LinuxFocus article number 282
http://linuxfocus.org

External attacks

by Eric Detoisien
<valgasu(at)club-internet.fr>

About the author:

Eric Detoisien is an expert in computer security. Very fond of everything related to security, he is one of the experts of the rstack group - www.rstack.org -

Translated to English by:
Georges Tarbouriech
<gt(at)linuxfocus.org>

Abstract:

This article was first published in a Linux Magazine France special issue focusing on security. The editor, the authors and the translators kindly allowed LinuxFocus to publish every article from this special issue. Accordingly, LinuxFocus will bring them to you as soon as they are translated to English. Thanks to all the people involved in this work. This abstract will be reproduced for each article having the same origin.

This article presents the different types of external attacks that a cracker can use against the machines within a network. We will take up the main network attack, some attacks through applications and denial of service type attacks.

Network attacks

The network attacks rely on vulnerabilities directly related to protocols or to their implementation. There are many of them. However, most of these are variants of the five well known networks attacks.

Fragment attacks

This attack goes beyond the protection of IP filtering equipment. To implement it, the crackers use two different methods: Tiny Fragments and Fragment Overlapping. These attacks are somewhat historical,

accordingly today's firewalls have been managing them for a long time.

Tiny Fragments

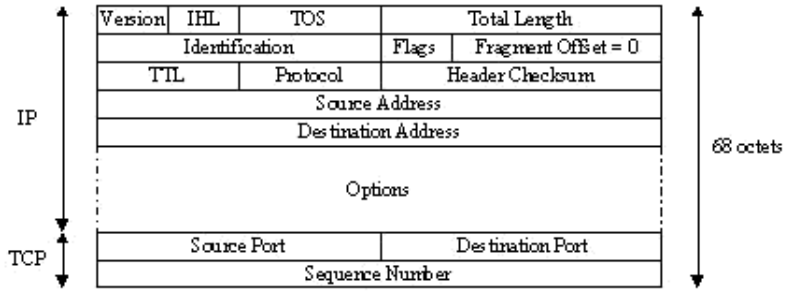
According to RFC (*Request For Comment*) 791 (IP), all Internet nodes (routers) must be able to transfer 68 bytes packets without fragmenting them. The minimum size of the header of an IP packet is 20 bytes without options. When options are present, the maximum size of the header is 60 bytes. The IHL (*Internet Header Length*) field holds the header length in 32 bit words. This field uses 4 bit, so the number of possible values is 2^4 - 1 = 15 (it cannot take the value 0000). So, the maximum size of the header is really 15*4 = 60 bytes. Last, the *Fragment Offset* field indicating the offset of the first byte of the fragment in relation to the full datagram is written in 8 bytes blocks. Thus a data fragment at least holds 8 bytes. This really makes 68 bytes.

The attack consists of requesting a TCP connection fragmented into two IP packets. The first IP packet of 68 bytes only holds the 8 first bytes of the TCP header (source and destination ports and sequence number). The data in the second IP packet then holds the TCP connection request (SYN flag is 1 and ACK flag is 0).

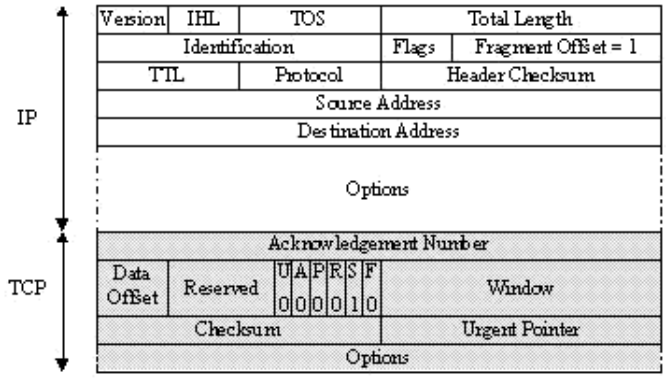
However, IP filters apply the same rule to all the fragments in a packet. The filter of the first fragment (Fragment Offset = 0) defines the rule, accordingly it applies to the other fragments (Fragment Offset = 1) without any other type of control. So, when defragmenting at IP level on the target machine, the connection request packet is rebuilt and passed to the TCP layer. The connection is established despite the IP filter in between which should have prevented it.

Pictures 1 and 2 show both fragments and picture 3 shows the defragmented packet on the target machine:

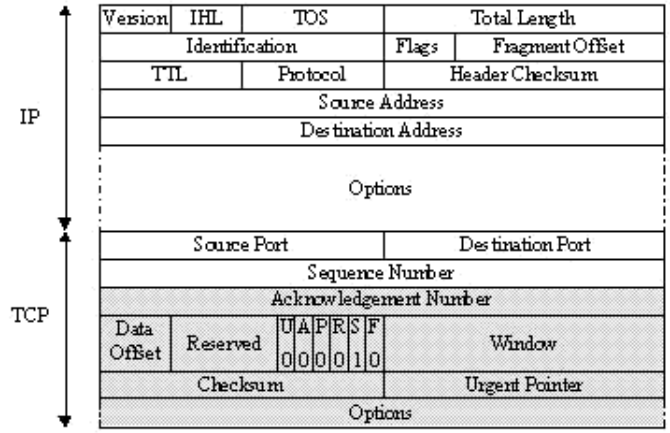
Pic.1: Fragment 1



Pic.2: Fragment 2



Pic.3: Defragmented packet

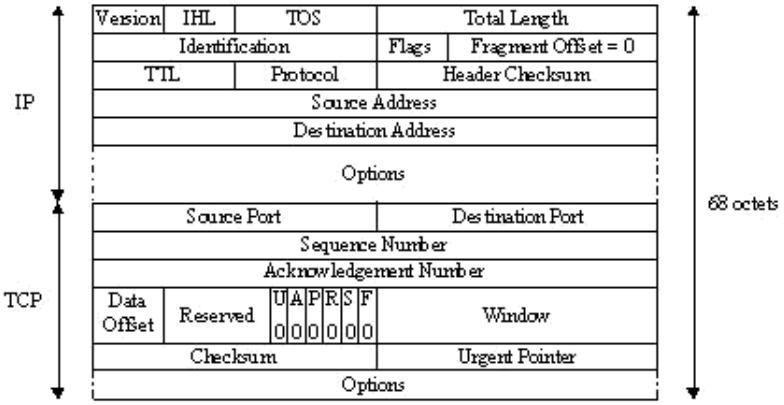


Fragment Overlapping

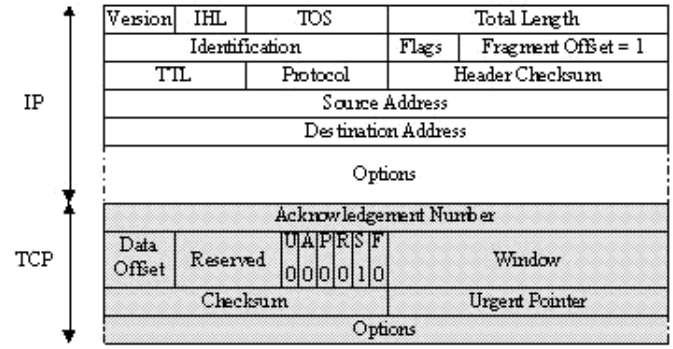
Still according to RFC 791 (IP), if two IP fragments overlap, the second one overwrites the first one. The attack consists of dividing an IP packet into two fragments. The IP filter accepts the first one holding 68 bytes (see Tiny Fragments) since it does not request a TCP connection (SYN flag = 0 and ACK flag = 0). Again, this rule applies to the other fragments of the packet. The second one (with a Fragment Offset = 1) holding the real connection data is then accepted by the IP filter because it does not see that a connection is opened here. Thus, when defragmenting, the data of the second fragment overwrites the data in the first one, starting after the 8th byte (since the fragment offset = 1). The reassembled packet is then a valid connection request for the target machine. The connection is established depite the IP filter in between.

Pictures 1 and 2 show both fragments and picture 3 shows the defragmented packet on the target machine:

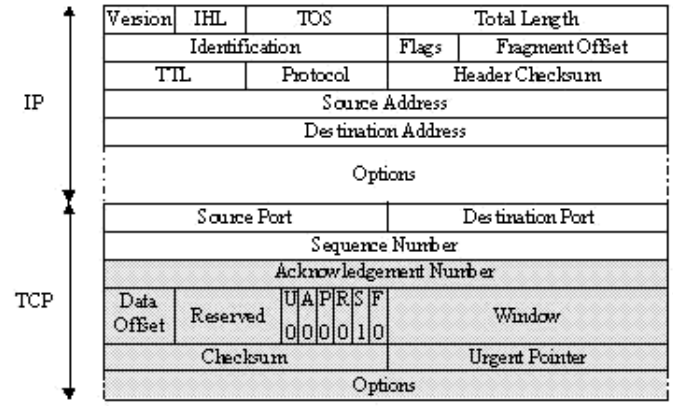
Pic.4: Fragment 1



Pic.5: Fragment 2



Pic.6: Defragmented packet



IP Spoofing

The goal of this attack is to usurp the IP address of a machine. This allows the cracker either to hide the origin of his attack (used in Denial of Service attacks) or to benefit from a trusted relationship between two machines. Here, we will explain this second use of IP Spoofing.

The basic principle of this attack consists, for the cracker, in forging his own IP packets (with programs

such as `hping2` or `nemesis`) in which he will change, among other things, the source IP address. IP Spoofing is often called Blind Spoofing. Since the answers to the false packets cannot go to the cracker's machine since the source has been altered. So, they go to the spoofed machine. However, there are two methods to get the answers back:

1. *Source Routing*: the IP protocol has an option called Source Routing which allows you to define the route the IP packets should take. This route is a series of router IP addresses that the packets will have to follow. Enough for the cracker to provide a route for the packets to a router he controls. Nowadays, most of the TCP/IP stack implementations reject the packets using this option;
2. *Re-routing*: router tables using the RIP protocol can be changed sending them RIP packets with new routing information. This is done to reroute the packets to a router that the cracker manages.

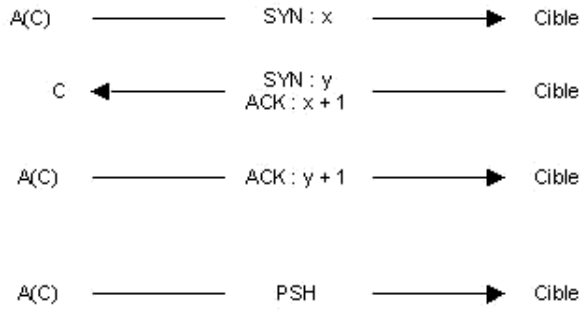
These techniques are hardly usable: the attack is carried out without knowing the packets coming from the target server.

Blind Spoofing is used against services such as rlogin or rsh. Their authentication mechanism only relies on the source IP address of the client machine. This relatively well known attack (Kevin Mitnick used it against Tsutomu Shimomura's machine in 1994) requires various steps:

- finding the IP address of the trusted machine using, for instance `showmount -e` which indicates where the file systems are exported, or `rpcinfo` which provides more information;
- taking the trusted host out of service using a SYN Flooding, for instance (more on Denial of Service later on). This is compulsory to prevent the machine from answering the packets sent by the target server. Otherwise, it would send TCP RST packets which would stop the connection attempt;
- prediction of TCP sequence numbers: every TCP packet is associated to an initial sequence number. The OS TCP/IP stack generates it in a linear way, depending on the time, random or pseudo-random, according to the systems. The cracker only can attack systems generating predictable sequence numbers (linear generation or dependent on time);
- the attack consists of opening a TCP connection on the desired port (rsh for example). For better understanding, we will remind you how the opening mechanism of a TCP connection works. It is done in three steps (TCP Three Way Handshake):
    1. the caller sends a packet holding the TCP SYN flag and a *x* sequence number is sent to the target machine;
    2. the target answers with a packet in which the TCP SYN flag and the ACK flag (with a *x+1* acknowledgement number) are activated. Its sequence number is *y*;
    3. the caller sends a packet containing the TCP ACK flag (with a *y+1* acknowledgement number) back to the target machine.

During the attack, the cracker does not receive the SYN-ACK sent by the target. For the connection to establish, he predicts the *y* sequence number in order to send a packet with the right ACK number (*y+1*). The connection is then established through the IP address authentication. The cracker can now send a command to the rsh service, such as`echo ++ >> /.rhosts` to get higher access rights. To do this, he forges a packet with the TCP PSH flag (*Push*): the received data is immediately sent to the upper layer (here the rsh service). He can then connect to the machine through a service such as rlogin or rsh without IP Spoofing.

Picture 7 shows the different steps of IP Spoofing:
Pic.7: IP Spoofing applied to the rsh service



The cracker uses the A machine while C represents the trusted machine. The A(C)statement means that the packet is sent from A with the C spoofed IP address. Note: there is a program called `mendax` which implements those IP Spoofing mechanisms.

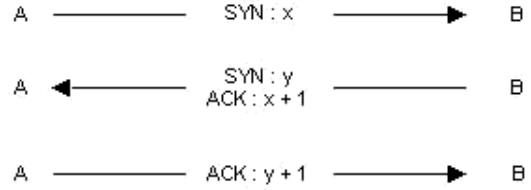
TCP Session Hijacking

TCP Session Hijacking allows the cracker to redirect a TCP flow. Then, a cracker can overstep a password protection (like in telnet or ftp). The need for listening (sniffing) restricts this attack to the physical network of the target. Before detailing this attack, let us explain some basic principles of the TCP protocol.

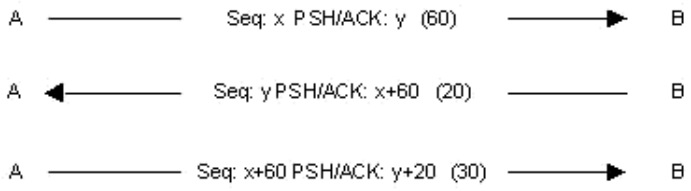
Here we will not unveil the mystery of the TCP protocol, but we will concentrate on the main points required to understand the attack. The TCP header holds various fields:

- the source port and the destination port, identifying the connection between two machines;
- the sequence number identifying every byte sent;
- the acknowledgement number corresponding to the last byte received;
- the interesting flags are:
  - SYN which synchronizes the sequence numbers when a connection is established;
  - ACK, the acknowledgement flag of a TCP segment;
  - PSH which tells the receiver to send the data to the application.

Picture 8 shows how to establish a TCP connection (Three Way Handshake):
Pic.8: Three Way Handshake



Here, the A machine initiated a TCP connection on the B machine.
Picture 9 shows a transfer of TCP data:



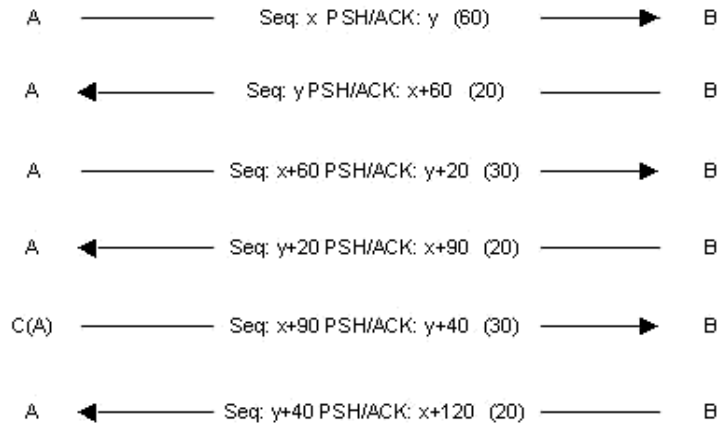
The sequence numbers will change according to the number of data bytes sent. The sequence number is represented by *Seq*, the acknowledgement number is found after the PSH and ACK flags and the number of data bytes sent is found between brackets.

This attack creates a desynchronization state on both side of the TCP connection, allowing the session hijack. A connection is desynchronized when the sequence number of the next byte sent by the A machine is different from the sequence number of the next byte to be received by B. And the other way round too.

In the example of picture 9, at the end of the first step when B receives its packet, A waits for a packet with an acknowledgement number of *x+60*. If the next packet sent by B does not have this acknowledgement number, then A and B are considered as desynchronized.

So, a cracker with a C machine wants to hijack an established Telnet session between A and B machines. First, the C machine sniffs the Telnet traffic (TCP port 23) between A and B. Once the cracker thinks A had time to authenticate to the Telnet service on the B machine, he desynchronizes the A machine against B. To do this, he forges a packet having the source IP address of the A machine and the TCP acknowledgement number expected by B. Of course, the B machine accepts the packet. Besides desynchronizing the TCP connection, this packet allows the cracker to inject a command through the Telnet session previously established by A. As a matter of fact, this packet is able to carry data (PSH flag = 1).

Picture 10 shows this attack:
Pic.10: TCP Session Hijacking



The B machine accepts the command sent by C, it acknowledges this packet sending a packet to A with the ACK flag. In the meantime, if A sent a packet to B, it has been rejected since the sequence number is not the one expected by B.

A problem then appears: the Ack Storm. A lot of ACK are generated. This happens when A sends a TCP packet with an invalid sequence number (since A is desynchronized), B rejects it and sends to A an ACK with the sequence number it expects. A receives this ACK, and since the sequence number does not match the expected one, it also sends an ACK to B and B does it again...

This Ack Storm problem can be solved if the cracker uses the ARP Spoofing. In that case, the C machine will poison the ARP cache of the B machine telling it the A IP address is now associated to the C MAC address. These techniques are implemented by the `hunt` program.

ARP Spoofing

This attack, also called ARP Redirect, redirects the network traffic from one or more machines to the cracker's machine. It is done on the physical network of the victims. Let us remind you what the ARP protocol is and how it works.

The ARP protocol (*Address Resolution Protocol*) implements the resolution mechanism from an IP address to an Ethernet MAC address. Network equipment communicates by exchanging Ethernet frames (obviously in an Ethernet network), at the data link layer. To be able to share this information, it is required for the network cards to have a unique Ethernet address: it is the MAC address (*MAC=Media Access Control*).

When sending an IP packet, the sender machine needs to know the MAC address of the receiver. To get it, a broadcast ARP request is sent to every machine in the local network. This request asks: "What is the MAC address associated to this IP address ?". The machine having this IP address answers through an ARP packet, providing the sender machine with the requested MAC address. From there, the source machine knows the MAC address corresponding to the IP address where to send the packets. This match will be kept for a while in a cache (to avoid making a new request each time an IP packet is sent).

This attack poisons the cache of the target machine. The cracker sends ARP answers to the target machine telling it that the new MAC address is one corresponding to a gateway (for instance) IP address is the crackers address. The cracker's machine will then receive the whole traffic sent to the gateway. So, enough for him to listen to the traffic (and/or to modify it). After that, he will route the packets to the real destination so that nobody notices the change.

ARP Spoofing is useful when a local network uses switches. These redirect the Ethernet frames to different ports (cables) according to the MAC address. Then a sniffer can hardly capture frames beyond its own physical wire. Thus, ARP Spoofing allows to listen to the traffic between machines situated on different switch ports.

To implement an ARP Spoofing attack, the cracker will use an ARP packet generator such as `ARPSpoof` or `nemesis`. Example: the "victim" machine `10.0.0.171`, its default gateway `10.0.0.1` and the cracker's machine `10.0.0.227`. Before the attack, the result of a traceroute is:

```
[root@cible -> ~]$ traceroute 10.0.0.1
traceroute to 10.0.0.1 (10.0.0.1), 30 hops max, 40 byte packets
 1  10.0.0.1  (10.0.0.1)  1.218 ms  1.061 ms  0.849 ms
```

And the ARP cache of the target machine is:

```
[root@cible -> ~]$ arp
Address      HWtype  HWAddress          Flags Mask   Iface
10.0.0.1     ether   00:b0:c2:88:de:65   C           eth0
10.0.0.227   ether   00:00:86:35:c9:3f   C           eth0
```

The cracker then runs `ARPSpoof`:

```
[root@pirate -> ~]$ arpspoof -t 10.0.0.171 10.0.0.1
0:0:86:35:c9:3f 0:60:8:de:64:f0 0806 42: arp reply 10.0.0.1 is-at 0:0:86:35:c9:3f
0:0:86:35:c9:3f 0:60:8:de:64:f0 0806 42: arp reply 10.0.0.1 is-at 0:0:86:35:c9:3f
0:0:86:35:c9:3f 0:60:8:de:64:f0 0806 42: arp reply 10.0.0.1 is-at 0:0:86:35:c9:3f
0:0:86:35:c9:3f 0:60:8:de:64:f0 0806 42: arp reply 10.0.0.1 is-at 0:0:86:35:c9:3f
0:0:86:35:c9:3f 0:60:8:de:64:f0 0806 42: arp reply 10.0.0.1 is-at 0:0:86:35:c9:3f
0:0:86:35:c9:3f 0:60:8:de:64:f0 0806 42: arp reply 10.0.0.1 is-at 0:0:86:35:c9:3f
0:0:86:35:c9:3f 0:60:8:de:64:f0 0806 42: arp reply 10.0.0.1 is-at 0:0:86:35:c9:3f
```

The packets sent are ARP packets poisoning the ARP cache of the `10.0.0.171` machine, with ARP Reply saying that the MAC address associated to `10.0.0.1` now is `00:00:86:35:c9:3f`.

The ARP cache of the `10.0.0.171` machine becomes :

```
[root@cible -> ~]$ arp
Address      HWtype  HWAddress          Flags Mask   Iface
10.0.0.1     ether   00:00:86:35:c9:3f   C           eth0
10.0.0.227   ether   00:00:86:35:c9:3f   C           eth0
```

To check that the traffic now goes through the `10.0.0.227` machine, enough to run a new traceroute to the `10.0.0.1` gateway:

```
[root@cible -> ~]$ traceroute 10.0.0.1
traceroute to 10.0.0.1 (10.0.0.1), 30 hops max, 40 byte packets
 1  10.0.0.227  (10.0.0.227)  1.712 ms  1.465 ms  1.501 ms
 2  10.0.0.1  (10.0.0.1)  2.238 ms  2.121 ms  2.169 ms
```

Now the cracker can sniff the traffic between the `10.0.0.171` and `10.0.0.1` machines. He must not forget to activate IP routing on his `10.0.0.227` machine.

DNS Spoofing

The DNS protocol (*Domain Name System*) converts a domain name (for example www.test.com) into its IP address (for example `192.168.0.1`) and vice versa. This attack uses false answers to the DNS

requests sent by a "victim". This attack relies on two main methods.

DNS ID Spoofing

The header of the DNS protocol holds an identification field to match answers and requests. The goal of the DNS ID Spoofing is to send back a wrong answer to a DNS request before the real DNS server can answer. To do this, the ID of the request has to be predicted. Locally, it is simple to predict just by sniffing the network. However, it becomes a bit more tricky remotely. There are various methods:

- testing every possibility in the ID field. Not very realistic since there are 65535 possibilities (this field is 16 bit);
- sending a few hundred DNS requests in the right order. Obviously, this method is not quite reliable;
- finding a server generating predictable IDs (for example, ID incremented by 1). This sort of vulnerability can be found in some Bind versions or on Windows 9x machines.

In any case, it is required to answer before the real DNS server. This can be done for instance by crashing it with a denial of service attack.

To succeed, the attacker must control a DNS server (`ns.attaquant.com`) having authority over the domain `attaquant.com`. The target DNS server (`ns.cible.com`) is supposed to have predictable sequence numbers (incrementing by 1 at each request).

The attack requires four steps:

1. the attacker sends a DNS request for the name `www.attaquant.com` to the DNS server of the `cible.com` domain, as seen in picture 11;

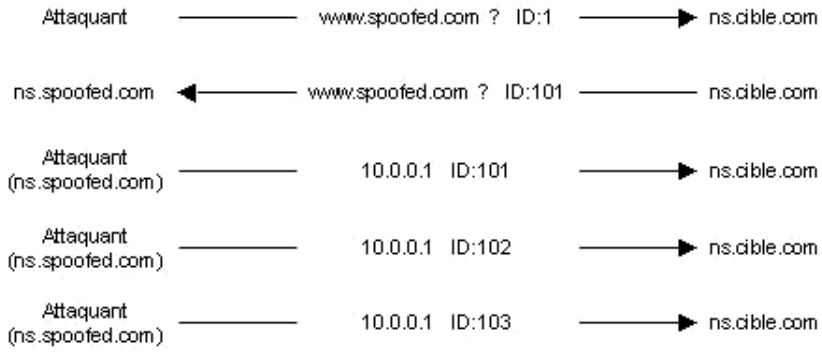
   Pic.11: DNS request sent to `ns.cible.com`

2. the target DNS server relayed the request to the DNS of `attaquant.com` domain;
3. the attacker is able to sniff the request to get its ID (in our example the ID is 100);
4. the attack alters the IP address associated to a machine name, here the "victim" machine is `www.spoofed.com` which should have `192.168.0.1` as its IP address. The cracker sends a DNS request for resolving the name `www.spoofed.com` to `ns.cible.com`. Immediately afterwards, he sends a bunch of altered DNS answers (giving as IP address the one of the attacker's site `10.0.0.1`) to this same request having spoofed the source IP address with the one of the DNS server of the `spoofed.com` domain. The ID of each answer will be incremented by 1 starting from the one received during the second step (ID = 100) to improve the chance of getting the right ID

number. This is, just in case `ns.cible.com` should have answered other requests and thus incremented its DNS ID. Picture 12 shows these steps.

Pic.12: DNS ID Spoofing



The cache of the target DNS server is then poisoned and the next machine asking for a resolution of the `www.spoofed.com` name will get the IP address of the attacker's machine and will be redirected to his site. This last can be a "copy" of the real site to fool the internet users and steal confidential information.

DNS Cache Poisoning

DNS servers use a cache, to locally keep the answers to previous requests data for a while. This is to avoid spending time in always asking the name server having authority over the requested domain. This second type of DNS Spoofing will consist of poisoning this cache with false information. Here is an example:

We keep the parameters of the previous example. Here are the different steps of the attack:

- send a DNS request to resolve the `www.attaquant.com` name to the DNS server of the `cible.com` domain;
- the target DNS server sends a request to resolve the `www.attaquant.com` name to the attacker's DNS server;
- the attacker's DNS server sends an answer with altered records allowing it to assign a machine name to an IP address belonging to the attacker. For example, the `www.cible.com` site could have an altered DNS record sending back the IP address of `www.attaquant.com` instead of the right one.

Applications attacks

Applications attacks rely on specific vulnerabilities found in some applications. However, some of them can be classified by type.

The configuration problem

One of the first security problems found in applications comes from configuration mistakes. There are two types of mistakes: default installation and wrong configuration.

Software, such as web servers, with default install files often provides example sites that can be used by crackers to access confidential information. For example, they can provide scripts to get the source data via faulty dynamic pages. Furthermore, such an installation may provide a remote administration interface with a default login/password (found in the application administration guide). The cracker is then able to change what he wants on the site.

The main vulnerabilities generated by a bad configuration are access lists with wrong parameters. So the cracker can access private pages or private databases.

As a classic example of misconfiguration, the mistakes in Lotus Domino web server parameters are frequent. When installing this server, Lotus configuration databases do not have any access control list. Clearly, if the *names.nsf* Lotus database can be accessed with a web browser without authentication, it is possible to get a lot of information such as every Lotus username.

Bugs

Bad software programming always leads to bugs. These are the most important vulnerabilities. When they are discovered, they allow to execute unauthorized commands, to get the source code of dynamic pages, to make a service unusable, to take control of the machine, etc. The most known of these bugs and the most interesting in term of exploit is the buffer overflow.

Buffer overflow

The buffer overflow is a vulnerability caused by bad programming. It appears when a variable passed as an argument to a function is copied into a buffer without checking its size. If the variable has a bigger size than the memory space reserved for this buffer, it is enough for the buffer overflow to happen. It will be exploited passing to the variable a program fragment. If a cracker succeeds in this attack he will get the ability to remotely execute commands on the target machine with the rights of the attacked application. More on this in the article series about secure programming:

- Avoiding security holes when developing an application - Part 1
- Avoiding security holes when developing an application - Part 2: memory, stack and functions, shellcode
- Avoiding security holes when developing an application - Part 3: buffer overflows
- Avoiding security holes when developing an application - Part 4: format strings

- Avoiding security holes when developing an application - Part 5: race conditions
- Avoiding security holes when developing an application - Part 6: CGI scripts

Scripts

Bad script programming often affects the security of a system. There are means of exploiting vulnerabilities found in Perl scripts which will allow to read files out of the web root or to execute unauthorized commands. These programming problems are presented in CGI security articles above (the Part 6).

Man in the Middle

The main goal of this attack is to divert the traffic between two machines. This is to intercept, modify or destroy the data circulating during the communication. This attack is more a concept than a real attack. There are various attacks implementing the Man in the middle principle, such as the DNS Man in the Middle which uses DNS Spoofing to divert the traffic between a web server and a web client. A recent application has been created to divert SSH traffic.

Denial of service

This attack is well named since it will lead to the unavailability of a service (specific application) or of a target machine. We will distinguish two types of denial of service: on the one hand, the ones exploiting an application bug and on the other hand the ones related to the bad implementation of a protocol or to the weaknesses of a protocol.

Application denial of service

If the vulnerabilities of an application can lead to the ability of taking the control over a machine (buffer overflow example), they can also lead to a denial of service. The application will become unavailable either by lack of allocated resources or by a crash.

Network denial of service

There are different types of denial of service using the protocols features of the TCP/IP stack.

SYN Flooding

We already saw that a TCP connection is established in three stages (TCP Three Way Handshake). SYN Flooding exploits this mechanism. The three stages are sending a SYN, receiving a SYN-ACK and sending an ACK. The idea is to leave on the target machine a big number of TCP connections waiting. To do this, the cracker sends a lot of connection requests (SYN flag = 1), the target machine sends the SYN-ACK back to answer the received SYN. The cracker will not answer with an ACK, thus for each received SYN, the target machine will have a pending TCP connection. Since these half-open connections use memory resources, after a while, the machine is saturated and cannot accept any other connection. This type of denial of service only affects the target machine.

The cracker uses a SYN Flooder such as synk4, indicating the target TCP port and using random source IP addresses to prevent the attacker machine from being identified.

UDP Flooding

This denial of service exploits the unconnected mode of the UDP protocol. It creates an UDP Packet Storm (bunch of UDP packets) either to a single machine or between two machines. Such an attack between two machines leads to a network congestion and to a resource saturation on both hosts. The congestion is more important since UDP traffic has priority over TCP traffic. The TCP protocol has a mechanism to control congestion, in case the acknowledgement of a packet arrives after a long time: this mechanism adapts the frequency in sending TCP packets, then the rate decreases. The UDP protocol does not have this mechanism: after a while the UDP traffic uses the whole bandwith, leaving a very small part of it to the TCP traffic.

The most known example of UDP Flooding is the *Chargen Denial of Service Attack*. The implementation of this attack is simple: It is enough to establish a communication between the `chargen` service of a machine and the `echo` service of another one. The `chargen` service generates characters while `echo` resends the data it receives. The cracker then sends UDP packets to the port 19 (`chargen`) to one of the "victims" spoofing the IP address and the source port from the other one. In that case, the source port is the UDP port 7 (`echo`). The UDP Flooding leads to a bandwith saturation between both machines. A whole network can then be the "victim" of an UDP Flooding.

Packet Fragment

The denial of service of Packet Fragment type uses weaknesses of some TCP/IP stacks concerning IP defragmentation (reassembling IP fragments).

A known attack using this, is Teardrop. The fragmentation offset of the second segment is smaller than the size of the first one and so for the offset added to the size of the second one. This means that the first fragment contains the second one (overlapping). At defragmentation time, a few systems do not manage this exception and this leads to a denial of service. There are variants of this attack: bonk, boink and

newtear for example. The Ping of Death denial of service exploits a bad management of ICMP defragmentation, sending more data than the maximum size of an IP packet. These different types of denial of service lead to a crash of the target machine.

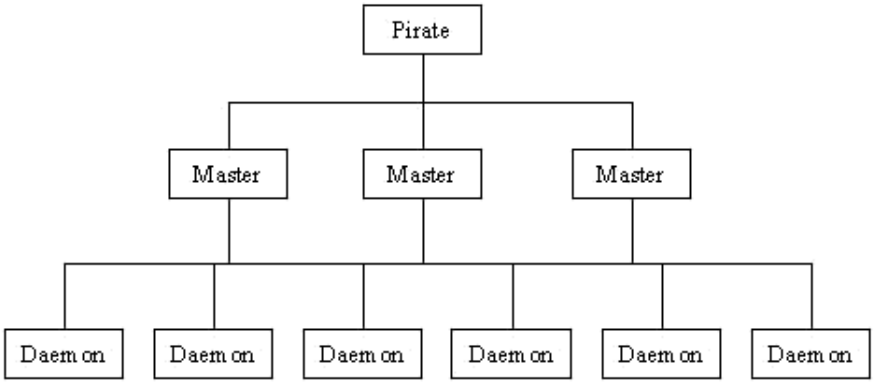
Smurfing

This attack uses the ICMP protocol. When a ping (ICMP ECHO message) is sent to a broadcast address (for instance 10.255.255.255), this last is reduced and sent to every machine in the network. The principle of the attack is to spoof the ICMP ECHO REQUEST packets sent using the target as source IP address. The cracker sends a continous ping flow to the network broadcast address and all the machines answer the target with an ICMP ECHO REPLY message. The flow is then multiplied by the number of hosts in the network. In that case, the whole target network will be affected by the denial of service, since the big traffic generated with this attack leads to a network congestion.

Distributed denial of service

The distributed denial of service saturates the attacked network. The idea is to use various sources (daemons) for the attack and masters to control them. The most known DDoS (*Distributed Denial of Service*) tools are Tribal Flood Network (TFN), TFN2K, Trinoo and Stacheldraht. Picture 13 shows a typical DDoS network:

Pic.13: DDoS network



The cracker uses masters to easily control the sources. Obviously, he needs to connect (TCP) to the masters to configure and prepare the attack. The masters only send commands to the sources via UDP. Without the masters, the cracker should have to connect to each source. The origin of the attack would be detected in a much easier way and its implementation would last much longer.

Every daemon and master talks to each other exchanging specific messages depending on the tool used. These communications can also be encrypted and/or authenticated. To install the daemons and the masters, the cracker uses known vulnerabilities (buffer overflow on services such as RPC, FTP, etc). The attack itself is a SYN Flooding or a Smurf Attack. The result of such a denial of service is to make a

network unreachable.

Conclusion

Today, the security against remote attacks is getting stronger but unfortunately this is not true for internal security. This "poor relation" of protection against crackers still leaves nice perspectives to local attacks such as TCP Session Hijacking, ARP Spoofing and DNS Spoofing. Furthermore, the prediction of sequence numbers (heart of IP Spoofing) and the Fragment Attack variants appear just because of bugs found in the network equipment's OS. Concerning applications attacks, they still have good times ahead because of the growing complexity of the web related applications and of the shorter deadlines set to developers and administrators. The denial of service attack will stay fearsome in its distributed form as long as every user fails to realize the need for protecting his machine.

Links

- RFC 1858 - Security Considerations for IP Fragment Filtering: sunsite.dk/RFC/rfc/rfc1858.html
- IP Spoofing Demystified - Phrack 48: www.phrack.org/
- Simple Active Attack Against TCP - Laurent Joncheray: www.insecure.org/stf/iphijack.txt
- DNS ID Hacking - ADM Crew: packetstorm.securify.com/groups/ADM/ADM-DNS-SPOOF/ADMID.txt
- The DoS Project's "trinoo" - David Dittrich: staff.washington.edu/dittrich/misc/trinoo.analysis
- The Strange Tale of the DENIAL OF SERVICE Attacks against GRC.COM: grc.com
- hping2: www.kyuzz.org/antirez/hping.html
- nemesis: www.packetfactory.net/Projects/nemesis/
- mendax: packetstorm.securify.com/Exploit_Code_Archive/mendax_linux.tgz
- hunt: lin.fsid.cvut.cz/~kra/index.html
- dsniff: www.monkey.org/~dugsong/dsniff/
- fragrouter: packetstorm.securify.com/UNIX/IDS/fragrouter-1.6.tar.gz

---

Webpages maintained by the LinuxFocus Editor team	Translation information:
© Eric Detoisien	fr --> -- : Eric Detoisien <valgasu(at)club-internet.fr>
"some rights reserved" see linuxfocus.org/license/	fr --> en: Georges Tarbouriech <gt(at)linuxfocus.org>
http://www.LinuxFocus.org	

2005-01-14, generated by lfparser_pdf version 2.51