



White Paper

Dude!

You Say I Need an Application Layer Firewall?!

Authored by Marcus Ranum

Table of Contents

Introduction	2
The Premise of a Firewall.....	2
Proxy Firewalls: Some History	3
Network Layer Firewalls	5
Looking OK	5
Enter IPS	6
So Why Isn't It Working?.....	7
Some Predictions	8
Summary.....	9

About the Author:

Marcus J. Ranum is a world-renowned expert on security system design and implementation. He is recognized as an early innovator in firewall technology, and the implementor of the first commercial firewall product. Since late the 1980's, he has designed a number of groundbreaking security products include the DEC SEAL, the TIS firewall toolkit, the Gauntlet firewall, and NFR's Network Flight Recorder intrusion detection system. He has been involved in every level of operations of a security product business, from developer, to founder and CEO of NFR. Marcus has served as a consultant to many FORTUNE 500 firms and national governments, as well as serving as a guest lecturer and instructor at numerous high-tech conferences. In 2001, he was awarded the TISC "Clue" award for service to the security community, and the ISSA Lifetime Achievement Award. Marcus is Chief Of Security for Tenable Security, Inc., where he is responsible for research in open source logging tools, and product training. He serves as a technology advisor to a number of start-ups, established concerns, and venture capitol groups.

Sponsored by:

SECURE
COMPUTING



www.securecomputing.com

Introduction

Internet firewalls have been a popular tool for security practitioners since the early 1990's. Today, they are considered a mandatory component of any industry or government network. Unfortunately, many consumers of these fundamental networking tools buy and rely on them without understanding that there can be dramatic differences between firewalls that are manufactured by competing security practitioners and their unique engineering teams. Firewall products that are brought to market based on significantly different technical design philosophies and different go-to-market strategies quite naturally introduce consumer trade-offs that should be weighed when making buying decisions. Certain firewall design trade-offs, for example, favor security over convenience, and certain firewall go-to-market strategies favor platform performance over security.

As a result of robust global market competition in the firewall space, and the growing demand for ever-improving perimeter security, software and appliance products sold as firewalls have evolved into a collection of products falling along a broad spectrum of features, benefits and, in some cases, pitfalls to take note of. From the author's point of view, there is a clear and easily observable divide in firewall types available for purchase today when they are sorted into two simple categories based upon the manufacturer's security design objectives. There are firewall product designs ranging from highly conservative and security-focused architectures, to designs that are highly appealing to the broad market because they offer good security "theater" in the look, feel, and marketing story but under the surface offer only simple security controls. Not all firewalls are created equal...on purpose.

In this paper, we will describe the evolution of firewalls from the standpoint of the *controls that they apply on data*, and we will explain why the currently accepted "state-of-the-art" firewall really represents a step backwards in most cases for securing perimeters. To many, this may seem contradictory. **One only need consider the high growth rate of the installed-base of firewalls while simultaneously taking note of the dramatic increase in the number of networks being penetrated, to realize that something is going wrong in the world of perimeter security.**

The Premise of a Firewall

In 1989, Steve Bellovin, one of AT&T Bell Labs' early innovators in the field of firewalls, described a firewall as, "a device separating *them* and *us* for any value of *them*." Much has been made of the importance of a firewall's *policy*—the rules by which you instruct a firewall what to permit or deny—but the firewall's policy is only approximately half of the story. To use an analogy, the firewall is a guard at the perimeter of your network which you have instructed whom to shoot and whom to allow past. Obviously, it's important that your guard remembers the rules you gave him (the policy) but it's equally important that your guard uses reliable means to determine if the people he's letting back and forth are truly who they appear to be, and in addition that they are not carrying anything dangerous in their pockets. To stretch the analogy to near the breaking-point, firewalls have to deal with a more post 9/11-style environment in which even highly recognized, authorized individuals should be thoroughly checked for dangerous weapons. In fact, that is exactly the case with firewalls. Firewalls can no longer just concern themselves with establishing and then expediting traffic flows because the majority of damaging attacks being used on the Internet today are contained within data payloads passing to or from applications across an *authorized* connection. Our border guards now have to rapidly and accurately strip-search everyone passing by *without* slowing them down or inconveniencing them. As if that isn't bad enough: imagine that the people passing through security are important members of the legislature. If they are incorrectly stopped or delayed, they will raise the hue and cry. Yet, if they manage to sneak something hazardous through as a test of the security process they will gleefully turn it over to the press and demand our resignation!

In the early days, one popular firewall approach was to try to deal with the security guard problem almost exclusively at the IP packet or network layer. By taking advantage of a side-effect of the TCP protocol, these early network-layer firewalls offered good performance because they didn't have to do that much security work when examining TCP packets. The early firewalls had only to look '*just deep enough*' to see if they had the ACK flag set (which normally indicates that a packet belongs to a properly established TCP data stream). These early firewalls did their policy checking largely on the origin/destination of the first packet—the SYN (or connection request) packet. Such simple firewall packet checks had the advantage of being extremely quick; and they had an easy-to-build advantage as well because the firewall didn't



have to maintain any state information about any of the connections that it allowed through. These early firewalls doing simple ACK screening could handle an arbitrarily large amount of simultaneous connections because each packet was assessed as a separate event. This approach, however, proved too simple to deal with protocols like FTP which is a bi-directional client server protocol and so performs callback connections, and this approach offered absolutely no awareness or control over the actual data passing over authorized connections.

Because the early network-layer firewalls simply looked for a few values inside each packet header, they didn't have to alter the packets at all, which meant that the data entering and leaving the network remained unaltered. In fact, the data was almost completely unexamined except for a handful of fields in the packet header: source, destination, source-port, destination port, and ACK flag. At the time, protocol-over-protocol tunneling was virtually unknown, so the ease with which one could send TELNET traffic through an open SMTP connection, as one example, came as a surprise to a number of sites. In 1991 the author demonstrated a shell script implementing TCP/IP encapsulation over email, which dramatically illustrated an important point: **you cannot meaningfully secure traffic without looking at it.**

Proxy Firewalls: Some History

The other early 1990's approach to building firewalls relied on what were known as "application proxies" or "application gateways." The idea of a proxy firewall was to act as a "man in the middle" between two networks, forcing all traffic to stop, be checked against prior traffic and policy, be taken apart and searched to an appropriate degree, then reconstructed and allowed to continue if it looked OK. "Looking OK," is an extremely important issue that we'll touch on in a moment. Let's imagine a proxy firewall is a border guard that stops every car that drives up to his gate and rigorously checks the driver's passport, reads his or her visas, examines what the driver has declared on his customs cargo manifest, and has him unload his goods and step out of the car. Once the driver is standing there with his goods in his arms, the car is crushed (just in case something was hidden in it that was not on the manifest) and the driver is sent on his way in a brand-new, shiny car of an approved make and model.

To extend the car/border guard analogy further, imagine that each type of traffic coming to the border crossing was sent to a particular crossing guard that specialized in that kind of cargo. So, drivers who were bringing books were sent to a special line of librarians that might check the books against valid ISBN numbers and drivers bringing dogs would be sent to the line of canine experts, etc. The first commercial firewall, the DEC SEAL, was based on this kind of proxy design and included security proxies for the following protocols: FTP, TELNET, SMTP, and DNS. Granted, it was a firewall that didn't know how to do many things; but the things that it did, it did well. By restricting the number of lines at the border crossing, the firewall was able to perform more detailed examination of the traffic going through it. By being closely involved with how the data went back and forth (i.e.: crushing all the cars, unilaterally, and making it impossible to slip something past that wasn't in the manifest) the firewall reduced the likelihood that someone could hide something unexpected in their trunk. Well, in fact, they could, but it'd get crushed.

Another overlooked facet of the proxy firewall model is what is known as "protocol minimization." A networked application like SMTP, implements a set of commands that are passed back and forth across a network connection, to carry out some kind of transaction. For example, a minimal SMTP transaction for transmitting a message across the Internet consists of the following commands, issued by the client, in this order:

1. Client connects to server
2. Server sends client a "welcome" identifier such as: "220 server-name ..."
3. Client greets the server with: "HELO client-name"
4. Server replies "250-server-name blah blah..."
5. Client sends "MAIL From: senders-email-address"
6. Server replies "250 senders-email-address... Sender ok"
7. Client sends "RCPT To: recipients-email-address"
8. Server replies "250 recipients-email-address... Recipient ok"



9. Client sends "DATA"
10. Server replies "354 Start mail input; end with ."
11. Client sends the mail message including headers and then: "."
12. Server replies "250 message received and queued"
13. Client sends "QUIT"
14. Server replies "221 Goodbye"
15. Both sides disconnect and the transaction is completed

The only parts of the servers' responses that actually mean anything are the numerical codes at the beginning, and the rest of the line can be ignored. But what's interesting is the client side of the transaction: the client issues a total of five commands to send the message. Early versions of the SMTP server, Sendmail, even in 1989, implemented nearly a dozen commands, including HELP, VRFY, and WIZ—each of which, at one time or another, was a vector for a nasty security hole. The designers of early proxy firewalls made the observation that since only five commands were actually needed, there was no point in offering the complete set to the Internet. Many of the first generation proxy firewalls improved the security processing for different protocols by either placing extra checks around parts of the protocol, or by simply not implementing additional commands. For example, one firewall that the author developed would simply reply with an innocuous "OK!" message to any command a hacker attempted to issue, while simultaneously logging the attempt as a potential threat. Other additional checks used a technique called "boxing in" for certain commands. In the transaction above at point #5, suppose the client sent "MAIL From: ..." followed by five megabytes of data. If the server software was not prepared and programmed to deal with that much, it might crash or be vulnerable to attack using a technique called "stack smashing." Most of the proxy firewalls, however, included code to "box in" expected replies at various points during a transaction; the proxy might be coded with the assumption that "under fifty kilobytes of data is acceptable on a MAIL From: line." The "box" around fifty kilobytes allows normally behaving applications to function, but as soon as no longer fits in the "box" it is detected and halted. We will see this technique of "boxing in" a protocol again, later, when we look at intrusion detection and "protocol anomaly detection." From a programming standpoint, the application aware security proxy acts as a layer of error checking, detection, and blocking, as well as a narrowing of the possible lines of attack against the target network.

The following is a good example of the power of an application-aware proxy to stop attacks it has never seen before. The TIS Firewall Toolkit's HTTP proxy (a design going back into the early 1990s) not only mediated the entire HTTP transaction it also 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 proxy 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 anti-virus 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."

When comparing an application-layer proxy firewall to a network-layer firewall, it is also crucial to understand that the application proxy is the only system that communicates directly with the untrusted network. Network-layer firewalls, on the other hand, allow the external, untrusted client to have a direct packet flow between them and the internal application server. This key difference means that the only way an attack is going to get from the outside client to the internal application server through an application-layer proxy firewall is if there is an error in the proxy's logic, or the attacker is somehow able to invent a new attack that fits within the proxy's idea of 'acceptable traffic' for that protocol. Proxies, therefore, act *very* differently from the way a network-layer firewall operates at a fundamental level. In the case of a network-layer firewall, once the source/destination/origin has been deemed acceptable, the external client traffic is allowed straight through, unmodified, and passed directly to the internal system behind the firewall. So, while the proxy firewalls were forcing all the data coming through them to flow through a code-path of strict compliance checks, the network-layer firewalls were improving their performance and flexibility. Not surprisingly, proxy firewalls came under attack as being "slow."



Network Layer Firewalls

At the same time as proxy firewalls were coming under attack for being “slow,” network-layer firewalls were coming under attack for being too *simplistic* in their security posture. Simply checking for an ACK bit in a packet was not good enough, so the designers of network-layer firewalls added some additional state-tracking to their products, to make them harder to fool and more transparent. “Stateful packet inspection,” was the term coined to describe this next generation of network-layer firewalls.

What does “stateful inspection” mean? Generally, stateful packet inspection firewalls add a large lookup table to maintain connection information; including which interface the connection originated on (to prevent injection of spoofed packets), simulated UDP state tracking, and TCP sequence number tracking. The UDP state tracking simulation watches for a UDP packet from external machine A to internal machine B and allows a response from internal B to external A within a certain time-window, assuming the overall policy permits UDP on that port. TCP sequence number tracking (essential to prevent certain types of denial of service attacks and guessing attacks) was added relatively late in most network-layer firewalls. By the late 1990’s, most network-layer firewalls handled *network layer* attacks as effectively as proxy firewalls, but still fell *far short* in terms of handling application layer attacks.

In the market, network-layer firewalls have proven consistently to be more popular because of the perception that they are faster, and because they can quickly open connections to service just about any new application flow without waiting for any feature development from the firewall vendor. When a proxy firewall adds security filtering for a new application protocol, someone in the firewall manufacturer’s engineering team needs to analyze and understand the protocol, look for possible ways it could be exploited, and design the “boxes” to code around the various protocol commands before they actually build the new proxy. However, the *value* of this design exercise can be considerable. Let me provide an example to drive this important point home. In 1990, the author, as part of implementing a proxy for the FTP protocol, discovered a fundamental flaw in the way the PORT command operated. As a result, the finished proxy knew how to protect against it and extended that protection to all the vulnerable servers behind it. In the case of a network-layer firewall, a “fix” for this fundamental flaw in the FTP protocol was simply not possible since the flaw was application-layer. So the only way for users of network-layer firewalls to protect themselves against the attack was to replace the FTP daemon on every server behind the firewall that was Internet-accessible. From a purely practical standpoint, however, the proxy firewall’s greatest value (the additional application-layer checks) was also its biggest liability, since it took time to do the detailed protocol analysis and to then develop a proxy to carry the data correctly. This gave the network-layer firewalls a huge market advantage, since they could quickly and easily “support” new applications. Why do I put “support” in quotations? Because the way the network-layer firewalls “supported” new applications was to dynamically open and close holes to let the new application zip through with *no analysis whatsoever* being performed against the traffic other than to check source/destination, port, and session connection state. That amounts to hardly any security processing, at all! But at least the firewall administrator could quickly respond to their internal customer’s request.

In response to this disadvantage for the application proxy firewall manufacturers, they added stateful inspection as an option into their firewalls as well to service just these types of situations. Adding stateful inspection into these proxy firewalls was a trivial matter and their existing customers were satisfied. However, network-layer firewall manufacturers seized upon this issue and never missed an opportunity to use it in competitive engagements.

Looking OK

Now, let us explore the concept of data “looking OK,” because it, more than anything else, will decide the future of firewalls and Internet security. What does it mean for data to look OK? The answer to that question is crucial for a number of computer security products, including: Intrusion Detection Systems (IDS), Anti-virus protection, Anti-spyware protection and system log analysis. All of those technologies, along with firewalls, attempt to somehow sort the bad stuff from the good stuff. It’s the same problem that border guards face, and it’s a hard problem for the same reason that it’s hard for border guards: the enemy is creative, and gets to decide where and when to launch their attack.



There are three ways you can decide whether data looks threatening:

- Know what is expected and verify that what you're given matches your expectations
- Know what the bad guys are up to, and look for signs of their bad actions
- Do both

The approach that proxy firewalls took (i.e., implementing a complete application protocol stack, with error checking, and "boxes" to guard against unusual inputs) is the "know what is expected" approach. After all, if you can define "OK" reasonably closely, then anything that's "not OK" must be potentially hostile or at least unusual. Unfortunately, as the Internet boom of the mid 1990's played itself out, there was a massive explosion of new applications and new versions of old applications. For example, the old SMTP protocol we looked at earlier expanded into ESMTP and now sports a command set that is several times more complicated. Defining a minimized subset of ESMTP is hard because not only are there more commands, there are more slightly different implementations of server and client-side software. The proxy designers have had to increase the complexity of the proxy so that it can encompass a superset of the commands that it needs to handle correctly; obviously, this is more work for their engineering teams.

The other approach is to know what the bad guys are doing, and attempt to match for it. The best example of the effectiveness of that approach is anti-virus software. Since the first computer viruses began appearing in the wild in the mid 1980's, there are now over one hundred thousand identified unique pieces of viral code or "malware." And, have you noticed something? Systems still fall prey to new viruses all the time! The reason is that just like a biological virus, a new computer or network virus gets a chance to propagate freely until immunity is developed against it. The term for this is a "0 day attack" (i.e.: an attack that takes place on the first day that anyone has ever seen it, and before the computer security community has had a chance to react to it). If you consider the situation purely in terms of military strategy, you can see that this approach has serious limitations, since you're ceding the attack initiative totally to the enemy. Put differently; if your entire strategy is to dodge bullets and never shoot back—sooner or later, you're going to be a bit too slow.

Some security practitioners refer to the "know what the bad guys are doing" as "reactive security" to acknowledge the fact that you're always going to be responding to external stimuli. In spite of the obvious flaws with the reactive model, it has proven to be extremely popular over time. Why? Because when you match an attack against a known attack pattern, you can positively identify it for what it is. SQL Slammer worm version 1 consisted of a single packet targeted to a specific port, containing easily recognized byte-codes of machine instructions. A firewall that "knows" how to identify SQL Slammer can check for that pattern and then throw the packet out, if it's seen. It can also log "I saw an SQL Slammer attack and deflected it" for the administrator, who will naturally get a comfort factor from knowing that the firewall is actually doing something useful.

As with most interesting security problems, we have two approaches, each of which has some value. In the long run, which is better? Well, the "know what is good" approach is more *conservative* because, unlike the "know what the bad guys are doing" approach, it is more likely to prevent a "0 day" or never before-seen attack. It's still possible that a new form of attack might get through the "know what is good" approach, but it is far, far less likely since the attack would have to be based on a previously unknown property of something that was believed to be "good." Of course, there's obvious value in blocking well-known attacks, but the most sensible approach is to realize *both approaches* have merit, and to try to synthesize them into a unified, manageable solution.

Enter IPS

Intrusion detection systems (IDS) became a popular computer security technology during the late 1990's. While IDS worked fairly well, the majority of implementations were primarily pattern matching systems that operated on the "know what the bad guys are doing" approach. In an IDS, the patterns were known as "signatures" and were sometimes supplemented with the "boxing in" approach to protocol analysis. In IDS, this was referred to as "protocol anomaly detection", the idea being that the IDS was looking for anomalous branches or variations in the application protocol. IDS worked fairly well, but customers remained unsatisfied with it for the simple reason that having a device sitting there warning, "Look, I see SQL Slammer!" prompts the obvious question, "Well, why can't you *do something about it!*" Thus, "Intrusion Prevention Systems" (IPS) were born.



First generation IPS looked a lot like a network-layer firewall with a default policy of “allow everything through” and a set of IDS signatures hooked into a packet-copying layer so that, if the IDS signature identified an attack, the packets related to the attack would be dropped. Since many of the underlying operating system/engines for IPS were developed to run on open source UNIX or LINUX, their manufacturers often took advantage of the open source network-layer firewalls that come bundled with those operating systems. Suddenly the IPS was able to offer a synthesis between IDS and firewalls, without being the best IDS or the best firewall. Regardless, it turned out to be an attractive mix for customers struggling to cope with swarms of self-replicating worms that were ravaging networks in the late 1990’s. Many companies fielded first generation IPS simply to block devastating worms like SQL Slammer and CodeRed.

IPS turned out to be a huge boon for the manufacturers of network-layer firewalls. During the late 1990’s, the majority of attacks were becoming increasingly data-driven and application-centric. By 2000, the most popular attacks relied on “buffer overruns”—remote penetration by exploiting a known flaw, or vulnerability, in a target application. Many buffer overrun attacks worked straight through active firewall connections if the basic connection was permitted; and since a majority of the target applications were Web servers and browsers, the traffic was nearly always permitted. Network-layer firewalls were beginning to fail at protecting networks and applications, and to fail conspicuously as worm after worm took down corporate WANs in spite of the presence of network-layer firewalls. In 2002 there was a flurry of technology partnerships as the IDS manufacturers rushed to integrate blocking and firewalling into their IDS, and the firewall manufacturers moved to add IDS signatures into their simple permit/deny logic.

In terms of history, we’re now caught up to the present. To differentiate in the market, various vendors have adopted fancy-sounding names for “firewalls with signatures” or “IPS” but generally the technologies remain the same at the core. There is still a sharp difference between proxy firewalls and network-layer firewalls around the question of *how much* analysis is actually performed on the network traffic passing through them. Since the proxy firewalls are still rooted in the “no data will pass unless it looks correct” model, they remain less likely to fail under attack than the network-layer firewalls because of their less conservative design.

So Why Isn’t It Working?

The Internet security landscape remains in constant flux. Unfortunately, the bad guys have proven themselves to be just as innovative as the good guys. In the last few years, there have been three big threat innovations that have been keeping firewall practitioners on their toes:

- Injection attacks
- Tunneling attacks and ‘firewall friendly’ products
- Encrypted everything-over-everything

Injection attacks are a special case of a site-specific attack. Typically, the attackers are looking for flaws in Web site transaction semantics or forms used for shopping carts or searching. The reason these attacks are so problematic is because they are site specific; an IPS / IPS-firewall isn’t going to have a signature for the attack because it’s an attack that is only used against a particular site’s back-end systems. Worse yet, the attack is usually launched over HTTP or SSL, which has to be permitted inbound to a Web site. What can be done to protect against this type of attack? Really, the *only option* is to program “know what is good” filtering rules into the firewall, rather than trying to rely on “know what the bad guy is doing” rules. This can be time-consuming work for sites that do not have well-specified URL layouts. Amazingly, the author has seen a large number of sites fall victim to attackers because they mistakenly relied on an IPS/IPS-firewall and failed to understand that the signature-based approach cannot help to prevent unknown or custom attacks. “But we thought the firewall would protect us!”

Many of the popular hardware-based IPS/IPS-firewalls rely on pattern-matching accelerators built into network processing units. One of the dirty little secrets of these devices is that some hardware accelerators can only load a smallish number of patterns before the software layers have to begin swapping patterns into and out of the ASICs, which incurs a huge performance cost. The manufacturers get around this by heavily marketing benchmark figures based on being “optimized” for performance signature sets, not



security signature sets. One of the early versions of a popular IPS-firewall initially boasted 36 IPS signatures when the product was announced in 2005. At the time of the product's announcement, there were several *thousand* "in the wild" attacks being used on a regular basis, yet the product could only do "deep packet inspection" and "intrusion prevention" against a tiny percentage of the attacks that would be thrown against it. Consumer beware.

Tunneling attacks and 'firewall friendly' products offer a different challenge for the firewall designer. These tools attempt to bypass the firewall by originating from *inside* the firewall's protected zone. Typically, they masquerade as HTTP traffic, although the newer generation is moving towards encrypting the tunnels over SSL. For the firewall, the challenge is to distinguish legitimate Web surfing activity from illicit remote-control software. Some of the remote-control tools are extremely sophisticated and allow the hacker to alter the appearance of the tool's traffic to make it more difficult for an IPS/IPS-firewall signature to match. Encrypting the traffic only complicates matters further. Using "know what the bad guy is doing" techniques to identify this kind of traffic is doomed to fail because each sophisticated attacker will develop their own variant that cannot be matched by an existing signature. 'Firewall friendly' products are not as overtly malicious as the hacking tools, yet they offer nearly the same challenge. A number of voice-over-IP chat tools and instant messaging programs "know" literally dozens of different ways to attempt to get data out of your network right through your firewall. Composing a policy that effectively shuts down such traffic is very difficult unless the administrator is willing to risk reducing the scope of legitimate traffic, as well.

Encryption presents another significant problem for firewall designers. At this time, it is possible to execute a "man in the middle" interception of SSL traffic if you are able to interpose a device in the traffic's path. Eventually, the flaws—or features—of SSL that allow this to happen will be fixed, and the encrypted stream will be completely inaccessible to IPS/IPS-firewalls. In the meantime, computer security has been fortunate that hackers appear to be too lazy to add sophisticated encryption into their software. Over time it will inevitably happen. When it does, firewalls will either be phased out entirely, which is *highly* unlikely, or will be configured to be much more restrictive. The more conservative design approach of the proxy firewall will probably become the most popular, again, for all the now obvious reasons discussed earlier in this paper.

Some Predictions

What does the future hold for firewalls? The author believes that the typical firewall administrator is going to find (sometime in the next 5 years or so) that the network-layer signature-checking firewall is going to be increasingly subverted—to the point where it may become effectively *useless*. For high-security applications, a proxy-style firewall that does application protocol validation and allows the administrator to tightly define a more restricted use of the protected application will remain the preferred tool. A necessary feature-set for the firewall of the future will be:

- High performance;
- Rapid URL checking and matching, to allow Web site-specific correctness matching and white-listing;
- Strict protocol analysis, matching for correctness rather than known hostile behaviors;
- Exhaustive HTTP transaction checking and decoding of tunneled layers;
- Ability to support large numbers of specific rules, as rules become increasing precise—down to the individual host level;
- Centralization and rapid reaction to new rules;
- Ability to run IDS-style signatures to diagnose and identify known attacks.

The future firewall will be a complex piece of software indeed, because it will need to be able to decode and analyze an ever-increasing number of complex and layered software protocols. Is there an alternative? The old "look for what you know the bad guys are doing" approach to protection is clearly doomed to fail. Or, more precisely, it really never succeeded in the first place, it's just that the mass consumer was never well-informed to understand this. Consider the anti-virus industry's twenty-year-long effort, which has resulted in twenty years of virus outbreaks. "Old school" security wizards have been pointing out for decades that eventually, it is more cost-effective to identify the software that you want to allow to run rather than to try to identify all the malware that you do not want to *allow to run*. The same logic applies with firewalls.



www.securecomputing.com

As networks grow increasingly complex and the type and cleverness of hostile network applications begins to vastly outnumber the legitimate applications, firewalls will need to switch away from the IPS-firewall approach back toward a “permit only what you know is OK” model. As part of that process, network and system administrators will be forced to confront the vast mix of services and protocols that they allow back and forth between their “internal” network and the “outside” world. The complexity of that protocol mix is already too high to be effectively secured without rigorous checking, and many administrators have favored the easier route of simply installing a network-layer firewall, even though (as we have just discussed) they simply cannot do the job. As stated earlier: **you cannot meaningfully secure traffic without looking at it.**

Summary

Over the course of the next decade, it is going to become absolutely critical that we understand the traffic patterns of ingress and egress within our networks. The permissive model that has been popular for the last decade is clearly failing. In fact, some might argue that it has shown no sign of succeeding in the first place. Proxy firewall technologies have proven time and again to be more secure than “stateful” firewalls and will also prove to be more secure than “deep inspection” firewalls. The main point of comparison between stateful firewalls and proxy firewalls has traditionally been performance, which had been a trade-off with security. The good news is that high-performance proxy firewalls are available today which are easily capable of handling gigabit-level traffic.

Secure Computing Corporation

Corporate Headquarters

4810 Harwood Road
San Jose, CA 95124 USA
Tel +1.800.379.4944
Tel +1.408.979.6100
Fax +1.408.979.6501

European Headquarters

Berkshire, UK
Tel +44.0.870.460.4766

Asia/Pac Headquarters

Wan Chai, Hong Kong
Tel +852.2520.2422

Japan Headquarters

Tokyo, Japan
Tel +81.3.5339.6910

For a complete listing of all our global offices, see www.securecomputing.com/goto/globaloffices

Note from Secure Computing Corporation: The Sidewinder Network Gateway Security appliance is a high speed, enterprise-class application proxy firewall that has been protecting many of the most important networks in the world for over a decade. For additional information on this leading proxy firewall please visit <http://www.sidewinder.com>