ANALYSIS OF INTRUSION DETECTION TOOLS AND TECHNIQUES

STEFANO MARINELLI

Abstract. An introduction to network system security, mainly focused on the analysis of several IDS tools and techniques.

Part 1. Introduction to System Security

1. A General Introduction

One of the most important tasks of a System Administrator is to keep the machine safe from information disruption. This includes hardware failures, user errors, information incoherence, attacks from outside, and many other troubles. Too many to be listed, too many to be solved. While it's almost impossible to prevent hardware failures (e.g.: if the hard disk stops working, there's nothing you can do to recover the data, except from sending it to a specialized lab and pay a lot of money), you can do something to protect your system and your network. The most important thing to remember is that nothing is perfect, so your well-working computer in your fast and reliable network could suddenly say "goodbye"; the best assumption to do is that you can never feel safe. It could sound catastrophic, but that is.

A lot of things have been done during the last years, and now we have some valid solutions to help us. For example, we have the RAID for the hard disks, Firewalls for network security, and so on. But they can't always be perfect. Firewalls offer a first protection against external attacks, but can't do much if the attack comes from our machine or network. So we will focus our attention on **intrusions**, *actions that attempt to compromise the integrity, confidentiality or availability of a particular resource.* There are many ways to protect from unauthorized intrusions: we could try to prevent them or we can try to detect them. The first thing to try is, in fact, to keep unfair users out of our resources, but there are a lot of situations in which it's not completely possible. Setting a strict security policy could bring to an unusable system, and regular users could have hard times to keep working. On the other hand, setting a fair policy could expose our precious devices to a lot of attacks and our important *bytes* could be compromised. We want to prevent it, as information integrity and reliability is our main task.

2. System Administrator's Life and Duties

A general System Administrator's life could be described in some steps:

(1) Install the System
(2) Configure the System
(3) Spend hours,days,weeks to check the configuration

Date: 19/12/2002.

(4) Set correct permissions to programs/users keeping in mind the PLP, described in proposition 2.1
(5) Put the system on-line and check what happens, keeping an eye on any user and on anything happening
(6) Protect the system from unauthorized and malicious users
(7) Check data integrity
(8) While (system!=broken) goto step 6
(9) Repair system if broken

Proposition 2.1. *Principle of Least Privilege: Each program and each user should operate using the minimum set of privileges needed to perform his/its job.*

The above principle is simple: giving more permissions than needed could bring problems. If we're talking about a program, it could be used to hurt the system, or could have a bug. An example could be a wrong setuid[1] program: if a user launches a setuid program, and the program provides a shell, the shell will be a child process of the setuid program, i.e. it would be run with root privileges! **THAT IS ABSOLUTELY TO BE AVOIDED!**
If we're talking about a user, he could be able to run more than he should. If the user wants to attack the system, of if the account is compromised and someone else is using his resources, he could have more weapons and (as maximum damage) destroy the whole system or, at least, more than he could do with correct (lower) permissions.
It is important to remember that a good system administrator *should **always*** keep informed of the security bugs and issues found, discovered and (we hope) resolved. Mailing lists, newsgroups, forums and web pages are updated 3/4 times a day and filled with the strangest discoveries. Even the most innocent and secure program could be buggy, as bug control is one of the biggest troubles of the whole Computer Science. Bugs, for example, allowed a lot of hackers to get *root* access just sending an arbitrary (long) string to the *sendmail* mail server[2]. Nobody said that being a System Administrator is an easy job...that's why I like it!

2.1. **The First Line of Defense: The Firewall.** In order to allow a secure control on what comes in/gets out from our System (or from our LAN), we can set a powerful tool, which can be in the form of a program running on a machine (even our server, but that's not a good idea, especially for high loaded servers[3]) or a stand-alone machine (a dedicated machine, usually integrated with the router. It's a computer, too, but projected and optimized to do only those jobs and often in the form of a box with some ports behind for net cables). Both of them are secure as both of them are insecure, since their behavior is more or less the same.

[1] A program that, when launched, assumes another user's privileges and behaves as if was launched by the other user (e.g.:root)

[2] This is just one of the several security issues found in sendmail during last years. It led to an overflow and allowed root privileges to the attacker

[3] There are some reasons why it's discouraged. One of them is that it could add some (sometimes much) load to the server. The performance could be compromised and the system could slow down too much to keep working efficiently. As second reason, we must keep in mind that no program is perfect and, one day or another, someone could find a bug and break into the firewall, getting root privileges. If the machine was just a firewall, it should not be a big issue, since he would not get access to any important data (at most it would bring just into a DoS), but if it was our main server server...

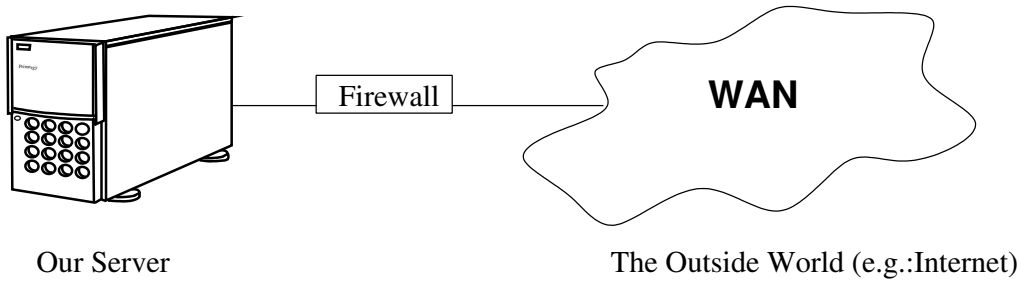

FIGURE 2.1. A General **Firewall**

Security Level	**Firewall Strategy**	**Effects**
Total Security	Everybody Out: no access to the system	Nobody can enter and do his job. Not a solution on a network environment.
High Security	Anything not explicitly allowed is denied	Users could have troubles doing their job.
Normal Security	PLP(Prop. 2.1 on the preceding page)	Ok,but needs other security layers.
Low Security	Filter out only known (bad) ports	System could still be compromised.

TABLE 1. Firewall Strategies

On Figure 2.1 we can see a general representation of the use of this solution. The scheme is general, so the "firewall" entity shown can be considered both as the program inside the server or the external device we've talked about before. All the connections coming from the WAN (that could be any external network, including the whole Internet) and directed to the internal server (or LAN, if the server is acting as a gateway) gets *filtered* before being routed. This allows to control and (if in doubt) discard or reject every single packet passing through the firewall, as specified by a proper rule-set. Some administrators could set some rules for the outgoing traffic, too. For example, an administrator can block the traffic on a particular connection port. If too many users are overloading the network with non job-related (or illegal, immoral,etc) traffic, *root* can block those ports' outgoing packets to be (almost) sure that the employers are just doing their jobs.

One of the biggest troubles to face when configuring and tuning a firewall is the choice of a good rule-set.
There are some approaches, but we must find a compromise between security and usability, as shown in Table 1.

There are different kinds of firewalls, but generally you can choice to let a port open, closed or filtered. The advantages of filtering ports are:

- You can redirect ports (as a transparent proxy): "All connections going out passing through port 80 on localhost should be redirected passing through port 8080 (on localhost, too), logging the traffic"
- You can log the traffic even if not redirecting: "Log traffic passing through port 25"
- All traffic directed to a particular port can be redirected to another machine on the private LAN without telling anything to the connecting host: "All traffic directed to port 3306[4] (on localhost) should be redirected to machine 192.168.12.33 on port 3306 (or anything else)"

There are other advantages, but also some drawbacks: filtering all the ports could overload the firewall and create enormous log files. That's not always the best solution: if you read personally all the logs, you could be flooded by lines containing regular information and there's the possibility of losing important lines and, if using another automatic log analyzer, it could be overloaded and really, really slow.

So, to sum up, firewall approach has some pros and some cons.

Pros:

- Can filter connections before they establish
- Can log, filter, redirect, discard, reject packets following a rule set
- Can be modified to adapt to the situation, even if it has to be done manually (for example, a new bug has been found and there's the possibility of a root exploit. We don't stop the service, just close the port from the WAN)

Cons:

- Too much control or blocking will stop the system from working. Must always find a good compromise between security and usability
- Cannot do anything to prevent attacks from an internal user or machine (it even won't log it)
- If installed on the server, it could potentially be another source of bugs and root exploits It's unusual[5], but it's still possible.
- If a port is not controlled by the firewall, it can be used to hack the system. The firewall, then, will be absolutely unuseful.
- Your network configuration could change out of your control. Let's explain it better: on figure 2.2 on the next page you've got a LAN, connected to a gateway behind a firewall. Anything works perfectly, bad guys stay out and the work continues. One day someone, in his office, plugs a modem to his PC and starts a connection to an external network (e.g.: Internet). The firewall can't control the traffic passing through the modem, so it's almost unuseful. The whole LAN is in danger, as an attacker coming from the WAN and connecting through the modem could gain access to machine 3 and, from that, to the other ones.

So, what can we do to cach'em all?

As matter of facts, a firewall alone is not enough to sleep well at night. We have to set another *layer* of protection. It should not replace the firewall, should work together with it, side by side.

---

[4] mySQL Server interface

[5] Well, I don't really know if it's ever happened, but it's not impossible. Look at note 3 on page 2 for more details

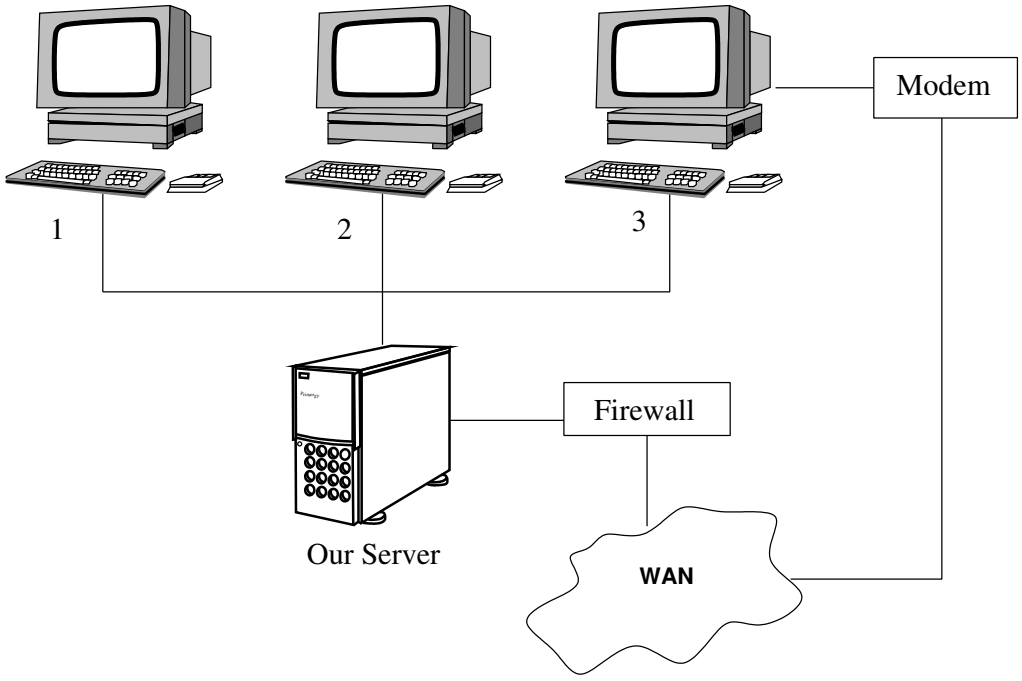

FIGURE 2.2. An Example of Firewall Bypassing

Part 2. The Second Line of Defense: Intrusion Detection Systems

3. WHAT IS AN IDS?

3.1. **How To Control System's Security and Reliability.** The firewall is a powerful tool to filter what's getting in and what's going out, but we've seen it's almost unuseful to find out internal attacks or attacks coming from a channel not controlled by the firewall. So we can't be sure we can prevent an intrusion, and we need a technique to discover it and reply. We can sum up this way:

(1) **Prevent intrusions** configuring and using a firewall. System policies should not be too restrictive and there should not be too much overload. If prevention fails, we should consider the system as *compromised*, even if the attacker hasn't done any kind of damage. Just the fact that he could get in is a hole in the system security. Then go to step 2
(2) **Detect the intruder**. The first way we could think of is to read all the system logs. Perhaps we will be able to trace his movements, his actions, his source ip (if not spoofed ip...that's another long topic) but we will have to check carefully hundreds (often thousands) of lines, most of them full of normal information (mail from: root@localhost to: root@localhost subject: cron log status: successfully delivered). We could be jumping important lines, or the attacker could have deleted them before our discovery. Anyway, if we could detect and recognize the intruder, go to step 3

Policy	Effect
Strict policy	System will think that we're often under attack: many **false positives**
Permissive policy	System will consider as "normal" even some unusual events: many **false negatives**
Balanced policy	System will try to be as correct as possible, when trying to catch intrusions. This will still lead to some **false positives**, but they should not be too many. **False negatives** should never occur.
Perfect policy	System will correctly catch the intrusions, reporting them immediately and replying as soon as it can. False positives won't occur, neither false negatives. This perhaps will be possible, one day, but for now it's just an ideal target (and the last goal of the IDS research)

TABLE 2. IDS choice and sensibility tuning

(3) **Response to intrusions**. We can choose the way we prefer: from calling the police to attack the attacker's system (I don't like this way), or we can block what he's doing trying to close his path to the system

About step 1 we've told enough (at least for now) in section 2.1 on page 2. We should now focus our attention to step 2, which is perhaps the most interesting (and hard to set up) process of a system configuration.

3.2. **An Overview of IDS Technology.** IDS is an acronym of **I**ntrusion **D**etection **S**ystems, that is something that allows to detect an intrusion as it happens and be able to respond to it in an efficient manner. IDS is a generic concept, there is not a fixed standard for it. There may exist a lot of different IDS, behaving in really different ways, implemented in different languages (perl, C, even Java) and, of course, with different configurations and efficiency, often regulated by a policy. This brings, as usual, some troubles to the poor administrator. Being an automatic system, sometimes it could not detect properly what's happening. So we could have **false positives** or **false negatives**. Have a look at table 2. As you can see, a strict policy can lead to a lot of false positives, which should normally be reduced, while a too permissive policy will lead to many false negatives, which **have to be** normally avoided.

Too many **false positives**, in fact, could lead to administrator's madness, since he could be awaken at 3am because the system detects an attack (which, of course, doesn't exist) and could lead him to turn off the IDS. No other harm can be done, but it is really annoying.

False negatives, in turn, won't detect a real intrusion and the attacker will have all the time to do anything he wants, including turning off the firewall, the IDS, or clearing the logs and deleting the evidences of his passage. He could, as well, install a trojan horse and create a door to get in again without troubles (e.g.: back door). *That's absolutely to be avoided*: if the attacker has arrived here, he's

already passed beyond the firewall, so no other protection will be available[6]. So, the questions could be:

- How much are we going to spend for the IDS?
  There are a lot of systems out there, some of them free, some commercial. If we don't want to spend, we have to find a free solution, but it can't always be the best choice. Someone could prefer the way proceding of a commercial product.
- How much noise could we tolerate?
  As said before, too strict policies could lead to noise. Some IDSes do a lot of noise as soon as we try to lower the false negatives.

The best solution is, of course, individual and not universal. Since the "Perfect policy" (as shown in table 2 on the facing page) is still almost ideal, we must find a compromise and the most fitting technique for our system. A lot of approaches have been tried, and anyone has its strong and weak points, depending on the configuration and on the system we want to keep controlled. We must always keep in mind the usual rules:

- System should be usable without too many restrictions
- The IDS (as for the firewall) should not have a big impact on the system's performance, otherwise we are violating the preceding rule
- It should not add new security issues. If it does, we have to balance the risks we are facing with and without it and decide which are more dangerous (or, better, find another solution). Feeling safe just because "We've just installed a new IDS" is the easiest way to put the system in danger. A close control is *always* needed. IDS is an instrument, not a permanent solution[7].

3.3. **Lack of IDS and other protection tools.**

3.3.1. *The Bastion Host.* A bastion host [Dillard1] is a computer that is fully exposed to attacks. The system is on the public side of the demilitarized zone, sometimes even unprotected by a firewall or filtering router. Frequently the roles of these systems are critical to the network security system. Indeed the firewalls and routers can be considered bastion hosts. Due to their exposure a great deal of effort must be put into designing and configuring bastion hosts to minimize the chances of penetration. Other types of bastion hosts include web, mail, DNS, and FTP servers. Some network administrators will also use sacrificial lambs as bastion hosts, these systems are deliberately exposed to potential hackers to both delay and facilitate tracking of attempted break-ins.

Effective bastion hosts are configured very differently from typical hosts. Each bastion host fulfills a specific role, all unnecessary services, protocols, programs, and network ports are disabled or removed. Bastion hosts do not share authentication services with trusted hosts within the network so that if a bastion is compromised the intruder will still not have "the keys to the castle". A bastion host is hardened to limit potential methods of attack. The specific steps to harden a particular bastion host depend upon the intended role of that host as well as the operating system and software that it will be running. Access Control Lists (ACLs) will be modified

[6] Apart from the single daemons' security routines, which could be buggy or intrinsically insecure. Often they have to get root access, at least to open the sockets. If an intruder knows it, he could easily get root privileges.

[7] At least for now...

on the file system and other system objects; all unnecessary TCP and UDP ports will be disabled; all non-critical services and daemons will be removed; as many utilities and system configuration tools as is practical will also be removed. All appropriate service packs, hot fixes, and patches should be installed. Logging of all security related events need to be enabled and steps need to be taken to ensure the integrity of the logs so that a successful intruder is unable to erase evidence of their visit. Any local user account and password databases should be encrypted if possible.

The last step to securing a bastion host may be the most difficult: securing whatever network application the host is running. Very often the vendor of a web or streaming media server doesn't consider security risks while developing their product. It is usually up to the system administrator to determine through testing what ACLs they need to modify to lock down the network application as thoroughly as possible without disabling the very features that make is a useful tool. It is also necessary to closely track the latest announcements from the vendor regarding security problems, workarounds, and patches. The more popular network applications also tend to inspire the creation of independent mailing lists, newsgroups, and websites that can be tracked for additional insights.

3.3.2. *The Honeypot.* Honeypots are programs that simulate one or more network services that you designate on your computer's ports. An attacker assumes you're running vulnerable services that can be used to break into the machine. A honeypot can be used to log access attempts to those ports including the attacker's keystrokes. This could give you advanced warning of a more concerted attack.

One honeypot program is called the Deception Tool Kit, which can be downloaded from http://www.all.net/dtk/. You can configure the responses for each port.

Honeypots are most successful when run on well-know servers, such as Web, mail, or DNS servers because these systems are often attacked. They can also be used when a system comes under attack by substituting a honeypot system for the target.

4. How Intrusion Detection Systems Work

4.1. **IDS techniques.** Year by year, the IDS technology has done a lot of steps and a lot of models have been invented and tested. Each model has its own features and its branch of develop, so we have to distinguish between some (main) categories:

(1) IDS based on Data Source
(2) IDS based on Model of Intrusions

About the first one, we it can then be:

- Host Based:
  Collects data from a single host and tries to detect intrusions
- MultiHost Based:
  Collects data from a number of hosts (more than one) and tries to use them to detect intrusions
- Network Based:
  Collects data from one or more hosts (like the others), but also uses data passing through the network and analyzes them all

About the second category, we have:

- Anomaly Detection Model:
  Collects continuously data from the system and creates some statistical databases, including medium CPU usage (on a per user basis), and so on. If it "thinks" that something strange is happening, it reports a positive and tries to reply to the attack.
- Misuse Detection Model:
  The system is monitored, and the IDS searches for known intrusion techniques. Acting like many anti-virus programs, it just searches for signatures. In this case, usually a "signature" can be a well defined pattern of actions (shell command or other) which is known to compromise the system security. No need to say, if an attack is different from the one the IDS knows about, it won't be detected. No heuristic scan is performed.

4.1.1. *Data Source: Host Based IDS.* A host based ID System consists mainly in loading a software program (or a set of programs) into the system you want to keep controlled. This software uses the system log and other security logs and/or auditing agents to see what's happening. This involves that the System Administrator must be enough competent and should know perfectly all the programs installed (in order to distinguish between normal using and suspect processes) and his user's habits (some careful SysAdms have detected some attacks just "feeling something different" on the system's behaviour or seeing users logged at strange hours).
So, it's important to remember that **without a careful Administrator, the system can never be considered safe!**

This kind of approach consists not only in checking network traffic getting in/going out, but also controlling the integrity of the system files and suspicious processes. If you're administering a network and you want to use this kind of solution, you have of course to install it on every single machine, as shown in figure 4.1 on the next page. There are two primary classes of host-based intrusion detection software: host wrappers/personal firewalls and agent-based software. Either approach is much more effective in detecting trusted-insider attacks (so-called anomalous activity) than is network-based ID, and both are relatively effective for detecting attacks from the outside.

- Host wrappers or personal firewalls can be configured to look at all network packets, connection attempts, or login attempts to the monitored machine. This can also include dial-in attempts or other non-network related communication ports. The main ones are the TCP Wrappers (for Unix), which can be found on *http://coast.cs.purdue.edu/pub/tools/unix*
- Host-based agents may be able to monitor accesses and changes to critical system files and changes in user privilege, since this should not happen (at least in a significant manner) automatically

If you're using a Unix operating system, you've got a lot of small independent programs to detect intrusions. No one of them will do anything for default, and you should configure each of them for your own system. For example, if a machine has only a handful of users, perhaps only the connections from the outside and the integrity of the system files need to be monitored; whereas, a machine with a lot of users or network traffic may need more stringent monitoring. There are a lot of those programs: syslog, disk quotas, process monitoring, *BSD process accounting, and so on.

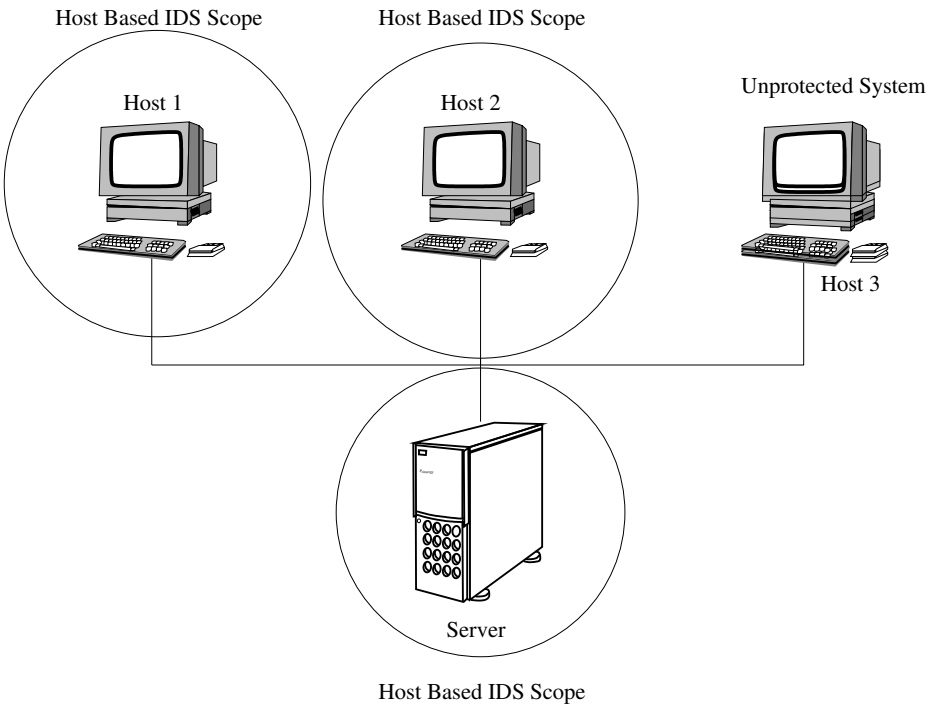

FIGURE 4.1. Host Based IDS Example

Of course, this kind of detection is as good as the logs can report: if you don't log anything, no intrusion will be found. In addition, if the log remains on the system, the attacker could delete the evidences of his passage. A good idea could be to send all the logs to another machine, via e-mail or so, and compute them there, so *nobody* can delete anything.

Before going on, I'd like to say a little more about figure 4.1: Host 1, as well as Host 2, are protected by a local IDS, absolutely independent one another. If any of the two gets compromised, the other one will continue to be protected, unless the attacker could use the same strategy to open the other one. Host 3 is unprotected, so the machine is completely open to any attack (at least concerning this kind of protection). It's clear, at this point, that a host based IDS alone could not be enough to be sure that the system will not be successfully attacked, so you'd better combine it with another kind of strategy. One of the most growing and exciting studies about this topic is how to merge all the data caught from different monitoring strategies.

4.1.2. *Data Source: Network Based IDS.* This kind of IDS monitors continuously its associated network. This means that the only source of data is the network traffic. Usually, this is performed by placing a "filter" on the network interface and capturing a copy of the packets to analyze them. Because of this, if another interface is connected to the machine (and the IDS is not updated to face the changes), it will have no effect on the new device (and, of course, on the data flow passing through that path). An example can be seen on figure 4.2 on the next page: two

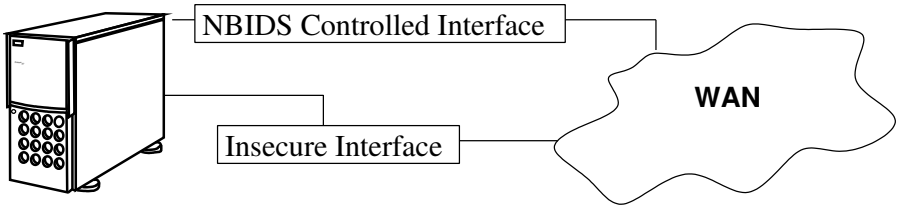

FIGURE 4.2. Network Based IDS: An example

network interfaces are present. The first one (let's call it eth0) is being monitored by the Network Based IDS. When the second one (let's call it PPP0) is enabled, no check will be performed on data passing through PPP0. So, if an intruder gets in using this channel, nothing will be detected[8].Keeping that in mind, we can say that both network-based and host-based ID sensors have pros and cons. In the end, you'll probably want a combination of both. According to the article [Northcutt1], network based IDS involves looking at the packets on the network as they pass by some sensor. The sensor can only see the packets that happen to be carried on the network segment it's attached to, as in figure 4.2. Packets are considered to be of interest if they match a signature. Three primary types of signatures are **string signatures**, **port signatures**, and **header condition signatures**.

- **String signatures** look for a text string that indicates a possible attack. An example string signature for Unix might be "cat "+ +" > /.rhosts" , which if successful, might cause a Unix system to become extremely vulnerable to network attack. To refine the string signature to reduce the number of false positives, it may be necessary to use a compound string signature. A compound string signature for a common Web server attack might be "cgi-bin" AND "IFS".
- **Port signatures** watch for connections coming to well known ports: an example is RPC, in which a lot of security issues have been found in the past. On port 25 there could be "sendmail", the first and yet more spread used MTA[9]. It's known to be rather buggy, even if now they should have been fixed[10].
- **Header signatures** watch for dangerous or illogical combinations in packet headers. The most famous example is Winnuke, where a packet is destined for a NetBIOS port and the Urgent pointer, or Out Of Band pointer is set. This resulted in the "blue screen of death" for Windows systems. Another well-known header signature is a TCP packet with both the SYN and FIN flags set, signifying that the requestor wishes to start and stop a connection at the same time.

It is important to keep in mind that a good IDS will use both host and network based systems. As seen, either used alone could be absolutely unuseful.

[8]This could be solved using another IDS, maybe a Host Based one.

[9]Mail Transport Agent

[10]Even if a lot of Unix versions install sendmail as a default (e.g.: FreeBSD), its use is discouraged: using a piece of software born buggy and insecure, even if fixed, is not a good idea. Today there are a lot of good alternatives (qmail, smail and my favourites: postfix and exim)

User	Usual Behaviour
System Administrator	Logs in as a root, accesses system's passwords database, edits users' access permissions, runs system configuration and monitoring tools.
Boss	Logs in from time to time, reads mail once per week and never sends it.
Secretary	Is logged in locally during company working hours, uses text preparation software, permanently reads and sends mails
Manager	Is logged in from early morning till late evening, permanently uses management tools and Internet browser, permanently reads and sends mails
Programmer	Is logged in from late morning till late night, runs software development tools, browsers Internet more often on the late evening than during a day time.

TABLE 3. Example: General common user profiles

4.1.3. *Model Of Intrusion: Anomaly Detection Model.* About the "Model Of Intrusion" approach, there are some good papers I will use to be clearer. One of them is [Gordeev1].
The first well known approach deals with detection of a certain anomaly in a user behaviour. Let's explain it in more detail. Each user of a computer system is capable of performing some tasks, according to his permissions. In other words, each user has a certain functionality within the system. Usually this functionality is observable and does not change a lot in time. For example, a secretary due to her or his job speciality usually deals with a limited number of tasks like typing various documents, reading and sending mail, etc. Unlikely he or she will start programming and using compilers. A system administrator accesses system configuration areas, run statistic, audit, installing and monitoring applications. A programmer's tasks is obvious - write a program, compile, and debug. This means that it is possible to define a set of actions usually performed by an user. Often, this set is also called *a user profile* that describes user's normal behaviour. Table 3 gives a rather general and intuitive example of a number of common user profiles[11].

This is of course a rather general example. Real profiles deal with more precise categories (sometimes up to system calls) and are more flexible in constrains of a user normal behaviour (e.g. boss could sometimes send mail and browse Internet, so they should be not too static).

After such profiles are defined, it is already manageable to trace current user behaviour and to search for some deviations from it. Such deviations are called anomaly and indicate in most cases an intrusion. An intuitive example could be, when a secretary logs in at 9:00 pm, accesses management software and, finally,

[11]Of course, profiles should not be too static. We can't forbid the boss to surf the net or do video conferencing, when needed. And we shouldn't be awaken up at 2AM because the secretary is trying to send a buisness mail she had forgotten of.

User	Anomaly in Normal Behaviour
System Administrator	Becomes a programmer, and accesses software development tools, and software sources belonging to some project
Boss	Logs in at midnight and start intensively sending mail, accesses confidential information
Secretary	Logs in form a remote host, becomes a manager
Manager	Tries to become a system administrator, changes users access permissions
Programmer	Becomes a secretary, accesses personnel database

TABLE 4. Example: General common user anomaly

tries to get system administrator privileges. See table 4 for an an example on how to trace an anomaly.

Usage of this approach intends in a need in *learning* a normal user behaviour. In other words, it means that IDS must have a knowledge about the user behaviour *priori* its normal operation[12].

Anomaly detection systems are trained on huge amounts of system audit information in order gain sufficient knowledge about user behaviour. Usually, this involves various intelligent techniques like rules generation, machine learning, neural networks, etc.

Advantages of behavior based approaches are that they can detect attempts to exploit new and unforeseen vulnerabilities. They can even contribute to the (partially) automatic discovery of these new attacks. They are less dependent on operating system-specific mechanisms. They also help detect "abuse of privileges" types of attacks that do not actually involve exploiting any security vulnerability. In short, this is the paranoid approach:

> **Everything which has not been seen previously is dangerous.**

The high false alarm rate is generally cited as the main drawback of behavior-based techniques because the entire scope of the behavior of an information system may not be covered during the learning phase. Also, behavior can change over time, introducing the need for periodic online retraining of the behavior profile, resulting either in unavailability of the intrusion detection system or in additional false alarms. The information system can undergo attacks at the same time the intrusion detection system is learning the behavior. As a result, the behavior profile contains intrusive behavior, which is not detected as anomalous.

4.1.4. *Model Of Intrusion: Misuse Detection Model.* Almost any intrusion can be described in terms of its indications and signs. First of all, signatures of all known attacks must be described in some sort of database and given to the IDS. This patterns[13] are used later by the IDS to identify an intrusion. This is done in order

[12]Often it's not easy, so you should trace the common user profiles *before* starting the IDS

[13]The "signature" is just composed by regular patterns

to study the system audit information,to find some signatures matching to patterns known to be a part of system intrusions.

A good example to demonstrate this approach could be a well known SYN flood denial of service attack. Its goal is to prevent the target host from accepting new connections on a given IP port. The implementation of this attack utilizes a three-step handshake schema of a TCP/IP connection establishment and usually exploits a resource exhaustion vulnerability that is common for many TCP/IP implementations. The basic idea is the following: when a client opens a TCP/IP connection, it sends a SYN packet to the server, which receives it and allocates an entry in a connection queue. Such connection is referred as being half-open. Then the server sends a SYN-ACK packet to the client that must be acknowledged by a ACK packet sent by the client to the server. After receiving the acknowledgement, the server releases the corresponding entry in the queue. This procedure can be exploited by an intruder that sends series of SYN packets to the server and does not acknowledge them. This will result in a queue overflow, since the finite connections queue of the server will get filled up and will be not emptied until timeout periods will not expire. The result is: the server is not able to accept connections on the attacked port any more.

This attack is characterized in general by the following indications:

- SYN packets arrive periodically in series
- A number of half-opened connections is rapidly increasing (ACK packets arrive with a not observable delay)

In order to recognize this attack, IDS must study information on TCP/IP traffic and try to find these indications in it. If the attack is detected, IDS should react. This reaction could be done in a form of signalling an alarm, removing corresponding entries from the connections queue, etc.

In a similar manner indications of other attacks can be figured out. They are represented in a certain form and coded to IDS. There is a number of methods for intrusions representation and their further recognition. Two of the most widely methods are **state transition analysis** and **rule-based expert system**.

4.2. **Methods to represent IDS knowledge.** The following sections are pieces of articles I've found surfing the net, which I've changed and adapted to satisfy our needs. They could give an idea of what can be done to "teach" the IDS how to work.

4.2.1. *State Transition Analysis.* State transition analysis was developed few years ago by the Reliable Software Group at University of California, Santa Barbara (see [Ilgun95] for more details). This method is used for representing a sequence of actions that an intruder performs to break into a system. These actions and requirements to them are represented by a state transition diagram. It is based on a premise that all intrusions have the following two common features:

(1) an intruder gets an access to a target system in one or another way
(2) Intrusion results in gaining by the intruder some abilities that he did not have before.

Therefore an intrusion is seen as sequences of an intruder actions that bring a system from an initial state to a compromised state through a number of intermediate (discrete) states. The initial state identifies a system state before the intrusion (usually the normal state, in a safe system), the compromised state reflects the

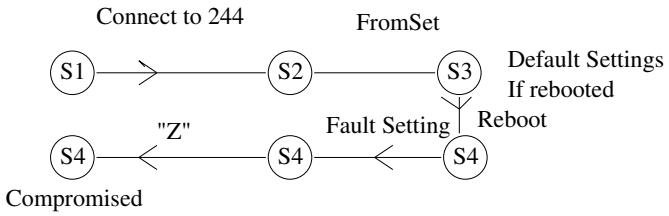

FIGURE 4.3. State Transition Analysis Example: IntelBusiness E-Mail Station Hacking

system state after the intrusion success (the total alert state). The steps that the intruder makes are represented by state transitions.

Apart from the states, signature actions are identified. Signature actions mean a minimal set of actions needed to complete the intrusion. If at least one of them is omitted, the intrusion will be not completed (e.g.: if the intruder is an internal user and can't get any higher permission and/or access to data he should not see, it's considered like if the attack. has failed, no alarm is generated).

Finally, the states, transitions and signature actions are represented graphically in a form of a state transition diagram. A good feature of this approach is that the threat scenario is represented in a visual form and very easy to read.

Lets demonstrate this approach with a small example. A number of security mailing lists have announced in January 2000 a vulnerability in a authentication schema of Intel InBusiness E-mail station, a small office application server. It lets remote users to connect to the server and perform some commands without any authentication. This could lead to giving an attacker a system command prompt under super-user privileges. This attack involves the following steps (see the visual form on figure 4.3):

- Connecting to port 244 (Status S1→ S2)
- Send a "FormSet" command. After the reboot, the system's settings are cleared, as well as the super user password, and everything is at factory defaults (Status S2 → S3)
- Induce the system to reboot. This can be accomplished performing other attacks that will overload the system (Status S3 → S4)
- After reconnecting to port 244, the "Z" command gives a login prompts and asks for super user password. Since we've deleted the system configuration, it can be omitted[14] (Status S4 ⇒ System is compromised)
- Now we've got root access to the system. We can do anything we want.

Real system do not operate with the graphical diagrams. Instead, they use some special languages for states and transition descriptions. Very often, state transition analysis is used as basis for rule-based expert system.

4.2.2. *Rule Based Expert System.* Rule-based expert system are used very often as the "engine" of an intrusion detection system. One of the most interesting things about this approach is that it is used in both anomaly detection systems and misuse

[14]In different situations, the factory password is not empty. But it's not hard to find it out, since usually any producer's web site has a link to download the device's user manual, where all default settings/passwords are revealed

detection systems. In such systems, like in any other expert system, declarative knowledge related to intrusions is separated from an inference engine performing a reasoning about the fact base[15]. In other words, it means that, in general, three main components can be distinguished:

- **Facts base** that contains facts on system states (already processed audit records)
- **Rules base** that contains rules representing intrusions scenarios
- **Inference** (deduction) **engine** that makes the logical process on facts and rules in order to identify an intrusion or an attack

Inference engine searches the facts space for those that match what is expected by a rule. If any match is found, the rule is activated and its consequent is fired. We shall demonstrate this process by a simple example. Lets build up a small rule describing a well known buffer overflow attack. This attack exploit a buffer overflow vulnerability of programs running under privileged accounts (e.g. root). An attacker calls a victim program with a long and carefully prepared argument that overflows program memory buffers and alters its execution. As a result, an attacker gets super-user privileges on the target host. This attack can be described by an heuristic rule based on the following facts[Lindqvist99]:

- The attack can be detected by analyzing system audit records on exec system calls
- Effective user id and real user id are different (exec call concerns a setuid program, which gives different privileges)
- The argument passed to the exec call is relatively long
- The argument contains characters in the range of ASCII control characters

So, if all those conditions are satisfied, the system can (reasonably) "think" that it's under attack. An example of this approach can be seen on

figure 4.4 on the facing page: we could see the system as a machine where each check produces a response: "ok" or "failed". If all the checks say "failed", the system will alarm and say it's under attack.

Expert systems are also used for anomaly detection. As already explained before (see section 4.1.3 on page 12), this approach needs some kind of learning of normal user behavior and anomalies in it. That is actually the basic difference in using rule based expert systems for anomaly and misuse detection. In the first case, the rules are generated using some other techniques. In the second case, the rules are given to the system in advance, so you probably will have to leave the system unprotected (or protected by other kinds of IDS) and trace a correct usage profile.

There is a number of methods used to obtain rules describing user behavior. One of the known methods is data mining[Stolfo98]. This method extracts descriptive models from huge amounts of data. In general, it uses three groups of algorithms originating from a variety of fields like statistic, pattern recognition, and machine learning:

- **Classification**: mapping data items into some predefined categories (e.g. "normal" and "abnormal")
- **Link analysis**: determining a correlation between various factors in the audit information (e.g. a correlation between "command" and "execution time")

[15]So the "brain" of the program can change, but the database can be left unmodified

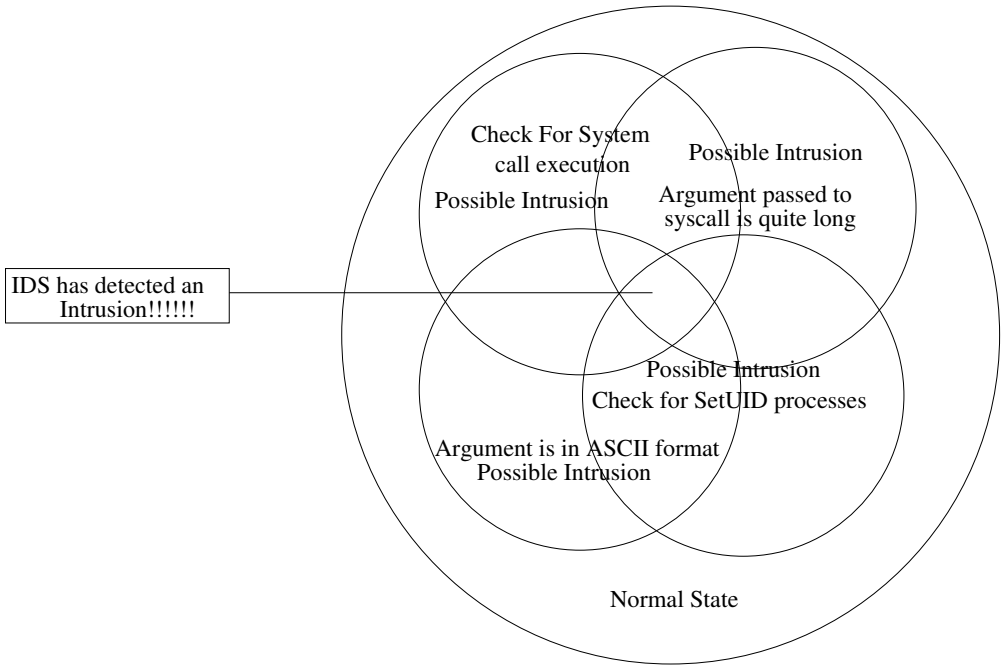

FIGURE 4.4. IDS: Example of heuristic analysis

- **Sequence analysis**: modeling of sequential patterns - audit events that occur together

There are some other methods like frequency based and Hidden Markov Models, but I won't explain them here.

4.2.3. *Neural Networks Approach.* Neural Networks offer an alternative means of maintaining a model of expected normal user behavior. They may offer a more efficient, less complex, and better performing model than mean and standard deviation, time decayed models of system and user behavior. Neural network techniques are still in the research stage and their utility have yet to be proven. They may be found to be more efficient and less computationally intensive than conventional *rule based* systems. However, a lengthy, careful training phase is required with skilled monitoring.

5. EXAMPLES

5.1. **An Open Source Example of Signature Based IDS: SNORT.** Many corporate networks and corporate security policies rely heavily on intrusion detection to alert administrators of intrusion. It would be impossible to do in other ways, even keeping in mind that it's not a perfect technology. With all of the features of modern intrusion detection systems there are some tragic flaws inherent in their design. These weaknesses apply to all other signature based intrusion detection engines.

5.1.1. *What is SNORT?.* Snort is an intrusion detection system written by Martin Roesch. Snort is was written as an open source project and is available for free under the GNU public license. The software is based upon a signature comparison engine optimized for speed. Snort offers many features that make it an ideal choice in the battle against Internet intruders. Here is a description of Snort from the Snort website:

> Snort is a lightweight network intrusion detection system, capable of performing real-time traffic analysis and packet logging on IP networks. It can perform protocol analysis, content searching/matching and can be used to detect a variety of attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, OS fingerprinting attempts, and much more. Snort uses a flexible rules language to describe traffic that it should collect or pass, as well as a detection engine that utilizes a modular plugin architecture.

Snort was written to take advantage of a highly modularized design. The application can take advantage of several different pre-processors to normalize, filter, and categorize data. Snort also has very powerful post-processors, or output plug-ins, that can be used to log the data generated by Snort in several different ways. Because Snort is an open source project and that it has many users its signature database is updated often and are simple to update.

5.1.2. *How Signatures Work*[16].

> Understanding how signatures work is essential to understanding how to defeat them. When Snort is given an incoming packet from the packet capture driver it compares that packet to its database of known signatures. The signature has some key aspect of the packet that it is compared against to look for a match. If a match occurs than Snort sends the output to a standard output mechanism or to one of the configured post-processor output plug-ins. For example if Snort received the following packet then it would compare it against its database:
>
> 03/21-13:02:34.978853 10.1.114.88:1272 -> 10.1.114.220:54320
> TCP TTL:128 TOS:0x0 ID:48408 IpLen:20 DgmLen:44 DF
> ******S* Seq: 0x2BC3D9 Ack: 0x0 Win: 0x2000 TcpLen: 24
> TCP Options (1) => MSS: 1460

and match it to rule:

> alert tcp $EXTERNAL_NET any -> $HOME_NET 54320 (msg: "BACKDOOR SIG - BO2K";)

This event would trigger an alert message. Most signatures do not just look for what port a packet is to or from, but it also examines part of the payload. As new security holes and exploits are found new signatures are written to counteract the danger.

What Snort and other signature based intrusion detection systems count on is that malicious traffic will have unique patterns to it that can be matched against

[16]According to [Richard01] and on [SNORT01]

rules in the database. It then seems obvious that there are many ways of circumventing this signature. The first thing that we could do is vary the destination port, but it's often undesirable since the destination machine could be running a particular service listening to a particular port. Because of this, we could do a large scan of ip numbers trying to connect to that port. But, in this case, Snort could detect it. Let's do another example: Back Orifice, a trojan for windows, installed itself at port 54320 (as in the example written before). The port cannot (easily) be changed, so, to avoid signature matching, we've got to change something else on the header. This could be accomplished by using some very simple form of encryption, passing over Snort's checks but arriving where we want to. . Another twist of this technique could incorporate public key/ private key encryption. The private key for the server and the public key for the client could be sent or bundled with the original install. This would render all communication between the 2 hosts unintelligible and undetectable by intrusion detection systems.

It's now clear that Snort and other intrusion detection systems do their best job in detecting attacks on services that require an exploit and that cannot be encrypted. Attacks like this would include buffer overflows, scanning attempts, etc. These types of attacks are based on existing bugs within the victim machine. These flaws can typically only be exploited using a certain attack routine that will have a certain *signature*. In these cases signature based intrusion detection does very well at detecting these patterns and alerting (or stopping them).

The problem with intrusion detection as it relates to attacks on services is that it may take some (sometimes long) time for a new exploit to become known, especially if talking about rarely used pieces of software. After the exploit is known then a new signature can be written for it and distributed. This leaves many systems vulnerable to unknowin attacks for a certain period of time. It is possible that a well-executed attack will leave no trace of intrusion thereby rendering all of the effort placed into intrusion detection wasted. IDS are also hurt by a lack of supporting data for attacks that were not immediately recognized. To explain it better, we could say that the IDS will start logging only when an intrusion is detected. Sometimes, before being caught the attacker could have done a lot of things and damages. More, the source of his penetration could not be identified.

There are also some other ways to confuse an IDS. One of them is the *"Denial Of Service"* technique. The attacker can do a lot of noise before starting the real attack, confusing the IDS (that, trying to face and reply, could get overloaded) and the operators, which will have to spend a lot of time before catching out which one of the attempted attacks is the successful one. Another possible method of implementing a denial of service against an IDS would be to exhaust the resources of that IDS. This denial of service would flood the IDS with traffic that will generate alerts until the IDS runs out of resources. This would cause the IDS to have an incomplete log of the events that took place. Have a look at figure 5.1 on the following page for a graphical example of this approach.

5.2. **Anomaly Detection Tools: SPICE/SPADE.** Silicon Defense's SPICE (Stealthy Portscan and Intrusion Correlation Engine) project is a DARPA-sponsored development-effort whose aim is to build a better mousetrap capable of detecting stealthy port scans. SPICE consists of two components, an anomaly sensor and a

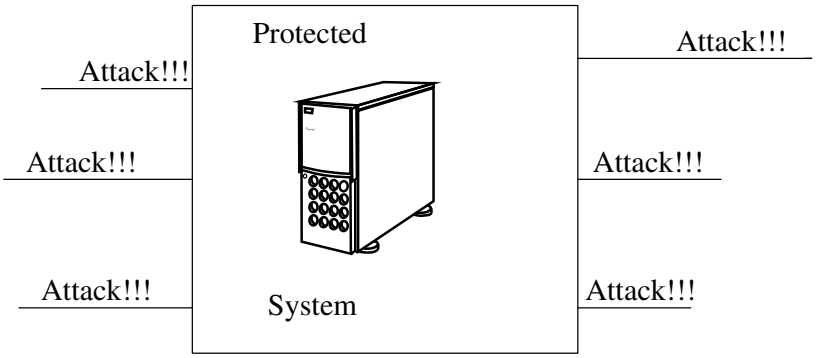

FIGURE 5.1. An Attempt to DoS a Signature Based IDS

correlation engine. SPADE is the anomaly detector, which acts as a plug-in preprocessor to Snort(described on page 17) . The correlation engine is still under development.

SPADE looks carefully at the network traffic, performs some calculations (I won't explain here, we'd go too deep) and tries to figure out what's happening. If everything is "in line" with the normal system's work, no alert is launched. Those calculations are made assigning an anomaly level (A) to any event occurring, which is dependant on the probability (P) that a particular event will happen. For example, a connection to the port 80 for a machine that is acting as a web server will have an high probability value and a low anomaly one. The higher P is, the lower A is, of course. The settings for SPADE need tuning to match the protected network. One tunable factor is the alert-threshold. This can be set manually and tuned by the analyst, or SPADE itself can be set to set its own threshold level. In default learning mode, SPADE will monitor network traffic for 24 hours. Then it will calculate the threshold level required to create 200 alerts in that monitoring period. The length of the monitoring period and the number of alerts to generate are selectable.

As it runs, SPADE will generate a probability table of observed network traffic. This table contains critical information and should be protected and backed-up. If this file is lost, SPADE will need to be retrained, exposing your network as the history is rebuilt and alerts are not generated. When operating in survey-mode, SPADE will generate reports on observed probability distributions of the network traffic. SPADE can be placed into statistical mode if one wishes to view the probability tables on a regular basis.

Currently, SPADE simply generates alerts on packets whose anomaly score (as calculated by SPADE,) exceeds the anomaly threshold level. These alerts are logged along with the other Snort alerts. It is the correlation engine, which is still under development, which promises to detect the stealthiest of port scans. The correlation engine is fed alerts from the anomaly detector. The alerts contain the event, and the anomaly score. The correlation engine will keep an event in memory based on its anomaly score. The higher the score, the more anomalous the event, thus the longer it will keep its state. The correlation engine then attempts to link the events into groups to possibly link rare events to a single cause. Links between a given pair of events are calculated by a series of heuristic functions. If the source IP or

the destination port or network are the same in the two events, a given heuristic would fire.

There is still much work to be done in the field of anomaly based detection. Misuse detection based on signature matching has limitations; these limitations can be lowered through the use of anomaly based detection. The fusion of both misuse detection and anomaly detection techniques will result in a more effective and efficient Network Intrusion Detection System.

5.3. **Can we feel safe?** The IDS research is new in computer sciences. There are still a lot of steps to do and today's solutions are still young. However, the experience of last years demonstrated that there is a strong demand on intrusion detection systems. Very often, a need in them is expressed on a state level. For instance, the USA government has announced in January 2000 that a global intrusion detection system covering the governmental computer network will be created. A lot of enterprises of different size already use these system for ensuring their business security.

References

[Northcutt1] Stephen Northcutt (SANS Institute), "What is a Network Based Intrusion Detection?"

[Gordeev1] Mikhail Gordeev,"Intrusion Detection: Techniques and Approaches"

[Dillard1] Kurt Dillard (Collective Technologies, Inc),"Description of a Bastion Host"

[Lindqvist99] U. Lindqvist, P. Porras, "Detecting computer and network misuse through the production-based expert system toolset (P-BEST)", In Proceedings of the 1999 IEEE Symposium on Security and Privacy, California, May, 1999

[Stolfo98] W. Lee, S. Stolfo, "Data mining approaches for intrusion detection", In Proceedings of the 7th USENIX Security Symposium (SECURITY-98), January, 1998

[Ilgun95] K. Ilgun, R. Kemmerer, P. Porras, "State transition analysis: a rule-based intrusion detection approach", IEEE Transactions on Software Engineering, Vol. 21, No. 3, March, 1995

[Richard01] Matthew Richard, "Are there limitations of Intrusion Signatures? ", April, 2001

[SNORT01] "What is Snort", http://www.snort.org/what_is_snort.htm ,Apr. 2001