Network Protocol Security Testing with the Packet Construction Set

George Neville-Neil
Consultant
The Problem

- Writing network protocol code is hard
- Testing network protocols is as hard as writing the protocol in the first place
- Most current systems are incomplete
  - Only support a small number of packets
  - Not extensible
  - Written in write-once languages
- Proprietary systems are expensive and incomplete
  - ANVL
  - SmartBits
One Solution

• A packet programming language
• Programming languages are hard
• Most people don’t want to learn a special language
• Adoption rate might be low
• Maintenance is hard
Packet Construction Set

- A Python Library for creating new objects that represent packets
- Re-use a well known language
- Make it easy to create packet objects
- Most of the code in the scripts is pure Python
- Python is readable and easy to work with
- Python is not a write-once language
We need to get from this...

<table>
<thead>
<tr>
<th>Version</th>
<th>IHL</th>
<th>Type of Service</th>
<th>Total Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification</td>
<td>Flags</td>
<td>Fragment Offset</td>
<td></td>
</tr>
<tr>
<td>Time to Live</td>
<td>Protocol</td>
<td>Header Checksum</td>
<td></td>
</tr>
<tr>
<td>Source Address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destination Address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options</td>
<td>Padding</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
struct ip {
    #if BYTE_ORDER == LITTLE_ENDIAN
    u_int ip_hl:4;  /* header length */
    ip_v:4;        /* version */
    #endif
    #if BYTE_ORDER == BIG_ENDIAN
    u_int ip_v:4,  /* version */
    ip_hl:4;      /* header length */
    #endif
    u_char           ip_tos;  /* type of service */
    u_short          ip_len;  /* total length */
    u_short          ip_id;   /* identification */
    u_short          ip_off;  /* fragment offset field */
    #define           IP_RF 0x8000 /* reserved fragment flag */
    #define           IP_DF 0x4000 /* dont fragment flag */
    #define           IP_MF 0x2000 /* more fragments flag */
    #define           IP_OFFMASK 0x1fff /* mask for fragmenting bits */
    u_char           ip_ttl;  /* time to live */
    u_char           ip_p;    /* protocol */
    u_short          ip_sum;  /* checksum */
    struct           in_addr ip_src,ip_dst; /* source and dest address */
} __packed;

Packet Construction Set                     pcs.sf.net                                                  www.neville-neil.com
...er I mean this

```python
version = pcs.Field("version", 4,
                      default = 4)
hlen = pcs.Field("hlen", 4)
tos = pcs.Field("tos", 8)
length = pcs.Field("length", 16)
id = pcs.Field("id", 16)
flags = pcs.Field("flags", 3)
offset = pcs.Field("offset", 13)
ttl = pcs.Field("ttl", 8, default = 64)
protocol = pcs.Field("protocol", 8)
checksum = pcs.Field("checksum", 16)
src = pcs.Field("src", 32)
dst = pcs.Field("dst", 32)
```
What does PCS provide?

- A simple system for laying out packets
- A way to access packet fields programmatically
- A special set of classes called Connectors which provide easy access to:
  - TCP
  - UDP
  - IP
  - PCAP
- Access to pcap and bpf interfaces by an extended version of py-pcap
  - Thanks Doug Song!
A Simple Example

```python
from pcs.packets.arp import *
from pcs.packets.ethernet import *

arppkt = arp()
arppkt.op = 1
arppkt.sha = ether_atob(ether_source)
arppkt.spa = inet_atol(ip_source)
arppkt.tha = "\x00\x00\x00\00\x00\x00"
arppkt.tpa = inet_atol(target)
```

A Simple Example continued

ether = ethernet()
ether.src = ether_atob(ether_source)
ether.dst = "\xff\xff\xff\xff\xff\xff\xff"
ether.type = 0x806

packet = Chain([ether, arppkt])

output = PcapConnector(interface)

out = output.write(packet.bytes, len (packet.bytes))
What just happened?

- We handcrafted an ARP packet
- We handcrafted an Ethernet packet
- We chained two packets together
- We transmitted them
- We did this in 15 lines of code
A quick on line demo
Why do I care?

- You now have programmatic access to any field in any packet
- It is now trivial to write
  - Protocol conformance tests
  - Fuzzers
  - Experimental protocols
What about security?

- Triggering kernel panics
- Attempting to get servers to give up their data
- Interactively interrogate a server
  - Using PCS in python’s command line interpreter
- Using the Python unittest module to generate repeatable tests for known security issues
Google Summer of Code Project

• Clement Lecigne took PCS and created IPv6 specific protocol fuzzers
• Several issues were found
• All were fixed (of course)
• Validates our approach
Too Big Packet Generator

- Try to trigger a kernel panic with PCS
- Send a “packet too big” message to the system
- The entire program including options parsing, is 125 lines
Making the packet

```python
def makepkt(sip, smac, dmac, dip, len = 8):
    ip = ipv6.ipv6()
    ip.traffic_class = 0
    ip.flow = 0
    ip.next_header = IPPROTO_ICMPV6
    ip.length = len + 8
    ip.hop = 64
    ip.src = inet_pton(AF_INET6, dip)
    ip.dst = inet_pton(AF_INET6, sip)
    # icmp6 header
    icmp6 = icmpv6(ICMP6_ECHO_REPLY)
    icmp6.type = ICMP6_ECHO_REPLY
    icmp6.code = 0
    icmp6.id = os.getpid()
    data = "A" * len
    icmp6.checksum = icmp6.cksum(ip, data) + 1
    chain = pcs.Chain([ip, icmp6])
    return chain.bytes
```

Putting on the headers

def toobig(iface, mtu, dstip, dstmac, srcip, srcmac, pkt):
    """send fake icmp TOO BIG packet"""
    # ethernet header
    eth = ethernet()
    eth.dst = eth.name2eth(dstmac)
    eth.src = eth.name2eth(srcmac)
    eth.type = ETHERTYPE_IPV6
    # ipv6 header
    ip = ipv6.ipv6()
    ip.traffic_class = 0
    ip.flow = 0
    ip.next_header = IPPROTO_ICMPV6
    ip.hop = 255
    ip.length = 8 + len(pkt)
    ip.src = inet_pton(AF_INET6, srcip)
    ip.dst = inet_pton(AF_INET6, dstip)
# icmp6 header
icmp6 = icmpv6(ICMP6_PACKET_TOO_BIG)
icmp6.type = ICMP6_PACKET_TOO_BIG
icmp6.code = 0
icmp6.mtu = mtu
icmp6.checksum = icmp6.cksum(ip, pkt)
chain = pcs.Chain([eth, ip, icmp6])
c = pcs.Connector("IPV6", iface)
c.write(chain.bytes + pkt)
c.close()
Results

• Code did not cause a panic
• In some cases it caused the system to be unable to communicate using IPv6
• More tests are necessary
Access to PCAP

- Most tools that work with PCAP files are one-offs
- Some tools are too graphical
- What is needed is a library for working with PCAP on which to build more tools
- PCS provides extensions to pypcap for better integration with scripting
DDOS Analyzer

- Sometimes a site DDOSs itself
- How can we detect a real DDOS from a mistake?
- Look at the source addresses and see if they are clustered
- `ddos_analyze.py` is 67 lines including options parsing
- 5000 packet pcap file
- Test data retrieved from a public server
- Snaplen of 9000 bytes
Analysis Output

? ddos_analyze.py -f pcaptestfile -s 255.255.255.0 -n 10.0.0.0 -m 5

5001 packets in dumpfile
5 unique source IPs
0 packets in specified network
Top 5 source addresses were
Address 204.152.184.203 Count 2473 Percentage 49.450110
Address 64.13.135.16 Count 2 Percentage 0.039992
Address 64.13.134.241 Count 1 Percentage 0.019996
Address 195.137.95.246 Count 1 Percentage 0.019996
Address 64.13.134.241 Count 1 Percentage 0.019996
Address 195.137.95.246 Count 1 Percentage 0.019996
1.898u 0.214s 0:02.12 99.0% 0+0k 0+7io 0pf+0w
Analyzing the packets

while not done:
    try:
        packet = file.read()
    except:
        done = True
    packets += 1
    ip = ipv4(packet[file.dloff:len(packet)])
    if (ip.src & mask) != network:
        if ip.src in srcmap:
            srcmap[ip.src] += 1
        else:
            srcmap[ip.src] = 1
    else:
        in_network += 1
Doing the analysis

hit_list = sorted(srcmap.itervalues(), reverse = True)
for i in range(1,max):
    for addr in srcmap.items():
        if addr[1] == hit_list[i]:
            print "Address %s \t Count %s \t Percentage %f" % (inet_ntop(AF_INET, struct.pack('!L', addr[0])), addr[1], (float(addr[1]) / float(packets)) * float(100))
Current Status

- Alpha 0.3 currently available
  - But most people would call it Beta

- Packets
  - Link Layer: Localhost, Ethernet
  - Network Layer: ARP, IPv4, ICMPv4, IPv6, ICMPv6, ND6
  - Transport Layer: UDP, TCP
  - Application Protocols: DNS, DHCPv4
  - Every protocol has a test suite!
Scripts

- arpwhohas.py
  - Generate a fake ARP

- ddos_analyze.py
  - Determine majority source addresses in a pcap file

- dns_query.py
  - Generate a fake DNS query

- http_get.py
  - Grab a web page

- pcap_info.py
  - Print out various bits of info about a pcap file
Scripts Con’t

- **ping.py**
  - A simple ICMPv4 packet generator

- **snarf.py**
  - A trivial packet sniffer

- **pcap_slice.py**
  - Carve up pcap files analogous to tcpslice

- **udp_echo.py**
  - Generate a fake UDP packet
Future Work

• Add more packets
  – Attempt to cover 80% of all known protocols

• Add more scripts

• Integrate into a protocol conformance test framework
  – NetTest is another project

• More documentation
  – Manual exists but is incomplete
More Information

- Project hosted on Source Forge
- BSD License
- http://pcs.sf.net
- More scripts and packets welcome!

Questions?