Scapy

- What is Scapy
- Why use Scapy?
- Important concepts to keep in mind
- Crafting your first packet
- Sending and Receiving Packets/Frames
- Basic examples and dealing with the OS's TCP/IP stack using iptables
- Quick discussion of the Super Socket

What is Scapy

- A Python framework used for crafting and transmitting packets.
- Capable of Sniffing or replaying packets for troubleshooting or fuzzing network services.
- Can be used as the interactive Python interpreter or the framework can be imported as a python module and used for further coding
- Everything is an object!

Why use Scapy?

- Blue Team
  - Test IDS/IPS
  - Test Firewall
  - Learn more about TCP/IP (down and dirty)
  - Application response(Fuzzing)
- Red Team
  - Fire teh lazorz (DOS/DDOS)
  - More Fuzzing
  - Penetration Testing

Important Concepts

- Everything is an Object – treat it as such
  - IP(), TCP(), UDP(), ICMP()
- Important commands to remember:
  - help() - displays help
  - ls() - displays packet classes
  - lsc() - displays commands available to you
- When assigning Field Values(either works)
  - ip=IP(src=”1.2.3.4”, dst=”google.com”)
  - ip=IP()
  - ip.src=”1.2.3.4”
  - ip.dst=”google.com”

Important Concepts Continued

- Displaying Values of Variables
  - ls(ip) – shows what you have set and default vaules
  - ip – shows only what you have set
  - ip.show() - omits variable classes and default values
- Assembling the Network Layers
  - packet=IP(dst=”1.2.3.4”)/TCP(flags=”S”,dport=443)
  - frame=Ether(type=0x8100)/Dot1Q(vlan=99)/packet
- Payload attribute
  - Will Display all the layers after the initial

Crafting your first packet

- ICMP echo(type 8) request to dst – 192.168.1.103
  - send(IP(dst=”192.168.1.103”)/ICMP(type=8))
- Using Variables
  - packet=IP(dst=”192.168.1.103”)/ICMP(type=8)
  - send(packet)
- Inoking Scapy into a python script
  - #! /usr/bin/python
    from scapy.all import *
    i=IP(dst=”192.168.1.1”)
    t=TCP(dport=80, flags=”S”)
    packet = i/t
    send(packet)

Sending and Receiving

- Frames (Layer 2)
  - sendp() - layer 2 sending
  - srp() - send and receive on Layer 2
  - srp1() - send and receive a single response
- Packets (Layer 3)
  - send() - layer 3 sending
  - sr() - layer 3 send and receive
  - sr1() - send and receive a single response

Basic Examples

- Start a TCP connection
  - sr1(IP(dst=”192.168.1.1”)/TCP(flags=”S”, dport=80,seq=100))
- Send to Multiple IPs and Listen for responses
  - sr(IP(dst=[”192.168.1.1”, “192.168.1.2”])/ICMP())
- Send to Multiple ports and see responses
  - sr(IP(dst=”192.168.1.1”)/TCP(dport=[80, 443, 22,445]))
  - Then to view them: ans,unans=_
  - ans.summary()

More useful features

- Fuzzing Values
  - send(IP(dst=”192.168.1.1”)/fuzz(ICMP(code=0, seq=0, id=0)), loop=1)
    - This will fuzz all values what are not assigned and stay in a loop until you Ctrl+C out of it.
- conf – allows you to modify default values and change scapy configurations
  - Conf.route – shows the routing table scapy will use
- Wireshark Interface
  - wireshark(packet) – will launch a wireshark interface showing the packet you crafted

Cooked Sockets / Raw Sockets and IP Tables

- Cooked sockets uses Native TCP/IP Stack
  - Kernel builds packet
  - Assigns correct IP/UDP/TCP header values
  - You supply the payload
- Raw Sockets circumvents Native TCP/IP Stack
  - You build packet
  - You assign header values
  - You supply the payload

IP Tables helps us

- Block TCP Outbound Resets
  - iptables -A OUTPUT -p tcp –tcp-flags RST RST -s 192.168.1.1 -d 192.168.1.2 –destination-port 80 -j DROP
- Block UDP Outbound ICMP port unreachables
  - iptables -A OUTPUT -s 192.168.1.1 -d 192.168.1.2 -p ICMP –icmp-type port-unreachable
- Call IPTables in your python script
  - Import subprocess
    cmd = [iptables rules]
    subprocess.call(cmd, shell=False)

A look at the Three-way Hand Shake

```
#!/usr/bin/python

from scapy.all import *


ip=IP(src="192.168.1.1", dst="192.168.1.2")

SYN=TCP(sport=1030, dport=80, flags="S",
seq=10)

SYNACK=TCP(sr1(ip/SYN)


my_ack =SYNACK.seq + 1

ACK=TCP(sport=1030, dport=80, flags="A"
```

Super Socket

- Network socket is a function that opens, reads, writes, and closes an instance of network communications
- Using both Scapy “sniff” to read DNS query and “send” to write new spoofed responses requires the use of two separate sockets
  - This method takes to longer and will never “beat” the DNS server
- Scapy uses a “super-socket” which takes care of both reading and writing with a single socket
  - Less time and can beat the server

In closing

- Scapy is a very useful tool for:
  - Gaining an in depth knowledge of the TCP/IP stack
  - Great tool for security/network analysts and testers
- Great features:
  - Cross Platform
  - Easily read, write, craft packets on the fly
  - Easily incorporate Scapy into an existing python script
  - Replay pcaps back onto the network

Sources

- SANS Security Course: Power Packet Crafting with Scapy
- Research done on Wikipedia

http://webstersprodigy.net/2012/07/06/some-practical-arp-poison-attacks-with-scapy-iptables-and-burp/