenes, first 1000 primes) in Byte Magazine at the time. When the reviewers tested the compiler with their benchmark, it immediately spat out the correct answer without actually performing any computation! The author of the compiler rightly pointed out that this was a legitimate, though unusual optimization for a C compiler. He did, however, explain the trick to the baffled benchmarkers at the magazine and used the opportunity to educate them on testing methods and benchmark design.

production networks that experience predictable loads. This would entail setting up the IDS on a mirrored network, then turning the traffic on so that all the IDS would be "hit" simultaneously. Into this stream of traffic, inject some attacks using a predictable attack generation tool that generates real attacks against real target systems. Once the test has run, disconnect the mirrored network and compare results, then repeat as necessary on more loaded networks.

Since few networkers have access to repeatable and predictable high-volume networks at various load-levels, another approach is to generate synthetic loads with standardized payloads. In order to make sure that the loads represent real traffic, it is best to set up a system to perform packet collection of complete sessions between "background traffic" systems and attacker/defender systems. One advantage of this approach is that the captures that make up the loads are infinitely replayable and can be mixed and combined *as long as they are not duplicated*. Capture files of varying types of traffic can be built to characterize different sorts of networks (e.g.: a busy website, a big file server, a corporate LAN, a DMZ network), which can also be mixed and matched. Into this mix may be injected other mixes representing real attack traffic captured by launching a standardized set of attacks against a real target system and recording both sides of the traffic dialog. To replay the traffic, multiple computers may be needed, and/or systems with high speed disks may be needed if the simulation is of a high speed network. With current network architectures, this approach breaks down when simulating loads greater than full-duplex 100-base-T, since current gigabit architectures are largely switched and the switch may cause trouble if it sees duplicated addresses in the data-mix being sent through it. Also, inexpensive hard disks begin encountering bandwidth limitations at 20-40MB/second - which indicates that a single system will not effectively saturate a gigabit network with real traffic. At present, most of the techniques that can be used to simulate a loaded gigabit network are actually more painful than simply finding a busy network to use for "live" testing.

What does this mean for the customer who is interested in testing an IDS? Probably the most effective approach is to perform comparative tests using a "live" network (DMZ networks are good) and perhaps running some attack tools past the IDS to check their responses. There are a number of freely available tools for generating traffic that causes problems for various IDS products, including tools like *fragrouter*, *whisker*, and *stick*. For capturing packets and replaying them, many sites rely on *tcpdump* and *tcpreplay*. As of this writing, there are no standard available attack suites that can be used for testing. Currently, most IDS testers have to resort to taking a sample of attack tools from one of the many hacker sites and wrapping them with an automation layer to launch the attacks at a predictable rate.

Summary and Observations

We have presented some experiences in benchmarking IDS, with a focus on the painful aspects of poorly designed tests and their effects. The purpose of this paper is not to deter the reader from being interested in the results of IDS benchmarks or comparisons, but rather to equip them with a background regarding how difficult it is to actually perform such a test. Hopefully, this background will also enable you to detect the many flaws present in published articles and test lab results. As with every new technology, a certain amount of skepticism is essential – results that sound too good to be true probably are. Systems that boast amazing performance most likely only achieve those results under carefully constructed conditions. If you encounter a situation in which someone presents their results and you detect the signs of a rigged benchmark, we hope we've provided you with enough of a background that you can ask the tough questions and learn the truth. In the long run, the purpose of testing and benchmarking is to give you an idea how a system is likely to perform if you actually purchase it and install it. In the end, everything else is secondary.

As IDS become more coupled collections of NIDS, HIDS, NNIDS, and management/aggregation systems, artificial benchmarks regarding simplistic measures such as packet rates and throughput will become increasingly meaningless. Eventually, the "bottom line" performance measurement will revert to the real purpose of IDS: ability to detect intrusions. Secondary measurements will be false positive count and ability to successfully aggregate and

correlate IDS data from multiple sources. In order to measure such large-scale and complete systems, testers will move away from the packet-oriented approach into a more system-oriented approach that focuses on sending predictable numbers of attacks and measuring the results returned by the IDS entity as a whole. There will still be plenty of room for qualitative analysis but it will mostly relate to the quality of the diagnosis in alerts and their relevance to the installed systems on the network.