THE THEORY OF BUILDING LARGE P2P BOTNETS

by **pest**
posted by **Insider**

This article was first posted on Inception E-Zine, issue 2013 and brought to GREYSEC.NET. This document contains the thread’s first message. To see the complete discussion, please visit https://greysec.net/showthread.php?tid=7537.

***P2p botnet** sounds truly grandiose. Many people think that only pros are capable of creating such a botnet. The truth is, the most important thing you need is to understand the theory of p2p, which is unbelievably simple. The main objective is to connect IP bots and pass commands from bot to bot.*

Architectures

1. Temporary Node Exchange (IP of Bots)

Every bot has a table stored (routing table) with the following structure:

```
struct NODE{
uint32 ip;
uint16 port;
uint32 time;
};
```

Where '*ip*' stands for the bot IP, '*time*' is the time the bot was added, '*port*' is the port. Let the table be limited to the size of 255 elements NODE [255]; It means that the bot will have the maxim of 255 neighbors. The table contains unique IPs and is arranged by time. This can be done with the help of the *qsort* function:

```
8.5.1.2 - 10:55:10 01.09.2013
1.5.1.2 - 10:53:10 01.09.2013
1.1.1.2 - 10:53:01 01.09.2013
1.1.1.2 - 06:33:10 01.09.2013
[…]
```

The bot "*looks at*" the table, extracts elements subsequently and sends messages with requests for a new list. The bot receiving this kind of message responds with current IPs (from the top of the table). It does not send all 255 IPs though, rather than a few, for example, 10 elements NODE[10]; The bot that initialized the request for a new IP list gets these NODE[10] and checks if they are on its list. If they are, it updates the time. If they aren't, it adds the new ones. If the table is full (with 255 elements), it updates the 'oldest' element.

That way, a constant exchange of IPs is supported. Only current IPs are added to the table. As time for transferring IPs between bots, it's best to use delta time_delta=now()-time, where now() is current time (to avoid certain attacks related to transferring the "excessive" time value). The entire network represents one large segment that has time as its coordinates, so the entire network approaches current time. We can distinguish **ZAccess** as a real example of such a network. It uses the same p2p network architecture as described above.

2. Exchange of Nodes by Distance (distributed hash tables DHT)

Every bot has a routing table:

```
struct NODE{
uint8 nid[16]; //ID of node (bot)
uint32 ip; //IP
uint16 port; //port
uint32 time; //time
};
```

The number of elements is limited by NODE[255]. When installed, every bot generates an ID and remembers it, it's a *nid*, for example, 0x00000000000000000000000000000004. When exchanging nodes, the bot adds to the routing table only *nids* close to it. Only then it's guided by the time. For example, there is a list:

```
0x00000000000000000000000000000000
0x00000000000000000000000000000001
0x00000000000000000000000000000002
0x00000000000000000000000000000003
0x00000000000000000000000000000005
0x00000000000000000000000000000006
0x00000000000000000000000000000007
0x00000000000000000000000000000008
```

The following will be the closest to 0x00000000000000000000000000000004:

```
0x00000000000000000000000000000003 (4−3) = 1
0x00000000000000000000000000000005 (5−4) = 1
0x00000000000000000000000000000002 (4−2) = 2
0x00000000000000000000000000000006 (6−4) = 2
```

To work with close nids, their ranges have to be somehow calculated to be put into order later. For that, a bit-to-bit (byte-to-byte) XOR operation is used. Let's see what happens if we XOR the list by 0x00000000000000000000000000000004:

```
(0x00000000000000000000000000000000^0x00000000000000000000000000000004)=0x00000000000000000000000000000004
(0x00000000000000000000000000000001^0x00000000000000000000000000000004)=0x00000000000000000000000000000005
(0x00000000000000000000000000000002^0x00000000000000000000000000000004)=0x00000000000000000000000000000006
(0x00000000000000000000000000000003^0x00000000000000000000000000000004)=0x00000000000000000000000000000007
(0x00000000000000000000000000000004^0x00000000000000000000000000000004)=0x00000000000000000000000000000000
(0x00000000000000000000000000000005^0x00000000000000000000000000000004)=0x00000000000000000000000000000001
(0x00000000000000000000000000000006^0x00000000000000000000000000000004)=0x00000000000000000000000000000002
(0x00000000000000000000000000000007^0x00000000000000000000000000000004)=0x00000000000000000000000000000003
(0x00000000000000000000000000000008^0x00000000000000000000000000000004)=0x0000000000000000000000000000000C
```

Arrange by result:

```
(0x00000000000000000000000000000004^0x00000000000000000000000000000004)=0x00000000000000000000000000000000
(0x00000000000000000000000000000005^0x00000000000000000000000000000004)=0x00000000000000000000000000000001
(0x00000000000000000000000000000006^0x00000000000000000000000000000004)=0x00000000000000000000000000000002
(0x00000000000000000000000000000007^0x00000000000000000000000000000004)=0x00000000000000000000000000000003
(0x00000000000000000000000000000000^0x00000000000000000000000000000004)=0x00000000000000000000000000000004
(0x00000000000000000000000000000001^0x00000000000000000000000000000004)=0x00000000000000000000000000000005
(0x00000000000000000000000000000002^0x00000000000000000000000000000004)=0x00000000000000000000000000000006
(0x00000000000000000000000000000003^0x00000000000000000000000000000004)=0x00000000000000000000000000000007
(0x00000000000000000000000000000008^0x00000000000000000000000000000004)=0x0000000000000000000000000000000C
```

You can arrange with the help of *qsort* function because the number of elements in our tables is not too large. However, if you are planning to use large routing tables, you should use the binary tree. That way, every bot accumulates *nids* that are the closest to it, breaking the network into a multitude of segments whose number equals the number of bots. The **Zeus GameOver (P2P)** network is a real example of how it works.

You can perform a nid search in such a network. If you associate, say, a file with a nid, you can also perform a search for a specific file. Let's say md5 h1 is calculated from the file (just the 16 bytes for our example). In the routing table, a nid close to h1 is searched for (or several ones, because not all nodes may be online) and a request to the list of close nids with h1 request is made. A remote node received this request and searches its list for nids close to h1, sending them back. After a few of such requests, if the distance decreases, nid^h1 can exceed a certain value. You can consider the file found and execute a request to download the file from the node. This is how files are searched for in **KAD networks**.

Files in P2P Network

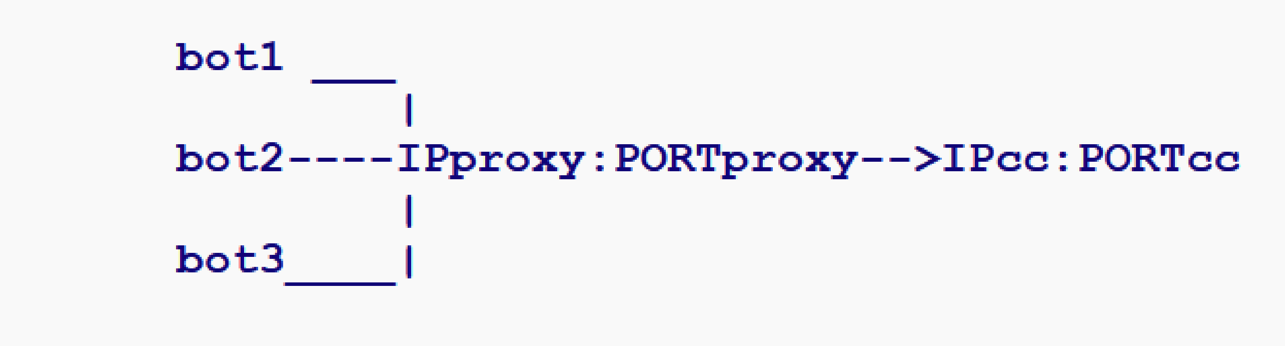
The types of files reviewed above connect bots between themselves, although this happens in different ways. This means that bots can send messages to their "neighbors". Files can be such messages. If you upload a file to one bot, it can pass it to its neighbors, and they will pass it on to their neighbors etc. until the file is disseminated across the network. That file can also be a bot update, another bot, a configuration file. Remember to such your files with an e-signature (private key), and the bot will check it with its public key upon receipt. Then your bot will know you were the one that signed it. Otherwise, your botnet will be hijacked very soon…

Tunnels (proxy)

You may have noticed from the information above that this kind of network does not have a command center, while it would be good to have one, to gather statistics on bots, form grabber data, web injects etc. This command center should also be concealed behind the nodes. To implement this idea, you can enable every bot to tunnel traffic.

- The *botmaster* selects from the list of bots those that will be tunneling traffic;
- The *botmaster* connects to them and sends a packet signed with its e-signature containing IPcc:PORTcc of its CC, that way tuning the bot into a tunnel;
- A configuration is created in which the *botmaster* signs IPproxy:PORTproxy (the bot it turned into a tunnel) with its e-signature;
- The configuration is sent throughout the network the standard way, just like a usual file.

The bots start going to the tunnel that sends packets to CC:



```
bot1 ___
        |
bot2----IPproxy:PORTproxy-->IPcc:PORTcc
        |
bot3____|
```

Usually, the UDP protocol is used to exchange IP lists, while TCP is used to download/upload files, for TCP tunnels.

Good luck with bot-building.