Design and Implementation of Secure, Content-Based Traffic Control

Internet Conference 2002

31 October 2002

Joe Stevens
j.stevens@jens.co.jp
+81-3-3500-2373

Shadan Saniepour
s.shadan@jens.co.jp
+81-3-3500-2536

JENS Corporation
Hibiya Dai-biru 10F
1-2-2 Uchisaiwai-cho
Chiyoda-ku, Tokyo 100-0011
JAPAN
FAX: +81-3-3500-2442

Abstract

The exponential growth of Internet traffic has made public servers increasingly vulnerable to unauthorized accesses and intrusions. In addition to maintaining low latency for the client, filtering unauthorized accesses has become one of the major concerns of a server maintainer. In this article we introduce the design and implementation of a load balancer that distinguishes between the traffic coming from clients and the traffic originated from the attackers, in an attempt to simultaneously mitigate the problems of both latency and security. We then present the results of a series of stress and scalability tests, and suggest a number of potential uses for such a system.

1. Introduction

Over the past decade the internet and online services have become an increasingly important asset to many businesses. To cope with the ever expanding role of internet services such as the World Wide Web, industry has developed a variety of solutions for load balancing and high availability which can be used to ensure high quality service despite network equipment failure or administrative error. At the same time, however, there has been a steady increase in the threats presented to such sites by unauthorized intrusion and appropriation of these systems and the data they contain by so-called *hacker*. The most advanced clustering and balancing technology can be rendered useless if the security of the underlying server is inadequate to repel these threats.

In this article we introduce the design and implementation of a load balancer that distinguishes between the traffic coming from clients and the traffic originated from the attackers. This system is an attempt to simultaneously solve the problems of load balancing and unauthorized intrusion. If, in the process of forwarding requests, the balancer detects traffic is an attack on the server ('an exploit'), it is then directed to an alternative server - a type of honeypot. Conventional detection and forensics methodology can then be used to gather information on the intruder, who will be unaware that they are not using a "real" server. Thus, the system will not only protect mission critical servers from unauthorized access in a manner transparent to the user, but allow for detailed data to be collected, which can later be used to take appropriate legal action against the intruder.

This article is organized as follows: Section 2, "SecureDirect" describes the goals of the project, as well as the design and implementation of our content based load balancer. The experimental design and results are discussed in section 3, "Experimental Results". Section 4, "Applications" discusses a number of potential uses for this type of security mechanism, and section 5, "Future Directions" presents a number of areas where further research is required. The paper concludes in section 6.

2. SecureDirect

Recent years have seen a drastic increase in the popularity of Network Intrusion Detection systems (NIDS). A plethora of commercial products, as well as the availability of open source solutions such as Snort [7], have moved NIDS into mainstream usage by both security administrators and providers of managed security service.

Existing products provide substantial network monitoring capacity, and have been largely successful at providing accurate reports on unauthorized activity. While these products have gotten fairly good at monitoring and reporting on unusual activity, it has been stated, that the biggest problem with IDS is "intelligently reacting to their output" [3]. If an IDS detects an attack in progress, what action should be taken? Furthermore, even if the organization deploying the IDS has the foresight create an attack response plan, if an administrator receives a page at 4:00 am, what are the chances that s/he will be able to react quickly enough to prevent the intrusion? Under most existing systems chances for timely prevention of the intrusion highly depend on the quick reaction of the human administrator. All too often IDS logs are only consulted long after the damage from an attack has been done [3].

SecureDirect is an attempt to address this problem: by providing a fully automated response to specific network intrusions, it can eliminate the need for human decision making, and thus mitigate slow human response times. While there is some existing work in this field, it has fallen into two categories: Honeypots, and Firewall-based solutions. The first of these, deploying network Honeypots, involve setting up fake-servers that look more attractive than the actual production machines (e.g., payroll.mycompany.com), in hopes that an attacker will target them. This type of project has seen a great deal of academic interest [5] and it has begun to see commercialization [6]. This approach provides excellent monitoring and forensic evidence. The downside, however, is that while it serves to distract potential attackers, if an attacker does decide to attack the production server, it offers no protection what-so-ever.

An alternative is to link an IDS system to a Firewall. The most well-known example of this technique is the open-source Hogwash project [4]. This methodology involves selectively blocking packets that are identified by the IDS system[1]. In a sense, this solution is exactly the opposite of Honeypot deployment, in that it provides excellent protection for the server, while offering no opportunity for forensic analysis, and very little monitoring capability, since anything identified as an attack is immediately blocked. An additional danger in this type of system is that if the IDS misidentify acceptable traffic as an attack, a site could end up blocking desired viewers. More worrisome is that in this type of system the attackers will be able to realize that they are being blocked, and thus may find a way to exploit the Firewall/IDS combination to create a denial-of-service situation.[2]

SecureDirect, by implementing content based load balancing, attempts to correct the shortcomings of both of these systems. Most importantly, it provides direct protection by not allowing "bad" traffic to the production servers. Additionally, by directing traffic to a Honeypot setup to look (from the outside) exactly like the production machine means that the security administrator will have ample opportunity to gather forensic evidence (through traditional NIDS deployment, etc), with the additional benefit of not allowing the attacker to realize that his access to the production system(s) has been cut-off.

To summarize, our objective for developing the current system can be described as follows:

The problem of high availability and security have thus far been dealt with independently, instead, we propose to deal with them as a single problem. Through this integrated solution we show that we can achieve a higher level of both security and availability. Additionally, we want to create an opportunity for adequate information to be gathered about an intruder, in order to facilitate legal action or prevent further intrusion.

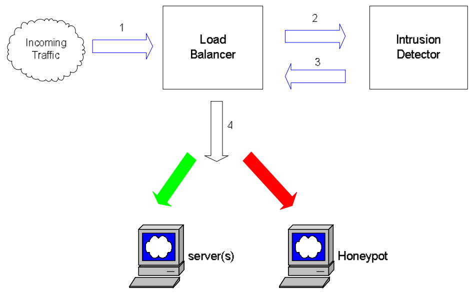
<u>2.1 Architecture</u>

SecureDirect consists of two main entities, namely, the load balancer entity and the intrusion detection system. These entities run as separate processes but information on user requests is shared between them. The architecture of SecureDirect is illustrated in Figure 1.

[1] Which in Hogwash's case (as the name suggests) is Snort.

[2] For example, sending packets that contain attacks with your IP spoofed as a popular web site, or the network DNS server.

Figure 1: The SecureDirect System



The activity at each numbered point in the above diagram can be described as follows:

1. The load balancer receives the request to the virtual IP address. If the packet containing the request has been fragmented, it is reassembled.
2. The Load balancer opens a TCP connection to the IDS Process, and sends the content of the packet (less the headers) over that connection.
3. The IDS process checks the content of the packet against its database of known attacks, and returns a Boolean result to the load balancer over the same TCP connection.
4. On receiving the result, the load balancer closes the TCP connection. If the result from the IDS was "true" (indicating the presence of an attack) the packet is forwarded to the Honeypot. Otherwise, a server is selected from the active server pool in a round-robin fashion, and the packet is forwarded to the server.

The design of this system entailed overcoming a number of challenges. What these challenges were, and how they were addressed is discussed in the remainder of this section.

2.2 Load Balancing

In designing a load balancer for SecureDirect we have three main focuses, namely, to provide High-Availability by handling hardware failure in the web-cluster, maintain high speed access to the cluster, and ensure the balancer itself does not become a security hole. High availability is achieved by simply pinging the servers at regular intervals, and removing them from the server pool if no response is received. In order to deal with the speed problem, we have designed our system so that the traffic coming from clients is passed through the load balancer and the load balancer has complete control over the traffic delivered to the web servers. However the traffic in the opposite direction, from server to clients, is directly passed to the clients. As the major part of the traffic usually consists of the contents delivered to the clients (e.g., web contents), this strategy will drastically decrease the traffic of our load balancer, and prevents it from becoming a bottleneck.

The second challenge is to secure the load balancer. Our strategy is to protect it from any irrelevant traffic. Only traffic to a specific port on the virtual IP will be processed by the load balancer and any other malicious access is simply ignored. The load balancer uses a technique known as 'Proxy-ARP' to respond to ARP requests from the router to the virtual IP address. This way the operating system of the server machines automatically skip all the packets destined to virtual IP address, while our load balancer daemon reads them off the wire and decides about its next action; whether to balance the load or redirect the Honeypot.

The web servers have their loopback interface[3] configured with the virtual ip address, but are set not to respond to arp packets. This, at the network layer, there is no way to tell which servers are configured with the virtual IP.

The load balancer process is implemented

[3] Interface lo0:1 on Solaris.

as a multi-threaded process in C. The main thread is responsible for reading packets off the wire and if they are destined to virtual IP and their destination port is our desired service port, it hands them to the control thread. The control thread then communicates with the IDS process and decides to pass the packet to the production servers or direct it to the Honeypot. The system implemented for this paper runs under solaris, and takes advantage of solaris kernel level threads to improve performance. Reading and writing packets to and from the Ethernet interface is done using Solaris' DLPI. The Data Link Provider Interface (DLPI) is a UNIX STEAMS standard that defines an interface to the Data Link Layer of the OSI Refrence Model. The use of DLPI and a multithreaded design allows us to capture packets while assigning a minimal load to the main thread, and thus prevent packet loss.

In case of TCP applications, after redirecting a request to a server, the load balancer process should hold the information about which request is forwarded to which server in order to forward all of the ACK packets in a particular session. We keep the information about each connection in a table. As the load balancer only passes the traffic from client to server, it is unable to see the last FIN sent by the server, and therefore there is no way that the load balancer can find out when the conversation between two hosts is over. Therefore we define a time stamp for each connection. Each time a packet is received on a connection its time stamp is updated. The connections will be removed from the table if its time stamp is not updated for a certain amount of time (which has been set at arbitrarily at 4 minutes during our tests).

The design of this load balancer is very similar to Linux Virtual Server [10]. A significant difference, however, is that we use ARP proxy to protect the load balancer. Comparing to other existing software [1,2,8] our load balancer has the advantage of passing only the incoming traffic and therefore the chance of the load balancer to become a bottleneck is decreased.

2.3 Intrusion Detection

Internally, the implementation of the IDS portion of SecureDirect is very simple. The IDS process runs as a concurrent TCP server, and listens for client requests on a specified port. When a connection is made, the IDS forks a child process when receives the content of the packet, and then checks it against a database of known attacks. It then returns the result of this check to the load balancer process, which makes the decision on whether to forward the request to the production server cluster, or the Honeypot. The IDS process is written in Perl [4] to take advantage of its strong and simple regular expression support, and fast pattern matching engine. The signature database is taken directly from the open-source IDS Snort.

The multithreaded design of the load balancer ensures that multiple requests from a client will not get 'mixed up', however, it is possible that an attack would occur from a single IP at the same time as a valid request. In this case, the initial, harmless packets may be safely forwarded to the real servers until the IDS process finds the attack-packet and detects the signs of intrusion. At this point, it immediately informs the load balancer process to discontinue forwarding packets to the real server, and to send an RST packet to the corresponding server to end the connection. Thus, the server will never receive the attack. In the attacker side, observing silence from the server side causes it to assume the server has crashed

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

and possibly causes it to try to re-connect. However from this point, after detecting the intrusion, all the incoming traffic from the attacker's IP will be forwarded to the Honeypot.

One of the key points in maintaining speed of this system is that, once an IP is recognized as an Attacker IP, packets coming from that source are not passed to the IDS process any more. The load balancer process directs the incoming traffic from attackers to the Honeypot.

2.4 The TCP Layer

SecureDirect is implemented for applications that use TCP/IP as their transport protocol. SecureDirect is designed (like any good security product) to be failed-closed [9] system, which means if it crashes, it is no longer possible to access either web server through the virtual IP. Consequently the attackers can not crash SecureDirect and access the unprotected system.

One of the major challenges for SecureDirect is to deal with attackers who try to fool the system by forcing it to analyze the packets inconsistent with what is received in the end-system. In the cases that the intrusion detection system runs at the different host-OS than the end-system, this can be a serious concern. The attacker can take advantage of differences between the IDS and the end-system in dealing with the packets that do not fully comply with the standard protocols, and send some packets that are discarded by the end-host but accepted by the IDS, or vice versa. If SecureDirect uses TCP/IP, the inconsistency between the IDS and the end-host may appear in IP level or TCP level [9].

TCP protocol uses sequence numbers to preserve the order of the incoming packets. The end-system waits until it receives all the sequence numbers required for re-assembling the data. If there is a missing sequence number, the end-system will not accept the consequent packets and waits until it receives the packet with the sequence number it is waiting for. Therefore one way to try to fool the system is to send two packets with the same sequence numbers, one containing false data to be accepted by the IDS and discarded by the end-host, and the other one containing the attackers desired data to be accepted by the end host, and skipped by the IDS. Thus, whenever it detects two different packets with the same sequence number (and a sane checksum) for one connection, it considers it as an attack and prevents the load balancer from sending that packet to the end-host. Furthermore it marks the source IP of this packet, as an attacker IP, causing the load balancer to forward all the consequent packets originated from this IP to the Honeypot.

An alternative way to break into the system is using IP fragmentation, hoping that IDS and the end-host follow two different methods for re-assembling IP fragments. SecureDirect, however re-assembles the IP fragmented packets in the load balancer and forwards the assembled packet to the end-host. Therefore what IDS analyzes is completely consistent with what end-host sees. This type of implementation should drastically reduce the chances of an attacker breaking into the system.

3 Experimental Results

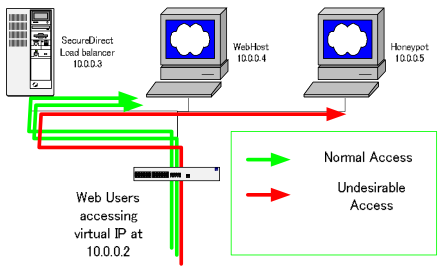
We conducted a simple experiment with total number of three servers. All were Sun Ultra 5[5] workstations running Solaris 9. In this section we describe the detail of this experiment.

[5] With a 270 mhz processor and 256 MB of RAM.

3.1 Test Environment

A simple test environment requires a minimum of 3 servers and 4 ip addresses. Each server is assigned a single IP address for basic TCP/IP connectivity, while the fourth is used as a virtual IP, which will be accessed by clients, and balanced by the balancer to either of the two servers depending on the contents of the client request. This environment is illustrated in Figure 2.

Figure 2: Test Network Environment


SecureDirect Load balancer 10.0.0.3
WebHost 10.0.0.4
Honeypot 10.0.0.5
Normal Access
Undesirable Access
Web Users accessing virtual IP at 10.0.0.2

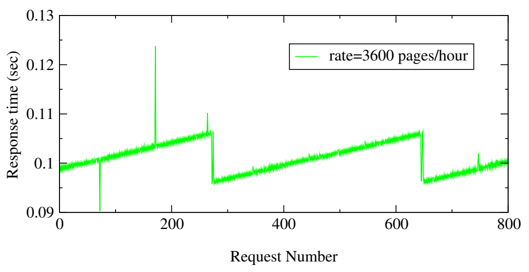
In this example, the virtual IP exists at 10.0.0.2. This is the only IP address ever seen by the clients, likewise in a production environment it would be the only IP to have a DNS record, etc. While this address is not actually bound to any of the computers in the network, the load balancer machine responds to ARP requests from the router using proxy-arp. When a request is received, it then reassembles the packet (if fragmented), checks the request against the IDS signature, and forwards it to appropriate server based on its content.
One of the keys to this system's security is that the only system being accessed by clients *doesn't actually exist*. Thus, in addition to the balancing being completely transparent from the perspective of the client, any attempts by an intruder to attack the virtual IP on another port than the one being balanced for, will be simply ignored.

Secondly, (as stated above) this type of system serves not only to protect the server intrusion, it can also be used to gather information about the intruder. While the load balancer itself doesn't gather any information internally, the addition of a Network Based Intrusion Detection System (NIDS) on the network can be used to perform this task, while the intruder is harmlessly attacking the Honeypot server.

3.2 Results

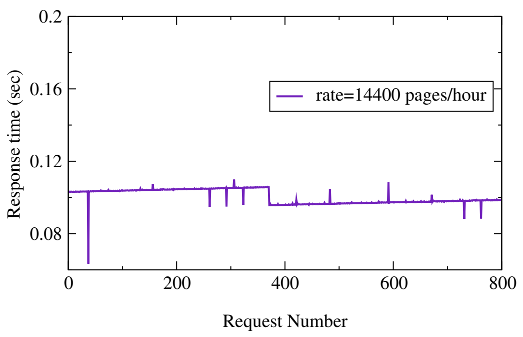
While the basic functionality of a content-based load balancer is relatively easy to achieve, such a system is only useful to the extent that it is scaleable and stable under load. At a modest load, the balancer showed very stable performance:

Graph 1: Response Time for 3600 pages/hour



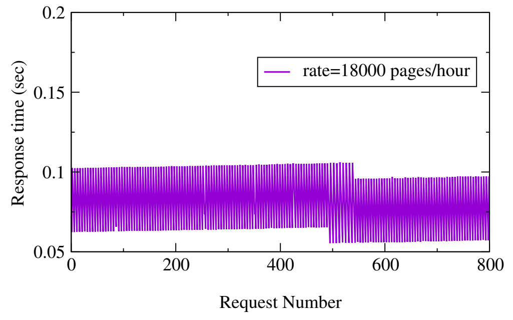
There are a number of spikes, likely due to server-side or network conditions, but the average response time is steady at approximately 0.1 seconds per request. This type of performance holds up well all the way through the 14000 pages/hour range, at which point variation in the response pattern begins to become evident. In Graph 2, there is a much larger number of spikes, and the average response time per page is slightly increased.

Graph 2: Response Time for 14400 pages/hour



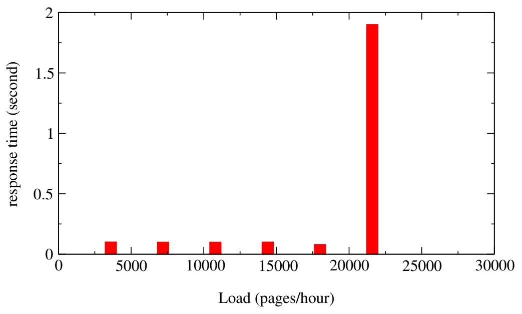
By the time the web-benchmark is turned up to 18000 pages per hour, the balancer is clearly starting to show signs of stress. The spikes have now become very regular, indicating that the load on SecureDirect is causing it to become less regular in servicing request.

Graph 3: Response Time for 18000 pages/hour



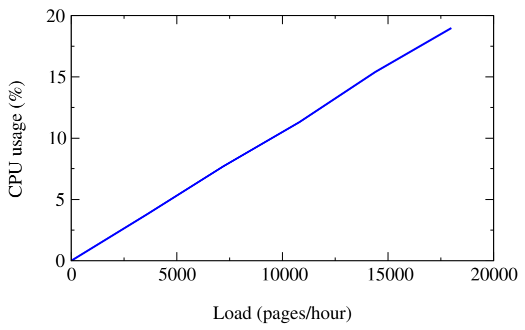
If the request speed is increased above 18000, the balancer becomes overloaded. At this point, the average response time per request increases dramatically, and after a period of 30-45 minutes under constant load the balancer begins to drop requests. The sharp increase in response time can be seen in Graph 4.

Graph 4: Average Response Time



An analysis of the source of the load shows that it is entirely due to the Intrusion Detection Process. The load balancer process consumed a nominal amount of CPU resource during every one of the trails. The IDS process, however, showed a load curve which increased in a linear manner as the number of pages per hour was increased.

Graph 5: CPU Load



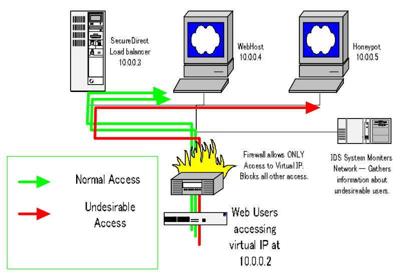
While the linear nature of the IDS process CPU usage suggests that SecureDirect able to scale gracefully, it also highlights the tradeoff between performance and security. While there is certainly room for optimization in our code, the act of pattern matching all incoming TCP traffic is an inherently CPU intensive activity. In our tests, we used the full set of Snort rules[6] pertaining to web attacks – which totaled 516 pattern matches per web request. This checked for attacks against both Unix and Windows based servers, as well as a variety of scripting and database applications. Performance could be improved by only checking for attacks against the type of software actually being used[7], however the tradeoff would still remain.

4. Applications

The introduction of SecureDirect into a production system would require a slightly more complex network design than the one used in our experiments. To realize the full potential of the system, a firewall and a traditional NIDS system would be required.

[6] As provided with Snort 1.8.3.
[7] For example, of the 516 rules, 94 of them checked for attacks against Microsoft's Internet Information Server. If the site being protected was based on Apache, these checks would be irrelevant.

Figure 3: Possible Application



While we believe that content based load balancing is a powerful security tool, it can not replace more traditional security practices such as firewalling, and network or host based IDS. In the figure above, the firewall[8] allows incoming TCP traffic on port 80 to the virtual IP address. All other traffic is blocked. Incoming traffic is picked up by SecureDirect, and balanced to the appropriate server. When the server responds, the response appears to be from port 80 of the virtual IP address, and thus no additional firewall rules are required. Any attempt to access the production web servers directly would be dropped at the firewall[9].

In the event of an Intrusion, in addition to being redirected to a non-critical server by SecureDirect, a NIDS monitoring the network is present to record the intruder's activity for latter action such as informing her ISP, or local law enforcement.

5 Future Directions

Research remains to be done in several areas. The current implementation of SecureDirect uses only simple pattern matching on incoming TCP requests to determine whether a particular request is good or bad. While we feel this is sufficient to protect against the majority web-server attacks seen in the wild, it isn't clear whether it is equally appropriate for other protocols (SMTP, FTP. Etc).

Additionally, our simple implementation of pattern based scanning is completely incapable of protecting against non-intrusive attacks such as Denial of Service (DoS).

This situation can be remedied in one of two ways: The IDS process can be made much more robust while still relying on pattern matching, or an alternate means of intrusion detection, such as one based on network traffic anomalies, can be implemented. Both of these systems have advantages and disadvantages. Reliance on a more complex IDS process for load balancing (such as the full output of Snort) would allow for a much wider range of detectable attacks, however, it would also significantly impact performance and increase administrator maintenance requirements.

Furthermore, the current system required two extra servers (the load balancer and the honeypot) in addition to the production machines being protected. If the system is used to its potential (with the addition of a NIDS and Firewall) an even greater number of machines must be purchased and managed. The financial and administrative cost of such a system will likely make it unattractive to all but the largest web-sites. Ways of providing the same functionality with a smaller number of machines (and a lower administrative overhead) are currently being investigated.

On a more technical level, the speed of the system could still use a good deal of improvement. There are two areas here

[8] This configuration assumes the use of a stateful firewall. Similar performance could be achieved by simply blocking all other ports at the router, although in such a case an additional rule allowing outgoing traffic from the virtual IP would be required.
[9] An even better configuration would be to give the entire load balancer and web server network segment private addresses, and perform destination NAT to the virtual IP at the firewall. The complexity of such a system, however, is outside of the scope of this paper.

most in need of attention. The first question involves how much data is processed by the ids, and how well it is examined. The experiment described in this paper used web traffic to test content based redirection. The text-based nature of web traffic (and thus web attacks) means that a simple pattern-based scan on incoming HTTP requests is sufficient to catch almost all known web attacks. This is advantageous because the traffic-scanning IDS process can be relatively simple and fast (in our case, only a few hundred lines of perl). A more versatile solution would necessitate a much more robust IDS, possibly even utilizing other detection technologies (such as anomaly based detection, etc), in order to provide protection for a wider variety of services. Such a system, however, would quickly become a bottleneck.

Additionally, in the current system the IDS and Load Balancer process are separated in order to prevent the Load Balancer from dropping packets if the IDS become loaded. While this yields a performance advantage for the both process internally, the communication between the two consumes considerable overhead. A number of different methods for interprocess communications were attempted (Unix Domain Sockets, FIFOs, shared filespace), and we finally chose TCP sockets for its robustness and concurrency, but this came with a relatively high cost in terms of latency and CPU overhead.

6 Conclusions

Solving the problem of high availability and security simultaneously offers the opportunity for more reliability than systems which solve the problems separately, in addition to being easier to implement, and offering increased opportunity for recording and data analysis. The integration of these two technologies, however, is a non-trivial task. A fine balance must be achieved between speed and robustness of IDS features – it is equally bad to have the web server crash because an attack was missed, or drop large amounts of traffic due to an overly through IDS becoming a bottleneck.

Additionally, while a content-based load balancer offers many advantages over separate Load Balancing and IDS solutions, it also suffers from the drawbacks of both. While a properly designed and maintained system can prevent the need for immediate administrator intervention, substantial resources in both time and equipment must be spent if such a system is to bear fruit.

It has been said that the three things most important to a server manger are Security (the ability to withstand hacking), High-Availability (the ability to withstand hardware failure), and low-latency (the ability to service requests quickly). With the current implementation of secure direct, the first two criteria, security and high availability, are satisfied. In regards to latency, our experiments have (once again) shown that there is a trade-off between security and performance. While improving the efficiency of the programming can mitigate this problem to an extent, the trade-off is certain to remain, and choosing the right balance will likely continue to be a difficult task for the system administrator. It is the authors' opinion that this type of system could provide a much needed additional security tool, however, a good deal of streamlining work remains before it could be widely deployed.

7 References

[1] Balance, An open Source Load Balancing TCP Proxy: http://balance.sourceforge.net

[2] Big/ip: http://www.bigip.com/bigip/

[3] Bruce Schnier, Decrets and Lies: Digital Security in a Networked World, Wiley Publications, 2000

[4] Hogwash: http://hogwsh.sourceforge.net/

[5] Honeypot Project: http://www.honeypot.org

[6] ManTrap: http://www.recourse.com

[7] Martin Roesch, Snort- Ligthweight Intrusion Detection for Networks, Proceedings of LISA'99 : 13th System Administration Conference, Seattle, Washington USA, 1999

[8] Pen A load balancer for simple tcp based protocols such as http or smtp for Unix: http://siag.nu/pen/

[9] Thomas Ptacek, Thimoty N. Newsham, Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection, Technical Report, Secure Networks Inc. 1998

[10] Wensong Zhang, Linux Virtual Server for Scalable Network Services, Ottawa Linux Symposium, 2000