barentsz.py Host Dicovery Tool

Author: ikkini <ikkini@gmail.com>
Version: 0.1
Revision: 0.1

> ***Note***
>
> "[...] the crew came across a herd of approximately 200 walruses and tried to kill them with hatchets and pikes. Finding the task more difficult than they imagined, they left with only a few ivory tusks."
>
> http://en.wikipedia.org/wiki/Willem_Barentsz

Initial setup

1. Be sure you have the right python in your environment
2. I will describe adding packages on OSX (using ports), as I get to install and run this on (Kali/BackTrack) linux I'll update these notes. Adding missing dependencies is usually as simple as adding a port (such as scapy) and sometimes a little more complicated, as for instance with adding netaddr:

```
sudo port install py27-distribute
--->  Computing dependencies for py27-distribute
--->  Cleaning py27-distribute
--->  Scanning binaries for linking errors: 100.0%
--->  No broken files found.
sudo easy_install netaddr
```

Why cb.py?

```
targets = ['192.168.0.0/24','10.0.0.1']
# -- Options
inter=0
retry=-2
timeout=3
concurrent=256
toptcpports=7 # all = 4238, taken from http://nmap.org/svn/nmap-services
from datetime import datetime
timestamp = datetime.now().strftime("%Y%m%d.%H%M%S")
database = timestamp + '.barentsz.db'
import random
randport = random.randrange(1025,65535)

# -- PROTOCOLS
ICMPtypes = [8,13,15,17]
UDPports = [161,500,4500,randport]
```

cb.py contains a collection of settings and stuff such as "payloads". Let me take you through them.

cb.targets

I've chosen to set the targets here and not on the command line because I'm a lazy bum. I actually have some code which will do this fine for one ip address or CIDR range, but not for a list of target(s/-ranges). For now, I just edit the ranges here, but probably this will change in the future.

The list of strings will go to netaddr, which will create lists of IP addresses and put them in the database. barentsz.py will read them in batches (as lists), limited by cb.concurrent.

cb.concurrent

Scapy will gladly build you a list of packets to be send out for a /8 with 65535 ports per hosts. It will completely rape your RAM and perhaps in the end fail because of allocation limits, but by the Gods, it will try. That is why I've added a **cb.concurrent** option.

If you decide to send a packet to 4238 ports on 16777214 hosts, you can use this option to select a suitable batch-size. The **itsAlive** function, using a SELECT statement, will filter in any hosts not yet found alive (alive=0). With a bit of luck, this means you'll send out less and less packets as you go.

```
def itsAlive(max,start):
    T = []
    cur.execute('SELECT ip from ips where alive=0 limit ?,?', (max,start))
    rows = cur.fetchall()
    for row in rows:
        T.append(row['ip'])
    return T
```

cb.inter, cb.retry, cb.timeout

Like the options themselves, their description is taken straight from scapy (documentation):

inter : *sr(inter=,)*

> Time in seconds to wait between 2 packets

retry: sr(retry=,)

> If retry is 3, scapy will try to resend unanswered packets 3 times. If retry is -3, scapy will resend unanswered packets until no more answer is given for the same set of unanswered packets 3 times in a row.

timeout : *sr(timeout=,)*

> How much time to wait after the last packet has been sent. By default, sr will wait forever and the user will have to interrupt (Ctrl-C) it when he expects no more answers.

Discovering hosts is a hard thing to do. Lots of things can go wrong, for unforseen reasons, with packets roaming the interwebs on their own. You will not know what you'll find (or not), until you try. Or to put it formally, discovery is by its very nature a stochastic process.

timeout

It can help to give packets some time to find their way back to you. Increasing **timeout** will probably help you there. Increase it too much, and you'll pobably find you've run out of patience before you've finished the scan.

retry

retry is one of the smartest functions of scapy (and there are a lot!), especially the minus (-) option. Scapy keeps track of unanswered packets and does not scan the whole batch again every retry. Not only does this decrease the footprint of your scanning on the target range, it also means that the (network) devices do not have to handle the full load every time. This could mean they can take their time to answer the packets they could not answer before.

inter

To decrease the network load even further, you can increase the **inter** settting. Again, there is a optimum which depends at least partly on your patience, so I have no idea what the "best" setting would be.