

What is "Deep Inspection"?

Introduction

Computer security technology is still in its infancy. Technologies such as firewalls, antivirus, and IDS have migrated from research labs into production networks, and have become required mainstays both as essential defenses and as legally mandated compliance systems. Computer security systems are complex devices that need to meet a variety of conflicting goals: high performance, fault tolerance, easy administration – and rigorous security processing. Some vendors have staked their claim based on speed, others on cost, and still others on the defensive posture and security of their products. Unfortunately, it's *extremely* difficult for the customer to sort through marketing fluff and dubious benchmarks, to determine which products actually work and which merely *appear* to work. Few customers are sufficiently sophisticated or willing to take the time to do their own testing and most are forced to rely on published results from trade magazines, recommendations from consultants, or industry analysts. Sadly, few of the trade magazines or analysts have the sophistication or time to perform adequate testing, either.

The author's experience indicates that large numbers of products on the market have excellent and attractive user interfaces, good performance, reasonable costs, rave reviews from loyal fans – and have taken shortcuts in their design that make them significantly less secure than other alternatives. This is not a recent development; it dates back to the early days of the firewall "market." In this article, we examine the evolution of packet filtering firewalls and their current incarnation as "Deep Inspection" firewalls. We compare the fundamental design philosophies of packet filtering firewalls with proxy gateways, and will conclude with a few historical observations regarding the relative effectiveness of conservative design philosophies when compared to their less-rigorous counterparts.

Early Firewalls, Packet Filtering Firewalls and "Stateful Firewalls"

The first firewalls were based on either a proxy design or a simple packet filtering ruleset. The proxy firewall operates by interposing itself in the middle of the application protocol and interpreting it while applying security controls to the application commands and data, where appropriate. The original value proposition of a proxy firewall is that the proxy is essentially a security-oriented reference implementation of the application protocol – in some cases omitting dangerous operations entirely, or providing additional controls on certain security-critical commands. Proxies have always been considered a conservative security design because the proxy reduces the likelihood of protocol backdoors or side-effects since the proxy's designer is

effectively performing a security assessment of the application protocol's features prior to implementing them. Early packet filter firewalls implemented a simple policy-table lookup based on { source-ip, destination-ip, source-port, destination-port, SYN-seen yes/no } permit or deny. Consequently, packet filters were extremely fast since they did very little computation. They were also extremely easy to implement since they required virtually no security expertise. The simple compute requirements of packet filters, and the fact that they required no security knowledge-base, made them easy to implement in silicon so they quickly became a feature of most routers. From the beginning, proxy firewalls were recognized as being more secure, because they effectively are implementing a *correctness check* upon the application protocols they gateway. This is still an important property of proxy firewalls. For example, when the author first implemented the FTP proxy in the DEC SEAL firewall, he simply left out unused FTP protocol commands that allowed users to issue remote commands to the FTP server. Years later, when hackers discovered those commands and attempted to exploit them, they simply did not work against proxy-protected networks because the proxy refused to gateway the command through to the target. Sites behind packet filtering firewalls were vulnerable, if the reachable systems behind the firewall were themselves vulnerable.

In 1993, "stateful" firewalls appeared on the market. The first popular stateful firewall, Checkpoint's Firewall-1, implemented a simple connection-origin table that tracked whether a connection had originated behind the firewall and permitted response packets for that connection. A layer-7 hook to parse FTP PORT commands and update the state table allowed FTP to work transparently through the firewall. Subsequent versions of the stateful firewall added TCP sequence number interpretation, and DNS query/response matching to ensure that return packets were only allowed in response to queries that had originated from the inside. It is important to note that stateful firewalls added these features *to overcome vulnerabilities in their design* – attacks such as TCP RST flood attacks and DNS cache poisoning. Proxy firewalls never had these kinds of vulnerabilities. Stateful firewalls have continued to evolve; often in response to new types of hacking techniques as they have been discovered. Proxy firewalls have evolved, as well, but mostly in response to ever-higher requirements for performance and transparency.

Near-Term History of Intrusion Detection, Intrusion Prevention, and Firewalls

Intrusion detection technologies (IDS) have enjoyed a spectacular rise and fall. Since Dorothy Denning first proposed the concept in 1986, there were relatively few products until 1997, when a number of vendors began offering commercial network monitoring IDS. The IDS market grew rapidly through 2002, when Gartner researchers announced "IDS is the 'pet rock' of computer security" and recommended their customers focus instead on the up-and-coming new technology termed "Intrusion Prevention" (IPS). According to Gartner analyst Richard Stiennon, IDS was a failure for two reasons. First off, IDS generated too many "false positives" – alerts in which the IDS mistook legitimate traffic for an attack. Secondly, IDS didn't actually

do anything but generate an alert when the attack was seen. Stiennon rightly pointed out that, if you can correctly detect an attack, it would be nice to *prevent* it. By the end of 2003, virtually every IDS vendor was re-branding their product as an IPS, by adding in-line blocking modes to their sensors. Initially, virtually no customers used the in-line blocking modes of their IPS, because they were afraid of traffic interruptions.

Some of the first IPS on the market were not effective security devices: they used simple signature-matching techniques and "knew" how to recognize and block only a few dozen worms and popular attacks. They did not sell particularly well until the massive worm outbreaks of 2004: corporations were desperate to contain worms in their networks, and worms are sufficiently easy to recognize that the IPS did a decent job blocking worm traffic on the network. IPS began to sell. The feature-similarity between IPS, IDS, and firewalls as well as single-purpose spam and anti-virus gateways, has triggered a move to consolidate these technologies into a single platform. A number of vendors have embarked on a course of technology acquisition, with the intent of embedding IPS/IDS/Firewall/AV/VPN termination capabilities into switching platforms. These combined systems offer an attractive feature-set, high performance, and a good price point when compared to having to integrate a series of separate products. The strengths and weaknesses of such systems will depend on the quality of the underlying detection and firewall engines and its IDS knowledge-base.

Enter "Deep Inspection"

According to David Flynn, NetScreen's Vice President of Marketing, a survey of 1000 network managers indicated their biggest security concern was "depth of protection" against worms, trojans, email viruses, and exploits against software vulnerabilities. Flynn is quoted as noting that these types of attacks "can traverse a traditional stateful firewall even if the firewall is deployed and working as it should be." Both NetScreen and CheckPoint have bolstered their stateful firewalls by adding application-oriented checking logic into their processing modules – essentially merging IDS signatures into the firewall traffic-processing engines of their products. What is "Deep Inspection"(DI) anyhow? According to NetScreen, DI firewalls "use an Attack Object Database to store protocol anomalies and attack patterns (sometimes referred to as signatures), grouping them by protocol and security level (severity)" Packet processing is typically described as "performing application level checks as well as stateful inspection." In other words, a "Deep Inspection" is just a catchy marketing term for a stateful packet inspection firewall with some IDS signatures and some application protocol anomaly detection rules. It is important to understand that a "Deep Inspection" firewall is going to provide all of the protections of a stateful firewall, as well as whatever signatures are loaded into it.

In the case of a NetScreen (circa 2003) the firewall "is designed to provide application layer protection for the most prevalent internet-facing protocols, such as HTTP, SMTP, IMAP, POP, FTP, and DNS." It is ironic to remember that one of the early complaints about proxy firewalls was that they only supported a small set of application protocols – what good is a DI firewall

that only does DI on 6 of the "most prevalent" internet-facing protocols? That may be "*deep*" inspection but it's certainly not very *broad*. When the NetScreen product was initially released, it included signatures to detect "over 250 application attacks." As far as an IDS signature set, this does not compare favorably to an IDS such as the open-source Snort, which at that time had signatures to detect over 3,000 attacks. Much like the early versions of the Snort IDS, NetScreen's DI engine uses regular expression pattern matching to specify its signatures – a technique that most serious IDS product designers stopped using years ago because it is too simplistic to represent complex network/application states effectively. Regular expressions, however, are easily processed in silicon, using off-the-shelf regex engines that provide hardware speedups for matching, so it makes sense for a switch-based DI firewall to rely upon them.

Designers of DI firewalls will probably not be able to provide sophisticated IDS engines in their products because of the performance overhead necessary to execute complex signature analysis and state tracking. The history of the IDS market shows that IDS has always been a performance-sensitive proposition – which is why most line speed IDS appliances are using multiprocessor main boards to keep up with the load. The author suspects that, unlike IDS, the DI firewalls will only be looking for a small subset of the attacks that are known at any given time, due to performance concerns. Even the "only look for a few attacks" strategy will eventually fail; Gartner reported that last year that 70 new vulnerabilities were identified per week, or over 3,500 per year. If even 10% of those are vulnerabilities that replicate across network data, the firewalls are going to very quickly have to handle very large signature-sets. Hardware acceleration will help somewhat, but typical hardware implementations in the fast-path of a switch are going to have built-in limitations on how many signatures they can effectively load at one time. DI firewalls will eventually have to deal with all of the complexity issues that caused Gartner to declare commercial IDS a "dead end."

Protocol Anomaly Detection Meets The Proxy

Protocol anomaly detection is an IDS technique in which rules are written that track the states of an application protocol and look for deviations from a proper state. A simple example of a protocol anomaly detection rule for FTP might be "if the user name given during the login process is longer than 45 characters, alert that as suspicious." It is possible to very tightly track the process of a protocol, at a cost of increased computing and memory use. For example, the NFR IDS protocol anomaly detector for SMTP tracked the entire SMTP command series from start to finish and would generate alerts if the client issued a "MSG From:" command before a "RCPT To:" command, or issued a "HELP" command, etc. Such tight protocol tracking is extremely powerful, because normal applications *always* send a message using the exact same sequence of steps but human hackers sometimes get the order wrong or attempt to trigger errors by issuing commands deliberately out of sequence. Protocol anomaly detection is a very powerful technique that is used to great effect in all commercial IDS and most DI firewalls. For detection accuracy, protocol anomaly detection *pales in comparison* to the

processing that is performed during the normal functioning of a proxy firewall. This is because the proxy, as part of its normal operation *executes* the protocol. Instead of having to follow along and try to figure out what the application protocol is doing, the proxy *is* the application protocol: protocol anomalies represent error conditions that the proxy detects. When the author implemented the first proxy firewall, the FTP proxy logged and error and shut down if, at any time, it received commands that were out of sequence or were incorrectly formatted.

Another topic of debate in the IDS and DI firewall community is the question of "detecting the exploit" or "detecting the vulnerability." When "detecting the exploit" the signature-writer develops a signature that will identify the particular attack tool. For example, the TESO mass-rooter for FTP daemons uses a specific set of buffer overruns against a variety of FTP servers. Coding to detect TESO, the signature-writer would write a minimal signature to identify the first steps of a TESO probe and generate an alert. To "detect the vulnerability" the signature writer would research the entire set of vulnerabilities that particular FTP servers might have, and then would develop a signature set that detected any one of them being exploited. Detecting the vulnerability takes considerably more research since it involves a larger number of signatures to develop. For the proxy firewall developer, this is a complete non-issue. If a new vulnerability is identified in an application, the proxy-writer assesses whether the attack vector is "legal" in the protocol, and, if it is not, the proxy wouldn't have let it through in the first place. In the case where the attack vector is encompassed in the protocol, the proxy can be updated with a protocol-specific signature at the exact point of the protocol processing where it's appropriate. In the long run, this approach is much more efficient and takes less work – both in terms of effort to keep the security knowledge-base up to date, but also in terms of processing speed. The proxy only needs to execute protocol-specific checks at the exact part of the protocol where they are needed, whereas a regular-expression based signature engine must compare each expression against each input as it arrives.

Some Philosophy About Security Design

Fundamentally, the tension between proxy firewalls and stateful filter + signature firewalls is a reflection of the deepest philosophic division that in security. It is a division that goes back to what we used to call "stance" back in the early days of firewalls:

- That which is not expressly prohibited is permitted (default permit)
- That which is not expressly permitted is prohibited (default deny)

The default deny stance is the default policy that virtually every credible firewall product executes unless told otherwise. In security, however, there are many places where we do not follow the rule of default deny. For example, antivirus products implement a "default permit" – "if you don't know it's a virus, then it's OK to run it." IPS implement default permit – "if it doesn't look like an attack, permit it." Fundamentally, default permit is not good security. A proxy firewall implements default deny. Unless the application that is talking to the proxy complies

exactly with the protocol as executed by the proxy, the traffic is not going to go through. It's not perfect, of course, since there might be an attack that is valid within the protocol, but is still damaging. That is why most proxies also add additional application specific signatures.

For example, the TIS Firewall Toolkit's HTTP proxy not only mediated the entire HTTP transaction it performed URL sanitization and did not permit unusual characters in URLs. Several of the authors' friends who still use components of the firewall toolkit were highly amused when a piece of software that was coded in 1992 successfully defeated worms in 2002 – 10 years before the worms had been written. Similarly, the DEC SEAL's FTP proxy from 1990 defeated FTP bounce scans when the hacker community discovered them in 1995. Peter Tippett, the author of Norton Antivirus and founder of ICSA/TruSecure once famously commented "The current philosophy of antivirus is to know the 50,000 bad things that can happen to you. It's much more sensible to enumerate the 50 things that you want to allow to happen, but the industry just isn't ready to think that way, yet." Perhaps that is true in the antivirus arena, but the proxy firewall's approach has always been:

- Nothing gets through unless it looks right

Rather than:

- Everything gets through as long as it doesn't look bad

Security products that adopt the default permit philosophy will always be fundamentally at risk from new attacks. A signature-based DI firewall can only block attacks that are known by its signature set. When a new worm breaks out that takes advantage of a never-before-seen attack, the signature-based DI firewall *will* let it through. Once the signature set is updated, the firewall will be able to block the attack, however, the time between the release of the attack and the development of the signature might be a matter of hours or even days. While that window of vulnerability is open, the customer will have no alternative but to either patch host software or disable that service.

Historical Performance of Security Solutions

The history of computer security shows that many customers do not want to take the time to fully understand what they are buying. In the mid 1990's the author was selling proxy firewall products that had a superlative history of resisting attack; yet the market leading products were simplistic "stateful" packet filters that were sold based on the fact that they were faster, cheaper, and more forgiving. Put differently: they didn't perform as rigorous checks, so they could be fast. They were easier to code, so they were cheaper. They were more forgiving, because they were more permissive. There is ample anecdotal evidence that default deny security survives better in the long term than default permit. There is ample anecdotal evidence that default deny security is less susceptible to nasty surprises, and requires less

patching. Systems that are based on the model "recognize a vulnerability and block it" inherently require "upgrade to fix the vulnerability of the week."

Customers need to understand their objectives and requirements, so they can best select technology that facilitates their mission. There are situations and circumstances where less-rigorous security systems can get the job done safely. The author believes, however, that few vendors take the time to explain the design tradeoffs inherent in their products to their prospective customers. With security being such a hot topic, it's hard for a customer to make sense of a plethora of products, all of which are asserted to be "secure." Analogies are dangerous, but choosing a security product is similar in many ways to buying a car: the customer needs to have a good idea of what kind of on-road properties are needed. Does it need to be economical and corner well, or does it need to be able to tow a 6 horse trailer safely? Or, perhaps it needs treads and 3" thick armor. What is frustrating to the author is that in the computer security space, there are some vendors asserting that what they offer is a general-purpose high fuel-efficiency sports-car main battle tank. Can you believe that?

mjr.

The speaker break room, Interop (Mandalay Bay Hotel) and McCarran Airport Gate 33, Las Vegas, May 6, 2005

Appendix 1: Summary Matrixes

Table 1: Pros and Cons of ASIC-based platforms V. General Purpose mainboard

<p>Packet Processing speed</p>	<p>Advantage: ASIC Platform</p> <p>Notes: Generally ASIC-based systems take advantage of off-the-shelf silicon for "fast-path" packet processing. Fast-path processing represents "best case" throughput; real throughput is often dramatically slower when running application logic (such as security processing beyond simple packet-filtering and state table updates)</p>
--------------------------------	---

<p>Matching performance</p>	<p>Advantage: ASIC Platform, up to a point</p> <p>Notes: off-the-shelf regular expression processing silicon is available and offers dramatic performance speed-ups for simple pattern matching. With most such hardware implementations the pattern-matching engine provides acceleration for a limited number of expressions in parallel; once this number is exceeded performance drops to un-accelerated levels because the higher-level controls need to "swap" in and out pattern-match sets. When this occurs it is usually no longer possible for the matching engine to run in the "fast path" so not only does matching performance suffer, packet processing performance can be degraded as well.</p> <p>Recommendation: Always evaluate such products using a ruleset that is slightly larger than the expected set of real-world signatures. A realistic test of an ASIC-based packet searcher should include ~3000 rules – approximately ½ the rules in the current Snort IDS signature set. Be extremely wary of performance figures in which the ruleset size may have been adjusted to fit within an optimal silicon space for the processor.</p>
<p>Signature set size</p>	<p>Advantage: General-purpose mainboard</p> <p>Notes: Devices based on general-purpose mainboards are generally more tolerant of larger rulesets since they can be expanded with additional RAM as necessary. General-purpose operating systems usually include virtual memory management algorithms and paging kernels, which allow virtual extremely large datasets and rulesets to be resident without performance impact.</p>
<p>Signature update</p>	<p>Advantage: General-purpose mainboard</p> <p>Notes: Devices based on general-purpose mainboards usually can take advantages of the presence of hard disk interfaces or removable media and are not space-constrained when storing administrative configuration data.</p>
<p>Search Space size</p>	<p>Advantage: General-purpose mainboard</p> <p>Notes: ASIC-based systems generally are designed without a hard-disk-backed virtual memory architecture, and consequently are unable to do long-term storage of information relevant to detection signature logic. This sometimes reveals itself in the form of hard capacity limits in ASIC-based devices that are not present in general purpose O/S-based systems. Such capacity limits might present themselves as hard limits on number of simultaneous sessions that can be tracked, maximum number of out-of-sequence packets that will be correctly re-sequenced, etc.</p>
<p>Storage</p>	<p>Advantage: General-purpose mainboard</p> <p>Notes: Most security products based on a general-purpose mainboard take advantage of the presence of a hard disk to store log data, event records, and security alert information. Persistent hard disk storage prevents loss of event/alert information if a network is partitioned temporarily.</p>

<p>Software Update</p>	<p>Advantage: General-purpose mainboard</p> <p>Notes: General purpose operating systems usually include a notion of software component architectures, allowing easy update of program logic, installation of patches, and operating system updates.</p>
<p>Hardware Update</p>	<p>Advantage: General-purpose mainboard</p> <p>Notes: The upgrade path for most ASIC-based systems consists of "buy a new one" while general-purpose mainboard based systems can be component-upgraded with additional RAM, hard disk, or a faster processor. Because ASIC-based systems can seldom be component-upgraded customers may encounter a "hard limit" in which one component of the system reaches capacity and the entire system must be replaced. I.e.: "the signature that broke the camel's back."</p>

Summary observations: ASIC-based solutions are appealing to the customer for whom raw speed is a paramount consideration. Generally, where there is a choice between "doing things fast" and "doing things securely" the design decision of ASIC-based products is to "do things fast." This, necessarily, affects the system's quality from a security standpoint. The author's experience is that custom hardware-based security products tend to look more like "a switch with a bit of security bolted on" rather than "a security product that is really fast."

Table 2: Pros and Cons of Signature Security Processing V Proxy Security

<p>Signature count</p>	<p>Advantage: Signature-based system</p> <p>Notes: Unlike signature-based systems, proxies do not derive their security through enumeration of attacks or vulnerabilities. Many proxy firewalls support signature/expert rule detection <i>in addition</i> to the proxy's correctness check. Signature counts are, in other words, not applicable to a proxy firewall. However, many customers attempt to make the comparison.</p>
------------------------	---

<p>Protecting Against Unknown Attacks</p>	<p>Advantage: Proxy</p> <p>Notes: Signature-based systems <i>utterly</i> fail to <i>prevent</i> unknown attacks. They may prevent new versions of previously known attacks, or new attacks that happen to trigger an existing signature. Signature-based IDS may detect protocol anomalies, but signature-based firewalls (AKA "deep inspection" firewalls) do not automatically block all the things they prevent; they generally only block things they can be <i>relatively certain</i> represent hostile traffic.</p> <p>Observation: It is important for customers to understand the difference between "signatures the signature-based firewall <i>detected</i>" and "signatures the firewall was <i>confident enough</i> to block traffic based upon." For example, when the Intruvert IPS first came out, it shipped with only 50 signatures (!) of which <i>none</i> were configured to block traffic. The product's designers were so concerned that false positives would cause the system to automatically shut down legitimate traffic that they only eventually enabled automatic blocking on fewer than 1 dozen high profile and easy-to-detect threats.</p>
<p>Performance</p>	<p>Advantage: neither</p> <p>Notes: As signature-based systems begin to load up on signatures their performance can rapidly degrade. Proxies do not have signature loading properties, so it's difficult to make a comparison.</p> <p>Recommendation: Customers should perform carefully constructed head-to-head comparisons based on real loads and realistic, useful signature set-loads.</p>
<p>Service Support</p>	<p>Advantage: neither</p> <p>Notes: One of the historical complaints against proxy firewalls is that they do not support as wide a set of services as a packet screening "stateful" firewall. With the addition of signatures to stateful firewalls, the stateful firewalls do not have signatures that perform meaningful security on any but a handful of protocols (e.g.: HTTP, FTP, DNS) The same complaint that used to be levelled against proxy firewalls now applies to "Deep Inspection" firewalls.</p>

Summary observations: Proxy firewall technologies have proven time and again to be more secure than "stateful" firewalls and will prove to be more secure than "deep inspection" firewalls. The main comparison point between stateful firewalls and proxy firewalls has typically been on *performance*, which is an orthogonal property to *security*. High-performance proxy firewalls are available, and some are capable of easily handling gigabit-level traffic. Customers need to be realistic about whether performance is really an issue in the expected deployment, and should be prepared to do operational testing. The author has seen many customers cheerfully settle for inferior security because of "performance concerns" that did not actually apply.