Homework 2 - Scanner Implant

Introduction

You will design a comprehensive network scanning implant that provides several different scanning capabilities. The goal of this implant is to be able to drop it on a foreign network and explore in-depth the machines and attack surface available to you. Your implant will have to discover both machines and open ports serving IPv4 with zero prior information about the network itself or the machines running on it. You will have to manually construct packets for host discovery with ARP.

Requirements

- Your program should accurately generate results from scanning on an unknown network.
- Your implant should be written in Python. The `netifaces` and `netaddr` libraries will be pre-installed on the test environment for your convenience
- You are prohibited from using *any* other external libraries. For example, the `struct` or `socket` libraries are allowed (and encouraged), but `scapy` is not. Your implant will be tested on a fresh Ubuntu machine outside of your control with only the `netifaces` and `netaddr` libraries installed.
- You must perform host discovery using manually crafted ARP packets.
- You must provide a singular executable named `implant` that can be invoked directly. For example, if you provide a Python script, we must be able to run it as follows:
  ```
  # ./implant
  ```
- While speed is not the main concern of this project, your program will be subject to timeouts when tested, and therefore multithreading your code is recommended if your implant is running slowly.
- You can assume that each interface will have a single IPv4 address.
- Your implant must generate a .json report of the machines on the network and ports open. This report should be named `results.json`.

Setup

Your code will be run on a computer in a container that is unknown to you. You will not be able to interact with the foreign network or computer in advance in any way, and it will not have Internet access. Instead, your code must automatically discover other computers on the network(s) and scan them, generating a JSON report of the networks. Please note that your box may be connected to more than one network, and your implant should be able to scan all of them.

Output

Your code will generate a results.json file with the following information:

```
# ./implant
# cat results.json
{
  "routers":{
    "eth0":{
      "ipv4":"10.1.0.254",
    }
  },
  "machines":{
    "eth0":{
      "10.1.0.2":{
        "mac" : "ab:bc:cd:de:ef:ff",
        "tcp":{
          "7":"echo",
          "21":"ftp",
          "80":"http"
        }
      },
      "10.1.0.8":{
        "mac" : "de:ad:be:ef:00:01",
        "tcp":{
          "23":"telnet",
          "25":"smtp",
          "80":"http"
        }
      },
    },
    "eth1":{
      "10.2.0.3":{
        "mac" : "af:b3:39:45:ff:0f",
        "tcp":{
          "80":"http",
          "36":"other",
          "3555":"other"
        }
      },
      ...
  }
}
#
```

A full example is provided in the ELMS assignment.

Notes:

- If there are no IPv4 machines associated with an interface, the interface key should exist, just with no contents.
- All machines that respond to an ARP request should exist in the machines table, *regardless* of whether they have any open ports.
- Your program will be run with root privileges.

- Your machine should be included in the scan, and should show up under all interfaces, with its own IP address.
- You are not responsible for any local interfaces. Local interfaces will be named "`lo`".
- Debug messages are *highly* encouraged if you think they will help us grade. Debug messages during implant execution can also help us grade in the event your implant crashes prematurely.
- Machines will not be connected to the internet.

Scans Required

Your code should be able to perform the following scans for IPv4:

- ARP discovery
- TCP SYN OR TCP Connect Scan (your choice)
- Service identification

In addition, for any open ports you find, your code should automatically try to identify which services are running on those ports. You will be responsible for identifying the following services directly:

- ssh
- ftp
- http
- echo
- telnet
- smtp

If a service does not fall into one of these categories, label it as "other". Your service identification solution need not be foolproof, and simple identification solutions are often better than complicated ones.

Optional Scans

For extra credit, implement router discovery using DHCP discovery. (This requires forging a UDP, BOOTP, and DHCP packet). You will need to create a thread for forging & sending packets and a separate thread for receiving the packets. If you do not implement this option, do not include the `'routers'` key in your output.

Testing

It is important to note that it is easy for networking implants like these to crash. Your program should still try to get as much information as possible. In a real operational environment, you will never totally know the network, and may not be able to test as you go. Therefore, your implants should still try to get as much information as possible. Submissions that crash but still have output will be manually hand-graded to see where the crash occurred. If it is a minor error, this will be kept in mind while grading. You are strongly encouraged to create your own testing environment on Cipherpath to rigorously test your implant.

Grading

Your code will be graded automatically based on how completely you profile the foreign target network. Your `json` output file should be valid. Points will be deducted for each missed piece of information (service, mac address, port, etc). You will not be graded on spelling/grammar/capitalization/spacing/order issues.

Deliverables

Please submit your code to the ELMS submit link. Your code should not have require any command-line arguments and should be just a python executable named “implant” in a zip file.