Disarming offense to facilitate defense*

Danilo Bruschi, Emilia Rosti
Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano
Via Comelico 39, 20135 Milano – Italy
{bruschi, rose}@dsi.unimi.it

ABSTRACT

Computer security has traditionally focused on system defense, concentrating on protection and recovery of victim machines. Moving from the opposite perspective, we propose a complementary approach that focuses on limiting the attacking capabilities of the hosts. Software design and implementation weaknesses usually are at the basis of computer offensive capacities. Since software redesign or patching on an extensive basis is not possible, we propose the adoption of a filtering strategy to block abuse attempts at the originating machines. As an example, applications of such an approach are presented at host level, in order to prevent root compromise attacks, and at network level, in order to prevent DoS attacks, among others.

The proposed solution is not a silver bullet and could be bypassed by sophisticated users. However, we believe it can effectively restrain the offensive capabilities of hosts that could be easily seized by crackers. We discuss the pros and cons of the proposed solution and present an application to host and network security.

Keywords

Computer and network security, defense, offense, disarm, attack, monitor

1. INTRODUCTION

Since its origins in the early '60, computer security has focused on system defense and protection of victim machines. A variety of tools and methodologies have been proposed, many of which proved to be quite effective in protecting systems and networks from intruders. As the computing paradigm started shifting from the host to the network in the mid to late '80 to become a full scale reality in the early '90, the focus of computer security should have shifted too. In the networked environment, "pacifist" hosts can suddenly and, sometimes, involuntarily become attackers, that is, a threat for the entire community. Million powerful computers that are just used to exchange emails or chat are connected on the network 24 hours a day, 7 days a week via cable modems or ASDL lines. Even worse is the scenario that will see embedded systems as powerful as fully equipped computers, e.g., playstations, will be constantly connected on the network for game updates and downloads. All these systems tend to be unattended as computer security is still an esoteric discipline for the average user, thus making such hosts easy target of attacks. Furthermore, a large number of hosts is and will keep probing other hosts on the network in order to find unattended ones that can be easily seized and used to attack yet other hosts. The network as an entity should be protected and made less dangerous.

Another consequence of the computer paradigm shift is the exacerbation of two security related problems:

- liability issues: there are countries, like Italy, where computer owners are liable for all the actions executed by their systems. This implies that they can be legally prosecuted when attacks to another system are launched from their computers, although intruders who had previously gained access to their computers, are responsible for the attack;
- sophisticated intrusion tools: a clever exploitation of the network centered computing paradigm with the realization that the "network is the computer" has lead to the development of distributed intrusion tools, which recently showed to be quite effective in launching DoS attacks at high profile Web sites.

Computer security does not seem to have an answer to these problems other than "improve your protections." On the other hand, it is becoming harder and harder for nowadays typical protections, such as firewalls and IDS's, to keep up with the ever increasing speed of network components, as processors cannot process packets fast enough. Furthermore the diffusion of encryption products that operate either at transport or network layer increases the complexity of the controls firewalls and IDS must apply. While networked computers are a mass market off the shelf commodity, or at least they tend to be used as such, computer security is not, not yet, neither technically nor economically.

In this paper we propose a new approach to address security

*This work was partially supported by the Italian M.U.R.S.T. 60% project.

New Security Paradigm Workshop 9/00 Ballycotton, Co. Cork, Ireland

problems. We move from the observation that a computer may as well be a victim and an attacker. Thus, if we want to improve security, we should not only protect our systems but also reduce threats, i.e., prevent systems from doing any harm. In a network where no, or just a few, hosts are a threat, global security results from individual harmlessness.

This paper is organized as follows. The new paradigm we propose is illustrated in Section 2. An application against network attacks is outlined in Section 3. In Section 4 the proposed approach is illustrated with the application to a host attack. Related work is presented in Section 5. The proposed approach is discussed in Section 6. Section 7 concludes the paper and outlines directions of future research.

2. DISARMING COMPUTERS

Based on the observation that reducing threats is another way to improve security, we propose a new research direction for computer security whose main goal is the definition of new techniques and methodologies for building non-offending, or *disarmed* computers. We define a disarmed host as the following:

> *a disarmed host is a host equipped with tools that turn off the host attacking capabilities and that force the host to be re-installed for it to be subverted.*

Offending capabilities may be turned off by tools that operate as filters that monitor the host activity and block it when it does not conform to a "good" behavior or, vice-versa, when it matches an "anomalous" behavior, depending on the approach followed. Such filters can be thought of as attack inhibitors that behave like intrusion detection systems but they are placed on the attacking host to block hostile activities. Our approach can be thought of as "extrusion" detection, where for extrusion we mean the attempt to attack, i.e., accomplish an intrusion. Although it is almost impossible to tell attacks from legitimate behavior in general, there are cases where a certain behavior can be clearly identified as offensive. We would like to block the latter, at least. Our mechanism cannot address DoS purely based on the quantity of honest packets.

With the introduction of our approach, attacks can be divided into two classes: attacks that can (easier and most successfully) be prevented at the source and attacks that can be blocked at the destination. The latter are well known to the security community since they have been among the major subjects of computer security studies. Their characterization has lead to the definition of signatures databases for intrusion detection systems. On the contrary, little if any interest has been shown for attack characterization at the source in order to block offending activities as they are being performed at the source. Although the two classes of attacks have a large intersection, they are different. As an example, IP spoofing is easier to detect at the source but can hardly ever be detected at the destination, even if heuristics such as DNS reverse lookup may be adopted to discover the spoofing in most cases.

We believe that, at the current state of the art, the disarming technology can be easily adopted in local environments, where operating systems can be installed and monitored centrally. In this case, its deployment can provide an effective solution to problems such as:

- liability: from a legal point of view, a disarmed computer could be considered adequately configured to comply with the law imposing that hosts not be attack sources, thus relieving the owner from liability issues;
- intranet security: disarming filters can protect an intranet from insiders' attacks and can help preventing insiders from using the internal hosts to attack computers outside the intranet perimeter. They are transparent to final users and applications, thus they do not require application customization, as it is the case with well known access control systems such as Kerberos [20].

The large scale deployment of a disarming technology would contribute to relieve the following problems:

- distributed tools for intrusion: with our approach, the network remains the "new computer" but not for intruders, who would have difficulties in finding hosts where their agents for distributed attacks could be installed;
- firewalls and IDS's: they can be easily circumvented by encrypted traffic, which prevents them from detecting attacks performed using it. The only way to block such attacks is to intercept the packets before they are encrypted, i.e., at the source;
- security tools performance: because a good part of the hosts on the Internet would operate honestly and fairly, thus never attacking other hosts, new and faster schemes can be investigated to identify packets originated from fair hosts. The spare time remaining could be actively used by firewalls and IDS's to deal with increased network speed [16].

Imposing such an approach, however, on a geographic scale and have it work is not an immediate task to realize but we believe it to be a reasonable one. Although this kind of disarming filters could be bypassed by sophisticated users like any software protection, they could be an effective protection against abuses by unexperienced users (the so called "script kiddies") that use ready made exploit programs down loaded from well known Internet sites. The cases considered in Section 3 provide an example of similar cases. Since we will implement them as kernel modules, an intruder who wants to bypass them would have to install a stripped version of the operating system, which may not be so immediate to do for unexperienced users. Furthermore, in order to be able to use successfully compromised victim computers to launch attacks, the OS should be reinstalled, which is quite conspicuous a task to perform to go unnoticed. On the other hand, a hardware implementation based on ASIC technology would overcome all these objections and will be

investigated as encouraging results will be obtained and the approach further refined.

As an example, in this paper we design two tools that can be used to block several well known attacks at the source, i.e., disarm the hosts with respect to the considered attacks. In particular, we propose a filter for blocking popular Denial of Service attacks. The solutions we present requires that a set of functionalities be added to a kernel device driver in order to detect harmful packets characterizing an attack in the outgoing flow. We also discuss a solution to avoid some cases of buffer overflows thus preventing intruders from exploiting such a vulnerability in order to get unauthorized access to a machine or possibly increase their privileges. Note that the latter represent an indirect form of disarming a computer, as in the first place it protects the local host from root compromises. Since root compromise is often the initial step of an attack launched from a victim host, preventing it represents a form of disarming the computer.

3. BLOCKING NETWORK ATTACKS

In this section we describe the design of a disarming filter against network attacks. Filtering components could be added as middleware between the device drivers and the kernel, so that they can monitor all the outgoing traffic, without changing the existing applications. They can be executed on host computers as well as network components such as routers. The packet flow is checked against attack signatures and blocked when an attack attempt is detected, similarly to what an IDS would do at the target host on the incoming traffic. The filters apply packet control rules based on a signatures that represent the 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 filters, i.e., the less false negative and false positive signals the filters will send. A separate module that handles critical situations, e.g., by raising alarm, suspending the allegedly offending program, or sending messages to the superuser according to a defined policy, is signaled by the filter whenever a tentative attack is detected.

Among the most (in)famous and disruptive network attacks are Denial of Service attacks such as SYN flood [8], Smurf [14], Ping of Death [9], Land [10], Teardrop [10]. 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 the crackers to protect their identity and often also damage an unaware indirect victim. Additionally, each of these attacks has a distinctive behavior. At the basis of the SYN flood attack is the limited backlog of uncompleted connections allowed during the establishment of a TCP connection when the three way handshake protocol is executed. The unrestrained use of the broadcast address is at the basis of the Smurf attack. The possibility to send oversized control packets is at the basis of the Ping of Death attack. The possibility of spoofing the $< host, port >$ source address and setting it equal to the $< host, port >$ destination address thus leading the victim host to a possibly lethal loop is at the basis of the Land attack. The need to fragment and re-assembly packets exceeding the minimum MTU of the intermediate networks traversed along the route from source to destination is at the basis of the Teardrop attack. We illustrate here how the middleware approach we propose can be employed to prevent a machine from launching some of these attacks or at least to mitigate their severity and impact on the target machine.

3.1 Source Address Spoofing

While verifying the authenticity of a packet source address at the destination is quite difficult, it is very easy to do it at the source itself. The filter we propose can apply the simple filtering rule [12] that prevents packets with a source address different from the one of the local machine to be passed to the network. Only packets carrying the proper source address, i.e., the one of the machine actually sending the packets, are allowed to the network card[1].

This simple rule is very strict and could limit network management activities, although it is sufficient to turn off most denial of service attacks as they usually forge packets with spoofed IP source addresses. Ad hoc less strict filtering rules can be adopted that verify the simultaneous presence of a spoofed IP source address and other attack specific conditions.

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

The Smurf attack also could not be performed if spoofed addresses were not allowed, or the attacker would hang his/her network. In this case, spoofing is combined with the abuse of the broadcast address of a network, i.e., address 255, in order to flood two networks. 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 will reply to the victim, thus flooding the alleged source network with ICMP ECHO_REPLY messages. The specific rule against the smurf attack would check for a spoofed source address associated with a broadcast destination address.

3.2 Uncompleted Connections

For a detailed analysis of the TCP/IP SYN flood attack, we refer the interested reader to previous works appeared in the literature (e.g., [24]). Critical factors for the success of this attack are the following:

1. the initiator of the bogus TCP/IP connections sends only the SYN of the SYN and ACK messages that it must send in order to complete the three-way handshake protocol. The initiator never replies with the expected ACK to the victim's SYN+ACK reply;

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

2. new bogus connections must be initiated by the attacking machine at a faster rate than the target machine's TCP timeout.

Because connection requests usually have spoofed source addresses of hosts that are not reachable from the victim, the traffic originated by a SYN flood attempt could be blocked by the simple filter against spoofing the source addresses. However, since this is a mere implementation technicality that is usually performed in order to disguise the attacker's real identity, someone might try a SYN flood using the authentic source address. In this case, the attack should be blocked based on the conditions that characterize it.

In order to detect an excessive number of half-open TCP/IP connections, the middleware monitors all the TCP connections requests sent to each machine and keeps a counter, on a per user basis or on a system basis. If the number of half-open TCP/IP connections to a single machine and the rate at which they are initiated exceed given thresholds, the middleware completes all the pending requests with an RST packet and blocks further connections to that destination for a period of time. The duration of such a period can be computed to be larger or equal to the number of half-open connections times the largest timeout defined in the TCP/IP specifications. Because in case of legitimate connections the ACK packet would be sent timely, we believe that the chances to hurt regular users are minimum, although false positives are still possible.

3.3 Oversize Packets

Although the IP specifications indicate a maximum packet size of 65535 bytes, dimension checks are not enforced either at the source or at the destination to prevent the destination to overflow its buffer when it reassembles fragmented packets. Teardrop and Ping of Death are examples of attacks exploiting the oversize packet vulnerability. It is very easy for the filter to check the packet size and block packets that exceed the maximum packet size.

3.4 Implementation

A prototype, HOstile Traffic Interceptor (HOT-I) [6], implementing the proposed filtering strategy has been developed on a Linux based system, kernel version 2.2.14, in order to demonstrate its feasibility and effectiveness. HOT-I operates at the IP layer as a static kernel module in order to prevent its easy removal. It applies packet filtering rules to the outgoing packets when they are ready to be passed on to the data link layer. The packet flow is checked against attack signatures of known attacks and blocked when an attack attempt is detected. In the current version, in case a hostile packet is detected, it is dropped by default. 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.

In order to take advantage of the in-depth security architecture provided by the firewalling extension of the Linux kernel [4], we register our module at level 1. Therefore, the routine that calls the registered firewalls, `call_out_firewall`, calls it before the system level firewall, if defined. Our module returns `FW_SKIP` if a packet is Packets accepted by HOT-I would still have to go through the system level firewall, if defined. Note that the controls acceptable packets for our module may not be acceptable by the system firewall if this one is used to implement a corporate defined policy, e.g., no ftp or telnet out of the perimeter defined by the firewall.

HOT-I is currently programmed to block a set of attacks comprising SYN flood, Smurf, Ping of Death, Land, and port scan. Preliminary performance results have been collected by instrumenting the IP level routines where HOT-I is called. The impact of the module on the time a packet spends in the IP stack is a function of the number of rules that must be checked before being able to make the right decision about the packet under examination. In case of legitimate traffic, the impact of the module is in the range of 5% of the processing time in the absence of the module. In case of hostile traffic, the delay introduced increases up to 50%. Optimization are under investigation in order to minimize performance degradation.

4. BLOCKING HOST ATTACKS

In this section we show how the disarming approach can be applied to protect a host from buffer overflow based root compromise, as this is a popular way to gain superuser privileges on a host and then start an attack from there. The filter we design requires minor kernel modifications in order to implement simple checks on some variables. The filter compares the behavior of the currently executing program as characterized by the value of a small set of parameters against the conditions we identify as characterizing a buffer overflow attempt. In this case, the disarming filters we propose are implemented as a set of kernel modifications.

Buffer overflows are still one of the most popular ways to perform a root compromise, i.e., the illegal acquisition of superuser privileges by an ordinary user. By overflowing a buffer of a setuid to root program [7] with a properly crafted content, the user executing that program may launch the execution of any command with superuser privileges. For the details about the "art" of writing buffer overflow exploits, we refer the interested reader to the wealth of publications on the issue (e.g., [1, 19]). Various pro-active and reactive solutions have been proposed that are characterized by different targets, i.e., source or executable code (e.g., [26, 11, 25, 5]). Unlike these, our suggestion is not a general solution to the problem but rather a point solution for some specific well known cases. We believe that with further research efforts it may be turned into a general solution to the buffer overflow problem.

In order to devise a filtering scheme that will protect the system from root compromise via buffer overrun, we characterize the necessary conditions for a buffer overrun to succeed. The critical components of such an attack are the setuid to root program and the possibility of passing adequately crafted inputs that will force the program to execute some piece of code while the program effective user id is 0 (i.e., root).

Exploiting the setuid feature is critical to privilege acquisi-

tion. Buffer overflow exploits usually have a process spawn a new process by forcing the execution of the exec system call, with the shell command interpreter /bin/sh as argument. An exec system call changes the code of the executing process to the code received as argument. The new process replaces the one that executed the exec and inherits from the latter its PID and real and effective uid's (RUID and SUID). When a root compromise occurs via buffer overflow, the process created by the exec has SUID = 0, i.e., is a superuser process.

In order to block this kind of attack, the exec system call can be modified so that it controls the EUID of the calling process. In case it is 0, the code of the object program to be executed is examined before loading it looking for a signature of the /bin/sh command, i.e., a characteristic sequence of bits in the executable code that identifies it. If the pattern matching is successful and the signature is found, an alarm is raised and the allegedly malicious program executing the exec is suspended. Provisions to handle the su root command and the login program should be made. As a matter of fact, executing such a command leads to a configuration where all the conditions above are satisfied but the program is legitimate.

The presented strategy can also be applied to prevent the exploitation of setgid programs, i.e., programs that change their group id during execution. Although not as critical as root compromise, buffer overflows on setgid programs can be a first step towards it or cause other problems [13].

5. RELATED WORK

In this section we compare the proposed filters with existing solutions that exhibit a certain degree of similarity and discuss the differences.

The first tool, from a chronological point of view, that was proposed to control actions performed by the system on a set of objects is the reference monitor [2, 17]. The reference monitor is the part of a security kernel that controls accesses to objects. It comprises access controls for devices, files, interprocess communication, memory, and all objects that may require access control. It is the single point of access to the objects it controls and cannot be modified nor circumvented. In order to enforce security it must function correctly, therefore it is usually small enough to be analyzed and tested thoroughly. The characteristics of the reference monitor are the following: it is always invoked whenever an operation on an object is performed, it is tamperproof, and it is small enough to be proved correct, secure, and complete.

A network filter like the one we described in Section 3 resembles a reference monitor for the network. It is always invoked whenever a packet is to be sent out. It is tamperproof or impossible to circumvent by a rogue process, as the only way it could be removed from the system is to install a different version of the operating system. However, because it is based on heuristics for attacks identification, it is impossible to prove it always work correctly. Thus it is not exactly a reference monitor, although it behaves as such. The filter against buffer overflows too shares some of the properties of the reference monitor, but like in the network case, it is prone to false positives, therefore cannot be proved to always work correctly.

Another approach to system security management similar to the one proposed here is the one adopted in CORBA [21]. Access control in CORBA environments is enforced by a system object called AccessDecisionObject. AccessDecisionObject consults two other objects when making a decision: the user's Credentials object (containing the privilege attributes of the user who is trying to access a resource) and the AccessPolicy object which applies to the resource being accessed (this AccessPolicy object is an attribute of a security policy domain which the resource belongs to). The AccessDecisionObject compares the privilege attributes in the user's credentials against those required for access to the object as specified in the AccessPolicy object, and makes a yes/no decision. Both the sending machine (the one from which the user's request originates) and the receiving machine have AccessDecisionObjects. The ORB (Object Request Broker – basically the object-oriented Network Operating System) on the client system can retrieve the same AccessPolicy object as the target system, and it can make an access decision based on the same information. Of course, the client's system may not be trusted by the target system, so the target will go ahead and make an access decision regardless of what the client system has decided. But in the case in which the client's system is trustworthy, and makes a valid decision to deny access, the request will be aborted inside the client system and will never be sent over the network, thus saving both network bandwidth and server processing.

In this case, the main difference is that, rather than checking for credentials and access permissions, our filters check for some predefined behavioral patterns or set of information in the packets or in the code to be executed. The policy enforced in this case may lead to false positive.

A recent family of tools, so called "personal firewalls," have been gaining larger popularity in the PC world, see e.g., [28, 22, 18, 27, 29], both as commercial products marketed by various vendors and as free software in various configurations. Besides the traditional protection from intrusions offered by all firewalls and the associated logging facilities, this kind of tools usually provide a varied set of additional functionalities. Among these functionalities are private information protection, by alerting the user every time one piece of such information is about to be sent across an insecure channel or by preventing web servers from retrieving personal information, e.g., email address, in background while browsing the network. Protection from mobile code such as Java applets and ActiveX controls, and from state related information, such as cookies, may also be offered. Some of them also integrate their firewall capabilities with antivirus functionalities. When installing such tools, the user is usually required to edit configuration settings to various extents, ranging from the choice of security level (low, medium, high) to the specification of ports and/or protocols allowed or denied on the machine. In the Linux world, packet filtering firewalling functionalities are provided by IPCHAINS, a kernel extension that is now part of the mainstream kernel distribution. In this case too, if not more, the user is required to specify various configuration settings and the firewall rules.

What we suggest with our approach is a patch to the kernel that cannot be removed unless the system is rebooted and a "clean" version of the OS is installed, and that the user does not need to be aware of in terms of configuration settings. Although installing a different version of the OS is not an unnoticeable operation, it is possible for hackers to develop scripts that will launch a "clean" install after the first spontaneous reboot of the machine, so as not to attract the user's attention. Nothing but preventing code downloads can be done to block such a hacker countermeasure to our approach.

6. DISCUSSION

In this section we discuss possible criticisms to our approach. Some of them have been briefly mentioned in the text already. We recall and collect them all here.

Because it is hard, if not impossible at all, to tell attacks from legitimate uses of a host, it seems impossible to prevent harmful behaviors. Although this is true in general, there are cases where it is easier to foresee possible danger in a certain stream of traffic, e.g., sending out spoofed packets, or in a certain set of actions. In such cases, blocking the action under execution is a safe reaction. It is true, however, that there are apparently legitimate behaviors that may result in an attack. Nothing can be done to prevent them, since they cannot be told apart from actually legitimate traffic.

Deploying a system like the one proposed in this paper may not be economically feasible or viable. Marketing problems due to what may be perceived as "reduced" functionalities may make it even harder. Liability suits may on the other hand make it economically viable if such a protection is values as sufficient by the legislation. Different countries have different legislations, this makes it difficult to claim absolute utility from the legal point of view of a tool like the one proposed here.

The use of a tool like the one proposed here would prevent the use of mobile IP as this protocol is based on the possibility of sending out spoofed IP packets from the base station. While this is true, we advocate the adoption of the proposed approach to protect home computers from easy exploitation. In case of mobile IP, we would expect the base station to be a reasonably administered server properly configured with other forms of protection. In case the proposed protection were in place, a customized version could be adopted that is automatically updated by the hand-off procedure so as to allow spoofed packets with the IP address of the mobile station. Such a list of temporarily spoofed connections would be updated upon a mobile station leaving/entering the cell under the base station control.

The protection against buffer overflows briefly outlined in Section 4 is not meant to be a general solution against the problem per se. Therefore, we do not expect this approach to be as effective as general solutions like Stackguard [11].

The system is aimed at protecting innocent and unexperienced users from being exploited by skilled ones that successfully attack the formers' hosts in order to launder their connections or start real attacks from there. It could also be very effective in protecting intranets from insiders launching attacks. Depending upon the jurisdiction and the legal system, our system may be a sufficient protection measure in both cases from the liability associated with being a source of attack.

An aspect that might be worth investigating is the kind of feedback a hacker receives indirectly from the presence of a module like the one we propose. It would be interesting to study the hackers' reaction, whether they would persist more or would be discouraged faster if they realized that a local protection like the one we propose.

A more radical approach to preventing some anomalous and malicious behaviors could be to patch the OS rather than distribute pieces of code that try to do the same. Although this would be the optimal strategy, we note that it does not affect the paradigm we propose.

7. FUTURE DEVELOPMENTS

In this paper we have proposed a new strategy to deal with computer security problems based on limiting attacking capabilities of systems. Its applicability to the specific cases of some network DoS attacks and buffer overflow attacks has been illustrated. An implementation of the kernel patch for a network filter for the various attacks considered has been briefly described together with the results of preliminary performance analysis. The implementation of the proposed strategy to prevent root compromise via buffer overflow is about to start at our laboratory.

Various issues remain open and are subject of current investigation:

- analysis and the characterization of other types of attacks;
- unique specification and description of attacks so as to eliminate false positives;
- mechanism to extend the set of rules that handle new attacks;
- preventing the module from being easily circumvented;
- porting to other platforms, i.e., Microsoft systems.

Ackowledgements

The authors are grateful to all the participants in the New Security Paradigms 2000, whose comments and discussion during the workshop have significantly improved the quality of this paper. A special thank goes to Bob Blakley, for providing excellent notes of the discussion and the references about CORBA.

8. REFERENCES

[1] Aleph One, "Smashing the stack for fun and profit," *Phrack Magazine* 49, Fall 1997.

[2] Anderson J., "Computer security technology plannng study," *U.S. Air Force Electronic System Division Technical Report* 73-51, October 1972.

[3] Bach M.J., **The design of the Unix operating system**, Prentice-Hall Int. Series, 1987.

[4] Bandel D. "**Linux Security Toolkit**," IDG Books, 2000.

[5] Banfi R., Bruschi D., Rosti E., "A tool for pro-active defense against the buffer overrun attack," *Proc. of ESORICS '98*, LNCS 1485, pp. 17-31, 1998.

[6] Bruschi D., Cavallaro L., Rosti E., "Less harm, less worry or hot to improve network security by bounding system offensiveness," *ACSAC '00, 16th Annual Computer Security Application Conference*, New Orleans, Dec. 2000.

[7] S. Bunch, "The setuid feature in Unix and security," Proceedings of the *10th National Security Conference*, 1987.

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

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

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

[11] C. Cowan, C. Pu, D. Maier, H. Hinton, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, Q. Ziang, "StackGuard: automatic adaptive detection and prevention of buffer-overflow attacks," Proc. of the *7th USENIX Security Symposium*, 1998.

[12] 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.

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

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

[15] R. Jones, P. Kelly, "Bounds checking for C," http://www-ala.doc.ic.ac.uk/phjk/BoundsChecking.html, July 1995.

[16] 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.

[17] Lampson B., "Protection," republished in *Proc. of the 5th Princeton Symposium, Operating System Review*, Vol 8, No 1, pp 18-24, Jan. 1974.

[18] McAfee, "McAfee.com Personal Firewall," http://www.mcafee.com/, 2000.

[19] Mudge, "How to write buffer overflow," http://www.l0pht.com/advisories/bufero.html 1997.

[20] B.C. Neuman, T. Ts'o, "Kerberos: an authentication service for computer networks," *IEEE Communications*, Vol 32, No 9, pp 33-38, 1994.

[21] Object Management Group, "CORBAsecurity Service version 1.5," OMG, 2000.

[22] Open Door Networks, "Door Stop Personal Firewall," http://www2.opendoor.com/doorstop/, 2000.

[23] Russel P.R., "Linux IPCHAINS-HOWTO," http://www.linuxdoc.org, 2000.

[24] 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.

[25] A. Snarskii, "FreeBSD Stack integrity patch," ftp://ftp.lucky.net/pub/unix/local/libc-letter, 1997.

[26] Solar Designer, "Non-Executable user stack," http://www.false.com/security/linux-stack/.

[27] Sygate, "Sygate Personal Firewall," htpp://www.sygate.com, 2000.

[28] Symantec, "Norton Personal Firewall 2001," http://www.symantec.com/, 2000.

[29] TinySoftware, "Tiny Personal Firewall," http://www.tinysoftware.com, 2000.