Less harm, less worry or how to improve network security by bounding system offensiveness

D. Bruschi, L. Cavallaro, E. Rosti
Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano
Via Comelico 39, 20135 Milano – Italy
{*bruschi@, lc529863@silab., rose@*}*dsi.unimi.it*

Abstract

In this paper we describe a new class of tools for protecting computer systems from security attacks. Their distinguished feature is the principle they are based on. Host or network protection is not achieved by strengthening their defenses but by weakening the enemy's offensive capabilities.

A prototype tool has been implemented that demonstrates that such an approach is feasible and effective. We show that some of the most popular DoS attacks are effectively blocked, with limited impact on the sender's performance. Measurements of the implemented prototype show that controlling the outgoing traffic does not affect performance at the sender machine, when traffic is not hostile. If traffic is hostile, the limited slow down experienced at the source is the price to pay to make the Internet a safer place for all its users.

The limited performance impact and the efficacy in attack prevention make tools like the one presented in this paper a new component of security architectures. Furthermore, such a type of tools represents an effective way to address security problems that are still unsolved or for which only partial solutions are available, such as the liability problem, intranet security, security tools performance and the use of distributed tools for intrusion.

1. Introduction

System defense and protection of victim machines have been among the main goals of computer security since its origins. A variety of tools and methodologies have been developed (i.e., firewalls, intrusion detection systems, antivirus, access control systems, etc.) that proved to be quite effective in protecting systems and networks from intruders [9]. They implement a common philosophy: improving system protection by stacking a series of barriers between the system and the outside world. They reflect a local perspective on security, where individual protection is the goal, regardless of what may happen outside on the network.

Such a philosophy and the tools based on it, however, are no longer sufficient to keep up with the current technological and legal trends. Due to the increasing popularity of encrypted traffic using IPSec, SSL, or VPN protocols, it is becoming harder and harder for nowadays protections, such as firewalls and IDS's, to transparently and effectively perform their tasks. In this case, the only viable solution would be to perform checks before the payload is encrypted. Furthermore, because of the steady increase in network bandwidth and speed, such tools cannot process packets fast enough to keep up with network speed [11].

Protection oriented tools do not help computer owners avoid liability since they do not prevent the systems they are applied to from attacking other systems. There are countries, such as Italy, where according to the local computer crime law, computer owners are liable for all the actions executed by their systems. In this case, even if a host becomes the source of an attack as a consequence of an intrusion it suffered, its owner can be legally prosecuted, although the owner is not physically responsible for the attack. When thousands of unaware hosts are exploited by sophisticated distributed intrusion tools [5, 6, 7] to launch distributed denial of service attacks, like the one of last February against high profile Web sites [16], the liability case can be serious.

In this paper we present a prototype of a tool or class of tools that complement the "classical" defensive approach to computer security as they contribute to realize a global perspective on security. In this perspective, security of the global network as an entity improves by restraining the harmful capabilities of its components, not only by hardening their defenses. The tools we propose provide an innovative way to solve security problems and represent a first answer to the limitations mentioned above.

The ultimate goal of our security tools, like any other one, is to protect systems from attacks but, unlike and opposite to traditional tools, our tools do not build thicker and stronger barriers around the system to be protected. Our tools achieve their scope by limiting the harm the "enemy" can do to the system. Such an approach has been proposed in [1] and is based on the consideration that a computer may be a victim and an attacker as well. Thus, in order to worry less about security, we should not only protect our systems but prevent them from doing any harm.

The tool we propose in this paper satisfies the following properties: when installed on a host, it turns off the host attacking capabilities and it requires the host to be re-installed without the tool, for the host to be exploitable for attacks. The tool is implemented as a kernel module that monitors the host activity and blocks it when it does not conform to a "good" behavior or, vice-versa, when it matches an "anomalous" behavior, depending on the approach followed. As an example, the prototype described in this paper is shown to effectively block several well known Denial of Service attacks at the source, thus disarming the hosts with respect to the considered attacks.

By characterizing attacks at the originator, we can then implement modules that recognize and block them. In our perspective such modules should be collected to form an "attack inhibition system." They behave like network intrusion detection systems (NIDS) applied to outgoing traffic on the attacking host, rather than to the incoming traffic on the potential victim host. The philosophy is similar in the sense that both our tool and NIDS apply a set of rules on the packets that allow to identify known attacks. Although similar in principle, the two systems cannot simply exchange their set of rules defining attacks as in our case the tool requires the characterization from the source point of view. The integration of the signature set, i.e., the set of rules defining the attacks, of our tool with that of NIDS such as Snort [12] is currently under analysis.

Solutions like egress traffic filtering [13], where filtering rules can be applied to the outgoing traffic on boundary routers, have been proposed in order to prevent malicious traffic to leave well defined boundaries such as corporate intranets. What we propose here is an operating system extension that locally stops offensive activities, thus preventing any host to easily turn into an attacker.

In a short period of timethe approach proposed in this paper could be effectively deployed in local environments, i.e., networks such as intranet or corporate LAN's where hosts may be centrally managed. In this case disarming technology can provide an effective solution to problems such as liability and intranet security. From a legal point of view, a computer adopting such a technology could be considered adequately configured to comply with the law that prohibits hosts to be attack sources, thus relieving the owner from liability. In case of an intranet, the disarming technology can protect it from insiders' attacks and help prevent insiders from using the internal hosts to attack computers outside the intranet perimeter. In the long term, the large scale deployment of disarming technologies would represent a significant step towards an improvement of the global security as fewer attacks would be possible, especially from unaware systems. Disarmed hosts would block a fair share of attacks at the source. A trusted list of disarmed hosts could be compiled and used by tools such as firewalls and IDS's to improve their operational speed. Furthermore, with our approach, intruders would have difficulties in finding hosts where their agents for distributed attacks could be installed and successfully activated. The proposed approach is not a silver bullet and limitations to its applicability exist. However, we believe that they can be overcome. The first limitation is the difficulty to impose our approach and have it work on a geographic scale. This would not longer be an issue if the proposed extensions became part of the distributions of the most popular operating systems as a static kernel module. The second limitation we see is that the proposed extensions could be circumvented by sophisticated users, like any other software protection. Yet, they would still protect a system from "script kiddies" that use ready made exploit programs downloaded from the network. Sophisticated users that tried to bypass the proposed extensions would have to remotely install a stripped version of the operating system, which may not go easily unnoticed nor be successful in all cases. In this case, a hardware implementation based on ASIC technology could be adopted.

This paper is organized as follows. Section 2 describes the architecture of the proposed tool. Section 3 illustrates the details of the prototype implementation. Results of the impact of the proposed module on system performance are given in Section 4. Section 5 summarizes the contributions of the paper and outlines directions of future research.

2. System architecture

In this section we describe the architecture of the proposed tool with respect to the TCP/IP protocol stack architecture. Details of the implementation on a Linux system are given in the next section.

We briefly recall that the path followed by an outgoing packet, generated at the application level (i.e., with the standard socket interface), on its way to the network traverses four layers. In a Unix system the data to be transmitted is first converted into a linked list of `mbuf` data structures (the internal memory allocation units of the TCP/IP protocols). The TCP/UDP module is then called to construct the corresponding header and checksum. The IP header and routing information are added to the packet by the IP layer and finally the data link layer, generally the Ethernet layer, maps

the destination IP address to an Ethernet address, constructs the Ethernet header and transmits the packet over the network. Stream and datagram sockets interface the TCP/UDP module, while raw sockets interface the IP layer directly. Direct data link access is provided as well that allows to bypass all of the above layers, either by specifying an address family such as AF_DLI, or by opening a BPF device, depending upon the Unix flavor.

The ideal position where to place our tool is the data link layer, as it works as a choke, the single point all packets have to go through. Such a layer, however, has strict performance constraints since it is the interface to the network device. Burdening it with additional operations besides the packet data link encapsulation would significantly hurt network performance. For this reason, we prefer to check the packets earlier on, at the IP level, in order to spare the lower level modules from potentially useless work.

The tool we propose in this paper operates at the IP layer. It is comprised of a static kernel module that applies packet filtering rules to the outgoing packets when they are ready to be passed on to the data link layer. The Hostile Outgoing Traffic Interceptor (HOT-I) is added to the native IP stack as a routine that manipulates a packet in the output chain, before the device driver receives it, as illustrated in Figure 1. This way the module monitors the outgoing traffic, without modifying the existing system. A data link layer module that can block hostile Ethernet frames could be developed in a similar way. Such a module would be useful to prevent attacks such as ARP cache poisoning that require the generation of malicious packets at the data link layer in order to bypass the legitimate ARP process [15]. We concentrate on the IP layer because attacks at this level are more popular. However, we successfully verified the applicability of our approach to the lower level so that a data link layer module is the next step in this project.

HOT-I modules can be executed on host computers as well as network components such as routers. In this case performance considerations are critical. The packet flow is checked against attack signatures of known attacks and blocked when an attack attempt is detected, similarly to what an IDS does on the incoming traffic. The interceptor requires attack characterizations, i.e., the behavioral patterns typical of the various attacks. The more unique the attack pattern behavior, the more precise the action of the interceptor, i.e., the less false negative and false positive signals the module will send. In case a hostile packet is detected, the default action is to drop it. However, alternative and/or additional actions could be considered, such as logging all the intercepted traffic or letting the packet out anyway but signaling the superuser for further actions to be taken. A separate module handles such a signaling part, e.g., by raising alarms, suspending the allegedly offending program, or logging the detected hostile activity.

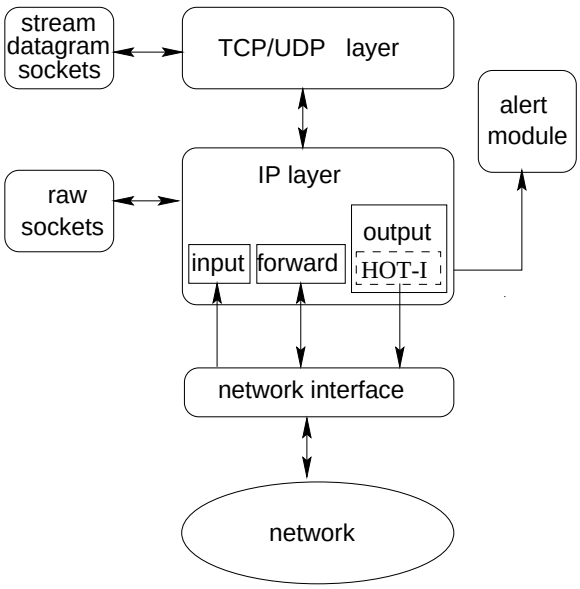

Figure 1. The modified TCP/IP architecture comprising the Hostile Outgoing Traffic Interceptor.

As the number of attack signatures grows, additional rules must be specified unless the new attacks share some general characteristics, e.g., source address spoofing, that already force packet drop. Because the module is compiled as a kernel static patch in order to prevent its easy removal, the kernel must be recompiled and the system rebooted for newly added rules to become active.

3. HOT-I prototype

In this section we describe the implementation of the HOT Interceptor prototype. We then illustrate the attacks it blocks.

3.1. The prototype

The prototype is implemented on a Linux based system whose source code is available under the GNU copyleft license. The kernel version we use, release 2.2.14, provides the firewalling extension, which has become a standard feature of later kernel releases. Such an extension is optional, as it can be switched off at kernel compile time. It allows to specify IP packet filtering rules for the input, forward, and output packet chains, to add user-defined chains and to register user-defined firewalls. The input, forward, and output packet chains are managed by the `call_in_firewall`, `call_fw_firewall`, `call_out_firewall` routines, respectively, which scan the list of registered firewalls and

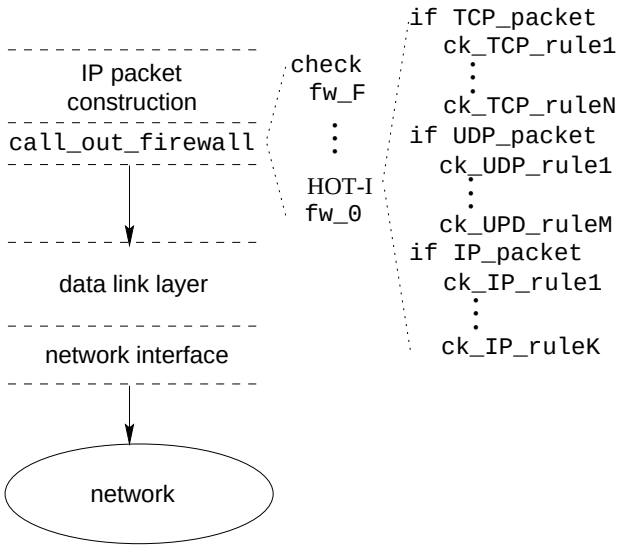

Figure 2. The Linux firewalling extension and the HOT-I module.

apply them to one packet at a time for the chain of interest. The `register_firewall` routine adds the firewalls defined by the user to the list at a priority level specified by the user and greater than 0, which is the system firewall priority level. Such a priority scheme allows to design an in-depth security architecture, as firewalls are checked in order of decreasing priority values. A packet is accepted and can proceed along its chain, skipping the next firewalls, when the current firewall returns the value `FW_ACCEPT`. A packet is passed on to the next firewall in the stack when the current one returns the value `FW_SKIP`. A packet is blocked and the next firewalls skipped when the returned value is `FW_BLOCK`.

In order to take advantage of the in-depth security architecture provided by the firewalling extension, we register our module at level 1, so that the `call_out_firewall` calls it before the system level firewall, as illustrated in Figure 2. Hostile packets are blocked by returning `FW_BLOCK`. Our module returns `FW_SKIP` if a packet is "acceptable" according to the rules implemented in it, so that the packet is passed on to the system firewall. This way, after the packet has passed the HOT-I checks, it has to go through the system firewall controls, if any are specified for the output chain. The host could be part of an intranet where a corporate level policy is specified and implemented using the system firewall. Such a policy could, for instance, prevent the use of telnet and ftp sessions. HOT-I would not block either type of packets as long as they are properly constructed. Because the purpose of the HOT-I module is to intercept hostile outgoing traffic, correctly constructed ftp or telnet packets should be blocked at a different level.

Note that HOT-I can operate independently of the Linux firewalling extension and can be ported to other Unix-based platforms that do not provide it. Furthermore, HOT-I rules can be disabled only when the system runs in single user mode. As systems usually run in multiuser mode, this means that the system must be rebooted in single user mode in order to disable the rules, which will also disable remote access and cut off the attacker.

As Figure 2 shows, HOT-I consists of a set of `if`'s that test various conditions, depending upon the type of protocol specified in the packet. Specific sets of rules are defined for each of the protocols TCP, UDP, and IP, that characterize known attacks from the point of view of the attack originator. The order in which such conditions are verified is critical from a performance perspective, as it can impact the module execution time.

3.2. The attacks

The HOT Interceptor is currently programmed to block a set of attacks comprising the most (in)famous and disruptive Denial of Service network attacks, namely SYN flood [2, 14], Smurf [10], Ping of Death [3], Land [4], and port scan. Blocking this type of attacks at the target is expensive and resource consuming, both in terms of network bandwidth and CPU time.

The common feature of all these attacks is the lack of strong authentication of the source address in IP packets that allows forged source addresses to be used. It allows attackers to protect their identity and often also damage an unaware indirect victim. Each of these attacks has a distinctive behavior.

- The TCP SYN flood attack is possible because of the limited backlog of uncompleted connections allowed during the establishment of a TCP connection when the three way handshake protocol is executed [14]. The attacker keeps sending TCP packets with the SYN flag on, at a sufficiently high rate so as to saturate the destination TCP server. Connections are never established as the SYN-ACK replies to the victim's ACK responses to the received SYN requests are never sent.

- The SMURF attack is possible because of the unrestrained use of the broadcast address. A conspicuous traffic of ICMP ECHO_REQUEST packets is sent to the IP broadcast address of a large network (the amplifier) with spoofed source addresses of another network (the victim). If the ECHO_REQUEST packets are delivered, most receiving hosts in the amplifier network will reply to the victim, i.e., the apparent source network, thus flooding it with ICMP ECHO_REPLY messages.

- The PING of DEATH attack is possible because of the lack of control on the size of control packets sent. Artificially crafted ping requests exceeding 64 KB in size are sent to the victim host. Because they exceed the MTU, oversized ICMP packets are fragmented at the source and then recombined at the destination. Reactions including crashing, hanging, and rebooting, may occur when systems receive oversized IP packets.

- The LAND attack is possible because packets with spoofed $< host, port >$ source address equal to the $< host, port >$ destination address are generally accepted although they lead the destination host to a lethal loop.

- The PORT SCAN attack is possible because any site can be freely contacted to see if a given service is available. With a port scan, such a control is performed systematically on all privileged ports and on some non-privileged ones where well known services run. This attack exists in two versions: the vanilla version uses stream sockets, the stealth version uses raw sockets and sends only the initial SYN packet. Although apparently innocuous, a port scan is usually the first step to collect information about the prospective victim site, as the identification active services is followed by the search for known vulnerabilities of those services.

While verifying the authenticity of a packet source address at the destination is quite difficult, it is an easy control to apply at the source itself. Our module checks all the system interfaces and their IP address and compares them to the source address of the packet[1]. This is the most expensive test to perform, as the host may have several interfaces. Such a rule is also recommended in the RFC 2267 as a proper router set up [8]. Although expensive to implement, simply blocking spoofed packets may be too strict a constraint and limit network management activities. Therefore, we always associate it to other conditions that characterize specific attacks.

In order to detect a SYN flood attack, the module checks for packets with the SYN flag on and a spoofed source address, as legitimate connections do not have spoofed source address. However, because an attacker may use a compromised machine and thus not care for the attack being traced back to it, we also consider SYN flood attacks with a legitimate source address. In order to block this type of attack, we also implement a control based on the frequency at which SYN packets are sent out. The threshold value of such a frequency that identifies an attack can be computed based on the number of half-open connections times the largest timeout defined in the TCP/IP specifications. In case of legitimate connections the ACK packet would be sent timely, thus we believe that the chances to hurt regular users are minimum, although false positives are still possible. For optimization reasons, spoofed SYN flood attacks are checked first. When the attack is detected, either spoofed or regular, the packet with the SYN flag on is dropped and the half-open connections, if any, completed with an RST packet.

The Smurf attack also could not be performed if spoofed addresses were not allowed, or the attackers would hang their network. A Smurf attack is recognized when source address spoofing is combined with the broadcast address of a network, as destination address, in order to flood both the destination network and the spoofed one. Similarly to the SYN flood attack, when an ICMP packet with the broadcast address in the destination field and a spoofed source address is seen, the packet is dropped.

The Ping of Death attack is blocked when oversized ICMP packets with (possibly) spoofed source address are identified in the outgoing traffic. Like with Smurf packets, Ping of Death packets are dropped.

The simple but dangerous Land attack can crash or hang the victim machine by sending to it packets with the same $< host, port >$ pair in the source and destination address fields. The ad hoc rule in this case checks for packets with the same destination and source address pairs.

In the case of port scan, the module recognizes the various stealth versions when it sees a TCP packet coming from a raw socket possibly with flags on. A check on the series of consecutive packets to the same destination with varying port numbers could be easily fooled by an attacker that separates the packet far enough in time.

4. Performance analysis

4.1. Preliminary considerations

When a tool or a set of tools like HOT-I is adopted, the performance impact of its operation cannot be overlooked, although system and global security is the main concern. In this section we present the results of a set of measurements of the prototype we implemented, in order to give an estimation of its impact on system performance.

The module we described in this paper affects the host network performance, in particular it can limit the host capability to sustain a given network traffic in terms of bandwidth. The peak traffic generation rate of a host with a given hardware architecture is a function of the latency, i.e., the execution time, of the network software components that complete the packet by adding all the control information in the packet header, such as address, checksum, flags, etc., to the payload. Note that the peak rate is not sustainable

[1] A statistical approach of the observed source addresses can be adopted to defeat possible changes of the computer IP address aimed at hiding the forged network traffic with spoofed source address.

in general because a host does not just generate packets but usually performs other activities. If a new routine is added to those components, the overall latency will increase, thus reducing the peak rate. The sustainable rate may not be significantly affected, however, because when real traffic is generated, the longest time might be spent processing the payload. In this case, the maximum throughput is limited by the time to process the payload rather than by the time to generate packets.

In the presence of HOT-I, outgoing packets are forced to flow through it, in order to perform a series of tests that verify their "harmlessness." Strategies have been adopted to minimize its impact when the host operates legitimately. If at some point the outgoing traffic becomes hostile, the impact of our module becomes non-negligible and network performance degrades. On the other hand, degrading the network performance of a system that is generating hostile traffic, although negative for the honest users of the system, has a positive effect on global security, as it slows down the rate at which the host might succeed to send out hostile packets. Another important consideration is that, except for servers, the outgoing traffic of a host connected on the Internet is a significantly small part of its overall network traffic, the large majority being input or at most forward traffic. This does not apply to servers, however, as they provide the information available on the network, thus their bandwidth usage is due to output traffic mostly. The size of a http request is typically few hundreds bytes while the average size of a page is few thousands bytes. If all the outgoing traffic of a server were to be checked by a tool like ours, server performance would degrade. In this case, the most efficient way to implement a filter like HOT-I that could sustain the heavy traffic of a server or a network components, e.g., a router, is to hardwire the controls in a chip.

On the other hand, because of their importance, servers are generally well protected and managed systems, which makes them unlikely to become sources of attacks without their administrators realizing it. Our module, on the contrary, is targeted to the protection of the global network from the potential misuse of individual machines that may be compromised without their owners realizing it and used to launch attacks.

4.2. Experimental results

We now describe the system used as experimental platform and then present the set of experiments we ran to collect the measurements we present in this section. The experimental platform we used is an Intel Pentium I processor with 120MHz clock, 48 MB RAM memory, running Debian GNU/Linux operating system with 2.2.14 kernel with firewalling support.

We measured the impact of the module on the time a packet spends in the IP stack by instrumenting the native IP routines `ip_queue_xmit` and `ip_build_xmit` that process packets generated via stream/datagram and raw sockets, respectively. We focus on these routines as individual servers and measure how much their execution time increases because of the module. This is not, however, the total time a packet spends within the TCP/IP stack. Our measurements do not take into account possible queueing delays a packet may suffer once it is enqueued by an application at the network protocols interface. Such a delay is caused by the possible congestion at network protocol level caused by the outgoing traffic generated by all the applications running on the system. The presence of the module might contribute to increase the congestion by slowing down the network level protocols.

We ran two sets of tests, with and without the module, with no other load on the system but the packet traffic generator. The first set of tests is the base case. It consists of legitimate traffic only, without the module. The measured execution times for the two instrumented routines are compared against the execution times when the module is installed and running on the system. For each run, we generate 1000 packets and compute the average measured execution time. The standard deviation computed on the collected data is not reported in the following tables because it is negligible. The second set of tests is run with a mix of hostile and legitimate traffic. In this case too the observed measurements are quite stable, so we do not report the standard deviation of the observed averages. We separate the measured execution times for legitimate traffic, which are the same as in the case above, and those for the hostile traffic. In the table below only the latter are reported.

As the upper table in Table 1 shows, in case of legitimate traffic only, i.e., all packets have been properly generated according to some known protocol, the delay introduced by HOT-I in case of stream/datagram sockets is in the order of 5%. If raw sockets are used, i.e., a user defined protocol is running, the range of execution time inflation is wider, as it depends on the number of conditions to test for each packet.

When a flow of hostile and legitimate traffic is generated, the execution times in the native system, i.e., with no module, are the same as no filtering rules are applied against it. In the presence of the module, the delay it introduces is a function of the type of packet, i.e., the type of attack, analyzed. In our implementation, the most expensive condition to test is the one for source address spoofing, as the module cycles through all the system interfaces and checks their IP addresses against the one in the packet. The lightest condition to test is the presence of the SYN flag on. Depending on how many cases must be tested, either fully or partially, before finding a matching one, execution times may vary on the given range.

The results provided in this section are just an indica-

with legal traffic only

	NATIVE	W HOT-I
stream/dgram socket	30	31.4
raw socket	50	55-81.7

with mixed (hostile and legal) traffic

	NATIVE	W HOT-I
stream/dgram socket	30	31.4
raw socket	50	60-100

Table 1. Execution times in ms of the packet construction routines at IP level with and without the HOT-I module in case of regular and raw sockets.

tion that the HOT-I approach is feasible and not disruptive of system performance, especially from the client point of view. Clients usually have heavier input traffic, from ftp downloads, http get requests, etc., than output traffic. The same is not true for servers. However, servers are usually more protected and better guarded against attacks than end users' clients, which are the typical targets of attackers' searches for victim machines to be exploited as the actual sources of Denial of Service attacks.

5. Conclusions and future developments

In this paper we have presented a complementary approach to computer and network defense from security threats. It consists in applying a set of protection rules that restrain hostile activities of hosts that could otherwise be easily exploited. Such an approach can be seen as an indirect form of defense as, by limiting the harm a host can do, it contributes to make the network a safer place for all the nodes that are connected to it. Its applicability to the analysis of possibly hostile outgoing traffic has been shown by the implementation of a prototype. Measurements of the impact on system performance have been presented for a limited set of attacks.

The directions of future research cover two main lines, namely performance optimization and the extension of the recognizable attack set. As for the latter, we are currently working on the characterization of the set of attacks that can be blocked more effectively at the source. The attack signatures as perceived at the victim site, e.g., the signatures used by NIDS, are also under analysis in order to adapt them to the originator's perspective. Furthermore, the possibility to cope with the distribution of malicious code, such as viruses, is being considered as one of the extensions to full or partial packet inspection.

References

[1] Bruschi D., Rosti E., "Disarming offense to facilitate defense," Computer Science Department TR 251-00, Università degli Studi di Milano, April 2000, to appear at "New Security Paradigms Workshop," Cork, Ireland, Sept 2000.

[2] CERT-CC, "TCP SYN flooding attacks and IP Spoofing attacks," CERT Advisory CA-96.21, http://www.cert.org, 1996-98.

[3] CERT-CC, "Denial of service attack via ping," CERT Advisory CA-96.26, http://www.cert.org, 1996-97.

[4] CERT-CC, "IP Denial of service attacks," CERT Advisory CA-97.28, http://www.cert.org, 1997-98.

[5] Dittrich D., "The DoS Project's "trinoo" distributed denial of service attack tool," http://staff.washington.edu/dittrich/misc/trinoo.analysis, 1999.

[6] Dittrich D., "The "Tribe Flood Network" distributed denial of service attack tool," http://staff.washington.edu/dittrich/misc/tfn.analysis, 1999.

[7] Dittrich D., "The "stacheldraht" distributed denial of service attack tool," http://staff.washington.edu/-dittrich/misc/stacheldraht.analysis, 1999.

[8] Ferguson P., Senie D., "Network ingress filtering: defeating denial of service attacks which employ IP source address spoofing," Network Working Group RFC 2267, http://www.rfc-editor.org/rfc/rfc2267.txt, January 1998.

[9] S. Garfinkel, E. Spafford, **Practical UNIX and Internet Security**, O'Reilly, 1996.

[10] Huegen C., "The latest in denial of service attacks: smurf-
ing. Description and information to minimize effects," http://users.quadrunner.com/chuegen/smurf.cgi, last update Feb. 2000.

[11] J. Kleinwaechter, "The limitations of intrusion detection systems on high speed networks," presented at the "*First International Workshop on Recent Advances in Intrusion Detection (RAID),*" http://www.zurich.ibm.com/˜dac/RAID98, Louvain La Neuve, Belgium, Sept. 1998.

[12] M. Roesch, “Snort - Lightweight Intrusion Detection for Networks,” *Proc. of the 13th System Administration Conference – LISA ’99*, Seattle, WA, Nov. 1999.

[13] SANS Institute, “Egress Filtering v 0.2,” http://www.sans.org/y2k/egress.htm.

[14] Schuba C.L., Krsul I.V., Kuhn M.G., Spafford E.H., Sundaram A., Zamboni D., “Analysis of a denial of service attack on TCP,” *Proc. of the 1997 IEEE Symposium on Security and Privacy*, pp 208-223, Oakland, May 1997.

[15] Tripunitara M.V., Dutta P., “A middleware approach to asynchronous and backward compatible detection and prevention of ARP cache poisoning,” *Proc. of the 15th Annual Computer Security Applications Conference*, pp 303-309, Dec. 1999.

[16] Wired.com, “A frenzy of hacking attacks,” http://www.wired.com/news/business/0,1367,34234,00.html, Feb. 2000.