Anatomy of an IP Fragmentation Vulnerability in Linux IPChains: Investigating Common Vulnerabilities and Exposures (CVE) Candidate Vulnerability CAN-1999-1018 (1)

GIAC Security Essentials Certification (GSEC) Practical Assignment, Version 1.4b, Option 1

Karim R. Sobhi

Submitted on May 29, 2003

Abstract

This paper investigates a potential IP fragmentation vulnerability in Linux IPChains. This candidate vulnerability is discussed in the Common Vulnerabilities and Exposures (CVE) database (1). A candidate vulnerability is one that has been identified, but has not yet been tested to establish whether it can be used to breach a system. The aim of this paper is to do exactly that: to establish whether or not this potential vulnerability can be used to compromise the security of a Linux IPChains firewall.

The candidate IP fragmentation vulnerability in question allows an attacker to bypass a Linux IPChains firewall by accessing ports on an internal network host that should be blocked by the firewall. Because of a glitch in IPChains code, an attacker can use IP fragmentation to disguise traffic that's prohibited by the firewall rules as traffic that is allowed.

After conducting a series of experiments, it appears that this is an actual vulnerability in IPChains. Whether it is exploitable, on the other hand, could not be shown. It was possible for a custom-made fragmented FTP SYN packet to reach an internal host running Red Hat Linux 8.0 and protected by a Linux Mandrake 6.0 IPChains firewall, although the firewall rules deny FTP traffic. However, the internal host did not issue a reply due to a failure to re-assemble the packet fragments.

The Groundwork

Vulnerabilities

A vulnerability is a "bug", either in a host's operating system or a running application, which causes the operating system or application to behave unpredictably or in a fashion not intended by the operating system or application's author under certain circumstances. Vulnerabilities can be used to (9):

- Gain unauthorized access.
- Simplify gaining unauthorized access.
- Take a system offline.
- Desensitize sensitive information.

Aside from software "bugs", some vulnerabilities exploit misconfigurations by system administrators to exploit a system. The vulnerability discussed in this paper is a combination of a software bug and some misconfiguration on the part of the firewall administrator.

IPChains

IPChains is the built-in firewall that comes with the Linux kernel version 2.2.x (4). It provides a packet-filtering framework, allowing the administrator to specify which packets are allowed into the system, out of the system, or through the system to other hosts according to a set of rules, specified in “chains”. A chain is simply a sequence of rules that are checked in order. As soon as a packet matches a rule, that rule’s “target” applies to the packet. This target can be to accept, deny, reject (drop and send an ICMP notification), masquerade (do Network Address Translation, or NATing) among others. For the purpose of this paper, only the ACCEPT and REJECT targets are relevant. Rules can match packets according to source IP address/port, destination IP address/port, incoming interface, fragmentation status…etc. IPChains has three basic “chains”, as described in (3) and (11):

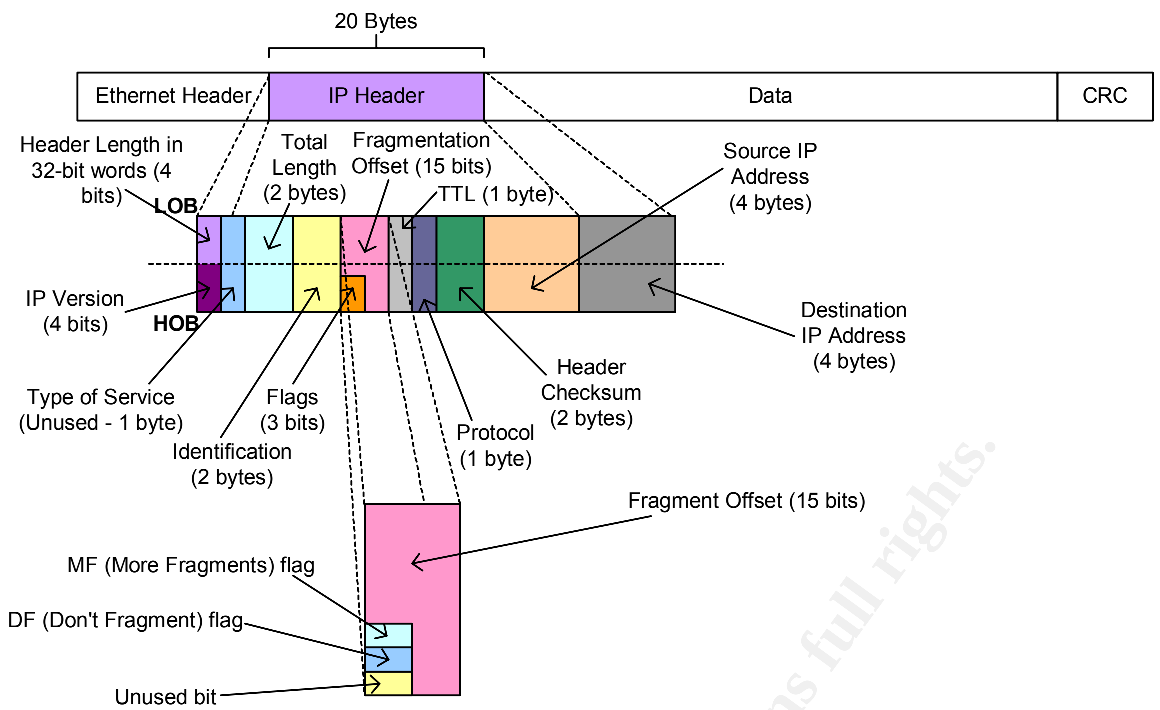
1. The INPUT chain: Packets arriving at the network interface are checked against the rules in this chain and, accordingly, are allowed or denied access to the host.

2. The OUTPUT chain: Packets about to leave the host are checked against this chain to determine if they should be allowed to leave.

3. The FORWARD chain: This chain is used for packets that arrive at the network interface, but are destined to other hosts, rather than the firewall. Note that packets that are to be forwarded are checked against all three chains.

IP Fragmentation

So, what is IP fragmentation? As packets traverse the Internet to get from one network to another, they may pass through different hosts on different physical networks. Discrepancies in the way these hosts operate, and are configured, require the IP protocol to do some extra work in order to accommodate for the diversity. IP fragmentation is an example of this fact. A host has a Maximum Transmission Unit (MTU), which is the maximum size of a datagram that can be placed into a frame (10). In the event that the MTU is smaller than the size of the packet to be transmitted, IP allows the division of this packet into “fragments”, which will be re-assembled into one packet by the IP layer at the receiving host. This is done transparently to the upper layer protocol (most commonly the Transmission Control Protocol or TCP). Following is a diagram of the IP header fields, with emphasis on the fields and flags used for IP fragmentation



The DF (Don't Fragment) flag indicates whether IP should be allowed to fragment this packet. If it is set, IP will drop the packet if it is larger than the MTU. The MF (More Fragments) flag indicates whether there are more fragments to come. It must be set in all fragments except the last. The "Fragment Offset" field is used during re-assembly to decide where each fragment fits in the original packet. The reason this field is required is that packets do not necessarily arrive at the destination host in the same order in which they were sent. Therefore, the destination host cannot depend on the order of packets to determine how they should be assembled.

When a packet is fragmented, the Ethernet header is copied as-is to every fragment. The IP header, on the other hand, must be modified. This list of steps is taken from (10):

1. The MF (More Fragments) flag bit is set in all fragments except the last. This indicates that this is not the last fragment. There are more to come.
2. The "fragment offset" field in each is set to the location this data portion occupied in the original datagram, relative to the beginning of the original unfragmented datagram. The offset is measured in 8-byte units.
3. The "header length" field of the fragment is modified.
4. The "total length" field of the fragment is modified.
5. The "header checksum" field is re-calculated.

The Vulnerability

This vulnerability candidate uses IP fragmentation in order to bypass IPChains. The details are discussed on the SecurityFocus BugTraq mailing list (2). Following is the relevant code excerpt from the IPChains source code (/net/ipv4/ip_fw.c). Note that the Linux kernel version 2.2.20 has some extra code to handle this issue. A browsable copy of ip_fw.c for the 2.2.20 kernel can be found at (5).

```
/* If we can't investigate ports, treat as fragment.  It's
 * either a trucated whole packet, or a truncated first
 * fragment, or a TCP first fragment of length 8-15, in which
 * case the above rule stops reassembly.
 */

if (offset == 0) {
      unsigned int size_req;
      switch (ip->protocol) {
      case IPPROTO_TCP:
            /* Don't care about things past flags word */
            size_req = 16;
            break;

      case IPPROTO_UDP:
      case IPPROTO_ICMP:
            size_req = 8;
            break;

      default:
            size_req = 0;
      }

      offset = (ntohs(ip->tot_len) < (ip->ihl<<2)+size_req);
}
```

The vulnerability takes advantage of the fact that IPChains treats packets with incomplete TCP header information in the same way as non-first IP fragments. The "if" statement condition will be true if the fragment offset header field of the packet is 0. This will be the case if:

1. This is not a fragment. It is carrying a complete TCP datagram.
2. This is the first fragment of a fragmented datagram.

The last line sets "offset" to 1 (TRUE) if the total packet length is less than the sum of the IP header length and the minimum allowable IP data segment size (16 bytes in case of TCP). Later on in the code, the "offset" variable is used to differentiate between non-first fragments and first fragments (or non-fragments). Therefore, if the firewall receives a first fragment with an incomplete TCP header, the "if" statement condition will evaluate to true, the code will be executed, and the "offset" field will receive the value of 1 (TRUE), and will later on be dealt with as if it were a non-first fragment.

The vulnerability also takes advantage of the fact that many firewall administrators create a rule to accept all non-first fragments, assuming that they will not be re-assembled correctly, and thus will be discarded, at the destination if the first fragment was blocked by the firewall. Usually this is done to ease the load on the firewall by relieving it from having to look into every single non-first fragment. However, this rule will be our ticket to bypassing the firewall. This means that, for this vulnerability to occur, two conditions must be satisfied (2):

1. The IPChains firewall must be configured to accept all non-first fragments.

2. The kernel must **NOT** be compiled with the kernel option CONFIG_IP_ALWAYS_DEFRAG, which will cause the firewall to attempt to re-assemble fragments instead of forwarding them as-is (2). Re-assembly will expose the attack, and the firewall will block the complete packet after re-assembly.

An attack to exploit this vulnerability would need to do the following (2):

1. Attacker sends a fragment, with offset 0, a set IP_MF bit, and a full transport protocol header which meets the packet filter and is passed to the victim machine.

2. Attacker sends a fragment, with offset 0, a set IP_MF bit, and a length of 4 bytes. This contains the (blocked) ports that the attacker wishes to access on the victim machine. This fragment will be accepted by the firewall and overlap - in the victim machine's reassembly chain - the port information contained in the fragment sent in step 1.

3. Attacker sends a fragment with a cleared IP_MF bit, starting where the first fragment left off, that completes the set of fragments.

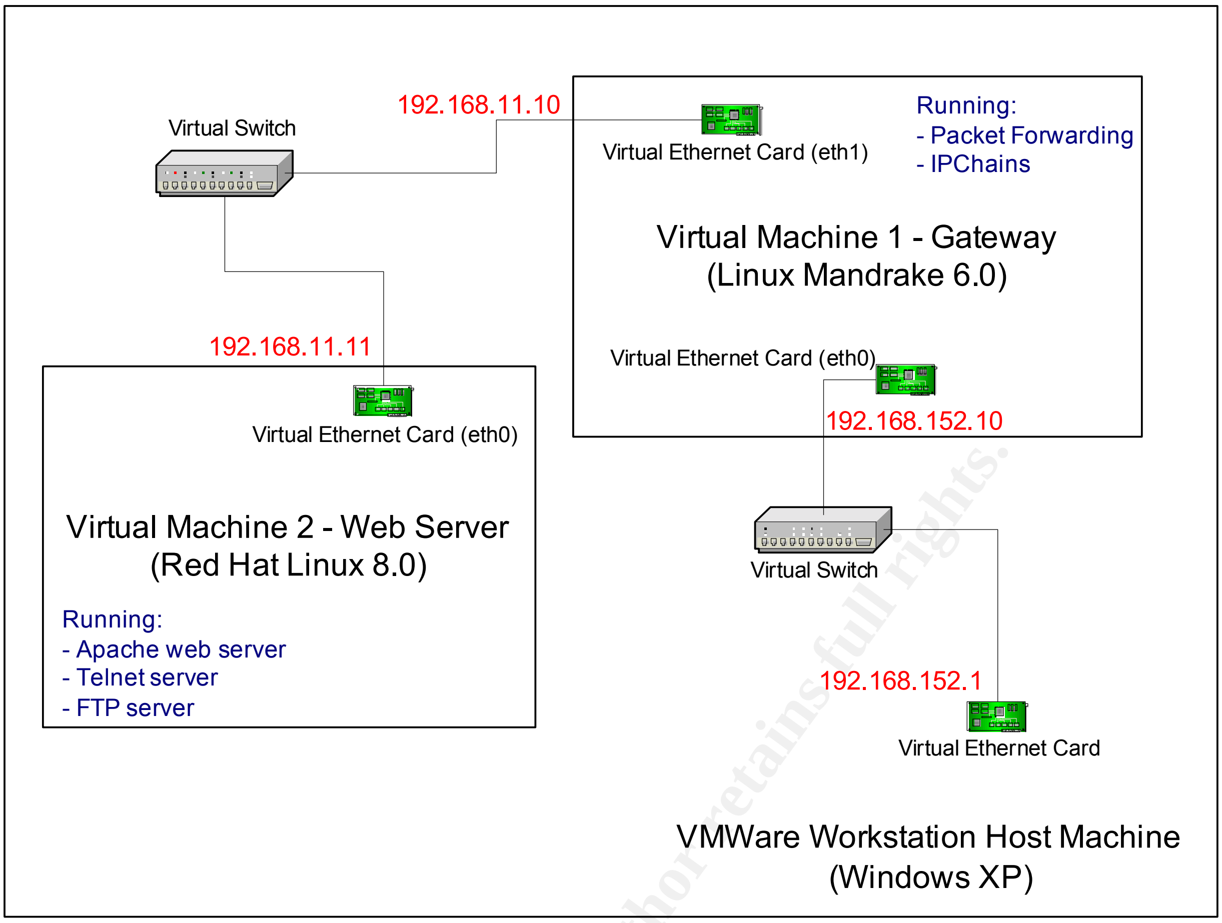
At the destination host, the IP protocol will re-assemble these fragments. Since both fragments are carrying an offset of 0, the second fragment will overwrite the port information on the first fragment. The re-assembled datagram will be passed to the TCP protocol, with the denied destination port number, thus allowing access to the blocked port.

The Experiment

Setting Up the Network

In order to test this vulnerability, I figured I need three machines. One will be the internal server, on which a service will be unlawfully accessed, a firewall box running the vulnerable IPChains, and a third box representing the "outside world", from which the attack will be launched. To run IPChains, I needed a Linux distribution that carries the 2.2.x kernel. Luckily enough, I was able to get my hands on Linux Mandrake 6.0, built on the 2.2.9 kernel. For the internal server, I decided to use Red Hat Linux 8.0. I configured it to run the Apache web server (the legitimate service), telnet (in case I need it) and FTP (the service that we are trying to illegitimately access). As the outside world, any machine would do.

Since I do not have three machines and two switches, I decided to set up my network virtually using VMWare Workstation® 4.0. I created the setup as follows:



Configuring IPChains

IPChains consists of three main built-in firewalling “chains”: the INPUT, OUTPUT and FORWARD chains.

For this setup, I configured the INPUT and OUTPUT chains to always accept all packets. This is done by setting the “policy” for the chain using the “-P” option.

```
#ipchains –P input ACCEPT
#ipchains –P output ACCEPT
```

As for the FORWARD chain, I set the policy to REJECT, meaning that it will drop all packets by default, and send an ICMP message back to the source to indicate that the packet was dropped. The next step was configuring what traffic is accepted. I allowed HTTP, Telnet and ICMP traffic, and I allowed FTP traffic from the inside of the network to the outside world, but not the other way round.

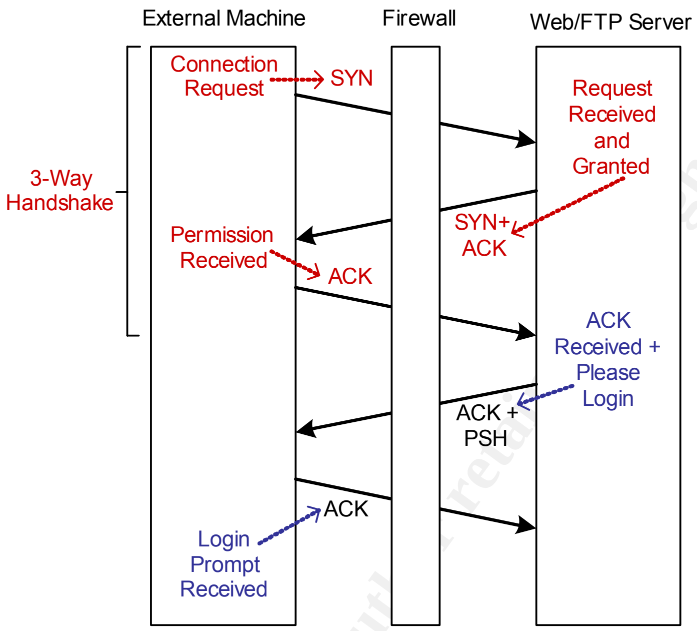
```
#ipchains –P forward DENY
#ipchains –A forward –j ACCEPT –p icmp
#ipchains –A forward –j ACCEPT –p tcp –d 192.168.11.11 80 –i eth1 –b
#ipchains –A forward –j ACCEPT –p tcp –d 192.168.11.11 23 –i eth1 –b
#ipchains –A forward –j ACCEPT –p tcp –s 192.168.11.11 21 –i eth0
```

Finally, the firewall must be configured to accept all non-first fragments

```
#ipchains –A forward –j ACCEPT –f
```

The Game Plan

Now that the setup is complete, I am ready to run the attack. I first started a regular FTP session while sniffing on the wire, in order to establish the exact sequence of packets involved in the process. The following diagram represents the packet sequence. At that point the firewall was configured to accept everything.



There were five packets sent to establish the connection:

1. SYN packet, requesting connection
2. SYN+ACK packet, request acknowledged and granted
3. ACK, permission received

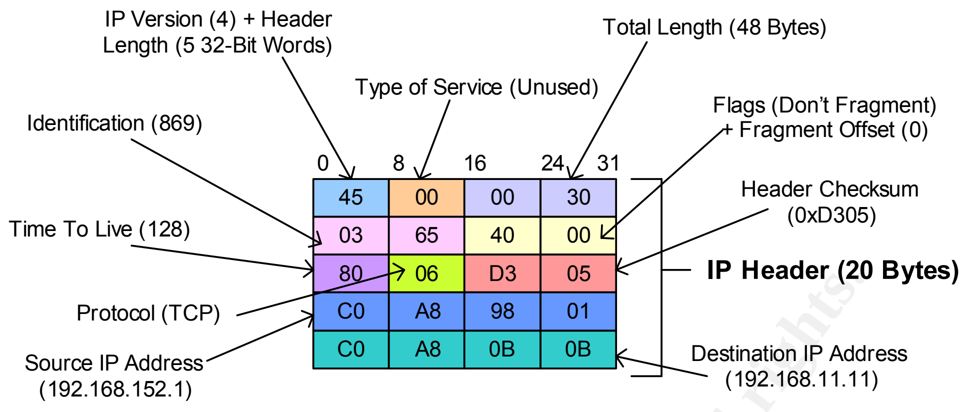
These three steps constitute the TCP 3-way handshake.

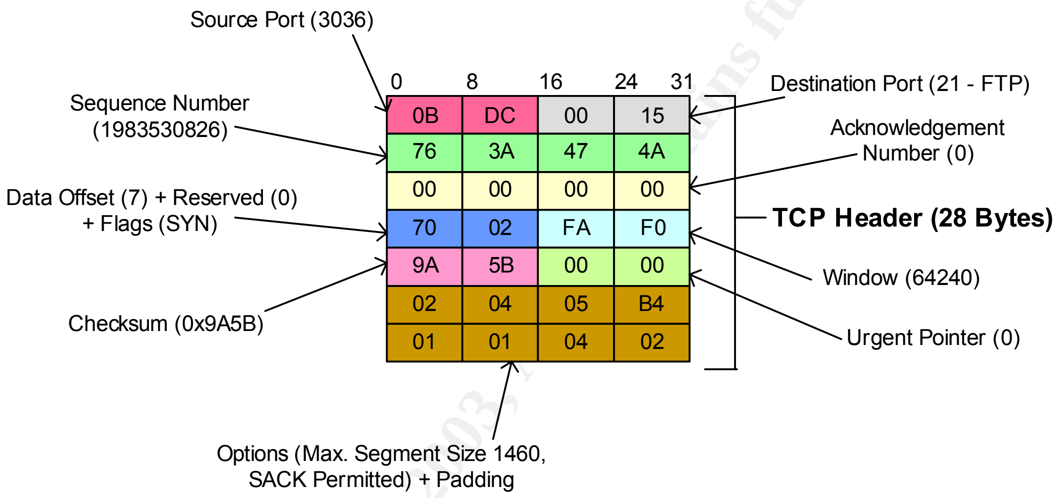
4. ACK+PSH packet, Received your acknowledgement, here's the login prompt (the PSH flag ensures that the data, the login prompt, is flushed from the server's TCP buffer) (8).
5. ACK, received your login prompt.

After careful consideration, I decided that my target would be to send a connection request (SYN packet) to the FTP server and receive a valid reply (SYN+ACK). This would provide a valid proof of concept, as it means that the attack was able to successfully contact the FTP service on the server machine through the firewall despite the fact that its rules deny FTP traffic.

The Attack

The attack consists of three fragmented packets, sent from the VMWare host machine (Windows XP), to the web server (Red Hat Linux 8.0) through the IPChains firewall (Linux Mandrake 6.0). When re-assembled at the web server, they will form the SYN packet directed at port 21 (FTP), the first step in the TCP handshake process. The following SYN packet was generated by a regular FTP request. It will be the target packet.





Source Port (3036)
Sequence Number (1983530826)
Destination Port (21 - FTP)
Acknowledgement Number (0)
Data Offset (7) + Reserved (0) + Flags (SYN)
TCP Header (28 Bytes)
Window (64240)
Checksum (0x9A5B)
Urgent Pointer (0)
Options (Max. Segment Size 1460, SACK Permitted) + Padding
0
8
16
24
31
0B
DC
00
15
76
3A
47
4A
00
00
00
00
70
02
FA
F0
9A
5B
00
00
02
04
05
B4
01
01
04
02

Note that there is no TCP data segment. It is simply a connection request. The first fragment involved in the attack will carry the first 16 bytes of the TCP header. The following changes will be made to the IP header:

1. The MF (More Fragments) flag will be set.

2. the IP header checksum will be re-calculated to accommodate for these changes.

3. In the TCP header, the destination port will be changed to 80 (HTTP). Note that the TCP checksum is not re-calculated. By the time the datagram reaches the TCP layer on the web server, the port will have been changed back to 21 (FTP), and therefore the checksum must remain as-is.

Since the packet has an offset of 0 and a complete TCP header, and since HTTP traffic is accepted by the firewall, the fragment will be allowed to pass through to the web server.

The second fragment will carry a complete IP header. However, it will only carry the first four bytes of the TCP header, specifying the same source port as the previous fragment, but a destination port of 21 (FTP). Since this is a packet with an incomplete TCP header, it will be treated by IPChains as a non-first fragment, will not be checked for ports, and will be allowed through to the web server.

As for the third packet, it will carry the remaining 12 bytes of the TCP header, and the following changes will be made to its IP header:

1. The fragment offset field will carry the value of 2, as it is calculated in units of 8 bytes (8*2 = 16 bytes sent in the fragment #1).

2. The IP header checksum will be re-calculated.

Since this is the last fragment, it's MF flag will remain cleared.

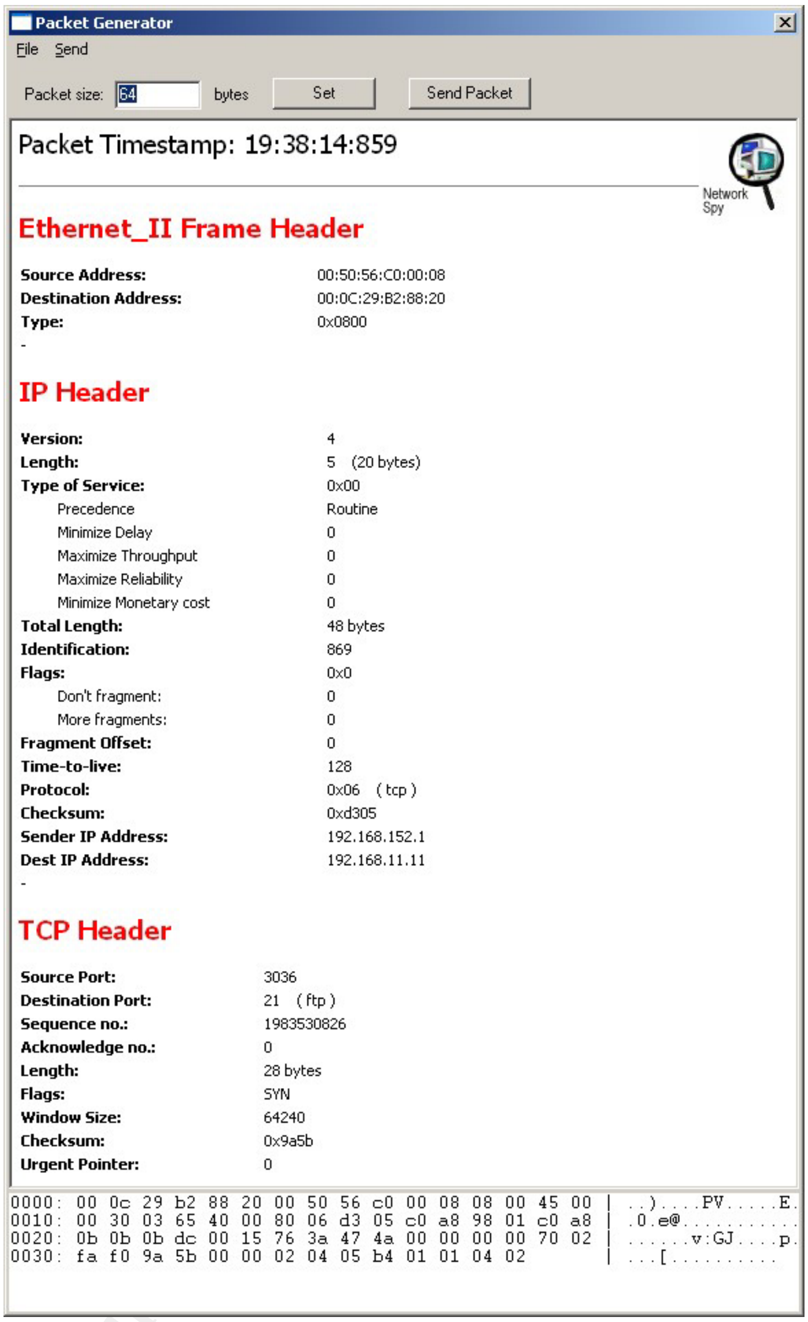
The FTP service will receive our custom-made TCP datagram, with the SYN flag set. It will issue a SYN+ACK reply, which will pass through the firewall (note that the firewall was configured to allow outgoing FTP traffic) and back to the host machine, thus proving the validity of the vulnerability candidate.

Tools Used

A number of Windows and Linux tools were used in order to conduct the experiment successfully. Following is a discussion of each tool and how it contributed to the experiment.

- **<u>VMWare Workstation 4</u>** (http://www.vmware.com): VMWare Workstation is a Windows/Linux application that creates virtual machines within the operating system on the "host machine". It encapsulates the host's hardware, so that the virtual machine sees it as if there were no operating system running. This "abstraction" is so complete that any operating system can be installed on the virtual machine, and will interact with it as if it were a physical machine. Moreover, VMWare Workstation allows for the creation of networked environment, through "virtual switches". Virtual network cards can be added to virtual machines and they can be "connected" to any virtual switch as desired. This is how the networked setup was created for this experiment, due to the lack of physical machines and switches. The guest operating systems used were Red Hat Linux 8.0 and Linux Mandrake 6.0.

- **<u>Network Spy</u>** (http://www.sumitbirla.com/network-spy/): Finding a raw packet generator for Windows was not an easy task. Most free tools are built for Linux and released under the GPL. I tried several packages for Windows, but none of them seemed to match my requirements. I was already investigating JPCap (Java Packet Capture library) and planning to develop my own packet generator in Java when I stumbled upon "Network Spy". This tool is mainly a protocol analyzer (similar to Ethereal, discussed below). However, it has an extra tool that came in very handy: the "Packet Generator". This tool allows for the creation of custom frames to be sent unmodified on the wire. The packet generator view is divided into two sections:



```
Packet Timestamp: 19:38:14:859

Ethernet_II Frame Header

Source Address:              00:50:56:C0:00:08
Destination Address:         00:0C:29:B2:88:20
Type:                        0x0800
-

IP Header

Version:                     4
Length:                      5   (20 bytes)
Type of Service:             0x00
    Precedence               Routine
    Minimize Delay           0
    Maximize Throughput      0
    Maximize Reliability     0
    Minimize Monetary cost   0
Total Length:                48 bytes
Identification:              869
Flags:                       0x0
    Don't fragment:          0
    More fragments:          0
Fragment Offset:             0
Time-to-live:                128
Protocol:                    0x06   ( tcp )
Checksum:                    0xd305
Sender IP Address:           192.168.152.1
Dest IP Address:             192.168.11.11
-

TCP Header

Source Port:                 3036
Destination Port:            21   ( ftp )
Sequence no.:                1983530826
Acknowledge no.:             0
Length:                      28 bytes
Flags:                       SYN
Window Size:                 64240
Checksum:                    0x9a5b
Urgent Pointer:              0
```

```
0000:  00 0c 29 b2 88 20 00 50 56 c0 00 08 08 00 45 00 | ..).....PV......E.
0010:  00 30 03 65 40 00 80 06 d3 05 c0 a8 98 01 c0 a8 | .0.e@.............
0020:  0b 0b 0b dc 00 15 76 3a 47 4a 00 00 00 00 70 02 | .......v:GJ....p.
0030:  fa f0 9a 5b 00 00 02 04 05 b4 01 01 04 02       | ...[...........
```

The packet shown above is the original unfragmented FTP SYN packet that I used to create my attack fragments.

1. Details: The details view shows every field in the different frame headers, namely the Ethernet (layer 2), IP (layer 3) and, in this case, TCP (layer 4) headers. Unfortunately, though, it does not allow changes to be made to the packet in this view. It is read-only.

2. Bytes: The lower part of the packet generator tool window shows the actual bytes (in hexadecimal) that will be put on the wire. This is the section where changes can be made. Any changes are immediately applied to the "details" view, which simply provides a more readable view of the frame.

A very useful feature of the packet generator was the ability to save and load packets. This allowed me to create my attack fragments once and then just load them and make changes or send them as appropriate. One feature I would have like to see in this software is the ability to define sequences of packets to be sent automatically, instead of having to deal with each fragment independently. However, this option was not there. There were several bugs in the software, all related to fragmentation:

1. The DF and MF flags in the "details" view did not change after they were edited in the byte view.

2. Possibly due to the previous bug, the packet generator treats all frames as complete packets. This was clear when I used the tool to create a non-first packet fragment. The "details" view attempted to interpret the IP data as a TCP header, although it was clear that this was not a complete packet, and therefore is not expected to carry a complete TCP header. Moreover, since it is a non-first fragment, it is clear that it will not carry a complete TCP header.

Although this software was very useful to me in running my experiment, I personally believe that a free and truly powerful packet generator for Windows is yet to be developed.

- **Ethereal** (http://www.ethereal.com): Ethereal is a very powerful and established protocol analyzer for Linux and Windows. I used it mainly on the destination (Red Hat Linux 8.0) host, in order to check on the receiving fragments to ensure that they arrive unchanged at the destination. It was very useful to filter packets to show only TCP traffic sometimes to avoid seeing a lot of unnecessary DNS, ARP and windows generated traffic.

- **tcpdump** (http://www.tcpdump.org): tcpdump is a classical console-based protocol analyzer that comes with most, if not all, Linux distributions. As it does not come with a GUI, it is not very easy to follow a packet's details. I only used tcpdump on the firewall host, as it was running Linux Mandrake 6.0 and did not come with Ethereal. I looked for an Ethereal version for Mandrake 6.0 but I gave up quickly. All I needed on the firewall was to establish which packets are being let through and which are not, which was well within the power of tcpdump. There are several useful options in tcpdump. One that was very helpful to me was the "-vv" option, which means "be VERY verbose". This provided more information about the dumped packet (including fragmentation information).

- **HEC Calculator** (http://byerley.cs.waikato.ac.nz/~tonym/hec.html): Since my experiment entailed the creation of custom packets, it was necessary to be able to calculate the IP header checksum, given the different IP header field values. At first, my search brought forward several web pages that explain how the IP header error checksum is calculated, complete with binary examples. A little more search yielded this HEC (Header Error Checksum) calculator, which made my life a lot easier.

- **Microsoft Visio** (http://www.microsoft.com/office/visio/): Microsoft Visio was used to create the four drawings included in the practical.

Conclusions

The conducted experiment was successful as far as proving that this vulnerability actually exists in IPChains. However, it was not possibly to establish that it is practically exploitable.

The fact that the vulnerability exists in IPChains was established completely by observing traffic arriving at the network interface of the web server. All three fragments that constitute the attack arrived unaltered at the network interface, which proves that IPChains did, in fact, treat a packet with incomplete TCP header information as if it were a non-first IP fragment.

As far as proving that the vulnerability is exploitable, however, the experiment was not as successful. After the fragments arrived at the interface, no SYN+ACK was issued in response by the web server. Several causes may be responsible for this:

1. The fragments are too small to be processed by the web server. The three fragments were 36 bytes, 24 bytes and 32 bytes long. However, it is worth noting that the original unfragmented SYN packet is 48 bytes long.

2. The web server employs some sort of packet filtering, although all NetFilter chains are running an ACCEPT policy, and have no rules defined.

3. The fragments time out in the receiver queue before they are all received, and are discarded. This option was ruled out after a revision of the code (6) showed that the fragmentation timeout is 30 seconds, and that an ICMP “fragment timeout exceeded” packet is sent in the event of such an occurrence. This did not occur in the experiment.

Heavy attempts were made to trace and follow the IP re-assembly process, with the help of (7), and the “Cross-Referencing Linux” code database at http://lxr.linux.no. However, it was not possible to establish the reason that no SYN+ACK is issued.

References

1. “CAN-1999-1018 (under review).” Common Vulnerabilities and Exposures (CVE). CAN-1999-1018. 12 Sep. 2001.

   URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-1018 (14 Apr. 2003).

2. Lopatic, Thomas. “Linux IPChains Firewall Vulnerability.” 27 Jul. 1999.

   URL: http://www.securityfocus.com/archive/1/19810 (14 Mar. 2003).

3. Russel, Rusty. “IP Firewalling Chains.” Linux IPCHAINS-HOWTO. v. 1.0.8. 4 Jul. 2000.

   URL: http://www.tldp.org/HOWTO/IPCHAINS-HOWTO-4.html (8 Apr. 2003).

4. “YoLinux: Using Linux and iptables / ipchains to set up an internet gateway for home or office.” Greg Ippolito.

   URL: http://www.yolinux.com/TUTORIALS/LinuxTutorialIptablesNetworkGateway.html (28 Apr. 2003).

5. “Cross-Referencing Linux: Linux/net/ipv4/ip_fw.c.” Linux Kernel 2.2.20. 21 Oct. 1999.

   URL: http://lxr.linux.no/source/net/ipv4/ip_fw.c?v=2.2.20 (8 May 2003).

6. “Cross-Referencing Linux: Linux/net/ipv4/ip_fragment.c.” Linux Kernel 2.4.18. 12 Jan. 2002.

   URL: http://lxr.linux.no/source/net/ipv4/ip_fragment.c?v=2.4.18 (17 May 2003).

7. Westall, James M. “IP Reassembly.” 12 Mar. 2003.

   URL: http://www.cs.clemson.edu/~westall/881/iprecv4.pdf (17 May 2003).

8. Meng, Xiannong. “TCP Header.” Transport Layer Protocols. 13 Nov. 1997.

   URL: http://www.cs.panam.edu/~meng/Course/CS6345/Notes/chpt-6/node7.html (8 May 2003).

9. Cole, Eric. Hackers Beware, First Edition. New Riders Publishing, August 2001.

10. Murhammer, Martin W. TCP/IP Tutorial and Technical Overview, Sixth Edition. International Business Machines (IBM), October 1998. 48 – 55.

11. The IPChains Manual Page.