

# Network packet forgery with xxd

Information Security Inc.

# Contents

- Why use xxd?
- The Layers
- Crafting a Packet
- Replaying the Packet
- Conclusions
- References

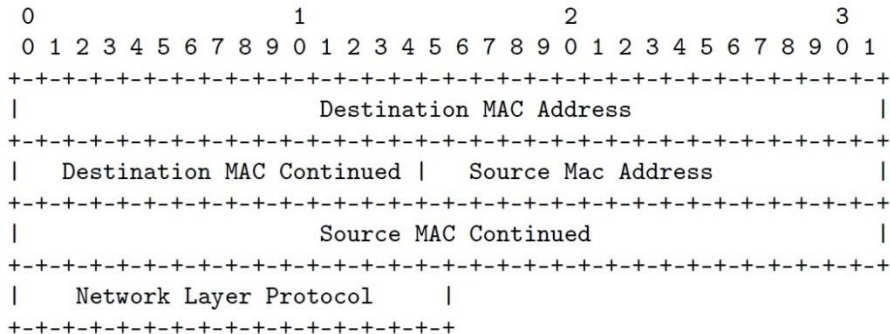
# Why use xxd?

- Installed by default on every Linux distribution
- Low level method to build a packet
- Full control of the packet fields

# The layers

## © Data Link Layer

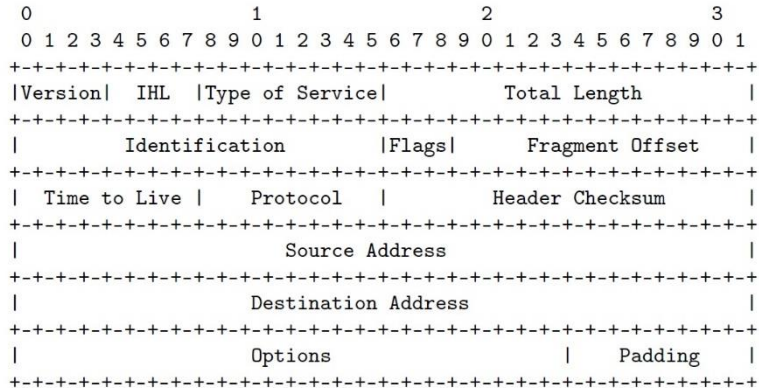
Has the destination and source MAC addresses and 2 bytes referring to what the Network Layer should be (0x8000 for IPv4)



# The layers

## © Network Layer (RFC791)

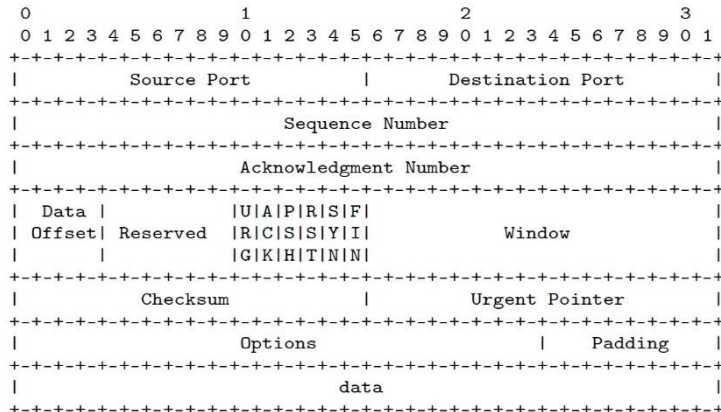
IPv4 interesting fields: Version, Total length, TTL, Source and Destination IP addresses, Checksum and the Protocol byte (0x06 for TCP)



# The layers

## © Transport Layer (RFC793)

Handles the transportation of data between two hosts. Simple UDP header but complex TCP headers.



# Crafting a packet

## © Link Layer

- Destination MAC address: b4:0c:25:67:5a:00
- Source MAC address: 00:0c:29:69:6f:c6
- Network-layer protocol of IPv4: 0x8000

# Crafting a packet

© Network layer (IPv4) -> Header length 20 bytes and no IP options

- First nybble (version is 0x4): 0x4
- Second nibble (Header length in 32 bit words): 0x5
- Service byte (QoS): 0x00
- Total length (61 bytes IP+TCP+Payload): 0x003D
- IP identification field: 0x1337
- IP flags + Fragment offset: 0x4000
- TTL (140): 0x8C
- Higher protocol TCP: 0x06
- Checksum:  $0x4500 + 0x003d + 0x1337 + 0x4000 + 0x8c06 + 0xC0A8 + 0x0101 + 0xC0A8 + 0x0102 = 0x2A7CD$  with 0x2 overflow -> add 0x2 back in ->  $0xA7Cd + 0x2 = 0xA7CF$ . Final checksum:  $0xFFFF - 0xA7CF = 0x5830$
- Source IP: 192.168.1.1 (0xC0A80101) ; Destination IP: 192.168.1.2 (0xC0A80102)



# Crafting a packet

## © Transport layer (TCP)

- Source port: 0x1337
- Destination Port (HTTPS 443): 0x1BB
- SEQ and ACK number no specificity so both: 0x00000000
- Data offset (TCP header length) 32 bytes (20 header + 12 bytes TCP options) 8 32-bit words + PSH ACK: 0x8018
- Windows Size no specific size so: 0x0000
- URG flag: 0x0000
- TCP Options (Two NOPs and and timestamp): NOPs -> 0x0101; TSval -> 0xDEADBEEF; TSecr -> 0xFFFFFFFF
- Checksum:  $(0xC0A8 + 0x0101 + 0xC0A8 + 0x0102 + 0x0006 + 0x0029) + 0x1337 + 0x01BB + 0x0000 + 0x0000 + 0x0000 + 0x0000 + 0x8018 + 0x0000 + 0x0000 + 0x0101 + 0x080A + 0xDEAD + 0xBEEF + 0xFFFF + 0xFFFF + 0xD796 + 0xC34F + 0x4FC7 + 0xE3C6 + 0xD600 = 0x963A3$  with 0x9 overflow -> add 0x9 back in ->  $0x63A3 + 0x9 = 0x63AC$ . Final checksum:  $0xFFFF - 0x63AC = 0x9C53$

# Crafting a packet

## ◎ PCAP Metadata

- Create pcap metadata to dissect or inject the crafted packet.
- First 6 PCAP 4 bytes fields : magic number ( 0xA1B2C3D4); PCAP version (0x00020004); timezone (GMT 0x00000000); sigfigs field (0x00000000); snaplen (0x0001000F); network data link type (Ethernet: 0x00000001)
- Per packet data: time to default day (0x4EBd02CF); microtime (0x00000000); packet length and capture length (0x0000004B)

Saving the Pcap: echoing hex data in order ( Pcap metadata + packet)

```
echo A1B2C3D4 00020004 00000000 00000000 0001000F 00000001 ¥  
4EBD02CF 00000000 0000004B 0000004B ¥  
¥  
000C2945 7E4A000C 29696FD0 0800 ¥  
¥  
45 00 003d 1337 4 000 8C 06 5830 C0A80101 C0A80102 ¥  
¥  
1337 01BB 00000000 00000000 8 0 18 0000 9C53 0000 ¥  
01 01 08 0A DEADBEEF FFFFFFFF ¥  
¥  
D796C34F4FC7E3C6D6 | xxd -r -p > SEC.pcap
```

# Crafting a packet

## © PCAP Metadata

- Bash script to create the packet.

```
# cat PACKET.sh
#!/bin/bash
echo A1B2C3D4 00020004 00000000 00000000 0001000F 00000001 4EBD02CF 00000000 0000004B 0000004B |
xxd -r -p > SEC.pcap
echo 000C2945 7E4A000C 29696FD0 0800 | xxd -r -p >> SEC.pcap
echo 45 00 003d 1337 4 000 8C 06 5830 C0A80101 C0A80102 | xxd -r -p >> SEC.pcap
echo 1337 01BB 00000000 00000000 8 0 18 0000 9C53 0000 01 01 08 0A DEADBEEF FFFFFFFF | xxd -r -p >>
SEC.pcap
echo D796C34F4FC7E3C6D6 | xxd -r -p >> SEC.pcap
```

- Sending the packet:

```
root@iucy64:~# # tcpdump -i eth1 SEC.pcap
Actual: 1 packets (75 bytes) sent in 0.001598 seconds
Rated: 46933.6 Bps, 0.375 Mbps, 625.78 pps
Flows: 1 flows, 625.78 fps, 1 flow packets, 0 non-flow
Statistics for network device: eth1
  Successful packets: 1
  Failed packets: 0
  Truncated packets: 0
  Retried packets (ENOBUFS): 0
  Retried packets (EAGAIN): 0

root@iucy64:~# #
root@iucy64:~# # tcpdump -nn -i eth1 host 192.168.1.1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
22:20:33.091100 IP 192.168.1.1.4919 > 192.168.1.2.443: Flags [P], seq 0:9, ack 0, win 0, options [nop,nop,TS val 3735928559 ecr 4294967295], len
  h 9
22:20:33.091323 ARP, Request who-has 192.168.1.1 tell 192.168.1.2, length 46
22:20:33.091330 ARP, Reply 192.168.1.1 is-at 00:0c:29:69:6f:d0, length 28
22:20:33.091407 IP 192.168.1.2.443 > 192.168.1.1.4919: Flags [R], seq 0, win 0, length 0
```

# Conclusions

Powerful low level technique to craft custom packets for testing network, endpoint devices etc.

# References

- Man xxd

<https://linux.die.net/man/1/xxd>

- IPv4 Header calculator

<http://www.n-cg.net/hec.htm>

- TCP checksum calculator

<http://www.netfor2.com/tcpsum.htm>