LinuxFocus article number 292
http://linuxfocus.org

IDS - Intrusion Detection System, Part I

by Klaus Müller
<Socma(at)gmx.net>

About the author:
Klaus Müller a.k.a. 'Socma' is still a student and engages in Linux programming and security issues.

Translated to English by:
Hubert Kaißer
<hubert(Q)faveve.uni-stuttgart.de>

Abstract:

This is the first article of a sequence about Intrusion Detection Systems. The first part will not only present typical IDS architectures but also typical attacks against Intrusion Detection Systems.

Introduction

Before I start just a few explanations so that no misconceptions arise: IDS stands for Intrusion Detection System. In the course of the text I also relate to IDS as a system that recognizes intrusions and starts counteractive measures (whereas to initiate counteractive measures is an optional feature). In the beginning, I will at first give an overview about the different kinds of IDS to be able to address different tactics afterwards. In the last part I will discuss several programs like Snort, bofh, ... and my project COLOID.
In advance, I give some annotations for terms which I will not explain directly in the text:

- false positive = an alarm that an attack has taken place whereas this was not the case
- false negative = the IDS does not detect and filter the attack

IDS overview

The idea of this chapter is to present different kinds of Intrusion Detection Systems and at the same time their advantages and disadvantages. But first, some words about the meaning and intention of Intrusion Detection Systems. An IDS sort of observes activities on the particular host or network. With several

methods which I will present here, it tries to find out if the security of the host is threatened (if there is an "attack") to initiate counteractive measures afterwards. Because, in the majority of cases, log files are created and it is one of the main issues to analyze them and react on the indication of an intrusion or an illegal attempt of a user to augment his rights.
Basically, there are the following three areas:

- Information Sources
- Response
- Analysis

Response and Analysis is covered again more precisely later , the different kinds of "Information Sources" right now. Because of the extensive possibilities to attack a PC or a network there are different kinds of IDS (of course, they can be combined with each other) which basically differ in the points where they control and what they control (Information Sources).

Host - Based Intrusion Detection

Host-Based Intrusion Detection Systems analyze information on a single host. As they normally do not look at the whole network traffic but "only" at the activities on the same host they are able to make more precise statements about the kind of attack. In addition, you can see directly the impact on the particular PC. E.g that a specific user started an attack successfully. Host-based IDSs use mostly two different sources to provide information about activities: system logs and operating system audit trails. Both have advantages and disadvantages as operating system audit trails are more exact, detailed and can give better information about activities while system logs mostly deliver only the most important information and are therefore considerably smaller. System logs can be controlled and analyzed better because of their size.

Advantages:

- you "see" the impact of an attack and can react better on it
- you can recognize Trojan horses etc. better as the available information/possibilities are very extended
- you can detect attacks which cannot be detected by Network based IDS because traffic is often encrypted
- you can observe activities on your host exactly

Disadvantages:

- they are not good in recognizing scans
- they are more vulnerable to DoS attacks
- the analysis of operating system audit trails is very time-consuming because of its size.
- they stress the host’s CPU performance (partly) immensely

Examples:

- Tripwire [http://www.tripwire.com/products/index.cfml]
- SWATCH [http://freshmeat.net/redir/swatch/10125/url_homepage/swatch]
- DragonSquire [http://www.enterasys.com/ids/squire/]

- Tiger [http://freshmeat.net/redir/tiger-audit/30581/url_homepage/tiger]
- Security Manager [http://www.netiq.com/products/sm/default.asp]

Network Intrusion Detection (NIDS)

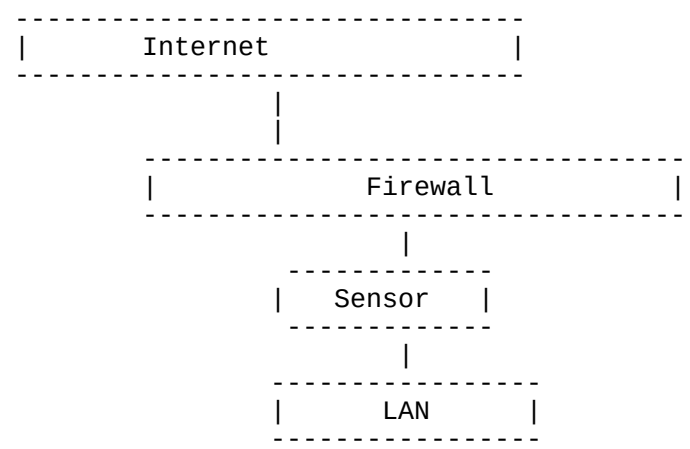
The main task of a Network IDS is the analysis and interpretation of packets transferred over the network. Signatures are used to examine packets for "conspicuous" content like e.g. /etc/passwd. We will discuss this in the course of the article. Mostly, so called sensors (often single hosts) are used which do nothing except analyze traffic and if necessary start an alarm. As this is their only task it is possible to secure those sensors better. In this context there exists often the so called "Stealth Mode", i.e., the sensors act "invisible" so that it is more difficult for the attacker to localize or attack them.

"Stealth mode can be used for network sensors to make the promiscuous mode network interface card (NIC) invisible because it simply doesn’t have an IP address in this mode. This is achieved by using a separate NIC in each network sensor machine to communicate with the console over, usually, a physically isolated secure network. Stealth mode makes it more difficult for a hacker to attack the network sensor itself."

This paragraph (taken from the description for RealSecure) clarifies again what a sensor in Stealth Mode is. The sensor switches into promiscuous mode (simply put: the mode in which the network device reads the whole network traffic) and has no own IP. With this it should be made as difficult as possible for the attacker to localize the sensor. By the way, this is the mode used by packet sniffers like tcpdump...

Basically, you can use sensors in following areas:

Within the firewall (simplified):



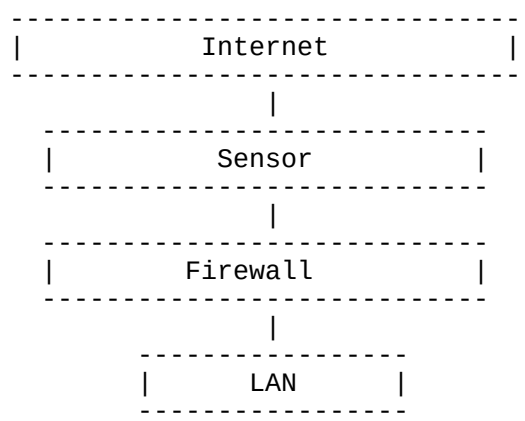
This diagram shows only one possible solution and shall only clarify that the sensors are not between DMZ and firewall. If the expression DMZ is unknown to you: it stands for DeMilitarized Zone and is an area which is secured from all sides.

If sensors are within the firewall it is easier for them to decide if a firewall was eventually configured

improperly and in addition you get to know if a potential attack came through the firewall or not. According to experience, sensors create less false positives as they act "less inconspicuously" and therefore traffic is less (whereby the probability of a false alarm drops).

Sensors placed within a firewall serve as intrusion detection. Should your sensor fulfill this task you should then put it within the firewall ...

Outside the firewall (simplified):



Sensors are often placed outside the external firewall as shown in the diagram. The reason for this is that the sensor can receive and analyze the whole traffic from the internet. If you place the sensor here it is not ensured that really all attacks are filtered and detected, e.g., TCP attacks. In this case, you would try to detect the attack by using so called signatures (more about this in the part about signatures). Nevertheless, this placement is preferred by many experts because there is the advantage to log and analyze the attacks (to the firewall...), thereby, the admin can see where he should change the firewall configuration.

Sensors placed outside the firewall serve as attack detection (different to placing in within the firewall!). If your sensors should detect these attacks you should put them outside the firewall...

- Outside and within the firewall

Actually, this variant connects both previously mentioned variants. But, the danger is that you configure the sensors and/or the firewall wrongly, i.e., you cannot simply add the advantages of both variants to this variant. These are not the only possibilities to placing sensors, you can, of course, place them somewhere else but the above mentioned places are the most commonly used.

Advantages:

- the sensors can be secured well as they "only" observe traffic
- you can detect scans better - on the basis of signatures... you can "filter" traffic (actually, we will show later that this is not always the case)

Disadvantages:

- the probability of so called false negatives (attacks are not detected as attacks) is high as it is difficult to control the whole network
- mostly, they have to operate on encrypted packets where analysis of packets is complicated
- as a difference to host-based IDS they do not see the impacts of an attack

Examples:

- NetRanger [http://www.cisco.com]
- Dragon [http://www.securitywizards.com]
- NFR [http://www.nfr.net]
- Snort [http://www.snort.org]
- DTK [http://all.net/dtk/dtk.html]
- ISS RealSecure [http://www.uh.edu/infotech/software/unix/realsecure/index.html]

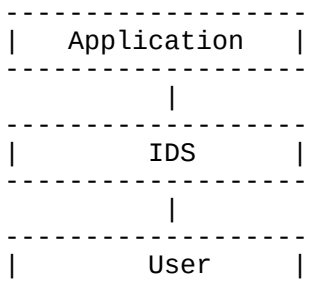
Though I distinguish between HIDS and NIDS the difference becomes smaller and smaller as in the meanwhile HIDS, too, have the basic functionality of NIDS. Known ID Systems like ISS RealSecure call themselves not only NIDS but "host-based and network-based IDS". In the future, the difference between both systems will become even less clear (so that both systems "grow together" more and more).

Network Node Intrusion Detection (NNIDS)

Basically, this new type (NNIDS) works like typical NIDS, i.e., you take packets from network traffic and analyze them. But it only concerns packets which are addressed to the network node (this is where the name comes from). Another difference between NNIDS and NIDS is that NIDS run in promiscuous mode while NNIDS does not run in promiscuous mode. As not every packet is analyzed the performance of the system will not suffer to much, such systems run very fast as a rule.

Application Based Intrusion Detection

Application Based IDS's are a subgroup of host-based IDS but I mention them here separately. It monitors the interaction between user and program whereby mainly additional log files are created to provide information about the activities. As you operate directly between user and program you can very easily "filter" conspicuous behaviour. You can visualize an ABIDS as in the following diagram:



Advantages:

- you work at unencrypted level, in opposition to, e.g., Network Based IDS, whereby analysis is more feasible
- you can detect and prevent conspicuous commands which the user wants to use on/with the program

Disadvantages:

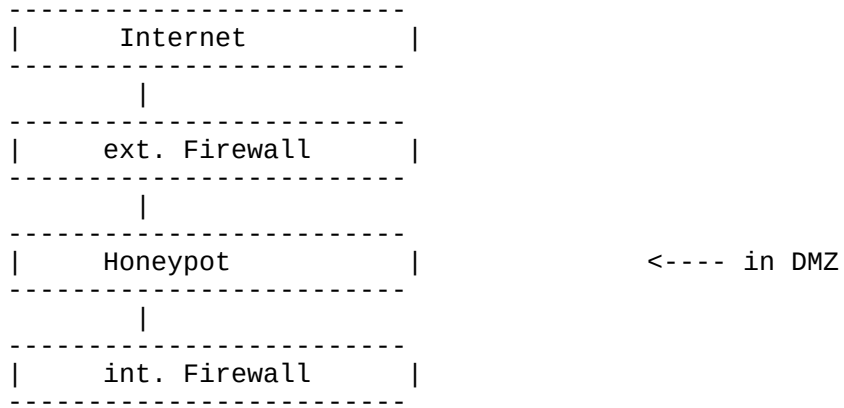
- no security and few possibilities to detect, e.g., Trojan horses (as you do not act in kernel space)
- the log files created by this kind of IDS are easy attack targets for "attackers" and not so secure like, e.g., operating system audit trails

Stack Based Intrusion Detection

Also a new kind of IDS is the so called Stack Based Intrusion Detection System (SBIDS). But momentarily, I do not have enough information which would allow me to brief you about this type of IDS. In a future version of this paper the description will certainly follow...

Honeypot

The word Honeypot is used in a lot of differnt contexts. To avoid confusion I will stick to the definition from the SANS Institute's Intrusion Deception FAQ: "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". In some cases, a honeypot is simply a "box". From the outside it appears vulnerable, while it logs traffic and also analyzes it. Thus, because honeypots appear vulnerable and no connections should be created every connection to the honeypot is seen as suspicious.



So, the internal network does not have to be exposed because a honeypot is normally placed in the DMZ (DeMilitarized Zone). The major task of a honeypot consists mostly in analyzing traffic, e.g., when certain processes were started, which files were changed..., so that you can create a quite good profile of potential attackers.

But not all honeypots are alike so you can distinguish between three "variants" which enhance security of the system in different ways:

Prevention: prevent an attack detection: notify that there was an attack (possibly a localization of the attacker, what kind of attack, which services are concerned...?) Reaction: the (counter)action, detection alone does not serve anything if you do not take steps against it. Padded Cell Systems offers special possibilities which concern counteractive measures. So, reaction implies what you do / the IDS does when you react to an attack.

Later, we will clarify the different response options of IDS again... In addition, honeypots can be classified in two bigger categories, research and production honeypots. Production honeypots are to help lessen potential risks in a network while research honeypots serve to collect and inquire information about the attackers.

Prevention:
Honeypots are surely not the most appropriate systems to avert attacks. As you will see later at Padded Cell Systems, a wrong programmed and wrong configured honeypot can even facilitate attacks whereby this covers the whole subject of IDS.

Detection:
Seemingly, that is the biggest advantage of honeypots as they can be excellent in detecting attacks. In this context, they can above all analyze and interpret system logs. The placement of the honeypot plays a decisive role, an admin can only benefit by honeypots if the are configured and placed properly. Often, they are placed between important servers to detect eventual scans of the whole network or in the proximity of one important server to detect illegal access with the help of port redirection (e.g., if someone tries to access the server at certain ports he will be redirected to the honeypot, as this access should not be allowed there should be a warning).

Reaction:
In the part about Responses (read beneath), you can see which possibilities a honeypot has to react on events. When using/creating a honeypot you should keep in mind that the host should look most attractive to an attacker, i.e., it should not be too difficult to attack the host. Basically, there should be few changes from the default configuration as too many changes only seem conspicuous. Nevertheless, you should not forget the actual idea, i.e., control traffic, log activities, ... One approach of which I have heard several times is to place the honeypot in its own subnet behind a firewall. This has normally alarm capabilities and so can display warnings. As logs are sought-after targets of attackers you should not log them on the host itself but send it to another server. Sometimes, sniffers are used to search traffic for certain signs and "log" traffic. If you collected enough information you should analyze the logs and look for weaknesses of the network, respectively rectify them. Because of the amount of information it should be easier to close security holes and take action against attackers. But you should consider that honeypots are not completely legal, respectively you should be attentive when using honeypots. The problem ist that a honeypot can be interpreted as an "invitation to attack" and if you realize that the host

was attacked (and you do not initiate counteractive measures) that could be interpreted additionally as an act of gross negligence. Some judiciary argumentation against honeypots sound banal but you should check the law at your place. If someone attacks another network from your network and creates damage you will (probably) pay for it for already mentioned reasons.
Further inquiries found that the application of honeypots is, e.g., in Europe is no problem (if you find a law which contradicts this, write to me, please. My search did not find such a law...).

Padded Cell Systems

These systems normally work together with one of the other systems. If the used IDS notifies about an attack the respective attacker is forwarded to a "padded cell host". As soon as they are in this padded cell host they cannot make any damage as the whole environment is simulated, a "bogus" environment. It is important that this environment is as realistically presented as possible, respectively the attacker has to think that the attack was successful (sort of). There is a possibility to log, analyze and follow every activity of the attacker. The problem with using padded cell systems is that it is possible that it is illegal to employ them in a particular country (as eventually such counter activities of the IDS can be seen as "attack", though they are only meant to collect information about the real attacker).

Advantages:

- once in a padded cell host, attackers cannot do any more damage as it is only a "bogus" environment
- the admin can follow/log the activities of the attacker directly to get more information about the attack and the target of the attack and so is able to initiate counteractive measures more easily

Disadvantages:

- eventually, usage of padded cell systems is not legal (the same as honeypots, in Europe this should be no problem)
- the implementation of such a system is very difficult and requires some knowledge as the whole environment has to be simulated correctly. If the admin makes a little mistake somewhere this system could by all means open additional security holes

Types of attack

Before you develop or use an IDS you should specify the potential dangers, also which attacks you expect. Despite the different possibilities attacks and their target can be defined in four categories:

1)Confidentiality
The consequences of the attack are that the mutual trust to a certain user changed (mostly to his favor ;), e.g., that you do not have to authenticate anymore for a certain program... Such circumstances are still abused these days, e.g., if there is a mutual trust between two hosts, i.e., the user of one host can log in another without password (or else). If an attacker accomplishes a comparable effect with his attack it is

sometimes very difficult to discover such "abused" mutual trust.

2)Integrity
If the user changes important configuration and system files, replaces existing binaries by his own... Often, existing binaries (like, e.g., /bin/login) are replaced by own binaries. There, the respective user only has to give a fixed password (in the source) and gets (mostly) root privileges. Such attacks serve to expand ones own privileges, e.g., by means of installing a login backdoor. In fact, there are tools like Tripwire but not everyone uses it. In addition, there are often devastating errors in configuration and use which make Tripwire an extra risk in such cases.

3)Availability
By this, the reachability of the system is affected. This could cause the ban of certain users to log in at all or that you can only log in at certain times... The aim is mostly to work "undisturbed" and cannot be discovered by any user.

4)Control
E.g, if the attack is for the purpose of "overtaking" the system that has control over files, programs... When this happens, you should consider that the attacker gets also all other previously listed options. If he has total control over the system he can change, restrict... what he wants.

Before I get to the different types of attack (DoS, DDoS, Scans, ...) I will make just a little excursion to the world of Integrity Checkers. Tools like Tripwire are part of host based IDS, the issue of an Integrity Checker is to check the integrity of diverse files and start an alarm if you detect changes on a file.

1) Creation of a database
The first step after the installation of an Integrity Checker like Tripwire is the creation of a database. This step should (must) be made in a situation at which the condition of the system is not compromised. If you create the database at a later time (and maybe the attacker has replaced the existing binaries) the use of an Integrity Checker would not work as it should. In such a case, the replaced binaries would be considered as "originals" and if the admin replaces these binaries by the actual "originals", an alarm would be started. Most of the tools offer extensive possibilities to specify files/directories of which you want to create checksums. We simply generate a fingerprint of the system. (Tripwire calls this Database Initialization Mode)

2) Check of Integrity
After the admin has a database (the fingerprint) of the system he can check his system when he wants to. This is done simply by comparing the checksums in the database with the currently available files on the disk. Tripwire offers more possibilities. Simply read the manpage for Tripwire or associated documentation.

Those two steps are actually found in most Integrity Checkers. Tripwire offers the possibility to update the database (when having installed new tools...) or to update the policy file, test the email notification system...

Which errors can be made when using Integrity Checkers ?

The first and grossest error an admin can commit is to create an MD5 hashsum database when existing binaries have been replaced by an attacker already. If an admin assumes that an attack was successful

and afterwards creates the hashsum database of this "compromised" system, the binary, replaced by the attacker (e.g., login backdoor), would be considered as the "real" binary. You should create the database as soon as possible after installation. One other big error when using Integrity Checkers (like, e.g., Tripwire) is to leave the hashsum database on hard disk. On first sight, it seems absolutely normal to leave the database on hard disk but if you have to leave it there you should at least make sure that the partition/the medium is read-only. You should make sure that no one can write-access our database. If the attacker could write the database he could change it as he wants. If you worked with Tripwire before you surely know that you can adjust in the configuration file of which files the integrity should be checked. But what if the attacker can read/write this configuration file? He could change search paths so that the directory with the changed binaries would not be scanned at all. It is best not to leave the configuration on hard disk and put it onto a read-only medium. Some will think that it is too much trouble to put the files on a read-only medium, actually, you should consider some more time exposure for more security (an interesting aspect with user-friendliness is that many security holes are/were opened because programs are made user-friendly. In our case it means that you do get less user-friendliness but a higher level of security). Tripwire (and other integrity checkers) are surely capable programs which can enhance security and prevent potential attackers from successfully attacking. But the best tool is of no use if the other security settings of the system are not okay. So, do not demand of integrity checkers that they do all your work.

A newer type of integrity checkers are the so called Realtime Integrity Checkers. In contrast to "normal" integrity checkers they check file integrity in realtime. Here is a short scheme:

1) Here too, the first step is the creation of a hashsum database
2) Before someone can run a program the hashsum of the file is procuced. If the value does not accord to the one in the database the binary was replaced. If this is the case, execution is prohibited.
This concept is only one possible for realtime integrity checkers (at least I arranged this concept for me). As a difference to integrity checkers you do not rely on the admin checking files regularly, you try to check the correctness of the binary before execution and not let it execute if necessary...

Besides the above mentioned categories (integrity, control, ...) attacks can be differentiated otherwise, of course, e.g., how to attack, respectively with what means. Also, here are naturally many possibilities, but the most common attacks against IDSs are:

- Scans
These days, scans are the common "attacks", the problem here is that the IDS should not produce too many false positives. Mostly, scans serve to get (further) information about a host/network (e.g., to start subsequent attacks). What information an attacker can get about your own network you can see with the following scan result (nmap - only a small example which does not at all show all possibilities of nmap):

```
....
Host (192.168.0.0) seems to be a subnet broadcast address (returned 1
extra pings).
Skipping host. Interesting ports on playground.yuma.net 192.168.0.1):
Port           State          Protocol               Service
22             open           tcp            ssh
111            open           tcp            sunrpc
635            open           tcp            unknown
1024           open           tcp            unknown
2049           open           tcp            nfs

TCP Sequence Prediction: Class = random positive increments
```

```
                           Difficulty=3916950 (Good luck!)
Remote operating system guess:
       Linux 2.1.122 - 2.1.132; 2.2.0-pre1 - 2.2.2

Interesting ports on vectra.yuma.net (192.168.0.5):
Port           State           Protocol                Service
13             open            tcp             daytime
21             open            tcp             ftp
22             open            tcp             ssh
23             open            tcp             telnet
37             open            tcp             time
79             open            tcp             finger
111            open            tcp             sunrpc
113            open            tcp             auth
513            open            tcp             login
514            open            tcp             shell

TCP Sequence Prediction: Class = random positive increments
                           Difficulty = 17719 (Worthy challenge)
Remote operating system guess: OpenBSD 2.2. - 2.3

Nmap run completed -- 256 IP addresses (2 hosts up) scanned in 6 seconds
```

Mainly, you can find out the following:

- which operating system?
- which versions of certain programs run
- which services run that you can "attack" eventually
- which ports are open
- ...

Using many of the most different scan techniques together result in a mass of information which allows the attacker to initiate his attack. Though, I will not describe/mention all scan techniques, I will mention some of the often used ones (if there are questions referring to protocols... look at the respective RFCs):

Ping scanning:
Ping scans are used to find out which hosts are online. For this, you send the host (or the hosts) an ICMP datagram of type 8 (i.e., echo request) and wait for an ICMP answer datagram of type 0 (i.e., echo reply). Sometimes, you do not send only an echo request but additionally an ACK as sometimes ICMP is blocked. If there is an RST answer the host is online.

Nmap parameter: -sP

TCP Scanning (Vanilla) :
With TCP scanning you (mostly) try to connect() on all ports after you did the three-way-handshake, i.e., you made a connection to the ports (the connections to the ports were successful) the reply value of connect() is checked. For the attacker the reply value informs if the used port (or the ports) is open or closed. The aim of a TCP scan is to find out if ports are open/closed.

Nmap parameter: -sT

The following nmap output is of one of my hosts, directly after a default installation. I deliberately made no changes (in the configuration):

```
 [Socma]$ nmap -sT localhost

Starting nmap V. 2.54BETA36 ( www.insecure.org/nmap/ )
Interesting ports on Diablo (127.0.0.1):
(The 1552 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
23/tcp     open        telnet
80/tcp     open        http
111/tcp    open        sunrpc
113/tcp    open        auth
6000/tcp   open        X11

Nmap run completed -- 1 IP address (1 host up) scanned in 1 second
```

A part of a tcpdump trace (of this scan):

```
 02:10:15.804704 Diablo > Diablo: icmp: echo request
                          4500 001c 2db8 0000 3501 5a27 7f00 0001
                          7f00 0001 0800 fc95 fb69 0000
 02:10:15.814704 Diablo > Diablo: icmp: echo reply (DF)
                          4500 001c 0000 4000 ff01 7dde 7f00 0001
                          7f00 0001 0000 0496 fb69 0000
 02:10:15.814704 Diablo.58725 > Diablo.http: . ack 110306597 win 3072
                          4500 0028 d223 0000 2a06 c0aa 7f00 0001
                          7f00 0001 e565 0050 ad48 0003 0693 2525
                          5010 0c00 e718 0000
 02:10:15.814704 Diablo.http > Diablo.58725: R 110306597:110306597(0)
         win 0 (DF)
                          4500 0028 0000 4000 ff06 7dcd 7f00 0001
                          7f00 0001 0050 e565 0693 2525 0000 0000
                          5004 0000 a070 0000
 02:10:16.114704 Diablo.1727 > Diablo.821: S 196002918:196002918(0)
 win 32767 <mss 16396,sackOK,timestamp 213509[|tcp]> (DF)
                          4500 003c 8663 4000 4006 b656 7f00 0001
                          7f00 0001 06bf 0335 0bae c466 0000 0000
                          a002 7fff 739c 0000 0204 400c 0402 080a
                          0003 4205
 02:10:16.114704 Diablo.821 > Diablo.1727: R 0:0(0) ack 196002919
        win 0 (DF)
                          4500 0028 0000 4000 ff06 7dcd 7f00 0001
                          7f00 0001 0335 06bf 0000 0000 0bae c467
                          5014 0000 d7c4 0000
 02:10:16.114704 Diablo.1728 > Diablo.440: S 195504823:195504823(0)
 win 32767 <mss 16396,sackOK,timestamp 213509[|tcp]> (DF)
                          4500 003c 68b2 4000 4006 d407 7f00 0001
                          7f00 0001 06c0 01b8 0ba7 2ab7 0000 0000
                          a002 7fff 0ecf 0000 0204 400c 0402 080a
                          0003 4205
```

UDP scanning:
The intention of UDP scans are analog to TCP scans as you want to find out open UDP ports. A scan runs differently as UDP is a connectionless protocol (TCP is connection-oriented). UDP scanning "uses" ICMP, if you send to the actual ports a 0 byte UDP packet to wait for an ICMP "answer". If there is a notify that the port is unreachable ("Port unreachable"/code value 3) this means that the port is closed. If the respective admin, e.g., in his /etc/inetd.conf, deactivated certain services and you try to send a packet

to the respective port this would result in the error message "Port Unreachable"...

Nmap parameter: -sU

```
 [Socma]$ nmap -sU localhost

Starting nmap V. 2.54BETA36 ( www.insecure.org/nmap/ )
Interesting ports on Diablo (127.0.0.1):
(The 1459 ports scanned but not shown below are in state: closed)
Port       State       Service
111/udp    open        sunrpc

Nmap run completed -- 1 IP address (1 host up) scanned in 4 seconds
```

The associated tcpdump trace:

```
10:41:55.954397 Diablo > Diablo: icmp: echo request
                         4500 001c cc8f 0000 2801 c84f 7f00 0001
                         7f00 0001 0800 8471 738e 0000
10:41:55.954397 Diablo > Diablo: icmp: echo reply (DF)
                         4500 001c 0000 4000 ff01 7dde 7f00 0001
                         7f00 0001 0000 8c71 738e 0000
10:41:55.964397 Diablo.63793 > Diablo.http: . ack 994287972 win 2048
                         4500 0028 79e3 0000 2506 1deb 7f00 0001
                         7f00 0001 f931 0050 06d8 0003 3b43 a164
                         5010 0800 cccd 0000
10:41:55.964397 Diablo.http > Diablo.63793: R 994287972:994287972(0)
     win 0 (DF)
                         4500 0028 0000 4000 ff06 7dcd 7f00 0001
                         7f00 0001 0050 f931 3b43 a164 0000 0000
                         5004 0000 dbb4 0000
10:41:56.274397 Diablo.63773 > Diablo.15: udp 0
                         4500 001c 8a0b 0000 3011 02c4 7f00 0001
                         7f00 0001 f91d 000f 0008 08af
10:41:56.274397 Diablo > Diablo: icmp: Diablo udp port 15
     unreachable (DF) [tos 0xc0]
                         45c0 0038 0000 4000 ff01 7d02 7f00 0001
                         7f00 0001 0303 fb18 0000 0000 4500 001c
                         8a0b 0000 3011 02c4 7f00 0001 7f00 0001
                         f91d 000f
10:41:56.274397 Diablo.63773 > Diablo.1459: udp 0
                         4500 001c 6c2f 0000 3011 20a0 7f00 0001
                         7f00 0001 f91d 05b3 0008 030b
10:41:56.274397 Diablo > Diablo: icmp: Diablo udp port 1459
     unreachable (DF) [tos 0xc0]
                         45c0 0038 0100 4000 ff01 7c02 7f00 0001
                         7f00 0001 0303 fb18 0000 0000 4500 001c
                         6c2f 0000 3011 20a0 7f00 0001 7f00 0001
                         f91d 05b3
```

Another variant of a UDP scan (UDPrecvfrom() and write() scan) consists in scanning every port twice. The just now mentioned method uses ICMP with "Port Unreachable", but only root receives this message. If you scan a closed port twice you get, after the second scan: "Error 13 : Try Again"...

ACK scanning:
With sending an ACK packet to a port of a firewall you find out which ports are filtered and which are

not. If you receive an RST answer it means that the referring port is "unguarded", respectively is not filtered, else you get an ICMP error message. So, you do not find out which ports are open but you get more precise information about the firewall (and its configuration).

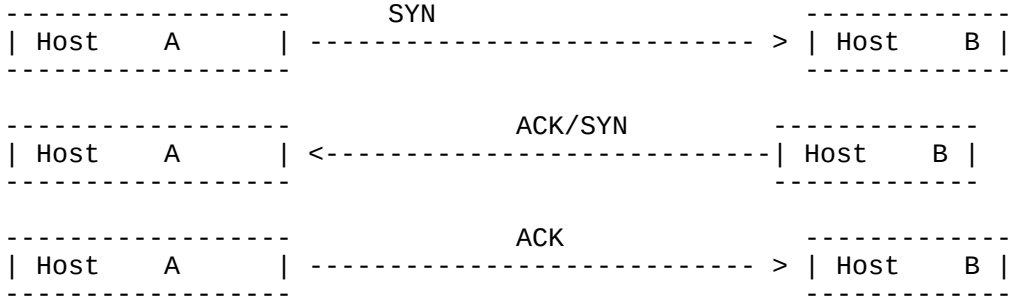
Nmap parameter : -sA

```
 [Socma]$ nmap -sA localhost

Starting nmap V. 2.54BETA36 ( www.insecure.org/nmap/ )
All 1558 scanned ports on Diablo (127.0.0.1) are: UNfiltered

Nmap run completed -- 1 IP address (1 host up) scanned in 6 seconds
```

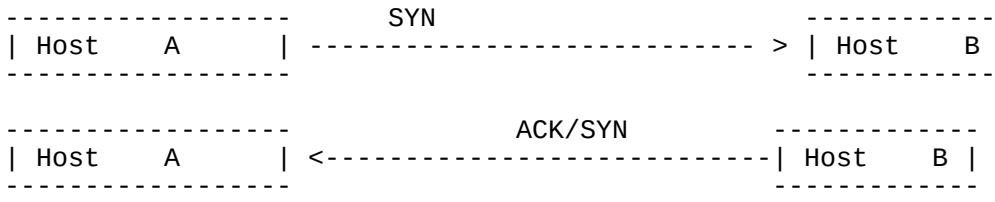
The tcpdump trace:

```
10:45:51.864397 Diablo > Diablo: icmp: echo request
                         4500 001c 1617 0000 3901 6dc8 7f00 0001
                         7f00 0001 0800 113d e6c2 0000
10:45:51.864397 Diablo > Diablo: icmp: echo reply (DF)
                         4500 001c 0000 4000 ff01 7dde 7f00 0001
                         7f00 0001 0000 193d e6c2 0000
10:45:51.864397 Diablo.53119 > Diablo.http: . ack 2682022466 win 3072
                         4500 0028 0dda 0000 3206 7cf4 7f00 0001
                         7f00 0001 cf7f 0050 0650 0003 9fdc 6a42
                         5010 0c00 c590 0000
10:45:51.864397 Diablo.http > Diablo.53119: R 2682022466:2682022466(0)
     win 0 (DF)
                         4500 0028 0000 4000 ff06 7dcd 7f00 0001
                         7f00 0001 0050 cf7f 9fdc 6a42 0000 0000
                         5004 0000 d7ef 0000
10:45:52.164397 Diablo.53099 > Diablo.14: . ack 2457451592 win 3072
                         4500 0028 218d 0000 3206 6941 7f00 0001
                         7f00 0001 cf6b 000e 5938 4710 9279 bc48
                         5010 0c00 e74d 0000
10:45:52.164397 Diablo.14 > Diablo.53099: R 2457451592:2457451592(0)
     win 0 (DF)
                         4500 0028 0000 4000 ff06 7dcd 7f00 0001
                         7f00 0001 000e cf6b 9279 bc48 0000 0000
                         5004 0000 93a2 0000
10:45:52.164397 Diablo.53099 > Diablo.imap3: . ack 2457451592 win 3072
                         4500 0028 a075 0000 3206 ea58 7f00 0001
                         7f00 0001 cf6b 00dc 5938 4710 9279 bc48
                         5010 0c00 e67f 0000
```

Stealth scanning (NULL, XMAS, FIN, SYN, ...): With stealth scanning you "abuse" the three-way-handshake. I present the three-way-handshake shortly:



The problem of TCP scans is that they are very conspicuous (as every time it does a three-way-handshake). With stealth scanning the following happens instead:



This diagram seems to look like the three-way-handshake but with a basic difference: The shown diagram would have no connection between A and B, respectively Host B would think the connection exists, though the connection doesn't exist until A sends a further ACK to B (it is also called a "half-open" port...). The above shown SYN scan implies that the port of the target host is open (because of the ACK/SYN), if it were closed you would receive a RST/ACK back.

Nmap parameter : -sS

```
 [Socma]$ nmap -sS localhost

Starting nmap V. 2.54BETA36 ( www.insecure.org/nmap/ )
Interesting ports on Diablo (127.0.0.1):
(The 1552 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
23/tcp     open        telnet
80/tcp     open        http
111/tcp    open        sunrpc
113/tcp    open        auth
6000/tcp   open        X11

Nmap run completed -- 1 IP address (1 host up) scanned in 3 seconds

Tcpdump trace:

10:47:41.674397 Diablo > Diablo: icmp: echo request
                          4500 001c 8f08 0000 3501 f8d6 7f00 0001
                          7f00 0001 0800 99a9 5e56 0000
10:47:41.674397 Diablo > Diablo: icmp: echo reply (DF)
                          4500 001c 0000 4000 ff01 7dde 7f00 0001
                          7f00 0001 0000 a1a9 5e56 0000
10:47:41.674397 Diablo.58038 > Diablo.http: . ack 1561498752 win 3072
                          4500 0028 afe5 0000 3206 dae8 7f00 0001
                          7f00 0001 e2b6 0050 82b0 0003 5d12 9480
                          5010 0c00 4e85 0000
10:47:41.674397 Diablo.http > Diablo.58038: R 1561498752:1561498752(0)
     win 0 (DF)
                          4500 0028 0000 4000 ff06 7dcd 7f00 0001
                          7f00 0001 0050 e2b6 5d12 9480 0000 0000
                          5004 0000 dd44 0000
10:47:41.984397 Diablo.58018 > Diablo.1488: S 2803535203:2803535203(0)
     win 3072
                          4500 0028 a4f5 0000 3206 e5d8 7f00 0001
                          7f00 0001 e2a2 05d0 a71a 8d63 0000 0000
                          5002 0c00 88ef 0000
```

```
10:47:41.984397 Diablo.1488 > Diablo.58018: R 0:0(0) ack 2803535204
     win 0 (DF)
                          4500 0028 0000 4000 ff06 7dcd 7f00 0001
                          7f00 0001 05d0 e2a2 0000 0000 a71a 8d64
                          5014 0000 94dc 0000
```

Now, other scans join the game: FIN scanning, NULL scanning and XMAS scanning. FIN scanning only sends a FIN message to the "target host", though no connection exists between them. At a closed port RST is sent back, else nothing happens.

Nmap parameter : -sF

```
 [Socma]$ nmap -sF localhost

Starting nmap V. 2.54BETA36 ( www.insecure.org/nmap/ )
Interesting ports on Diablo (127.0.0.1):
(The 1552 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
23/tcp     open        telnet
80/tcp     open        http
111/tcp    open        sunrpc
113/tcp    open        auth
6000/tcp   open        X11

Nmap run completed -- 1 IP address (1 host up) scanned in 6 seconds

Tcpdump trace:

10:48:28.704397 Diablo > Diablo: icmp: echo request
                          4500 001c b29d 0000 3401 d641 7f00 0001
                          7f00 0001 0800 a1a7 5658 0000
10:48:28.704397 Diablo > Diablo: icmp: echo reply (DF)
                          4500 001c 0000 4000 ff01 7dde 7f00 0001
                          7f00 0001 0000 a9a7 5658 0000
10:48:28.704397 Diablo.52201 > Diablo.http: . ack 2873378189 win 4096
                          4500 0028 cbeb 0000 2b06 c5e2 7f00 0001
                          7f00 0001 cbe9 0050 9020 0003 ab44 458d
                          5010 1000 54a3 0000
10:48:28.704397 Diablo.http > Diablo.52201: R 2873378189:2873378189(0)
     win 0 (DF)
                          4500 0028 0000 4000 ff06 7dcd 7f00 0001
                          7f00 0001 0050 cbe9 ab44 458d 0000 0000
                          5004 0000 f4d2 0000
10:48:29.004397 Diablo.52181 > Diablo.233: F 0:0(0) win 4096
                          4500 0028 10c6 0000 2b06 8108 7f00 0001
                          7f00 0001 cbd5 00e9 0000 0000 0000 0000
                          5001 1000 d522 0000
10:48:29.004397 Diablo.233 > Diablo.52181: R 0:0(0) ack 1 win 0 (DF)
                          4500 0028 0000 4000 ff06 7dcd 7f00 0001
                          7f00 0001 00e9 cbd5 0000 0000 0000 0001
                          5014 0000 e50e 0000
```

NULL and XMAS scans are of special interest (above all with practical implementation of protocol anomaly detection). It is called XMAS scan because all flags are set: SYN, ACK, FIN, URG, PUSH. As with FIN scanning you send back RST if the port is closed.

Nmap parameter : -sX

```
 [Socma]$ nmap -sX localhost

Starting nmap V. 2.54BETA36 ( www.insecure.org/nmap/ )
Interesting ports on Diablo (127.0.0.1):
(The 1552 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
23/tcp     open        telnet
80/tcp     open        http
111/tcp    open        sunrpc
113/tcp    open        auth
6000/tcp   open        X11

Nmap run completed -- 1 IP address (1 host up) scanned in 5 seconds

Tcpdump trace:

10:44:24.004397 Diablo > Diablo: icmp: echo request
                         4500 001c ffcc 0000 2a01 9312 7f00 0001
                         7f00 0001 0800 103d e7c2 0000
10:44:24.004397 Diablo > Diablo: icmp: echo reply (DF)
                         4500 001c 0000 4000 ff01 7dde 7f00 0001
                         7f00 0001 0000 183d e7c2 0000
10:44:24.004397 Diablo.36398 > Diablo.http: . ack 718216305 win 2048
                         4500 0028 2e28 0000 2906 65a6 7f00 0001
                         7f00 0001 8e2e 0050 9220 0003 2acf 1c71
                         5010 0800 41f0 0000
10:44:24.004397 Diablo.http > Diablo.36398: R 718216305:718216305(0)
     win 0 (DF)
                         4500 0028 0000 4000 ff06 7dcd 7f00 0001
                         7f00 0001 0050 8e2e 2acf 1c71 0000 0000
                         5004 0000 dc1f 0000
10:44:24.304397 Diablo.36378 > Diablo.1996: FP 0:0(0) win 2048 urg 0
                         4500 0028 7651 0000 2906 1d7d 7f00 0001
                         7f00 0001 8e1a 07cc 0000 0000 0000 0000
                         5029 0800 13d3 0000
10:44:24.304397 Diablo.1996 > Diablo.36378: R 0:0(0) ack 1 win 0 (DF)
                         4500 0028 0000 4000 ff06 7dcd 7f00 0001
                         7f00 0001 07cc 8e1a 0000 0000 0000 0001
                         5014 0000 1be7 0000
```

The other possibility, called NULL scan, means that no flag is set, if the port is closed an RST is sent back.

Nmap parameter : -sN

```
 [Socma]$ nmap -sN localhost

Starting nmap V. 2.54BETA36 ( www.insecure.org/nmap/ )
Interesting ports on Diablo (127.0.0.1):
(The 1552 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
23/tcp     open        telnet
80/tcp     open        http
111/tcp    open        sunrpc
113/tcp    open        auth
6000/tcp   open        X11
```

```
Nmap run completed -- 1 IP address (1 host up) scanned in 5 seconds

Tcpdump trace:

10:43:37.594397 Diablo > Diablo: icmp: echo request
                        4500 001c 2ecf 0000 2c01 6210 7f00 0001
                        7f00 0001 0800 8f87 6878 0000
10:43:37.594397 Diablo > Diablo: icmp: echo reply (DF)
                        4500 001c 0000 4000 ff01 7dde 7f00 0001
                        7f00 0001 0000 9787 6878 0000
10:43:37.604397 Diablo.34607 > Diablo.http: . ack 1932747046 win 4096
                        4500 0028 ee0f 0000 3706 97be 7f00 0001
                        7f00 0001 872f 0050 5b20 0003 7333 6126
                        5010 1000 ead5 0000
10:43:37.604397 Diablo.http > Diablo.34607: R 1932747046:1932747046(0)
     win 0 (DF)
                        4500 0028 0000 4000 ff06 7dcd 7f00 0001
                        7f00 0001 0050 872f 7333 6126 0000 0000
                        5004 0000 5605 0000
10:43:37.904397 Diablo.34587 > Diablo.408: . win 4096
                        4500 0028 e3bb 0000 3706 a212 7f00 0001
                        7f00 0001 871b 0198 0000 0000 0000 0000
                        5000 1000 192f 0000
10:43:37.904397 Diablo.408 > Diablo.34587: R 0:0(0) ack 0 win 0 (DF)
                        4500 0028 0000 4000 ff06 7dcd 7f00 0001
                        7f00 0001 0198 871b 0000 0000 0000 0000
                        5014 0000 291b 0000
```

You do not need the complete three-way-handshake, thus, stealth scanning (like that mentioned) is less conspicuous than TCP scanning. IDS should detect those abnormalities in all cases (XMAS and NULL)...

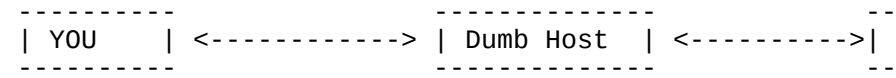
FTP bounce:
With some ftpds the PORT command can be abused to establish an arbitrary connection from the ftp server to another machine. But first, a little overview how this "normally" happens. First, the client makes a connection to the ftp server (port 21), the ftp server "creates" a second connection back to the client (to be able to send back data). For this second connection you use the PORT command. The interesting thing here is that the command contains the IP and the port (which is to be opened) of the client. Subsequently, the server creates a connection, where source port 20 and the destination port are the ports specified by the PORT command. The point of attack is the PORT command with which you can manipulate the port of the (supposed) client to connect to the victim instead of our host. After the IP and port were manipulated you can initialize the actual traffic by "list" or "get". Now, you check the answer of ftp, because if we get a "425: Can’t build data connection: Connection refused" this specified port is closed. If we receive "150 : File status okay about to open data connection" or "226: Closing data connection. Requested file action successful (for example, file transfer or file abort)" instead as an answer we know that the specified port is open.

Nmap parameter: -b

Fragmented packets: This method uses the fragmentation of an IP packet by TCP. Normally, fragmentation occurs when the datagrams are bigger than the possible size, this size limitation is called MTU (Maximum Transmission Unit). Fragmented datagrams are put together at the end of a node. This behaviour can be abused. Not every IDS/firewall works with fragmentation, i.e., there are sometimes errors with fragmented packets. Instead of sending our packet, normally, we subdivide it (in fragments).

These contain "common" data like source IP, destination IP, source port... Now, it is possible that the running firewall/IDS has problems with putting together fragments. These problems can manifest themselves in different ways, either it comes to a crash of the whole system or the packet goes through. Our packet could possibly get through because the putting together was erroneous and the packet was falsely specified "harmless". Sometimes, not all fragments are checked properly, i.e., only a certain fragment is checked so that our packet gets through again. With this technique you can scan less conspicuous as the traffic could be marked "harmless" and would not start an alarm. On the other hand this theory depends on the IDS (firewall) having problems with processing and putting together fragments.

Reverse Ident Scanning:
First, a section from RFC 1413, the RFC for Identification Protocol: "The Identification Protocol (a.k.a., "ident", a.k.a., "the Ident Protocol") provides a means to determine the identity of a user of a particular TCP connection. Given a TCP port number pair, it returns a character string which identifies the owner of that connection on the server's system." Reverse Ident Scanning uses identd to ask for the owner of the running process. This technique serves above all to find daemons which run as root in order to attack precisely these daemons.



Here Dumb Host should have as little traffic as possible. The reason for this will be more clear in the end. Why is Dumb Host used, why do we need one at all? Ok, this question leads to the actual attack and with this also to the explanation why a dumb host is necessary. In order that you can find out if a port of "TARGET" is open or closed you should examine the IP ID field of Dumb Host. For this, Dumb Host is sent a packet (echo request) and with its reply you can read the ID field, respectively the value of the ID field. Subsequently, you can send TARGET a packet where the source address is that of Dumb Host. The answer of TARGET is then received by our dumb host. If he receives a SYN/ACK of TARGET it means that the port is open. As an answer our dumb host will then send a packet were RST is flagged. If the dumb host receives a RST/ACK of the target machine it sends no answer to TARGET. To find out which answer Dumb Host got from TARGET we send Dumb Host another ping. If the port of the target was open and sent back RST the IP ID field of Dumb Host will be incremented, with port closed and nothing happens. By reading the new IP ID value of Dumb Host you can detect if the ports were open or closed. Now, it is hopefully clearer why we use a dumb host, i.e., a host with little traffic. If there is too much traffic on the host it would be more difficult to specify which ports are open (at TARGET), the probability of getting it wrong would be higher at higher traffic.

Fingerprint OS detection:
Fingerprint OS detection aims to detect the operating on the server. Most new scanners deliver not only, e.g., "Linux" or "Solaris" for an answer but a specification of versions (of the particular operating system). For this, you make a "fingerprint" (profile) of the operating system. These days, you cannot trust banners of telnet, ftp (see above) as they can be changed and manipulated. If the attacker finds out the exact version of the target host he can build scripts, exploits, ... for it and continue with his attack. This technique exploits the fact that operating systems differ in small details (what an insight ;). As there are a lot of documents on this technique I will only give a short overview over the most important test possibilities:

- FIN test: If a host receives a FIN packet at an open port it should rather not answer (RFC 793) but

there are also exceptions, e.g., with MS Windows, BSDI, CISCO, MPS, ... which send a RESET for an answer.

-ACK sequential numbers: When sending FIN|PSH|URG to a closed port the sequential number of the ACK packet is set to the own sequential number mostly. But here also Windows makes an exception ;). Windows sends back the own sequential number +1 ...

-BOGUS flag test: If an undefined TCP Flag is used in the TCP header (in a SYN packet), Linux hosts < 2.0.35 adopt these flag settings. Other OSs reset when receiving a SYN+BOGUS packet...

-TCP initial window: Most operating systems use (almost) constant window sizes (of the reply packets). E.g., AIX delivers 0x3F25, MS NT5, OpenBSD and FreeBSD use 0x402E....

-Don’t fragment bit test: Some operating systems differ in setting this bit in some packets or not. Thus, you can additionally differentiate which OS is available...

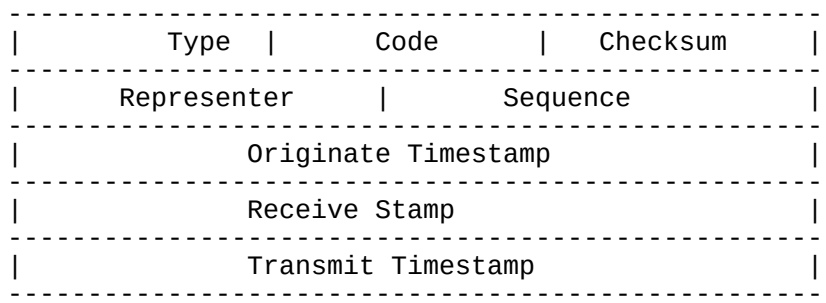
-TCP options test: The basis of this test is simply told: You send an inquiry to the particular host, set diverse options and look at the answer if there are also these options set. Those options contained in the answer are supported... As the case may be with the operating system (and version) certain options are in principal not supported, others are. Thus, the used operating system can be specified more exactly.

Nmap parameter : -O

There are many teste not discussed here but there are enough documents on the net about fingerprint OS detection. Fyodor (NMAP) has written a small paper about this. This part should only give a small overview about widespread scan techniques. For someone who works with protocol anomaly detection there were surely several starting points (certain flag combinations which you have to consider as anormal...).

Other ICMP related stuff
Aside from the mentioned echo reply/request ICMP supports further message types with which you can collect further information about the network (so-called NON - ECHO - REQUESTS):

ICMP Time Stamp Request / Reply (RFC 792):
The actual meaning of Time Stamp Requests (type 13) is to get the time settings of a remote system. If the remote host receives a Time Stamp Request it sends back a Time Stamp Reply (type 14). First, the structure of a time stamp (relative to RFC):



Before I get to the use of a Time Stamp Request for an attacker I tell you some basics about the time

stamp. For us, only the last three "fields" are of importance. Here, the respective section in the RFC:

```
The Originate Timestamp is the time the sender last touched the
message before sending it, the Receive Timestamp is the time the
echoer first touched it on receipt, and the Transmit Timestamp is
the time the echoer last touched the message on sending it."
```

As said in the RFC the sent back timestamp is the count of milliseconds since midnight UT (GMT).

Of what use is a time stamp request/reply for us?

If you are sent back a time stamp reply you would know that the host is reachable and on the other hand, with the Originate Stamp and the Receive Stamp you can estimate the load of the network (the difference is, of course, dependant on cables used, cards, ...).

At last, a tcpdump trace:

```
11:38:37.898253 Diablo > Diablo: icmp: time stamp query id 53763 seq 64548
(ttl 254, id 13170, len 40)
                4500 0028 3372 0000 fe01 8b60 7f00 0001
                7f00 0001 0d00 61fb d203 fc24 0211 c0ca
                0000 0000 0000 0000

11:38:37.898253 Diablo > Diablo: icmp: time stamp reply id 53763 seq 64548 :
org 0x211c0ca recv 0x211c0ca xmit 0x211c0ca (DF) (ttl 255, id 0, len 40)
                4500 0028 0000 4000 ff01 7dd2 7f00 0001
                7f00 0001 0e00 db43 d203 fc24 0211 c0ca
                0211 c0ca 0211 c0ca
```

ICMP Information Request / Reply (RFC 792): "This message may be sent with the source network in the IP header source and destination address fields zero (which means "this" network). The replying IP module should send the reply with the addresses fully specified. This message is a way for a host to find out the number of the network it is on. "

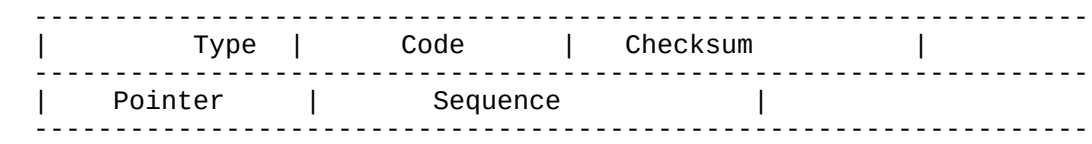
So, an Information Request (type 15) has the meaning to get the network number of the host which sent the request.

```
The address of the source in an information request message will be
the destination of the information reply message.  To form an
information reply message, the source and destination addresses
are simply reversed,..."
```

With an Information Reply (type 16) you take the source IP of the Information Request as destination IP of the reply (simply put: you send the reply to the host which requested the information). As source Ip of the reply you take the destination IP of the request...

Normally, the situation is that the sender of an Information Request sets 0 as destination address (which means "this network"). But there is also the possibility to set destination and source IP to 0 (when sending the request). In this case, the Information Reply would receive the network number of the host

in the source and the destination address field, i.e., if the source address field in the request does not equal 0 the network number of the host would only be sent back in the source IP field of the reply.

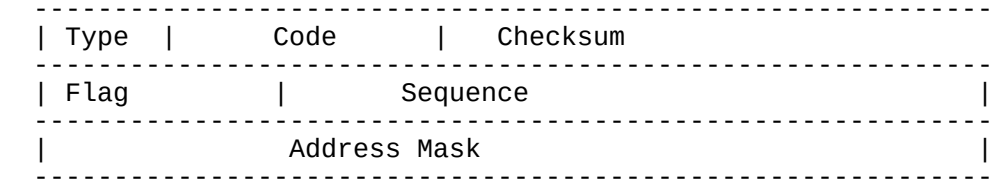


It seems that you could only send an information request within the network (see above) but that does not have to be. Some operating systems also answer an information request where the destination IP is not in the same network. In such an information reply we would receive the IP of the host (and not the network number).

At the end, again a short tcpdump trace:

```
11:42:35.608253 Diablo > Diablo: icmp: information request (ttl 255,
 id 13170, len 28)
                         4500 001c 3372 0000 ff01 8a6c 7f00 0001
                         7f00 0001 0f00 1afc d603 0000
11:42:36.608253 Diablo > Diablo: icmp: information request (ttl 255,
 id 13170, len 28)
                         4500 001c 3372 0000 ff01 8a6c 7f00 0001
                         7f00 0001 0f00 19fc d603 0100
```

ICMP Address Mask Request / Reply (RFC 950): The Address Mask Request (type 17) has been described in another RFC, for more information look at RFC950 and not in RFC792. The meaning and use of an Address Mask Request is to get the subnetmask of a connected network. If a gateway receives such a request it should send back relevant information to the respective node (Address Mask Reply - type 18)



With this, you can not only discover hosts in the network (which are online) but also get to know about the network configuration with further tests...

Tcpdump trace:

```
11:45:26.678253 Diablo > Diablo: icmp: address mask request (ttl 254, id
 13170, len 32)
               4500 0020 3372 0000 fe01 8b68 7f00 0001
               7f00 0001 1100 edd7 dc03 2524 0000 0000
```

I hope, this section showed you that you have more possibilities to get information about a network and that it does not always have to be the "normal" ping... In the respective RFCs are mostly such hints which describe what you should consider if you want to "support" the different types of request... Developers of ("real") IDSs should also consider such issues. If they should be supported you should consider that it works (read RFCs). But even this is not the end as you can read in the next section...

Even more information about the target: The use of packet filters (or more common: firewalls) is surely not very special, these days. Also the use of so called firewall modules in IDSs can be found more and more often. Often, an attacker does not have the possibilities to use some of the discussed scans as they are blocked, filtered... The aim of this small section is to show possibilities with which you can work out some of the filter/firewall rules of the target.

The principle of this idea is quite simple as you try to provoke ICMP error messages, respectively send "illegal" packets from which you can draw conclusions about the set of rules.

```
If the gateway or host processing a datagram finds a problem with
the header parameters such that it cannot complete processing the
datagram it must discard the datagram.  One potential source of
such a problem is with incorrect arguments in an option.  The
gateway or host may also notify the source host via the parameter
problem message.  This message is only sent if the error caused
the datagram to be discarded."
```

This section is from RFC 792 and is part of the description of the so called parameter problem (type 12). As you can see in this section a reason for the message "Parameter Problem" can be a false IP header, i.e., if we would send a packet with a false IP header to a host we should actually get back this error message. This error message has one additional advantage as support of this error message is "recommended" in RFC 1122:

```
A host SHOULD generate Parameter Problem messages.  An
incoming Parameter Problem message MUST be passed to the
transport layer, and it MAY be reported to the user.

DISCUSSION:
 The ICMP Parameter Problem message is sent to the
 source host for any problem not specifically covered by
 another ICMP message.  Receipt of a Parameter Problem
 message generally indicates some local or remote
 implementation error.   "
```

On the whole, this gives a very good possibility to detect hosts in a network (for the mentioned reasons).

Additionally, I refer to RFC 1812:

```
 4.3.3.5 Parameter Problem

A router MUST generate a Parameter Problem message for any error not
specifically covered by another ICMP message.  The IP header field or
IP option including the byte indicated by the pointer field MUST be
included unchanged in the IP header returned with this ICMP message.
Section [4.3.2] defines an exception to this requirement.  "
```

Nevertheless, different routers interpret this section (and also others) differently, for which reason it is not clear that a Parameter Problem is generated.

An IDS (respectively firewalls) should check fields in the IP header anyway as it can happen at times that you receive a packet with a false header, but this should happen rarely. An attacker who scans a whole network with this method (or any specified IP range) knows that the firewall/packet filter... does

not block, filter this error message. but if you do not get a Parameter Problem you know at least that this message is blocked.

This method to find out the set of rules is only the first step (the method is above all an alternative to a "normal" ping). To get an as exact as possible access control list (ACL) of possible filtering/firewall software we should get further information of the topology. To do this, you could, e.g., send the different ICMP message types to single hosts (with false IP header) and wait for a notification of a Parameter Problem. If there is a notification of a Parameter Problem on host "X" this means for us that the host does not filter this ICMP message type (and the respective host is reachable). Additionally, this means for us that the erroneous information in the IP header is not checked. If we get no Parameter Problem we can also make several conclusions. On the one hand, it would be possible that a filter filters/blocks the ICMP message type and on the other hand it is possible that the router checks the header and does not let forward this anormality, etc... As you can see, with both results you can make conclusions on possible filter rules, last and not least you get a wide representation of what is allowed and what not. Further possibilities could be to vary used protocols (TCP, UDP, ...) so that you could find out further rules. There would be, e.g., the possibility that a certain protocol is blocked/filtered.

I think the abstract on the Parameter Problem is now clear so that we can go on looking at further error messages.

The following section is again taken from RFC 1812 and describes what Destination Unreachable error is and what can be the reason for this error:

```
The ICMP Destination Unreachable message is sent by a router in
response to a packet which it cannot forward because the destination
(or next hop) is unreachable or a service is unavailable.  Examples
of such cases include a message addressed to a host which is not
there and therefore does not respond to ARP requests, and messages
addressed to network prefixes for which the router has no valid
route.

A router MUST be able to generate ICMP Destination Unreachable
messages and SHOULD choose a response code that most closely matches
the reason the message is being generated.

0 = Network Unreachable - generated by a router if a forwarding path
     (route) to the destination network is not available;

1 = Host Unreachable - generated by a router if a forwarding path
     (route) to the destination host on a directly connected network
     is not available (does not respond to ARP);

2 = Protocol Unreachable - generated if the transport protocol
     designated in a datagram is not supported in the transport layer
     of the final destination;

3 = Port Unreachable - generated if the designated transport protocol
     (e.g., UDP) is unable to demultiplex the datagram in the
     transport layer of the final destination but has no protocol
     mechanism to inform the sender;

4 = Fragmentation Needed and DF Set - generated if a router needs to
     fragment a datagram but cannot since the DF flag is set;

5 = Source Route Failed - generated if a router cannot forward a
```

```
      packet to the next hop in a source route option;

 6 = Destination Network Unknown - This code SHOULD NOT be generated
      since it would imply on the part of the router that the
      destination network does not exist (net unreachable code 0
      SHOULD be used in place of code 6);

 7 = Destination Host Unknown - generated only when a router can
      determine (from link layer advice) that the destination host
      does not exist;

 11 = Network Unreachable For Type Of Service - generated by a router
      if a forwarding path (route) to the destination network with the
      requested or default TOS is not available;

 12 = Host Unreachable For Type Of Service - generated if a router
      cannot forward a packet because its route(s) to the destination
      do not match either the TOS requested in the datagram or the
      default TOS (0).         "
```

If we now, e.g., try to send a packet to any port which uses a protocol that does not exist, there should actually be a notification about a Destination Unreachable with code value 2 (Protocol Unreachable). Further, with this example you should know first which protocols are "admitted". This, you can find out when you look at your /etc/protocols. After the installation on one of my hosts the /etc/protocols looked like this, e.g.:

```
 -------------- /etc/protocols ----------------
# /etc/protocols:
# $Id: protocols,v 1.2 2001/01/29 17:29:30 notting Exp $
#
# Internet (IP) protocols
#
#       from: @(#)protocols     5.1 (Berkeley) 4/17/89
#
# Updated for NetBSD based on RFC 1340, Assigned Numbers (July 1992).
#
# See also http://www.isi.edu/in-notes/iana/assignments/protocol-numbers

ip      0       IP              # internet protocol, pseudo protocol number
#hopopt 0       HOPOPT          # hop-by-hop options for ipv6
icmp    1       ICMP            # internet control message protocol
igmp    2       IGMP            # internet group management protocol
ggp     3       GGP             # gateway-gateway protocol
ipencap 4       IP-ENCAP        # IP encapsulated in IP (officially ‘‘IP’’)
st      5       ST              # ST datagram mode
tcp     6       TCP             # transmission control protocol
cbt     7       CBT             # CBT, Tony Ballardie
egp     8       EGP             # exterior gateway protocol
igp     9       IGP             # any private interior gateway
                                # (Cisco: for IGRP)
bbn-rcc 10      BBN-RCC-MON     # BBN RCC Monitoring
nvp     11      NVP-II          # Network Voice Protocol
pup     12      PUP             # PARC universal packet protocol
argus   13      ARGUS           # ARGUS
emcon   14      EMCON           # EMCON
xnet    15      XNET            # Cross Net Debugger
chaos   16      CHAOS           # Chaos
udp     17      UDP             # user datagram protocol
mux     18      MUX             # Multiplexing protocol
dcn     19      DCN-MEAS        # DCN Measurement Subsystems
```

```
hmp     20      HMP             # host monitoring protocol
prm     21      PRM             # packet radio measurement protocol
xns-idp 22      XNS-IDP         # Xerox NS IDP
trunk-1 23      TRUNK-1         # Trunk-1
trunk-2 24      TRUNK-2         # Trunk-2
leaf-1  25      LEAF-1          # Leaf-1
leaf-2  26      LEAF-2          # Leaf-2
rdp     27      RDP             # "reliable datagram" protocol
irtp    28      IRTP            # Internet Reliable Transaction Protocol
iso-tp4 29      ISO-TP4         # ISO Transport Protocol Class 4
netblt  30      NETBLT          # Bulk Data Transfer Protocol
mfe-nsp 31      MFE-NSP         # MFE Network Services Protocol
merit-inp       32    MERIT-INP       # MERIT Internodal Protocol
sep     33      SEP             # Sequential Exchange Protocol
3pc     34      3PC             # Third Party Connect Protocol
idpr    35      IDPR            # Inter-Domain Policy Routing Protocol
xtp     36      XTP             # Xpress Tranfer Protocol
ddp     37      DDP             # Datagram Delivery Protocol
idpr-cmtp       38    IDPR-CMTP       # IDPR Control Message Transport Proto
tp++    39      TP++            # TP++ Transport Protocol
il      40      IL              # IL Transport Protocol
ipv6    41      IPv6            # IPv6
sdrp    42      SDRP            # Source Demand Routing Protocol
ipv6-route      43    IPv6-Route      # Routing Header for IPv6
ipv6-frag       44    IPv6-Frag       # Fragment Header for IPv6
idrp    45      IDRP            # Inter-Domain Routing Protocol
rsvp    46      RSVP            # Resource ReSerVation Protocol
gre     47      GRE             # Generic Routing Encapsulation
mhrp    48      MHRP            # Mobile Host Routing Protocol
bna     49      BNA             # BNA
ipv6-crypt      50    IPv6-Crypt      # Encryption Header for IPv6
ipv6-auth       51    IPv6-Auth       # Authentication Header for IPv6
i-nlsp  52      I-NLSP          # Integrated Net Layer Security TUBA
swipe   53      SWIPE           # IP with Encryption
narp    54      NARP            # NBMA Address Resolution Protocol
mobile  55      MOBILE          # IP Mobility
tlsp    56      TLSP            # Transport Layer Security Protocol
skip    57      SKIP            # SKIP
ipv6-icmp       58    IPv6-ICMP       # ICMP for IPv6
ipv6-nonxt      59    IPv6-NoNxt      # No Next Header for IPv6
ipv6-opts       60    IPv6-Opts       # Destination Options for IPv6
#       61                      # any host internal protocol
cftp    62      CFTP            # CFTP
#       63                      # any local network
sat-expak       64      SAT-EXPAK       # SATNET and Backroom EXPAK
kryptolan       65      KRYPTOLAN       # Kryptolan
rvd     66      RVD             # MIT Remote Virtual Disk Protocol
ippc    67      IPPC            # Internet Pluribus Packet Core
#       68                      # any distributed file system
sat-mon 69      SAT-MON         # SATNET Monitoring
visa    70      VISA            # VISA Protocol
ipcv    71      IPCV            # Internet Packet Core Utility
cpnx    72      CPNX            # Computer Protocol Network Executive
cphb    73      CPHB            # Computer Protocol Heart Beat
wsn     74      WSN             # Wang Span Network
pvp     75      PVP             # Packet Video Protocol
br-sat-mon      76      BR-SAT-MON      # Backroom SATNET Monitoring
sun-nd  77      SUN-ND          # SUN ND PROTOCOL-Temporary
wb-mon  78      WB-MON          # WIDEBAND Monitoring
wb-expak        79      WB-EXPAK        # WIDEBAND EXPAK
iso-ip  80      ISO-IP          # ISO Internet Protocol
```

```
vmtp    81      VMTP            # Versatile Message Transport
secure-vmtp     82      SECURE-VMTP     # SECURE-VMTP
vines   83      VINES           # VINES
ttp     84      TTP             # TTP
nsfnet-igp      85      NSFNET-IGP      # NSFNET-IGP
dgp     86      DGP             # Dissimilar Gateway Protocol
tcf     87      TCF             # TCF
eigrp   88      EIGRP           # Enhanced Interi
```

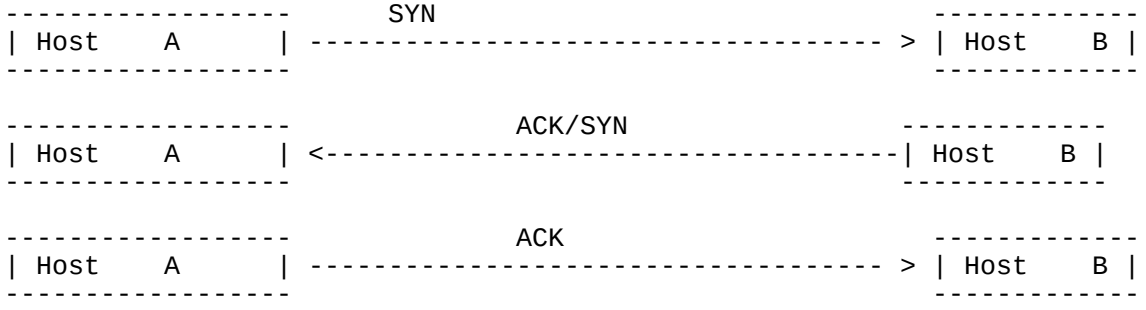
What if the host does not send back a Protocol Unreachable (though you used a protocol that actually does not exist)? This can have two reasons. First, it could be that you have found an AIX, HP-UX or Digital Unix machine or the set of rules of the host does not allow access to these ports. So, at first, verify what kind of host you scan (among other possibilities with fingerprint OS detection) or else you can assume that it gets filtered/blocked.

Note: The detection of such an attack is quite simple (and belongs to the section of Protocol Anomaly Detection). Anomaly Detection will be discussed in the next chapter (Possibilities of analysis), now, it is sufficient to know that traffic is searched for "anormalities" and the use of a non-existant protocol belongs to these "anormalities".

- Denial of Service (DoS)
DoS (Denial of Service) attacks normally aim to paralyze the target server so that it is not reachable for some time. There are so many techniques by wich you can "exhaust" the resources of the target host. In the following I present you the most common:

ICMP Flooding:
ICMP Flooding "uses" ICMP. If a host receives an ICMP echo request it will normally try to answer with an ICMP echo reply. This behaviour is used with ICMP Flooding: If you send the victim too many echo requests its resources will be working on it to reply to these requests (as echo replies). As the IP of the attacker is mostly additionally spoofed it does not get the replies but someone else.

SYN Flooding:
To understand SYN Flooding, I present again the "normal" three-way-handshake:



Host A sends Host B a SYN to say that he wants a connection, B answers with ACK/SYN and waits for the final ACK with which the connection would complete. But what if the last ACK is not sent? If B sends back its SYN/ACK it waits (as said before) for the ACK of A, until then it will be queued in the connectin queue of "B". If the connection is complete (A sent B an ACK) it will be removed from the connection queue. But as mostly the IP address is spoofed B never receives an ACK (it should be so). So, you can "fill" the connection queue as the host cannot make additonal connections.

UDP Flooding:
With this flooding attack the target server is flooded with UDP packets. If you send a UDP packet to a port on the target server it will first check which service is responsible for this "request". Now, you choose random ports to which you send the packets to rise the probability that a "Port Unreachable" is sent by ICMP. As the result of such a flood the performance of a network segment suffers (often) considerably.

Land:
Later you will see a filter which detects if there is a Land attack and can initiate counteractive measures, but first the actual attack. A Land attack has similar source and destination IPs. When sending, e.g., SYN packets (with same source/destination IP) to an open port there will start a race condition on the victims host which leads to paralysis of the whole system. Here, a trace of a Land attack:

```
23/06/02  23:12:48  194.157.1.1  80  -> 194.157.1.1
23/06/02  23:14:57  194.157.1.1  31337  ->     194.157.1.1
```

As you can see again, source and destination IP are the same.

```
12:35:26.916369 192.168.38.110.135>192.168.38.110.135: udp 46[tos0x3,ECT,CE]
        4503 004a 96ac 0000 4011 15c7 c0a8 266e
       c0a8 266e 0087 0087 0036 8433 6920 616d
       206c 616d 6520 646f 7320 6b69 6420 6275
       7420 6920 7265
12:35:26.916566 192.168.38.110.135>192.168.38.110.135: udp 46[tos0x3,ECT,CE]
       4503 004a 2923 0000 4011 8350 c0a8 266e
       c0a8 266e 0087 0087 0036 8433 6920 616d
       206c 616d 6520 646f 7320 6b69 6420 6275
       7420 6920 7265
12:35:26.916682 192.168.38.110.135>192.168.38.110.135:udp 46[tos0x3,ECT,CE]
       4503 004a 50a0 0000 4011 5bd3 c0a8 266e
       c0a8 266e 0087 0087 0036 8433 6920 616d
       206c 616d 6520 646f 7320 6b69 6420 6275
       7420 6920 7265
```

The attack which you can see here is also called Snork.

Teardrop:
Here, the possibility of fragmentation of IP packets is used. As you see in the description of the scans there is a fragmentation if datagrams are bigger than the size limit, this size limitation is called MTU (Maximum Transmission Unit). With this attack, fragments overlap and as a result of this overlap many OSs have (had) problems and mostly crashed the system.

```
10:13:32.104203 10.10.10.10.53>192.168.1.3.53: udp 28(frag 242:36@0+)(ttl64)
       4500 0038 00f2 2000 4011 8404 0a0a 0a0a
       c0a8 0103 0035 0035 0024 0000 0000 0000
       0000 0000 0000 0000 0000 0000 0000 0000
        0000 0000 0000

10:13:32.104272 10.10.10.10 >192.168.1.3: udp 28(frag 242:4@24)(ttl 64)
       4500 0018 00f2 0003 4011 a421 0a0a 0a0a
       c0a8 0103 0035 0035 0024 0000 0000 0000
       0000 0000 0000 0000 0000 0000 0000
```

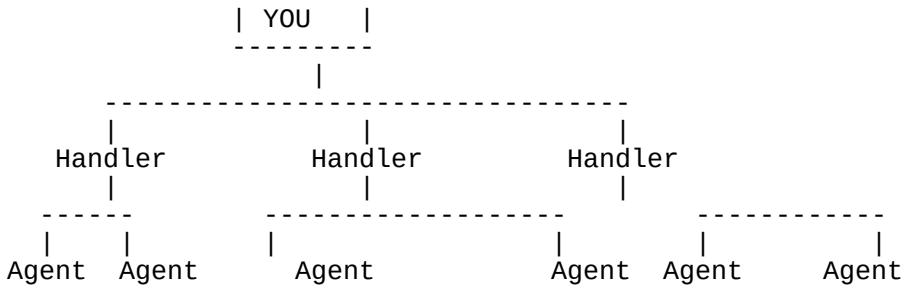
Ping of Death:
Here, too, fragmentation of the IP packets again plays a role. Here, I will go into more detail (only a little ;) on fragmentation. As told before, fragmentation means that the size of datagrams is "reduced", whereas single fragments are not to be bigger than the MTU. When fragmenting you should actually consider that you cannot fragment as you want, respectively that certain fields have to exist in every fragment. So, every fragment has to contain, e.g., the IP protocol header to choose the right route. That the router can rebuild fragments to a datagram every fragment receives a 16 bit flag (of the original, "big" datagram"). With this 16 bit flag it is later possible to sort the single fragments to the right datagram. Additionally, there is a fragment offset which tells at which position the fragment was in the original datagram. But the position is in 8 octet units (as the position is measured in 8 octet units). Additionally, the "more-bit" shows if further fragments of this datagram follow or not. If it is set to 1 further will follow if set to 0 it was the last fragment of the datagram. Now, we get to the actual ping-of-death attack. Ping-of-death is given an offset of the last fragment for which is: offset + fragment size > 65535 bytes. Thus, it is possible to flood internal 16 bit variables which would result, e.g., in a system crash.

```
17:43:58.431 pinger > target: icmp echo request (frag 4321: 380@0+)
17:43:58.431 pinger > target: (frag 4321: 380@2656+)
17:43:58.431 pinger > target: (frag 4321: 380@3040+)
17:43:58.431 pinger > target: (frag 4321: 380@3416+)
```

Smurf:
A ping is sent to the broadcast address, better said, many pings are sent. Packets sent to the broadcast address are sent to all hosts of the network. If you send many pings (ICMP echo requests) to the broadcast address (e.g., 10000 per second) and you have a relatively large network with 1000 hosts, it means that there are 10000 * 1000 ICMP echo replies per second at the victim’s host, that means 10000000 ICMP echo replies.

```
09:28:28.666073 179.135.168.43>256.256.30.255: icmp: echo request (DF)
       4500 001c c014 4000 1e01 6172 b387 a82b
       c0a8 1eff 0800 f7ff 0000 0000 0000 0000
       0000 0000 0000 0000 0000 0000 0000
09:28:28.696073 68.90.226.250>256.256.30.255: icmp: echo request (DF)
       4500 001c c015 4000 1e01 95cf 445a e2fa
       c0a8 1eff 0800 f7ff 0000 0000 3136 3803
       3133 3503 3137 3907 696e 2d61 6464
09:28:28.726073 138.98.10.247>256.256.30.255:  icmp: echo request (DF)
       4500 001c c016 4000 1e01 27ca 8a62 0af7
       c0a8 1eff 0800 f7ff 0000 0000 0332 3236
       3938 0331 3638 0769 6e2d 6164 6472
09:28:28.756073 130.113.202.100 > 256.256.30.255: icmp: echo request (DF)
       4500 001c c017 4000 1e01 704c 8271ca64
       c0a8 1eff 0800 f7ff 0000 0000 0231 3002
       3938 0331 3338 0769 6e2d 6164 6472
       ...
```

For some time, there also existed DDoS attacks (Distributed Denial of Service). As the name suggests it is a distributed/networked DoS attack. The attacker (client) looks for other hosts/networks... which are easy to exloit. These first infected hosts are the so called handlers. Handlers infect further hosts/networks, these infected hosts are then called agents, i.e. as a datagram:



Later, agents execute attacks.

This is the first article out of 2. We will continue in the next issue of LinuxFocus.

Translation information:
de --> -- : Klaus Müller <Socma(at)gmx.net>
de --> en: Hubert Kaißer <hubert(Q)faveve.uni-stuttgart.de>

2005-01-14, generated by lfparser_pdf version 2.51