Network Attacks

CS342, Handout 10

Tuesday, Oct. 3rd , 2006
Wellesley College
Daniel Bilar

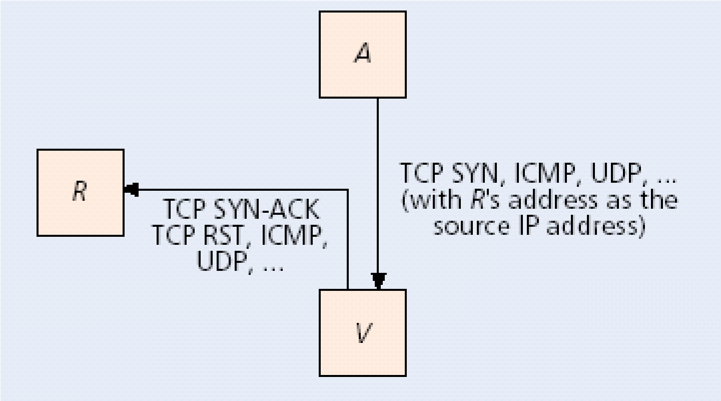
Today’s class goals

- Finish Anatomy of An Attack
- Some selected attacks against
  - Network and transport layer: TCP/IP protocol design and implementation
  - Application layer: Network services like DNS
- Appreciate how various trust assumptions are pre-conditions for these network attacks

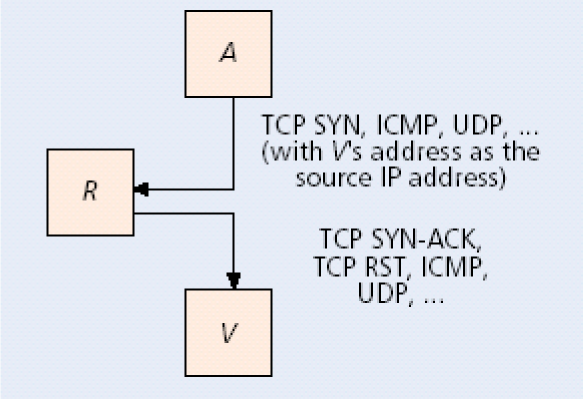
Attack classifications

- **Effect**
  - **Bandwidth depletion**: Flood the victim network with unwanted traffic that prevents legitimate traffic from reaching the victim system
  - **Resource depletion**: Tie up the resources of a victim host
- **Vector**
  - **Direct**: attacking host sends directly to victim machine
  - **Reflector** (indirect): Intermediate nodes are used as attack hosts
- Mechanism
  - Protocol design
  - Protocol implementation

Direct:



Reflector:



IP concepts needed this lecture

1. **IP Spoofing**

   Pretend to be host C when you are host A

2. **ICMP packets**

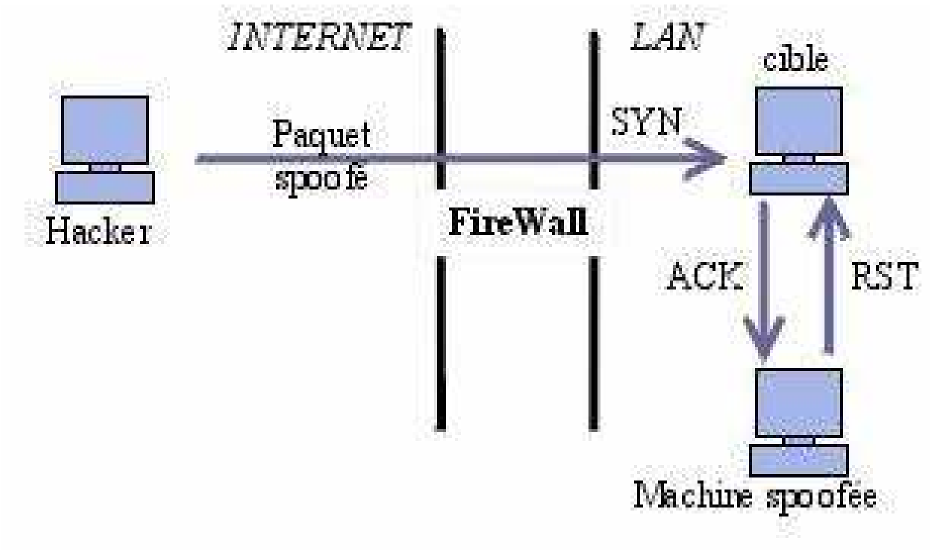
   Feedback mechanism/diagnostic message in IP networks

3. **Fragmentation** and (subsequent) **packet reassembly**

   Break up IP packets into smaller chunks when necessary for network relay

IP Spoofing

- Nothing prevents you from physically mailing a letter with an invalid return address, or someone else's, or your own.
- Likewise, packets can be inserted in the network with invalid or other IP addresses.



Any node can send packets pretending to be from any IP address

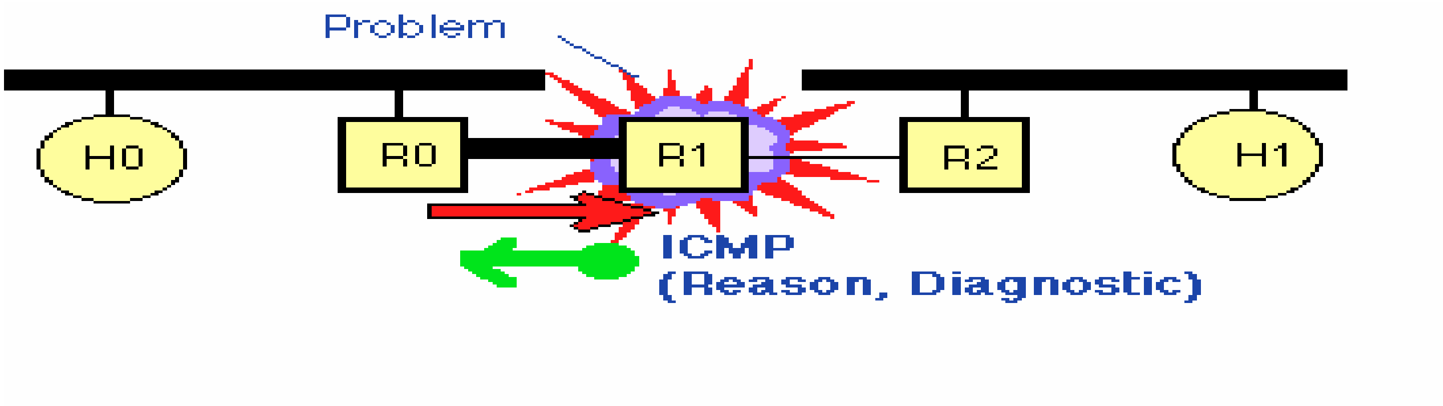
Attacker might not get replies if spoofing a host on a different subnet

For some attacks this is not important
For others, like **TCP hijacking** attacks, it is important

Refinement: IP Spoofing with Reflection

- Use broadcasts pretending to originate from victim
- All replies go back to victim
- Class B broadcast: 253^2 = 64 009 replies
  - □Assuming class C subnetting
- This may use any IP protocol (ICMP, TCP, UDP)
  - □Any application or service that replies using these protocols
  - □Famous attack: Smurf (using ICMP) DoS

ICMP (Internet Control Message Protocol)



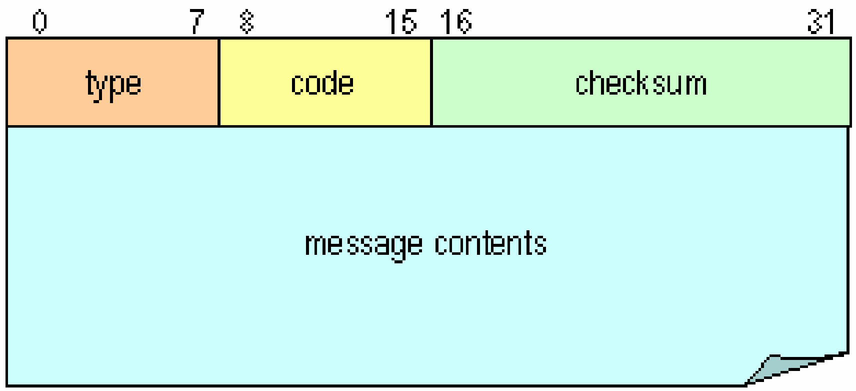
- IP network “feedback” messages
- Used to report problems with delivery of IP packets within IP networks, also for queries
- Encapsulated in an IP packet

Ethernet header	IP header	ICMP header	user data	Ethernet CRC

- Not authenticated!

Basic ICMP Message Types

Type	Code	Desc	Query/Error
0	0	**Echo reply** e.g. ping	Q
3	1	**Host unreachable**	E
3	3	**Port unreachable** (see traceroute)	E
8	0	**Echo request** e.g. ping	Q
11	0	**Time-to-live is zero** during transit (see traceroute)	E



Message types: 40 assigned, 255 possible, ~ 25 in use

ICMP Echo

- a.k.a. Ping
- Destination replies (using the "source IP" of the original message) with "echo reply"
- Data received in the echo message must be returned in the echo reply
- How can this be abused?

Fragmentation

- Networks have different frame sizes
  - “MTU” is the "Maximum Transmission Unit"
- Fragmentation allows oversized packets to be split to fit on a smaller network

- Reassembly is difficult
  - Have to keep track of all fragments until packet is reassembled
  - Resource allocation is necessary before all validation is possible
  - Lots of fragments from different packets can exhaust available memory

➔Perfect grounds for resource exhaustion attacks

Important Fields for Fragmentation

- Fragment ID
  - All fragments have the same ID
- More Fragments bit flag
  - 0 if last fragment
  - 1 otherwise
- Fragment offset
  - Where this data goes
- Data length
  - For how long

Fragmentation situations

- What do you do if you never get the last missing piece?
- What do you do when you get packets out-of-order?
  - This is a legitimate situation as per RFCs
- What do you do if you get overlapping fragments?
- What do you do if the last byte of a fragment would go over the maximum size of an IP packet, i.e., if the size of all reassembled fragments is larger than the maximum size of an IP packet?

Implementation Attack: **Ping of Death**

- **Attack**: Send ICMP echo with fragmented packets : `ping -L 65510 <victim IP address>`
- Maximum legal size of an ICMP echo packet: 65535 - 20 - 8 = 65507
- Fragmentation allows bypassing the maximum size: (offset + size) > 65535
- Reassembled packet would be larger than 65535 bytes
- **Goal**: OS crash

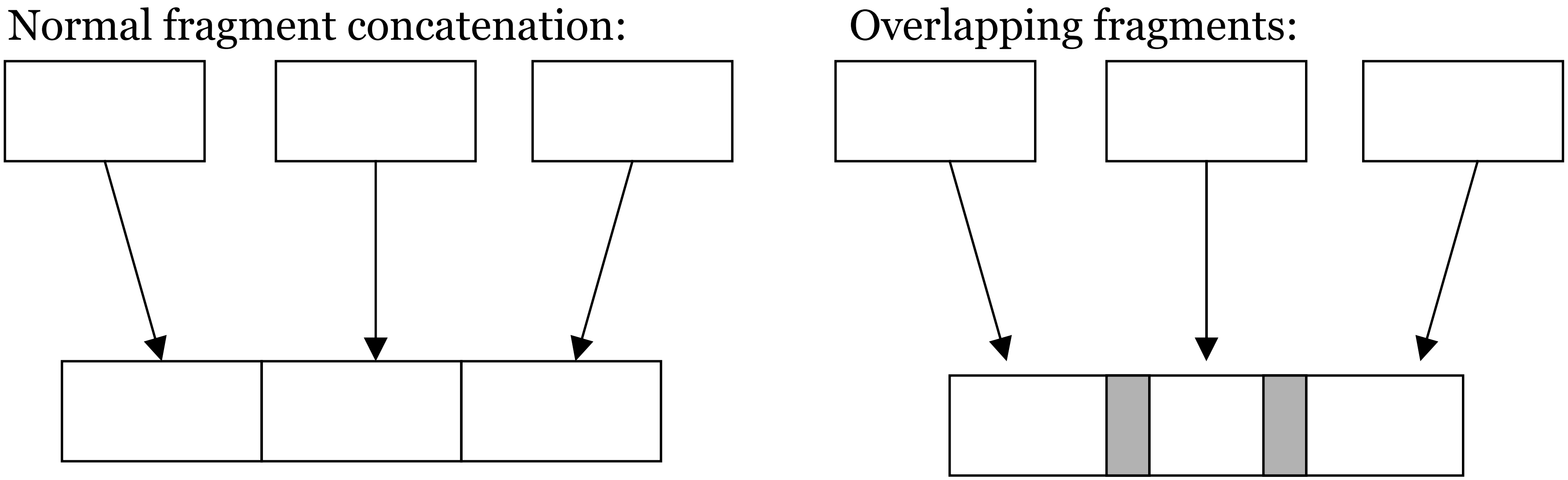
See http://insecure.org/sploits/ping-o-death.html

Implementation Attack: **Teardrop**

- IP packet can be broken, is called ‘fragmentation’

  Fragmented (i.e. broken) packet is reassembled using offset fields
- **Attack**: Send fragments that overlap
- **Goal**: Crash, reboot and hang machine

See http://attrition.org/security/denial/w/teardrop.dos.html



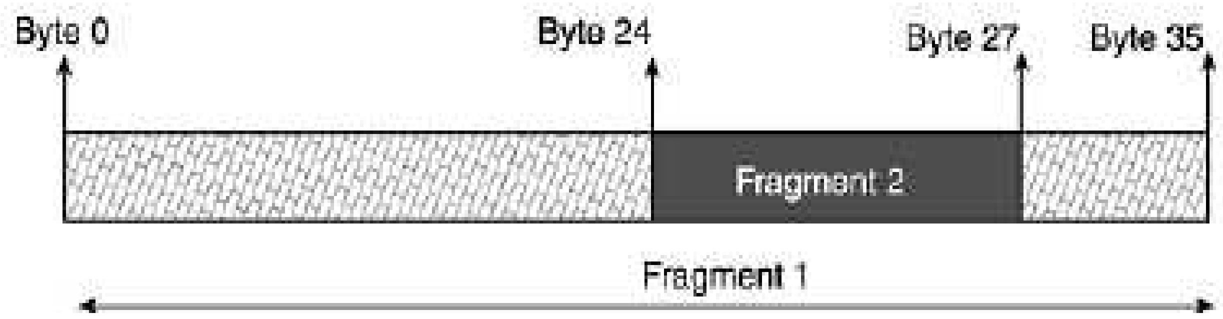
Teardrop: Overview

```
evilfrag.com.139 > target.net.139: udp 28 (frag 242:36@0+)
evilfrag.com.139 > target.net.139: (frag 242:4@24)
```

We see two IP packets with Frag 242, the fragment ID of an IP packet that was broken up.

Send Fragment 1 with offset = 0, payload size N, **More Fragments** bit on

Send Fragment 2 with offset + payload size < N, **More Fragments** bit off ➔ Fragment 2 fits entirely inside Fragment 1

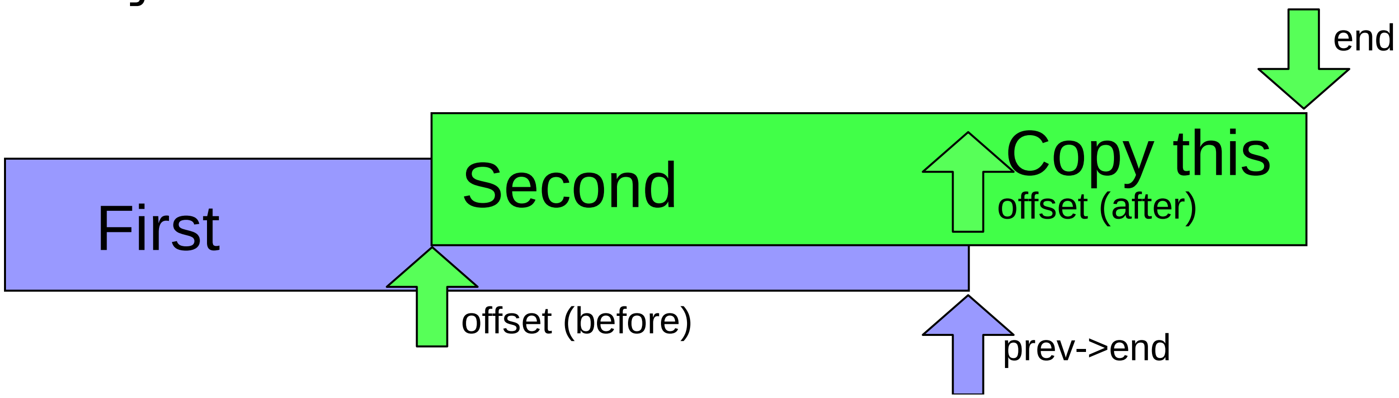


OS has to reassemble packets …

Teardrop: Mechanism
Deep in the protocol implementation

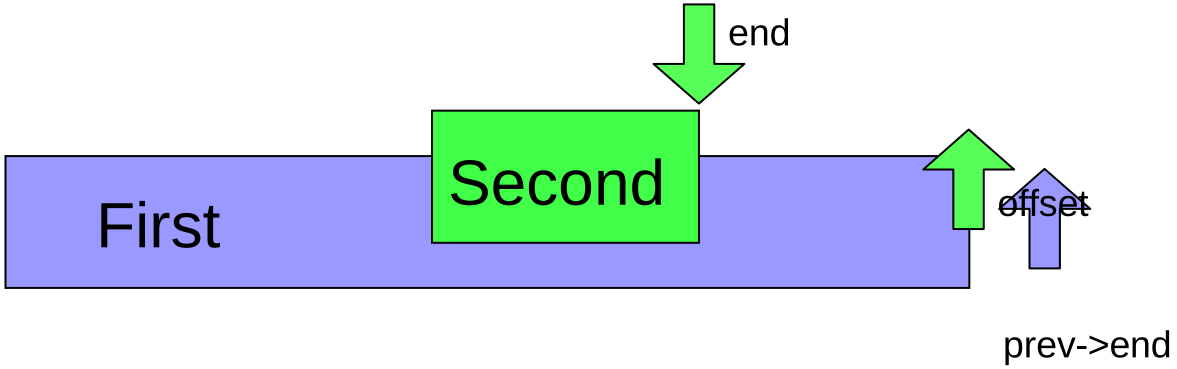
In ip_fragment.c@531 (ca. 1997)

```
if (prev != NULL && offset < prev->end)
// if there are overlapping fragments
   {
          i = prev->end - offset;
          offset += i;    /* ptr into datagram */
          ptr += i;       /* ptr into fragment data
*/
         //advance to the end of the previous
fragment
   }
```



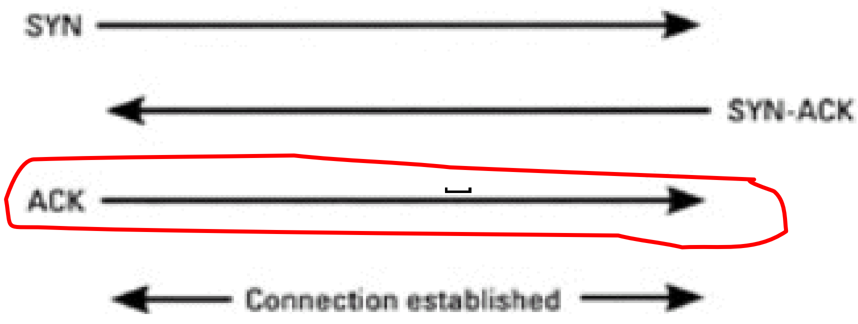
Teardrop: Result

- Offset now points outside of the second datagram's buffer!
- Program calculates the number of bytes to copy
  - fp->len = end - offset;
  - very large unsigned number… What can we do with this?
  - We’ll see more when we get to software vulnerabilities

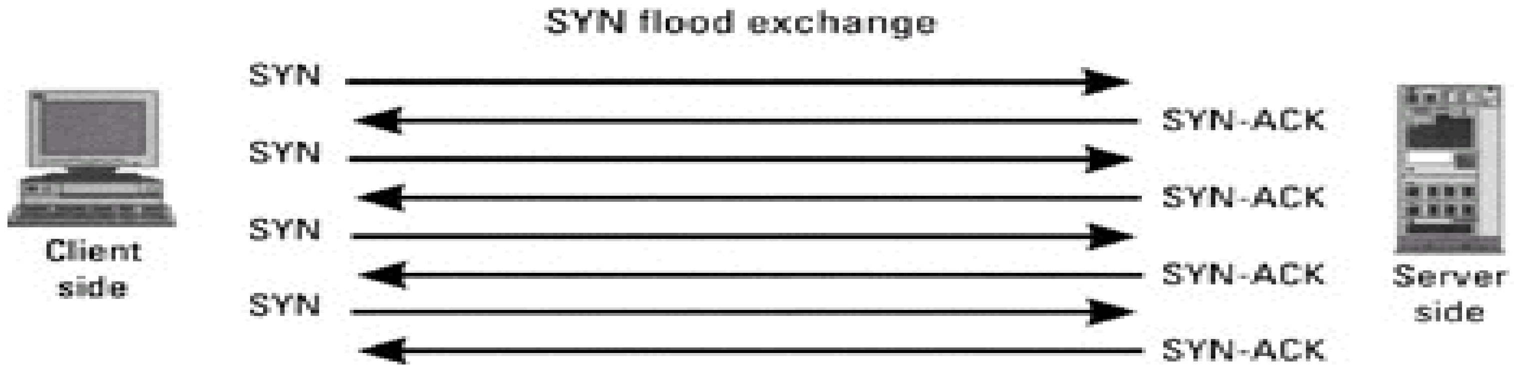


Protocol Attack: **SYN Flood**

Attack: Initiate, but do not finish 3-way handshake - don't send final Ack



Every TCP connection establishment requires an allocation of significant memory resources.



Goal: By sending overdosed connection requests with spoofed source addresses to the victim, an attacker can disable all successive connection establishment attempts including those of legitimate users.

Bandwidth Attack: **Smurf**

- **One level of indirection**
- **Attack**: Ping a broadcast address, with the (spoofed) IP of a victim as source address

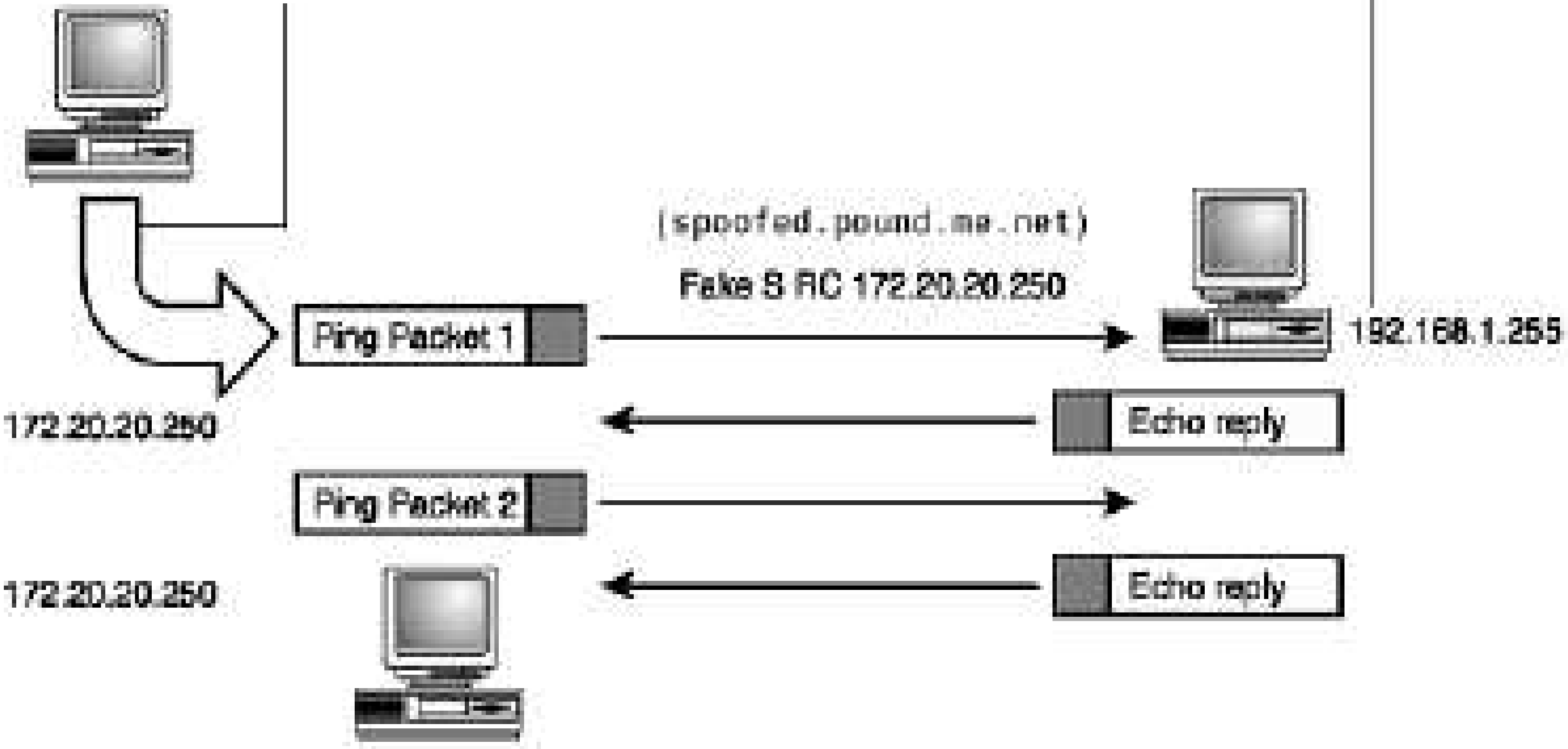
  All hosts on the network respond to the victim
- **Goal**: Overwhelm the victim

- **Mechanism**: Reflection, IP spoofing and protocol vulnerability
  - implementation can be “patched” by violating the protocol specification, to ignore pings to broadcast addresses

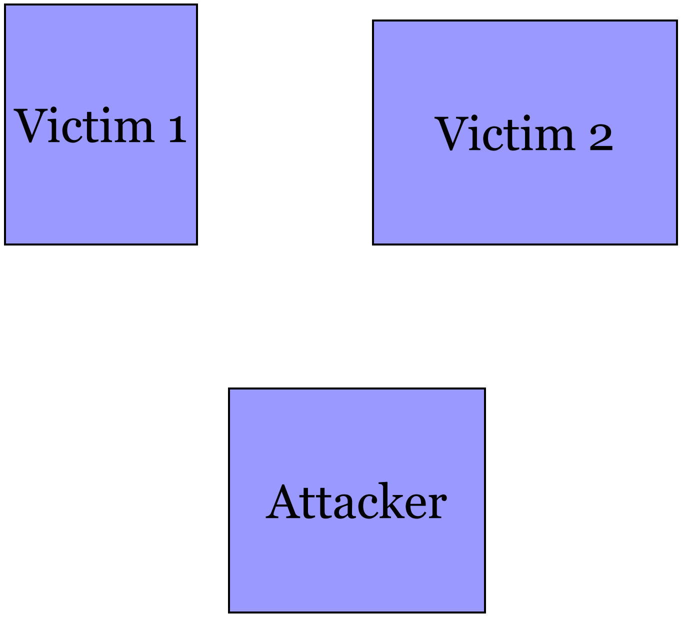
Smurf: Overview

Echo request with spoofed source address 172.20.20.250 to 192.168.1.255 (broadcast address of subnet 192.168.1.x)

All live hosts at subnet 192.168.1.x respond with echo reply .. to 172.20.20.50



Bandwidth Attack: **UDP Ping-Pong**



Victim 1

Victim 2

Attacker

- **‘Bootstrapping’ SMURF**
- **Attack**: Spoof a packet from host A's `chargen` service to host B's echo service
  - `chargen` service replies with a UDP packet to any incoming packet
- **Goal**: Computers keep replying to each other as fast as they can

```
08:08:16.155354 spoofed.pound.me.net.echo > 172.31.203.17.chargen: udp
08:21:48.891451 spoofed.pound.me.net.echo > 192.168.14.50.chargen: udp
08:25:12.968929 spoofed.pound.me.net.echo > 192.168.102.3.chargen: udp
08:42:22.605428 spoofed.pound.me.net.echo > 192.168.18.28.chargen: udp
```

Evolution of DoS Attacks: DDoS

- DDoS: Distributed Denial Of Service
- Attack against bandwidth and/or resources (like before) **using two (or more) levels of indirection!**

Attacker: used to coordinate attack

Handler: controls subservient computers

Agents: Actually do the attack

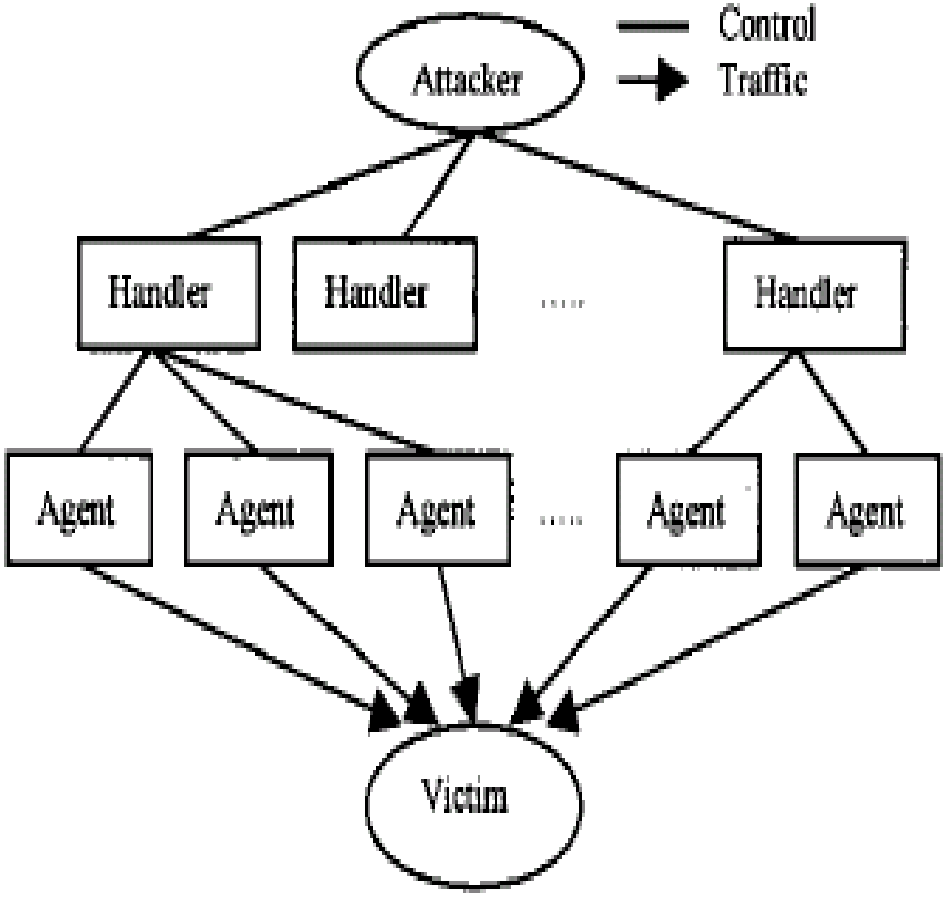


Figure 1- Architecture of DDoS attacks

DDoS examples

TRINOO

Sends UDP floods to random destination port numbers on victim

TFN

Sends UDP flood, TCP SYN Flood, ICMP Echo Flood, or a SMURF Attack

Master communicates to daemon using ICMP echo reply, changes IP identification number and payload of ICMP echo reply to identify type of attack to launch

TFN2k

First DDOS for windows. Communication between master and agents can be encrypted over TCP, UDP, or ICMP with no identifying ports

STACHELDRAHT

Combination of Trinoo and TFN

Authority on analysis of DDoS is Diettrich at University of Washington

`http://staff.washington.edu/dittrich/misc/ddos`

DDoS: Trinoo (1998/1999)

- All C source (Linux, Solaris, Irix)
- UDP packet flooder, no source address forgery
- Full control features, control traffic on TCP and UDP
- Menu operated
- Agent passwords are sent in plain text form (not encrypted)

DDoS: Tribe Flood Network (1999)

- Limited control features, control traffic via ICMP Echo Reply
- UDP packet flood ("trinoo emulation"), but also TCP SYN flood, ICMP Echo flood
- IP Spoofing: Either randomizes all 32 bits of source address, or just last 8 bits
- Handler keeps track of its agents in “Blowfish” (encryption algorithm) encrypted file

DDoS: TFN2k (1999)

- Improved version of TFN
  - Runs on *nix, Windows NT
  - Encrypted control traffic uses UDP, TCP, or ICMP
  - Same source address forgery features as TFN
- Agent can randomly alternate between the TFN types of attack
- Agent is completely silent - handler sends the same command several times, hoping that agent will receive at least one
- IP Spoofing: Random source IP address and port number
- Decoy packets (sent to non-target networks)

DDoS: Stacheldraht 2.6 (2000)

- Evolved TFN/TFN2k
- Several levels of protection:
  - □ Hard-coded password in client
  - □ Password is needed to take control over handler
  - □ Encrypted traffic: TCP for communication between client and handler, and ICMP_ECHOREPLY for communication between handler and agent
- Same basic attacks as TFN2k, adds TCP ACK flood ("stream") and TCP NULL (no flags) flood
- **New**: Adds "smurf" attack w/16,702 amplifiers (already `inet_aton()`d for speed!
- **New**: Automated update of agents (via rpc)

More attack modes

- Attacks against address translation services
  - DNS cache poisoning
    - Tricks a DNS server into believing it has received authentic URL->IP translation information
    - Difficult and serious problem (a form of semantic hacking)
    - http://www.lurhq.com/cachepoisoning.html

Summary

- The 'glue' of the Internet (TCP/IP protocol and associated services like DNS) was predicated towards communication (and limited recovery from random errors, i.e. noise)
  - Security (confidentiality, authentication, recovery from deliberate errors, i.e attacks) was an **afterthought**
- As such, **strong assumptions were made** while designing, implementing and running the protocols

➔ This makes attacks against the TCP/IP protocol and implementation, as well as network services such as DNS, realtively easy and feasible

Acknowledgments

Some materials/slides from

Pascale Meunier, Purdue U. (CERIAS)

Dave Dittrich, U. Washington

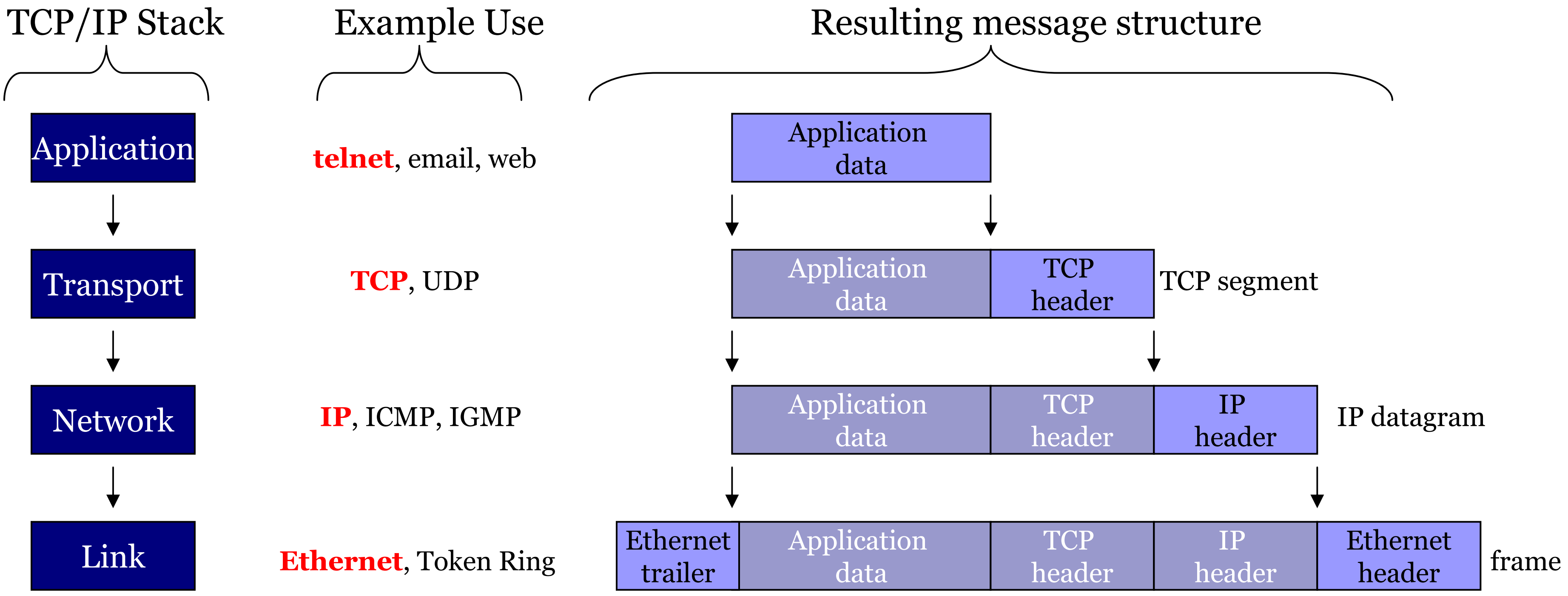
Steven Northcutt "Network Intrusion Detection", 3rd Edition

For Friday

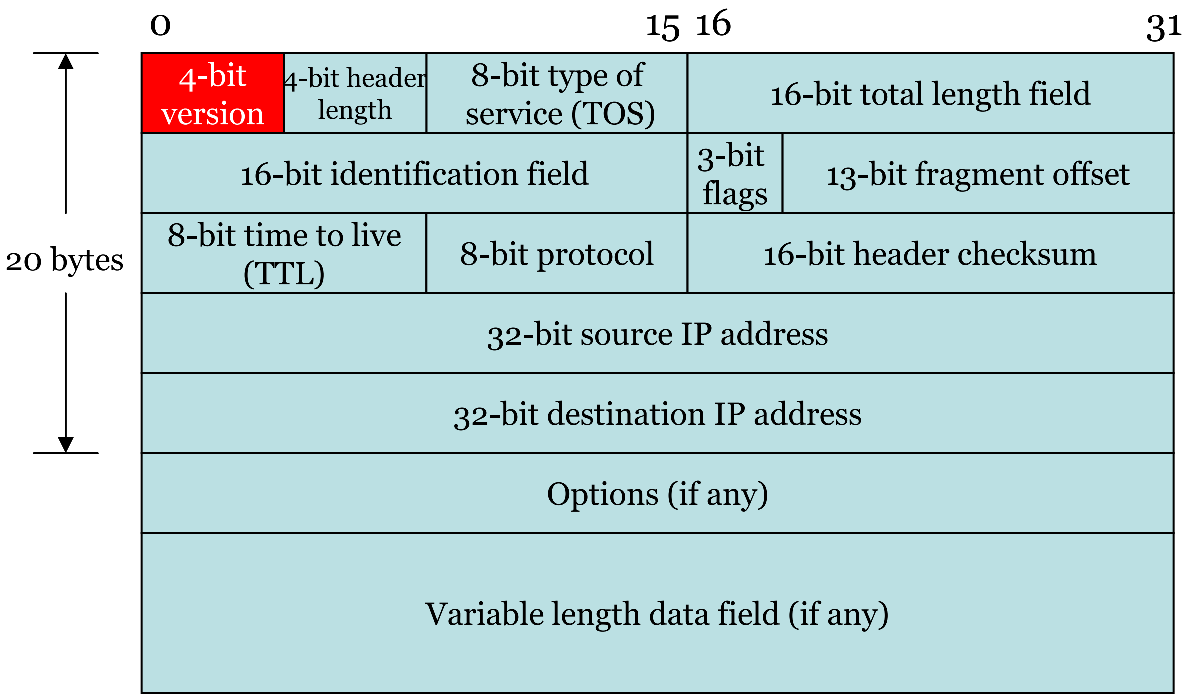
- Review notes and look at actual traces in attacks (posted as additional material in lecture notes)

Additional material

Review: Protocol Stack



Packet construction from recipient's point of view (up the protocol stack)



0 15 16 31

20 bytes

4-bit version

4-bit header length

8-bit type of service (TOS)

16-bit total length field

16-bit identification field

3-bit flags

13-bit fragment offset

8-bit time to live (TTL)

8-bit protocol

16-bit header checksum

32-bit source IP address

32-bit destination IP address

Options (if any)

Variable length data field (if any)

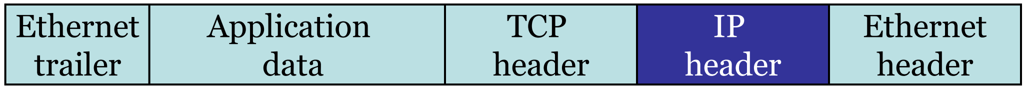
Defines the version of IP being used.

Normal: 4 (current) and 6 (emerging).

Abnormal: any values other than 4 or 6.

4

version 4



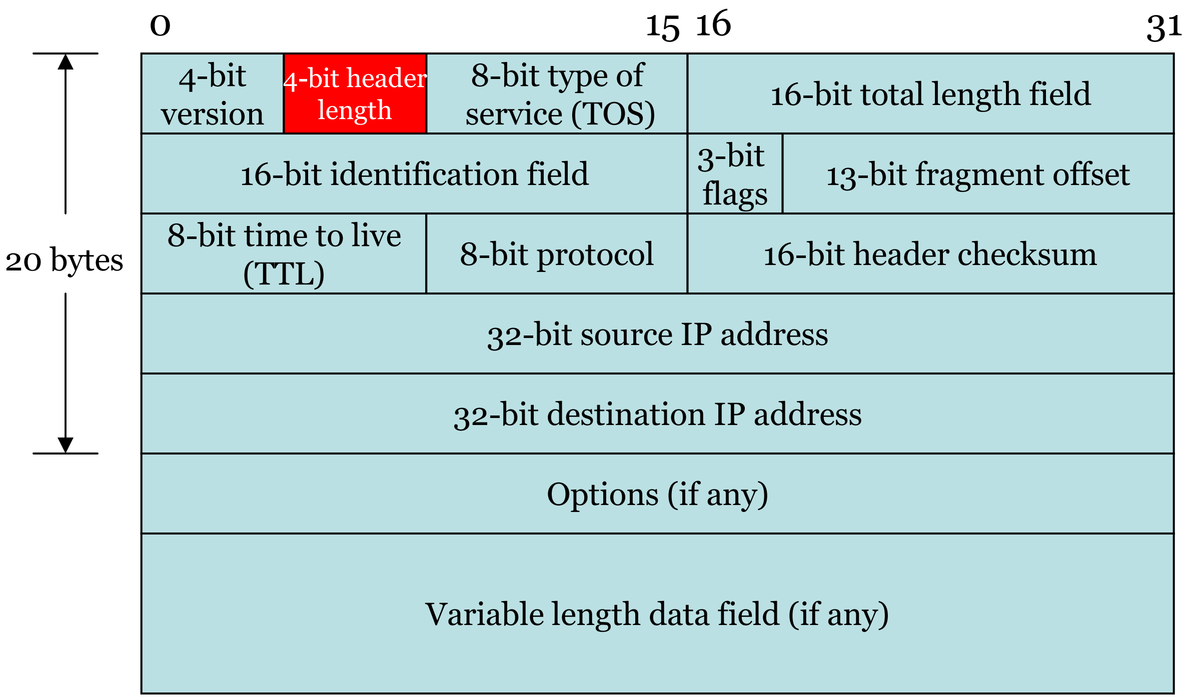
Ethernet trailer

Application data

TCP header

IP header

Ethernet header



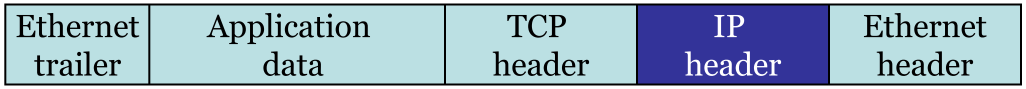
Represents the number of 32-bit (4-byte) words in the header. The minimum value is 5 (20 bytes) and the maximum value is f (60 bytes)

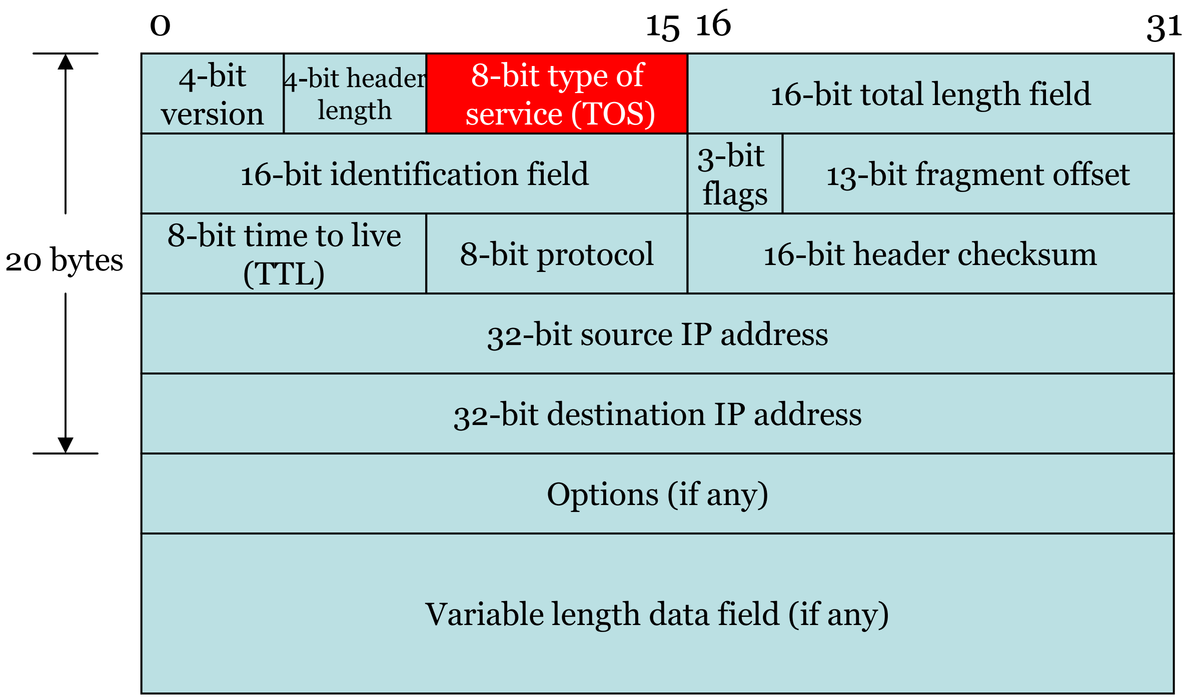
Normal: 5 (a 20 byte length), no options

Abnormal: values 0-4. values 0-f when not followed by the corresponding amount of data.

45

a 20 byte header





0 15 16 31

20 bytes

4-bit version | 4-bit header length | 8-bit type of service (TOS) | 16-bit total length field

16-bit identification field | 3-bit flags | 13-bit fragment offset

8-bit time to live (TTL) | 8-bit protocol | 16-bit header checksum

32-bit source IP address

32-bit destination IP address

Options (if any)

Variable length data field (if any)

Options for special case handling of datagrams.

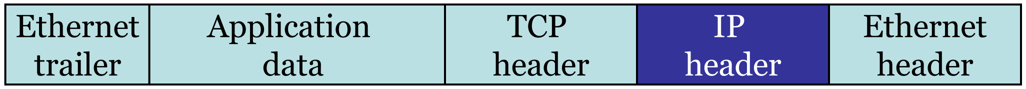
Normal:

normal service	0x00
minimize delay	0x10
maximize throughput	0x08
maximize reliability	0x04
minimize monetary cost	0x02

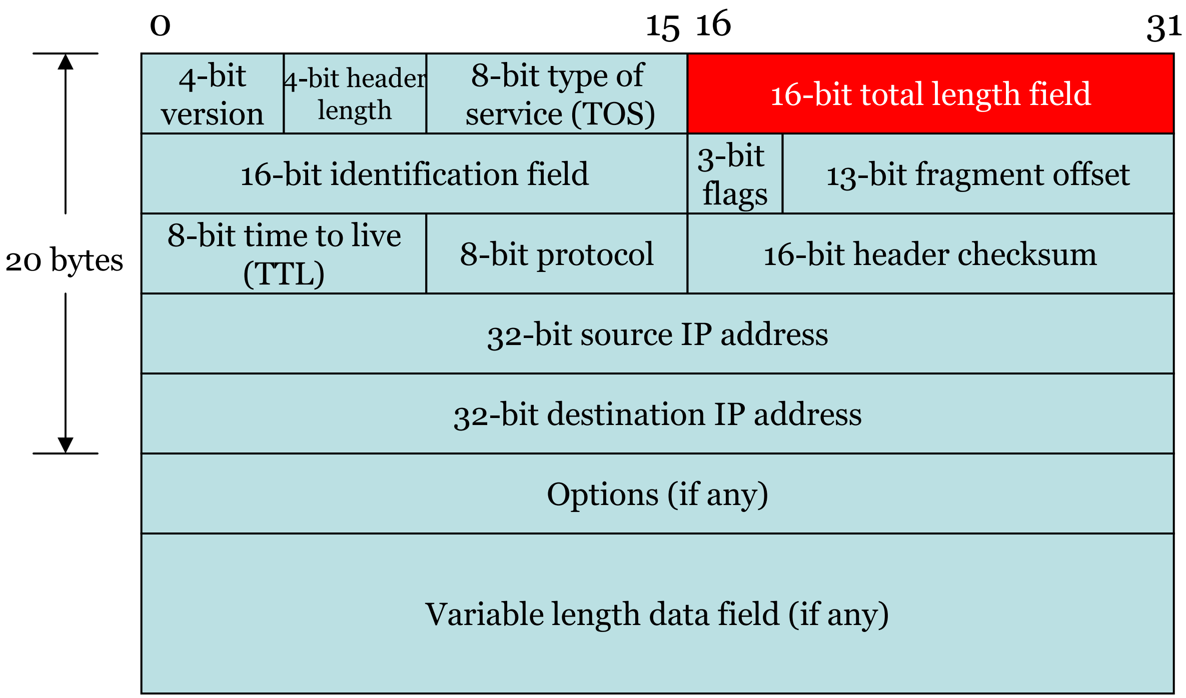
Abnormal: values other than the 5 shown above (there can be only one turned on at a time)

```
45 10
```

minimize delay



Ethernet trailer | Application data | TCP header | IP header | Ethernet header



0 15 16 31

20 bytes

4-bit version | 4-bit header length | 8-bit type of service (TOS) | 16-bit total length field

16-bit identification field | 3-bit flags | 13-bit fragment offset

8-bit time to live (TTL) | 8-bit protocol | 16-bit header checksum

32-bit source IP address

32-bit destination IP address

Options (if any)

Variable length data field (if any)

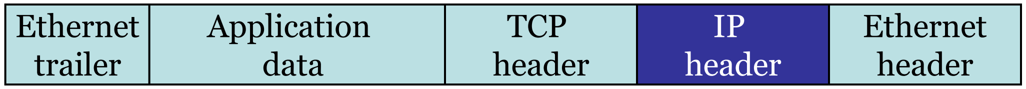
Total length of the datagram including IP header, transport layer header, and any data.

Normal: minimum length is 0x0014 (20 bytes) and maximum is 0xffff (65535). The maximum is actually limited by the link's MTU, which is 1500 on an Ethernet.

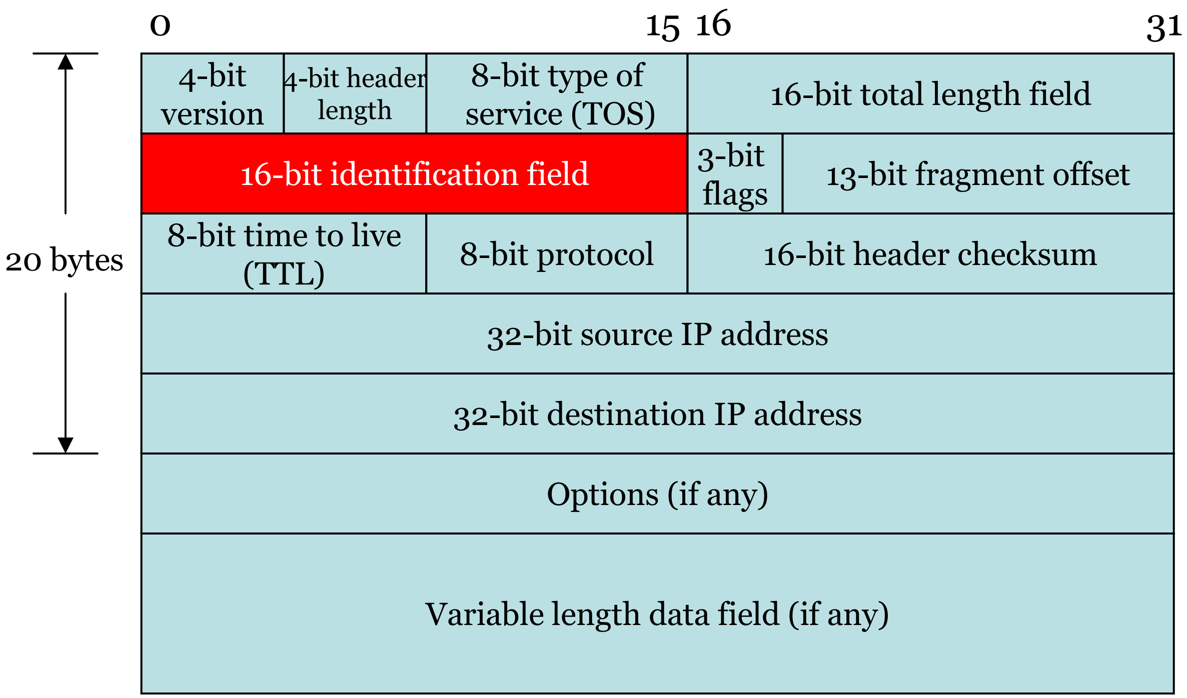
Abnormal: a value inconsistent with the actual number of bytes in the message.
A value larger than the networks path MTU thus causing fragmentation.

```
45 10 00 3c
```

a 60 byte total length



Ethernet trailer | Application data | TCP header | IP header | Ethernet header



0 15 16 31

20 bytes

4-bit version | 4-bit header length | 8-bit type of service (TOS) | 16-bit total length field

16-bit identification field | 3-bit flags | 13-bit fragment offset

8-bit time to live (TTL) | 8-bit protocol | 16-bit header checksum

32-bit source IP address

32-bit destination IP address

Options (if any)

Variable length data field (if any)

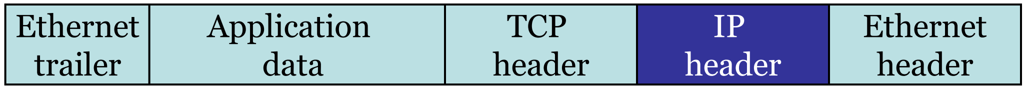
Uniquely identifies each datagram sent by a host. It normally increments by one each time a datagram is sent.

Normal: integers between 1-65535

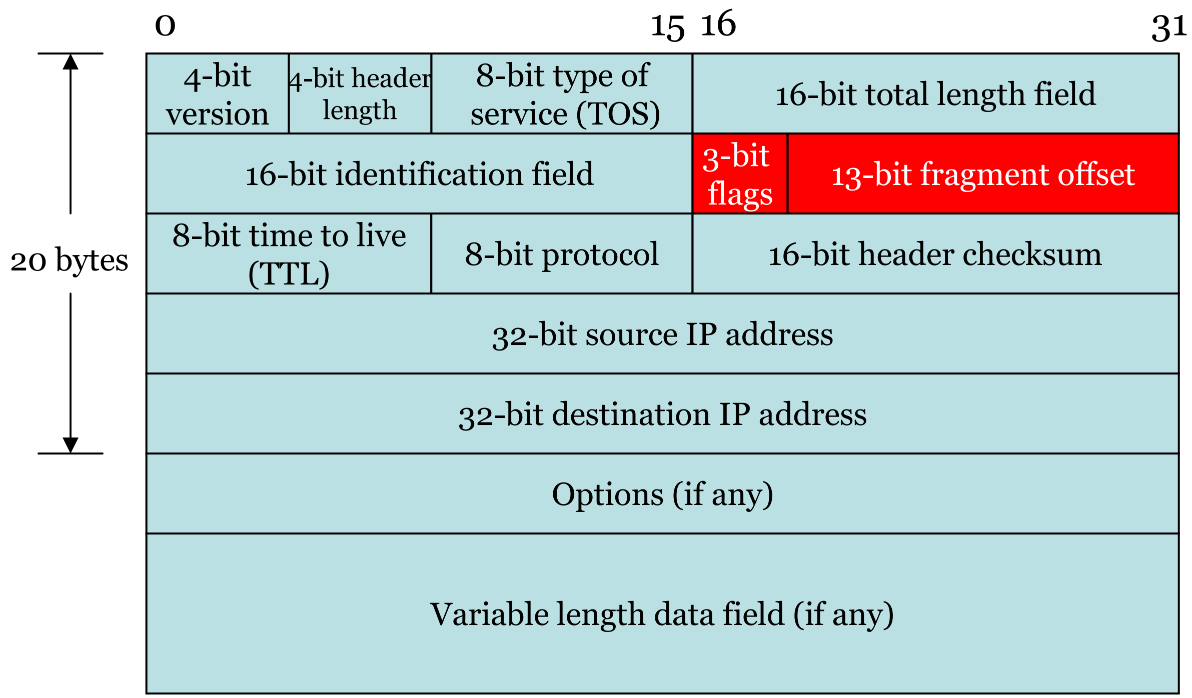
Abnormal: repeated datagrams from a single source using the same id number (no frags and no timeout & retransmission). Datagrams from 1+ sources using the same ID suggesting it is hard coded into an exploit (high false posItives)

```
45 10 00 3c 27 a7
```

IP ID 10151



Ethernet trailer | Application data | TCP header | IP header | Ethernet header



0 15 16 31

20 bytes

4-bit version | 4-bit header length | 8-bit type of service (TOS) | 16-bit total length field

16-bit identification field | 3-bit flags | 13-bit fragment offset

8-bit time to live (TTL) | 8-bit protocol | 16-bit header checksum

32-bit source IP address

32-bit destination IP address

Options (if any)

Variable length data field (if any)

Provides the information IP needs to re-order fragmented messages.

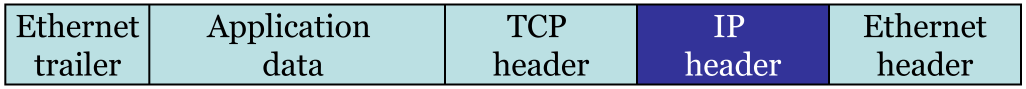
Normal: 0x4 sets don’t fragment (DF) bit. 0x2 sets more fragments (MF) bit.

MF bit	Frag. Offset	Meaning
Not set	zero	packet not fragmented
Set	zero	first fragment
Set	non-zero	middle fragment
Not set	non-zero	last fragment

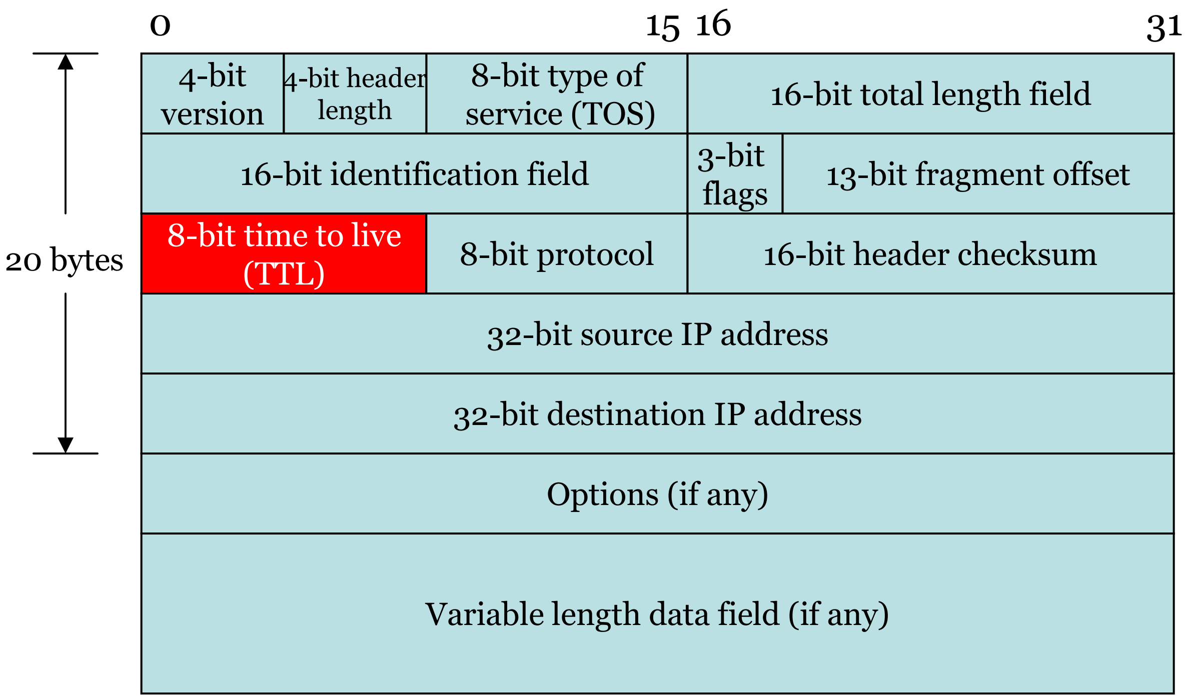
Abnormal: mismatched, overlapping, out of spec, or gapping fragment offsets.

```
45 10 00 3c 27 a7 40 00
```

don’t fragment



Ethernet trailer | Application data | TCP header | IP header | Ethernet header



0 15 16 31

20 bytes

4-bit version | 4-bit header length | 8-bit type of service (TOS) | 16-bit total length field

16-bit identification field | 3-bit flags | 13-bit fragment offset

8-bit time to live (TTL) | 8-bit protocol | 16-bit header checksum

32-bit source IP address

32-bit destination IP address

Options (if any)

Variable length data field (if any)

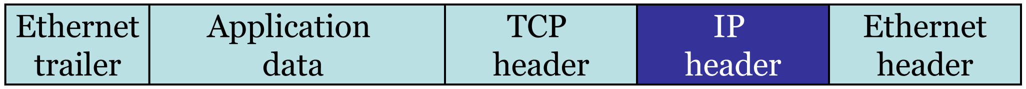
Initialized to some value and decremented by one by every router that handles the datagram. When the field reaches 0 it is thrown away, effectively limiting the lifetime of the datagram (preventing an infinite loop)

Normal: at least 64 (initially), 128, 255

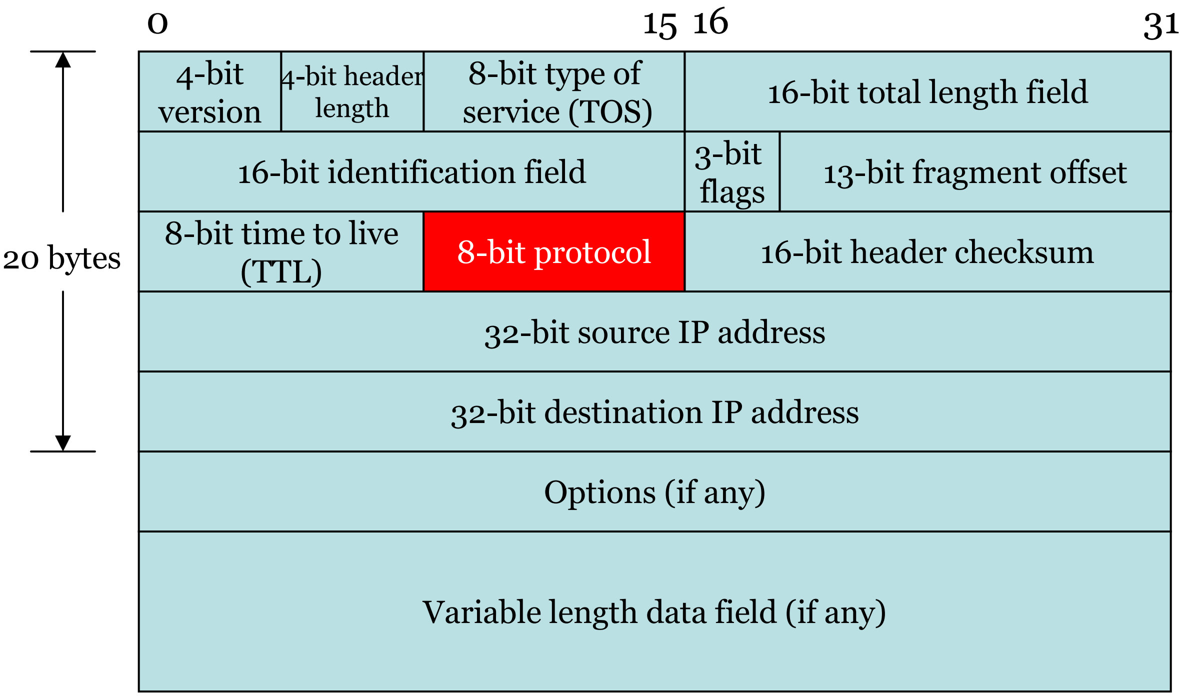
Abnormal: contextual.

```
45 10 00 3c 27 a7 40 00 40
```

64 hop TTL



Ethernet trailer | Application data | TCP header | IP header | Ethernet header



0 15 16 31

20 bytes

4-bit version

4-bit header length

8-bit type of service (TOS)

16-bit total length field

16-bit identification field

3-bit flags

13-bit fragment offset

8-bit time to live (TTL)

8-bit protocol

16-bit header checksum

32-bit source IP address

32-bit destination IP address

Options (if any)

Variable length data field (if any)

Which protocol is encapsulated in IP.

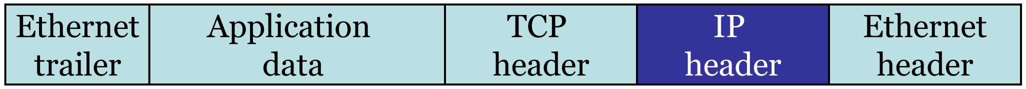
Normal: (see /etc/protocols)

ICMP	0x01
IGMP	0x02
IP	0x04
TCP	0x06
UDP	0x11

Abnormal: Values 0x88 – 0xfe are unassigned and 0xff is reserved. Others may or may not be valid depending on which protocol a network is intended to use.

```
45 10 00 3c 27 a7 40 00 40 06
```

TCP data follows the IP header



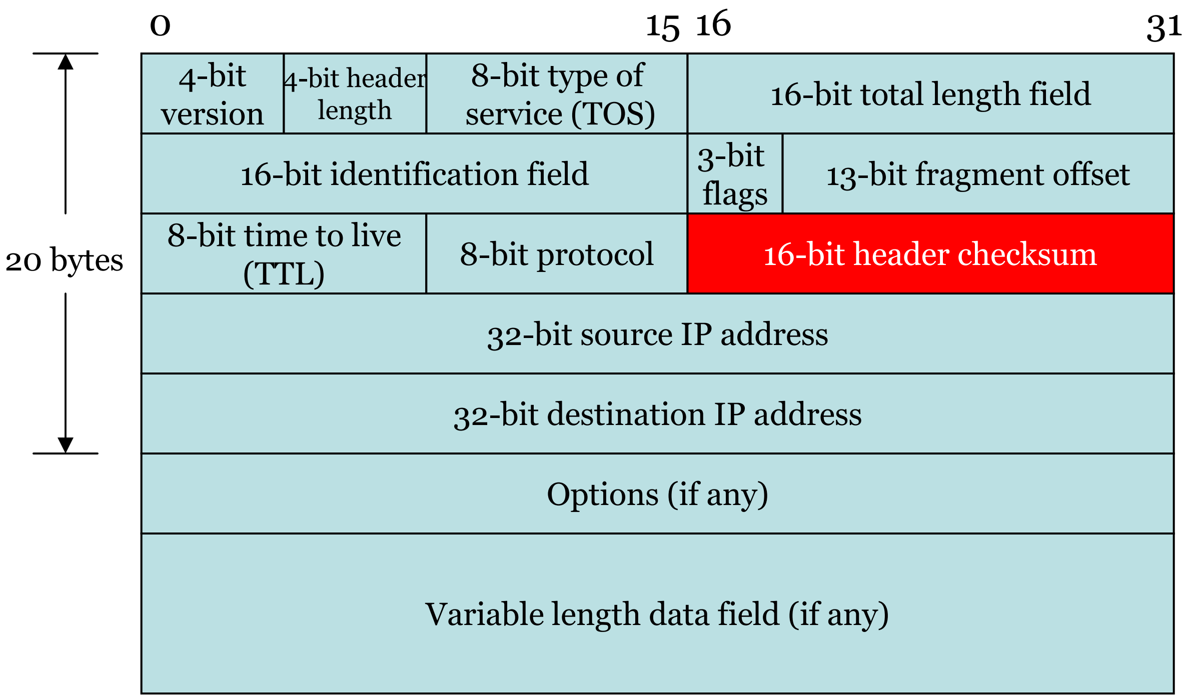
Ethernet trailer

Application data

TCP header

IP header

Ethernet header



0 15 16 31

20 bytes

4-bit version | 4-bit header length | 8-bit type of service (TOS) | 16-bit total length field

16-bit identification field | 3-bit flags | 13-bit fragment offset

8-bit time to live (TTL) | 8-bit protocol | 16-bit header checksum

32-bit source IP address

32-bit destination IP address

Options (if any)

Variable length data field (if any)

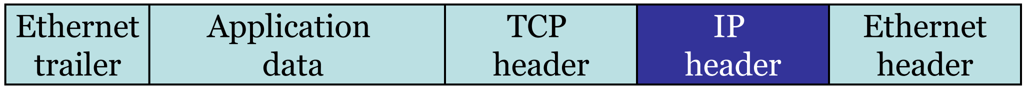
Calculated over IP header only – it does not cover any data that follows the header because UDP, TCP, ICMP, and IGMP all have a checksum of their own to cover their header and data.

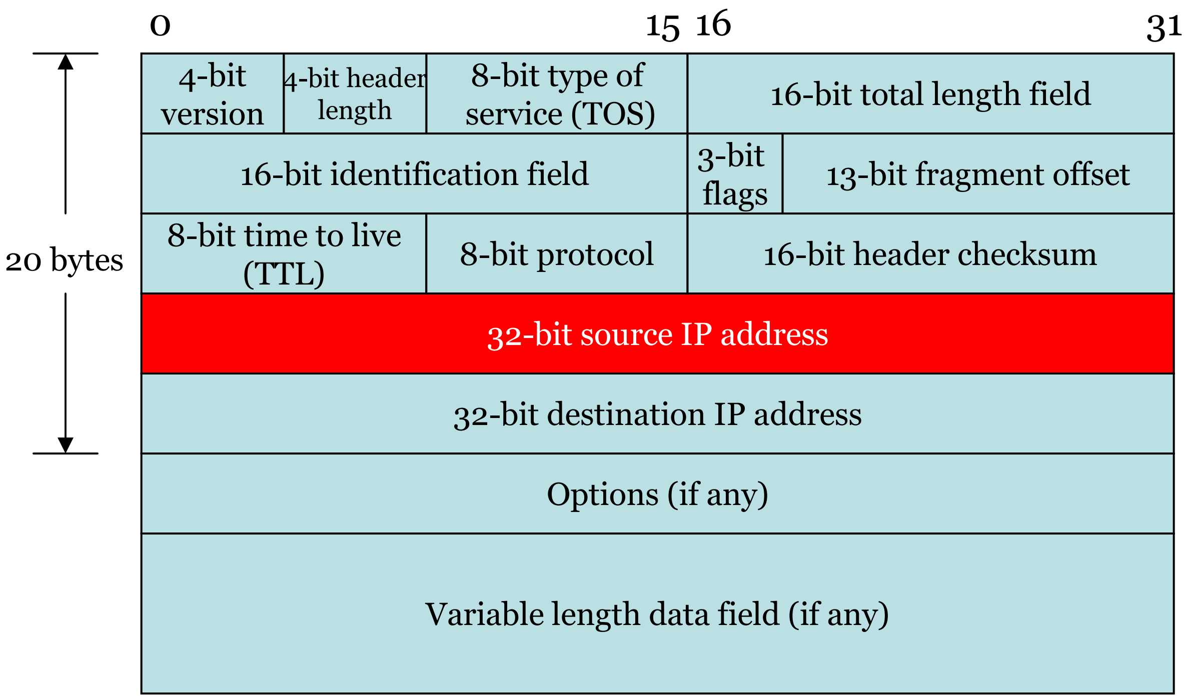
Normal: a correct checksum

Abnormal: contextual (errors in transmission do occur but not very often)

```
45 10 00 3c 27 a7 40 00 40 06 8f 56
```

checksum is 0x8f56 (dummy figures)





0 15 16 31

20 bytes

4-bit version | 4-bit header length | 8-bit type of service (TOS) | 16-bit total length field

16-bit identification field | 3-bit flags | 13-bit fragment offset

8-bit time to live (TTL) | 8-bit protocol | 16-bit header checksum

32-bit source IP address

32-bit destination IP address

Options (if any)

Variable length data field (if any)

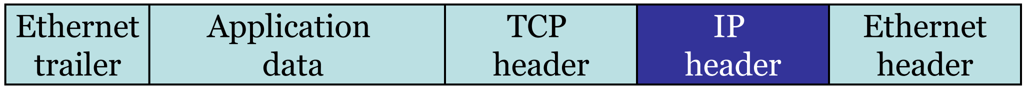
The alleged sender of the message.

Normal: contextual

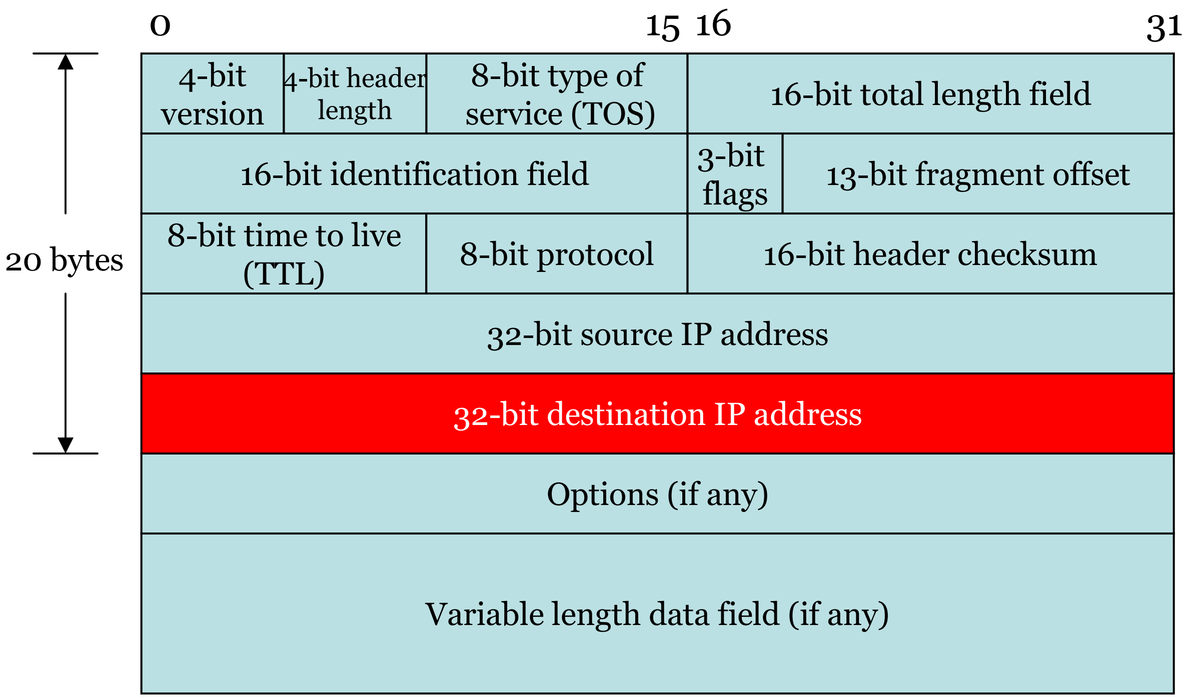
Abnormal: contextual. Non-routable, reserved, internal, or vacant addresses approaching an external interface should raise suspicion.

```
45 10 00 3c 27 a7 40 00 40 06 8f 56 c0 a8 01 01
```

source address is 0xc0a80101, which translates to 192.168.1.1



Ethernet trailer | Application data | TCP header | IP header | Ethernet header



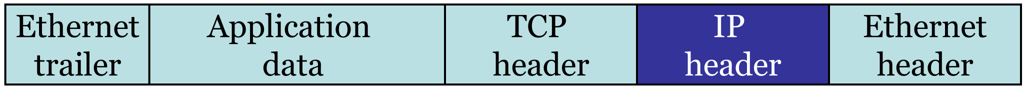
The IP address of the machine intended to receive this message.

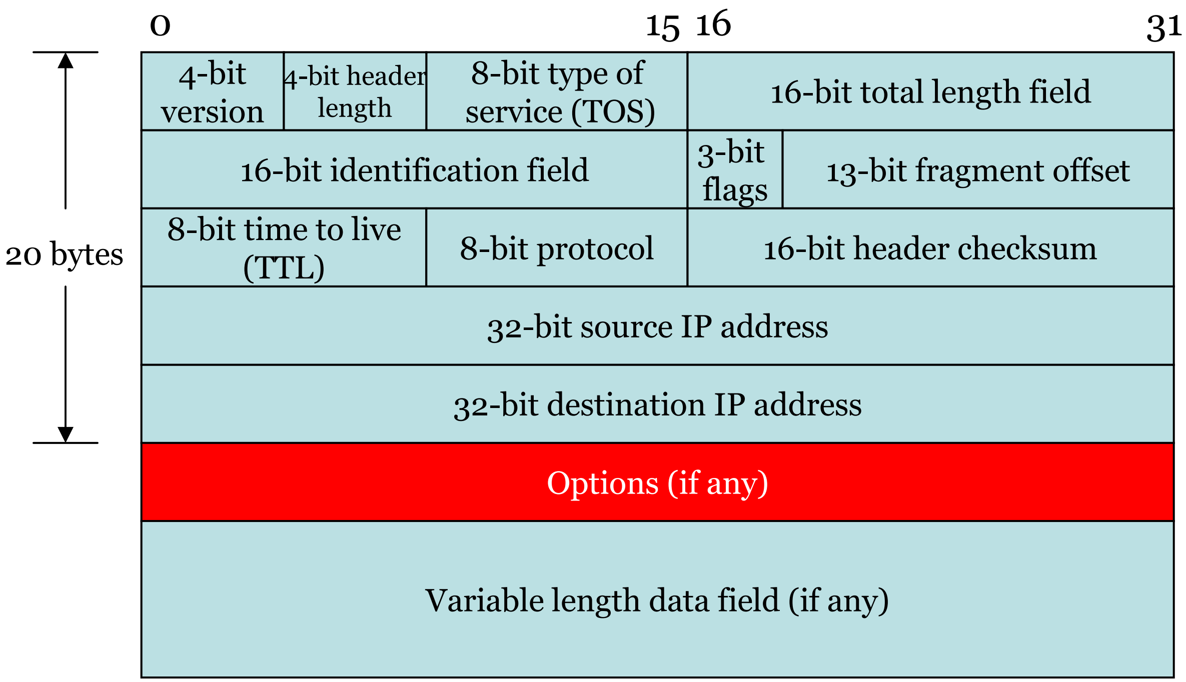
Normal: contextual.

Abnormal: contextual. messages
to a network's broadcast address from the outside (i.e. smurf), consecutive messages to all or part of a networks range of addresses.

```
45 10 00 3c 27 a7 40 00 40 06 8f 56 c0 a8 01 01
c0 a8 01 64
```

destination address is 0xc0a80164, which translates to 192.168.1.100





Options: record route, timestamp, loose source routing, strict source routing.

Normal: contextual. timestamp is most common.

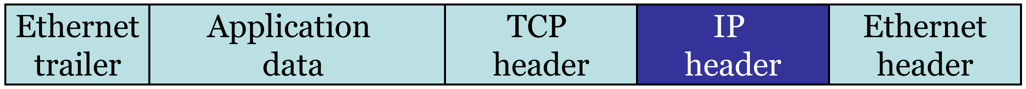
Abnormal: loose and strict source routing can be used by attackers to manually route packets (evasion technique)

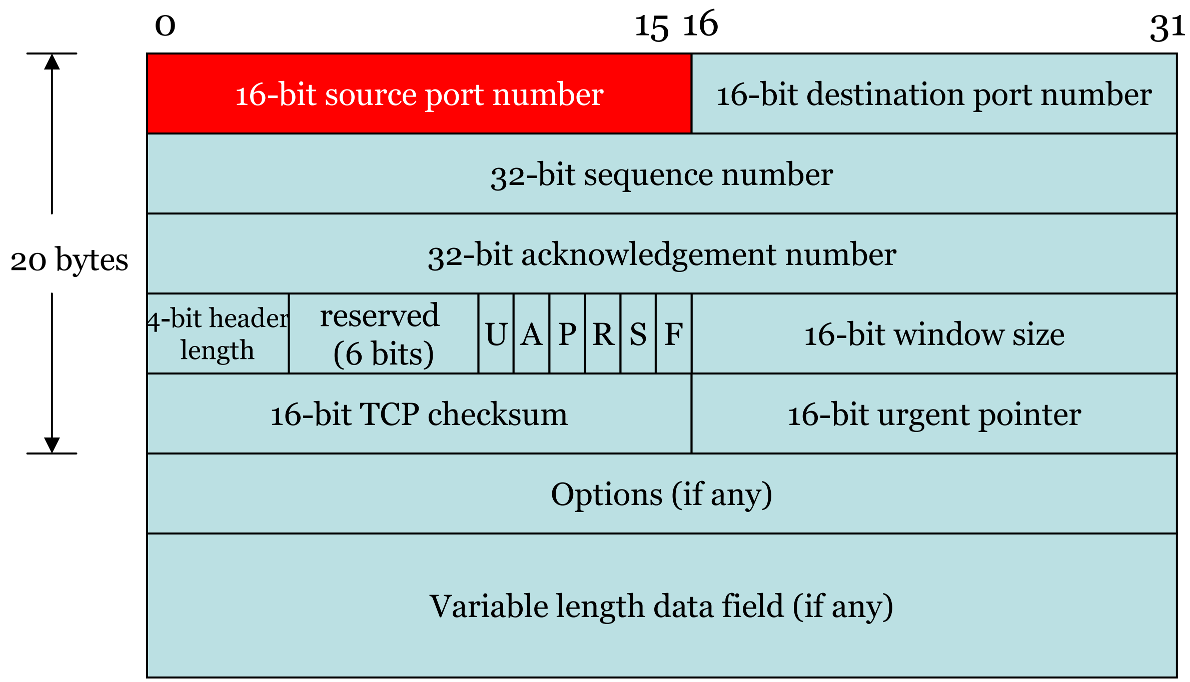
The variable length data field in this case is actually the start of the TCP header

How do we distinguish?

```
45 10 00 3c 27 a7 40 00 40 06 8f 56 c0 a8 01 01
c0 a8 01 64
```

no options





0 15 16 31

20 bytes

16-bit source port number

16-bit destination port number

32-bit sequence number

32-bit acknowledgement number

4-bit header length

reserved (6 bits)

U A P R S F

16-bit window size

16-bit TCP checksum

16-bit urgent pointer

Options (if any)

Variable length data field (if any)

The port through which the host will transmit this message.

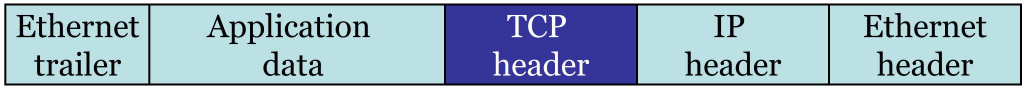
Normal: contextual. acting as server, the source port should be that of which the process is listening on. acting as client, the source port should be an ephemeral port above 1023.

Abnormal: datagrams to ports that are closed (trojan & service scanning), datagrams to open ports from untrusted sources.

See /etc/services

```
45 10 00 3c 27 a7 40 00 40 06 8f 56 c0 a8 01 01
c0 a8 01 64 80 ab
```

an ephemeral client port, 25894, sends the message



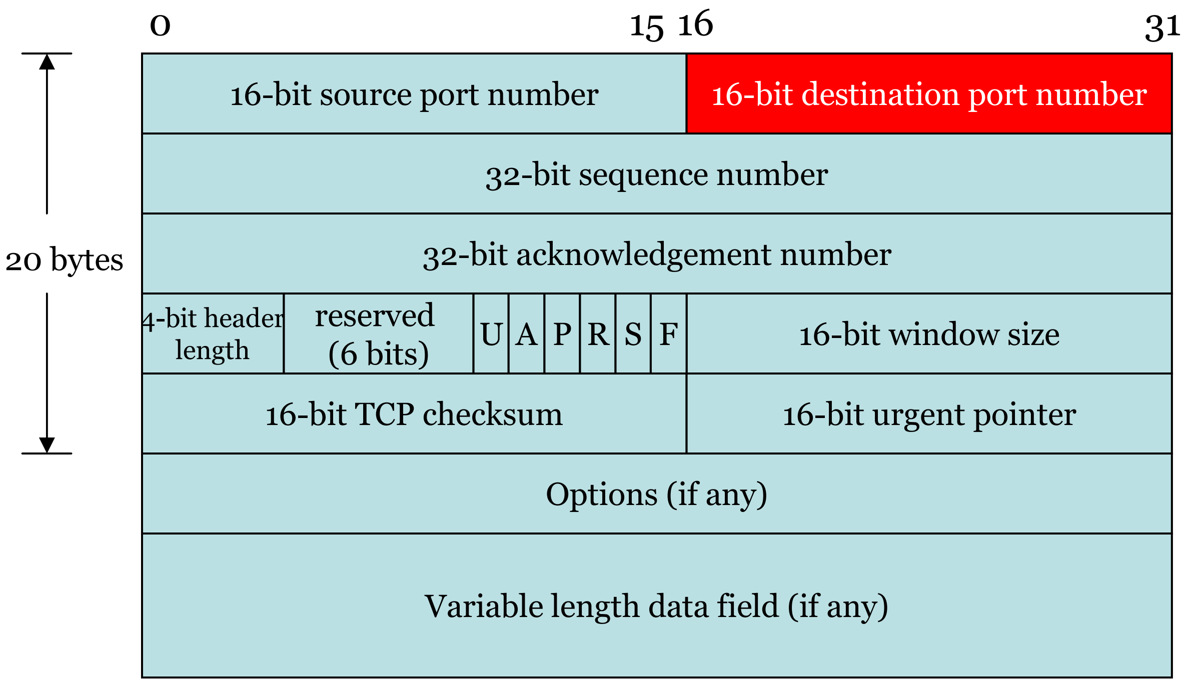
Ethernet trailer

Application data

TCP header

IP header

Ethernet header



0 15 16 31

20 bytes

16-bit source port number

16-bit destination port number

32-bit sequence number

32-bit acknowledgement number

4-bit header length

reserved (6 bits)

U A P R S F

16-bit window size

16-bit TCP checksum

16-bit urgent pointer

Options (if any)

Variable length data field (if any)

The port at which this message is directed.

Normal: contextual. acting as server, the destination port should be that of which the process is listening on. acting as client, it should be an ephemeral port above 1023.

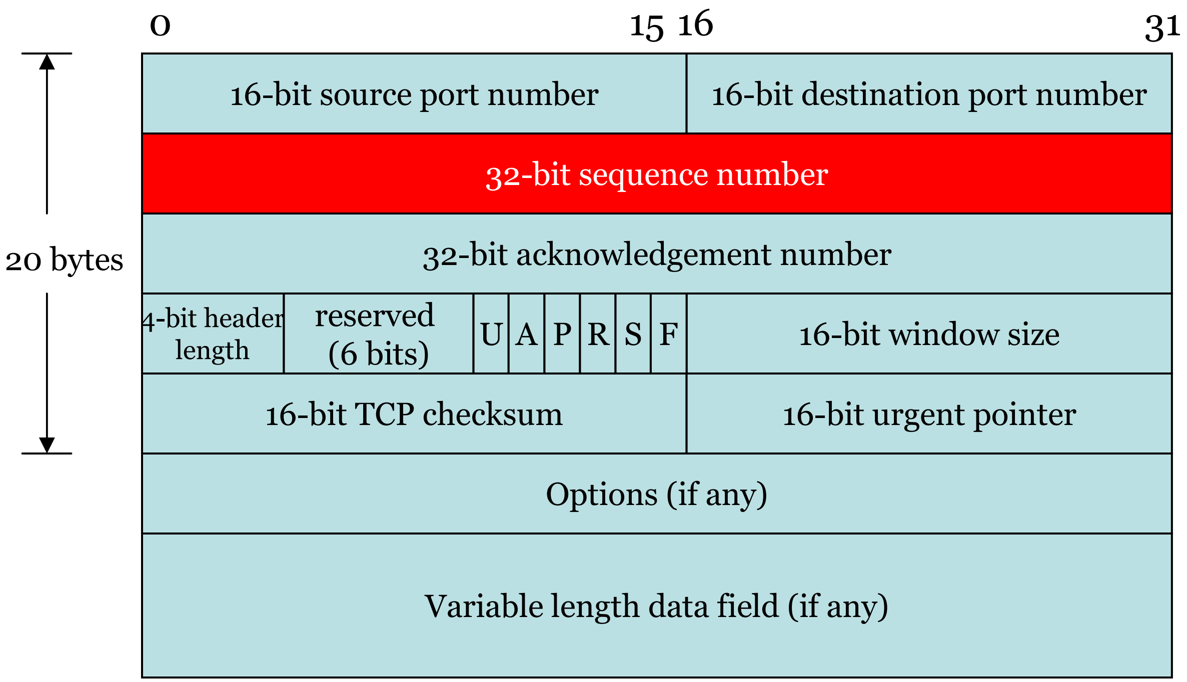
Abnormal: datagrams to ports that are closed (trojan & service scanning), datagrams to open ports from untrusted sources.

See /etc/services

```
45 10 00 3c 27 a7 40 00 40 06 8f 56 c0 a8 01 01
c0 a8 01 64 80 ab 00 17
```

port 23, the telnet server, will receive the message

Ethernet trailer	Application data	TCP header	IP header	Ethernet header



0 15 16 31

20 bytes

16-bit source port number | 16-bit destination port number

32-bit sequence number

32-bit acknowledgement number

4-bit header length | reserved (6 bits) | U | A | P | R | S | F | 16-bit window size

16-bit TCP checksum | 16-bit urgent pointer

Options (if any)

Variable length data field (if any)

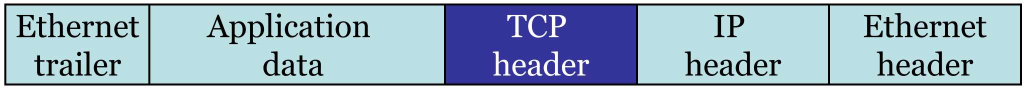
An initial sequence number (ISN) is chosen at random for each new TCP connection. Similar to how fragment offsets are used to reorder fragments into packets, sequence numbers are used to reorder packets into the data stream.

Normal: random ISN that increases by the number of bytes this host has sent since the beginning of the connection.

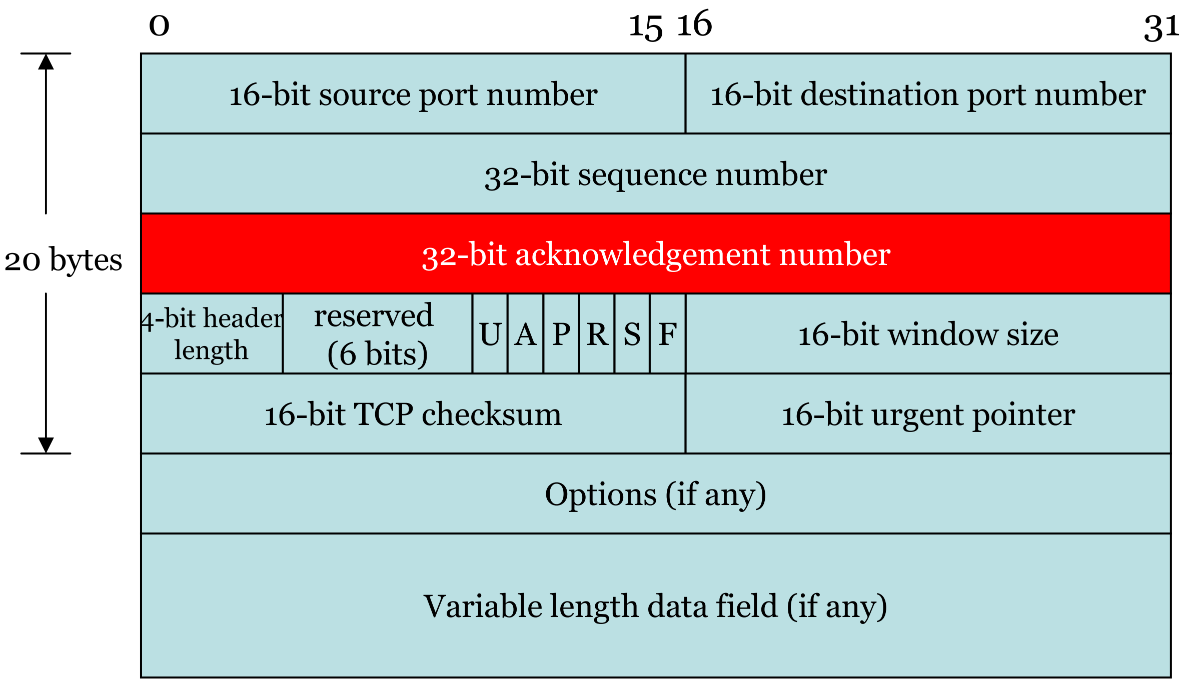
Abnormal: one of the values known to be coded into exploits. values that report in-accurate amounts of data have been sent.

```
45 10 00 3c 27 a7 40 00 40 06 8f 56 c0 a8 01 01
c0 a8 01 64 80 ab 00 17 a2 cf a9 10
```

sequence number is 2731518224



Ethernet trailer | Application data | TCP header | IP header | Ethernet header



0 … 15	16 … 31
16-bit source port number	16-bit destination port number
32-bit sequence number	
32-bit acknowledgement number	
4-bit header length | reserved (6 bits) | U | A | P | R | S | F	16-bit window size
16-bit TCP checksum	16-bit urgent pointer
Options (if any)	
Variable length data field (if any)	

20 bytes

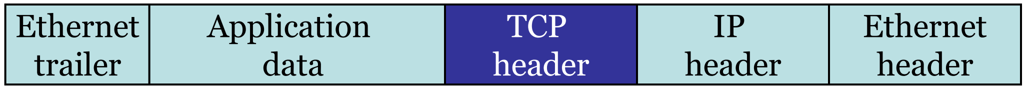
The acknowledgement number contains the next sequence number that the sender of the acknowledgement expects to receive.

Normal: AN = SN +1

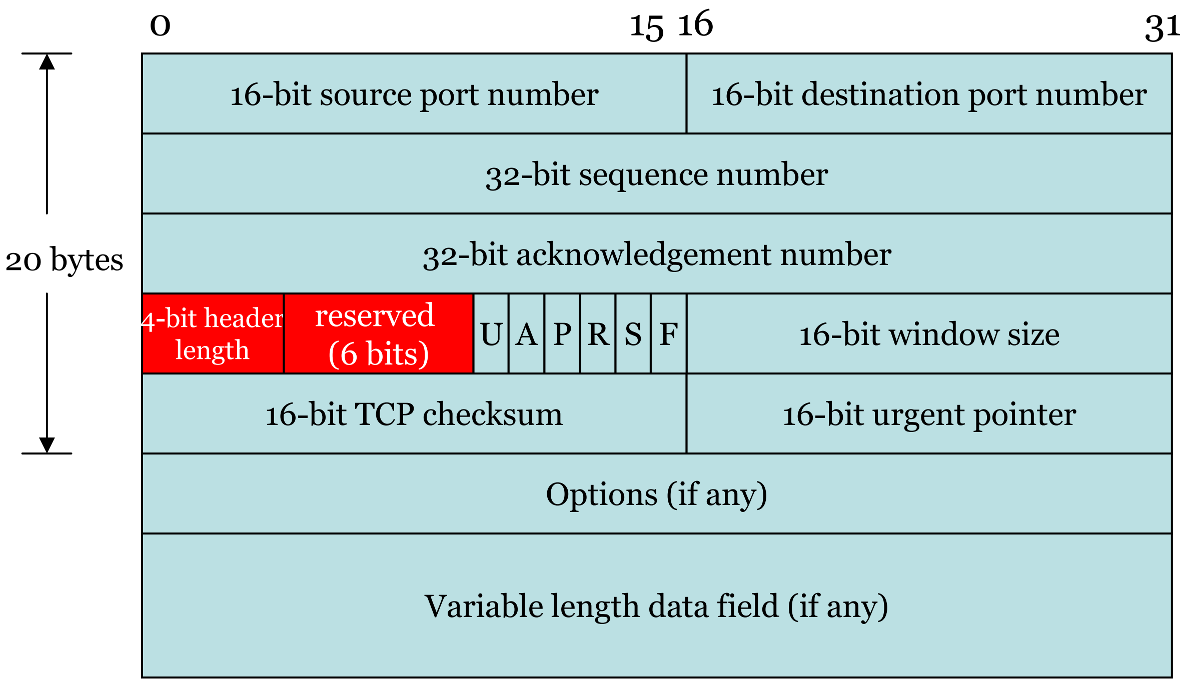
Abnormal: any non-zero value when the Ack flag is not set.

```
45 10 00 3c 27 a7 40 00 40 06 8f 56 c0 a8 01 01
c0 a8 01 64 80 ab 00 17 a2 cf a9 10 d5 6d b7 2d
```

acknowledgement number is 3580737325



Ethernet trailer	Application data	TCP header	IP header	Ethernet header



0 15 16 31

20 bytes

16-bit source port number | 16-bit destination port number

32-bit sequence number

32-bit acknowledgement number

4-bit header length | reserved (6 bits) | U | A | P | R | S | F | 16-bit window size

16-bit TCP checksum | 16-bit urgent pointer

Options (if any)

Variable length data field (if any)

Length of the TCP header.

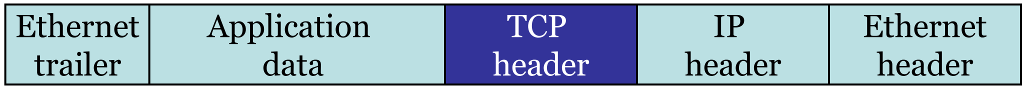
Normal: minimum is 0x5 (20 bytes). When options are set, the value can be 0xf (60 bytes) at maximum.

The 6-bit reserved field should always be zero.

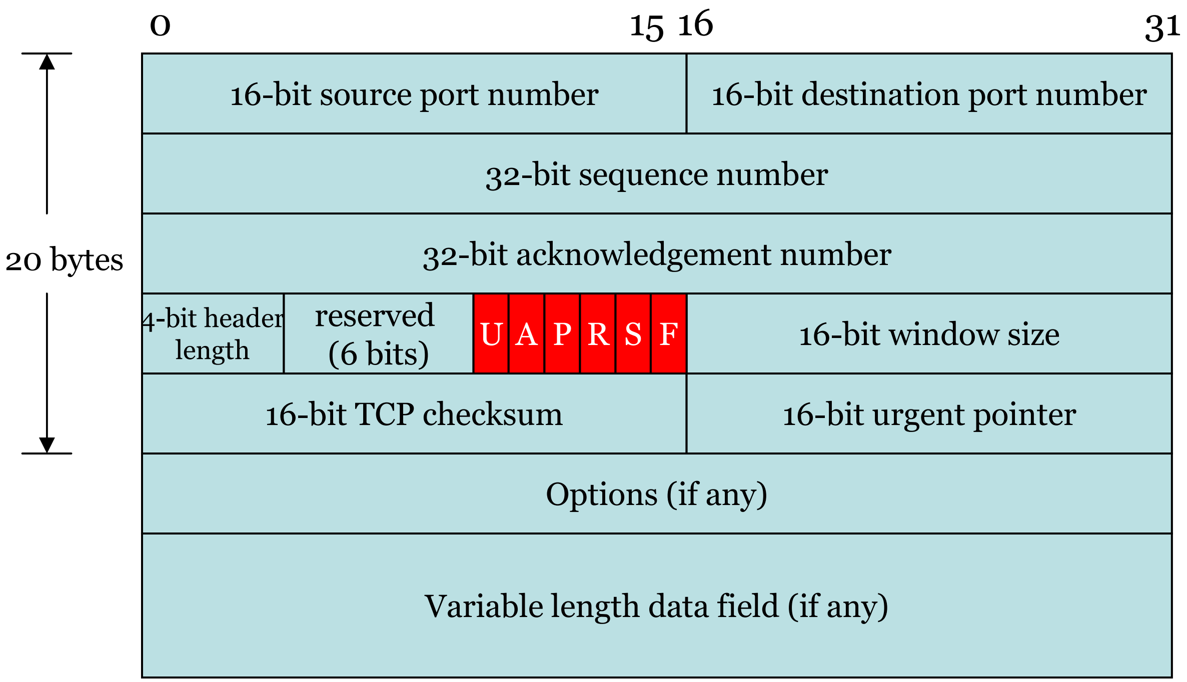
Abnormal: header length values inconsistent with the actual size. Non-zero reserved bit field.

```
45 10 00 3c 27 a7 40 00 40 06 8f 56 c0 a8 01 01
c0 a8 01 64 80 ab 00 17 a2 cf a9 10 d5 6d b7 2d
50
```

header length is 20 bytes, reserved bits are 0



Ethernet trailer | Application data | TCP header | IP header | Ethernet header



0 15 16 31

20 bytes

16-bit source port number | 16-bit destination port number

32-bit sequence number

32-bit acknowledgement number

4-bit header length | reserved (6 bits) | U A P R S F | 16-bit window size

16-bit TCP checksum | 16-bit urgent pointer

Options (if any)

Variable length data field (if any)

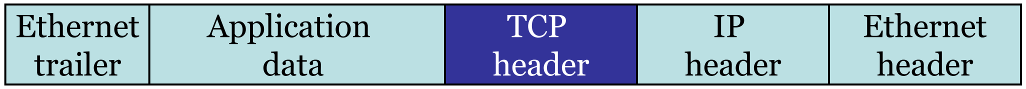
URG	the urgent pointer
ACK	the acknowledgement number is set
PSH	pass the data to the app. ASAP
RST	reset the connection
SYN	begin a connection
FIN	finished sending data

Normal: contextual. Possibly valid combinations: S, SA, A, R, RA, F, FA, FPA, UA, PA.

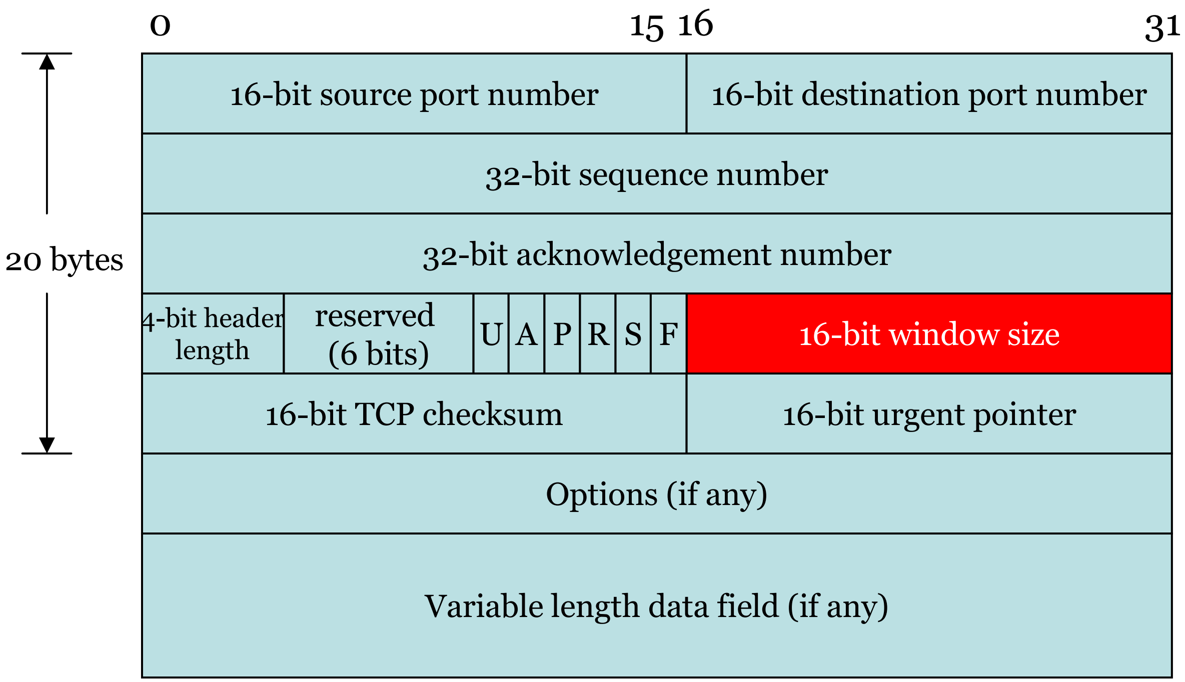
Abnormal: contextual – "out of spec Packets," SF (syn-fin), UAPRSF (xmas tree, nastygram, kamikaze, etc), 21****** (reserved bits set).

```
45 10 00 3c 27 a7 40 00 40 06 8f 56 c0 a8 01 01
c0 a8 01 64 80 ab 00 17 a2 cf a9 10 d5 6d b7 2d
50 18
```

Ack and Psh flags are set



Ethernet trailer | Application data | TCP header | IP header | Ethernet header



0 15 16 31

20 bytes

16-bit source port number | 16-bit destination port number

32-bit sequence number

32-bit acknowledgement number

4-bit header length | reserved (6 bits) | U | A | P | R | S | F | 16-bit window size

16-bit TCP checksum | 16-bit urgent pointer

Options (if any)

Variable length data field (if any)

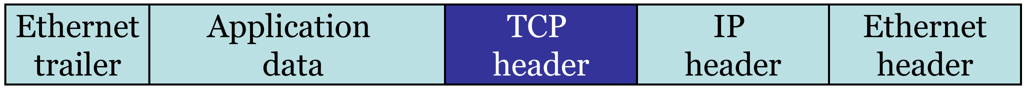
This value tells the transmitting host how much data it may transmit before it must stop and wait for acknowledgements from the receiver. It allows the receiver to control the flow of data.

Normal: if the receiver's input buffer is currently full, this value may be 0 telling the transmitter to discontinue data flow until further notice. Maximum window size is 65535.

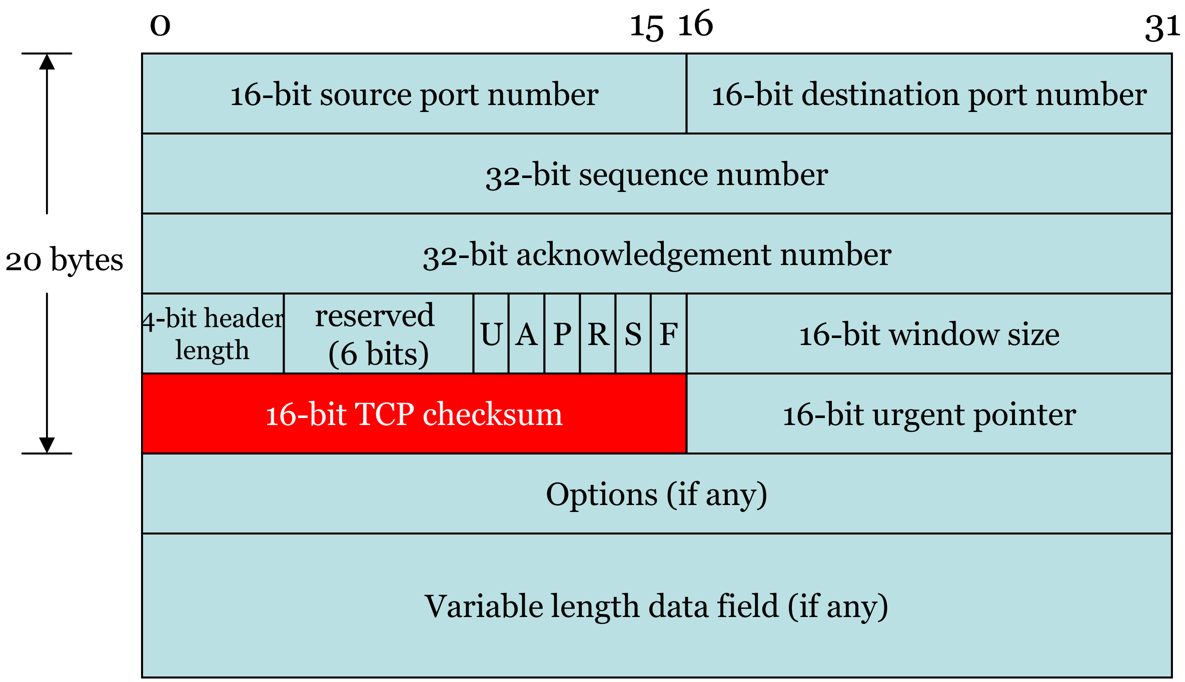
Abnormal: contextual. an aggressive flow of data after advertising a window size of 0 should be suspicious.

```
45 10 00 3c 27 a7 40 00 40 06 8f 56 c0 a8 01 01
c0 a8 01 64 80 ab 00 17 a2 cf a9 10 d5 6d b7 2d
50 18 16 d0
```

5480 bytes of data can fit into the input buffer



Ethernet trailer | Application data | TCP header | IP header | Ethernet header



0 15 16 31

20 bytes

16-bit source port number | 16-bit destination port number

32-bit sequence number

32-bit acknowledgement number

4-bit header length | reserved (6 bits) | U | A | P | R | S | F | 16-bit window size

16-bit TCP checksum | 16-bit urgent pointer

Options (if any)

Variable length data field (if any)

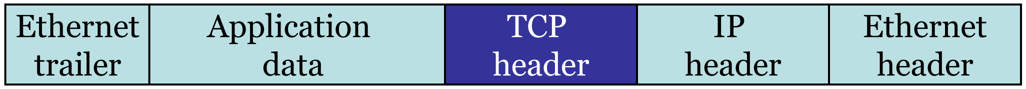
A mandatory checksum covering the TCP header and contents that is calculated by the sender and verified by the receiver.

Normal: a correct checksum

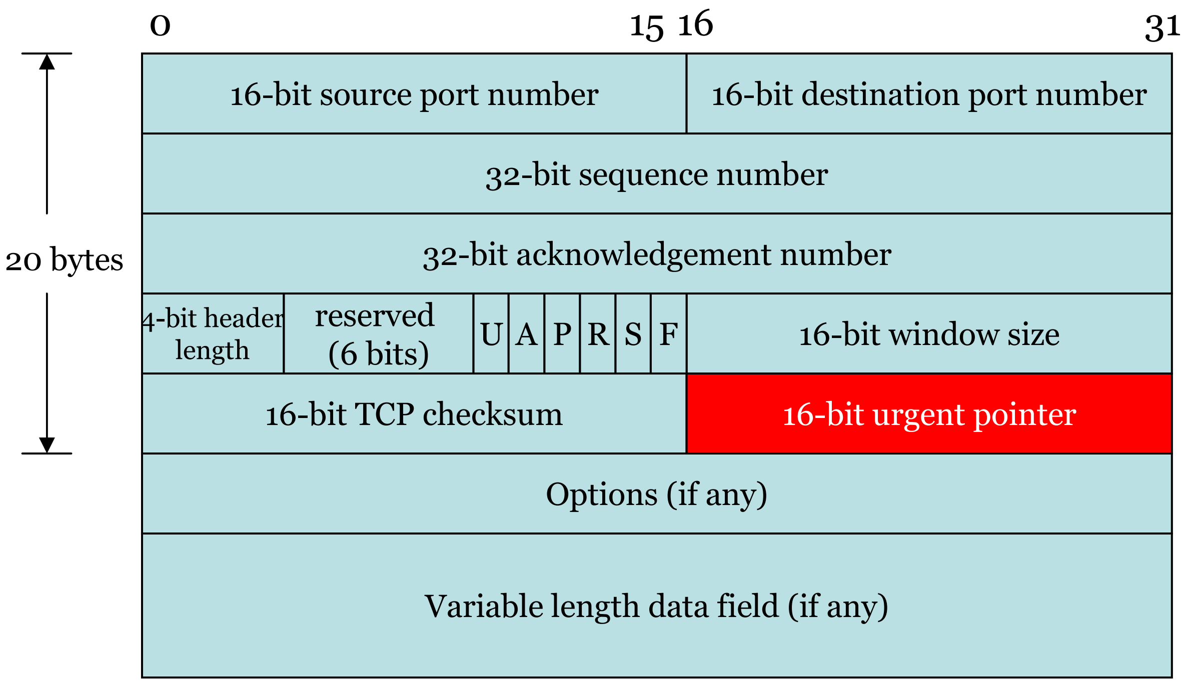
Abnormal: an abundance of incorrect checksums

```
45 10 00 3c 27 a7 40 00 40 06 8f 56 c0 a8 01 01
c0 a8 01 64 80 ab 00 17 a2 cf a9 10 d5 6d b7 2d
50 18 16 d0 ae ee
```

checksum is 0xaaee (dummy figures)



Ethernet trailer | Application data | TCP header | IP header | Ethernet header



This value, when added to the sequence number in the packet, points to the last byte of urgent data.

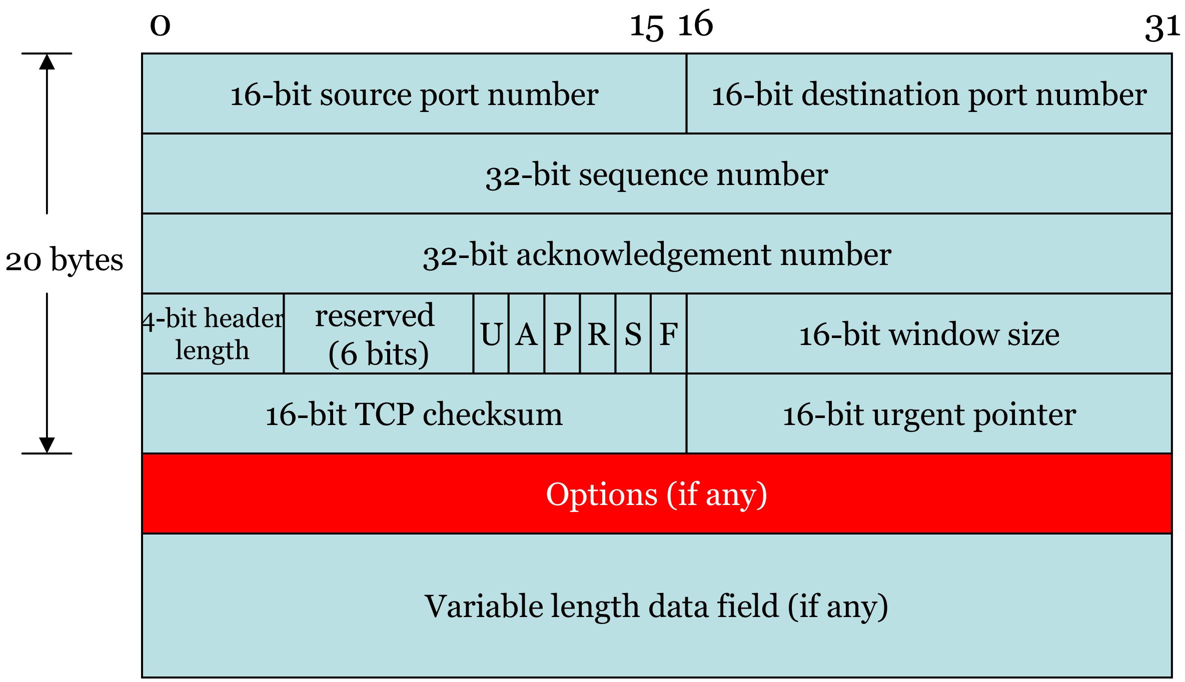
Normal: contextual. The URG flag is common when a telnet user presses the interrupt key or an FTP user aborts a file transfer.

Abnormal: a non-zero value when the URG (U) flag is not set.

```
45 10 00 3c 27 a7 40 00 40 06 8f 56 c0 a8 01 01
c0 a8 01 64 80 ab 00 17 a2 cf a9 10 d5 6d b7 2d
50 18 16 d0 ae ee 00 00
```

the Urg flag is not set, so the urgent pointer field is 0

Ethernet trailer
Application data
TCP header
IP header
Ethernet header



0 15 16 31

20 bytes

16-bit source port number

16-bit destination port number

32-bit sequence number

32-bit acknowledgement number

4-bit header length

reserved (6 bits)

U A P R S F

16-bit window size

16-bit TCP checksum

16-bit urgent pointer

Options (if any)

Variable length data field (if any)

Possible options include:

MSS maximum segment size
SackOK selective acknowledgement
Timestamp
NOP no operation
wscale window scale

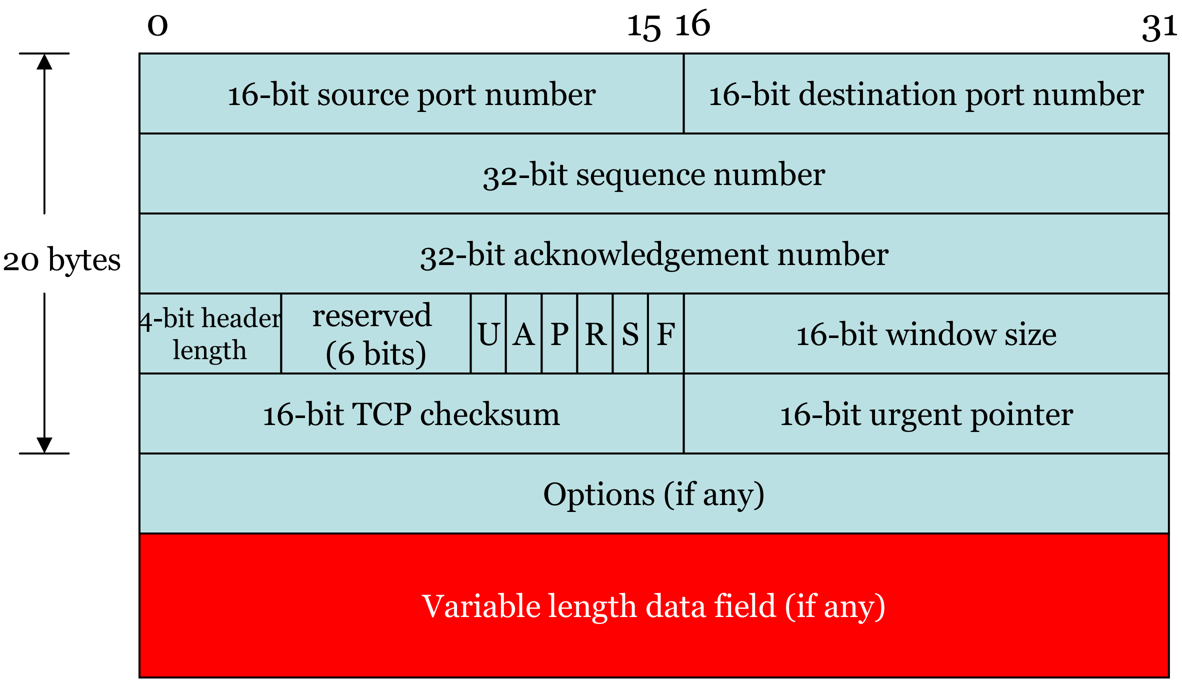
Normal: contextual.

Abnormal: contextual. MSS, SackOK, and wscale may only be set in connection establishment packets (the first three).

```
45 10 00 3c 27 a7 40 00 40 06 8f 56 c0 a8 01 01
c0 a8 01 64 80 ab 00 17 a2 cf a9 10 d5 6d b7 2d
50 18 16 d0 ae ee 00 00
```

no options

Ethernet trailer	Application data	TCP header	IP header	Ethernet header



0

15 16

31

20 bytes

16-bit source port number

16-bit destination port number

32-bit sequence number

32-bit acknowledgement number

4-bit header length

reserved (6 bits)

U A P R S F

16-bit window size

16-bit TCP checksum

16-bit urgent pointer

Options (if any)

Variable length data field (if any)

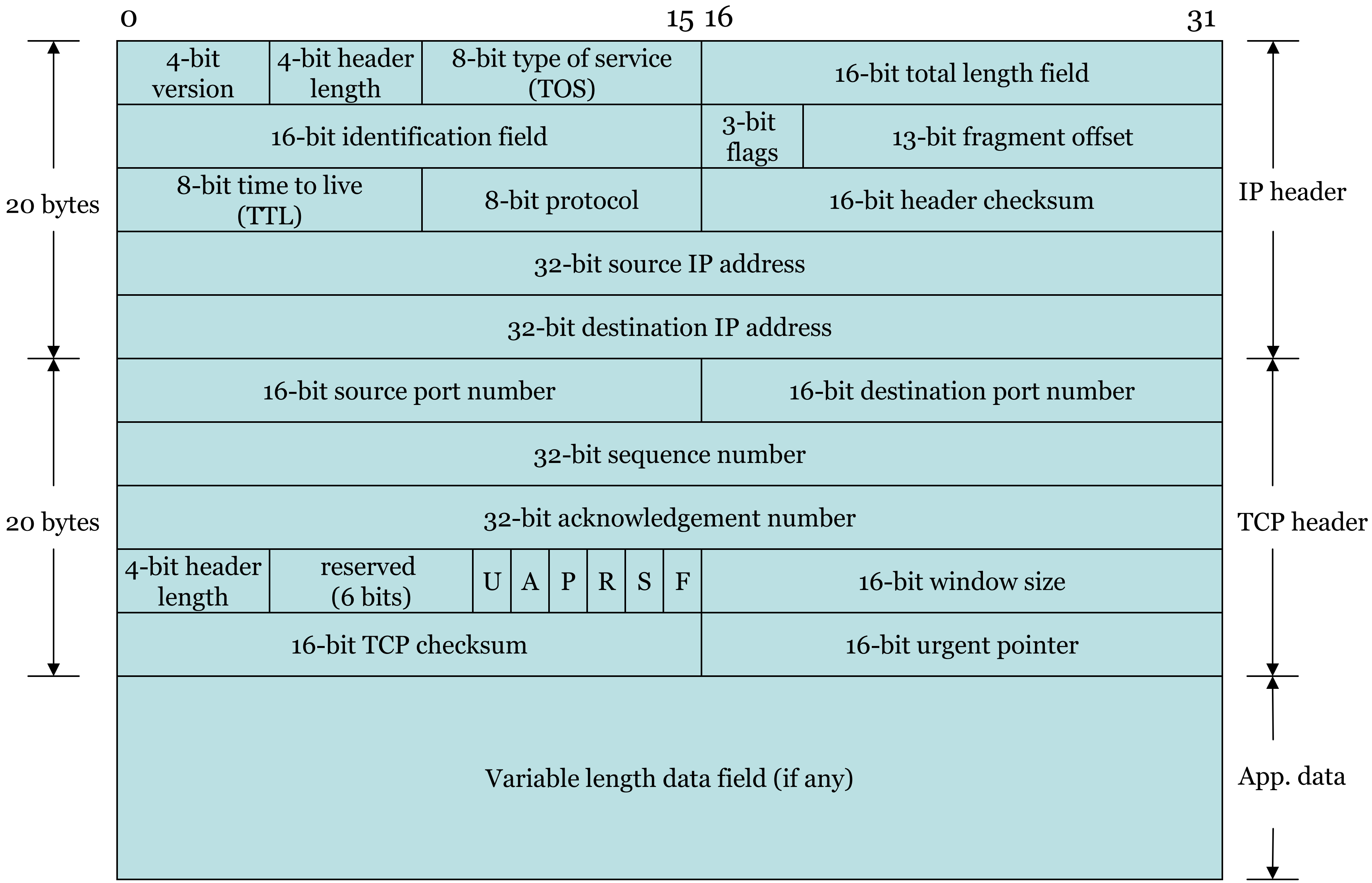
Variable length data field (application data).

In this example we are logging into telnet with the password “reveal77”

```
45 10 00 3c 27 a7 40 00 40 06 8f 56 c0 a8 01 01
c0 a8 01 64 80 ab 00 17 a2 cf a9 10 d5 6d b7 2d
50 18 16 d0 ae ee 00 00 72 65 76 65 61 6c 37 37
```

an 8-byte string, reveal77

Ethernet trailer	Application data	TCP header	IP header	Ethernet header



0 15 16 31

4-bit version | 4-bit header length | 8-bit type of service (TOS) | 16-bit total length field

16-bit identification field | 3-bit flags | 13-bit fragment offset

8-bit time to live (TTL) | 8-bit protocol | 16-bit header checksum

32-bit source IP address

32-bit destination IP address

16-bit source port number | 16-bit destination port number

32-bit sequence number

32-bit acknowledgement number

4-bit header length | reserved (6 bits) | U | A | P | R | S | F | 16-bit window size

16-bit TCP checksum | 16-bit urgent pointer

Variable length data field (if any)

20 bytes

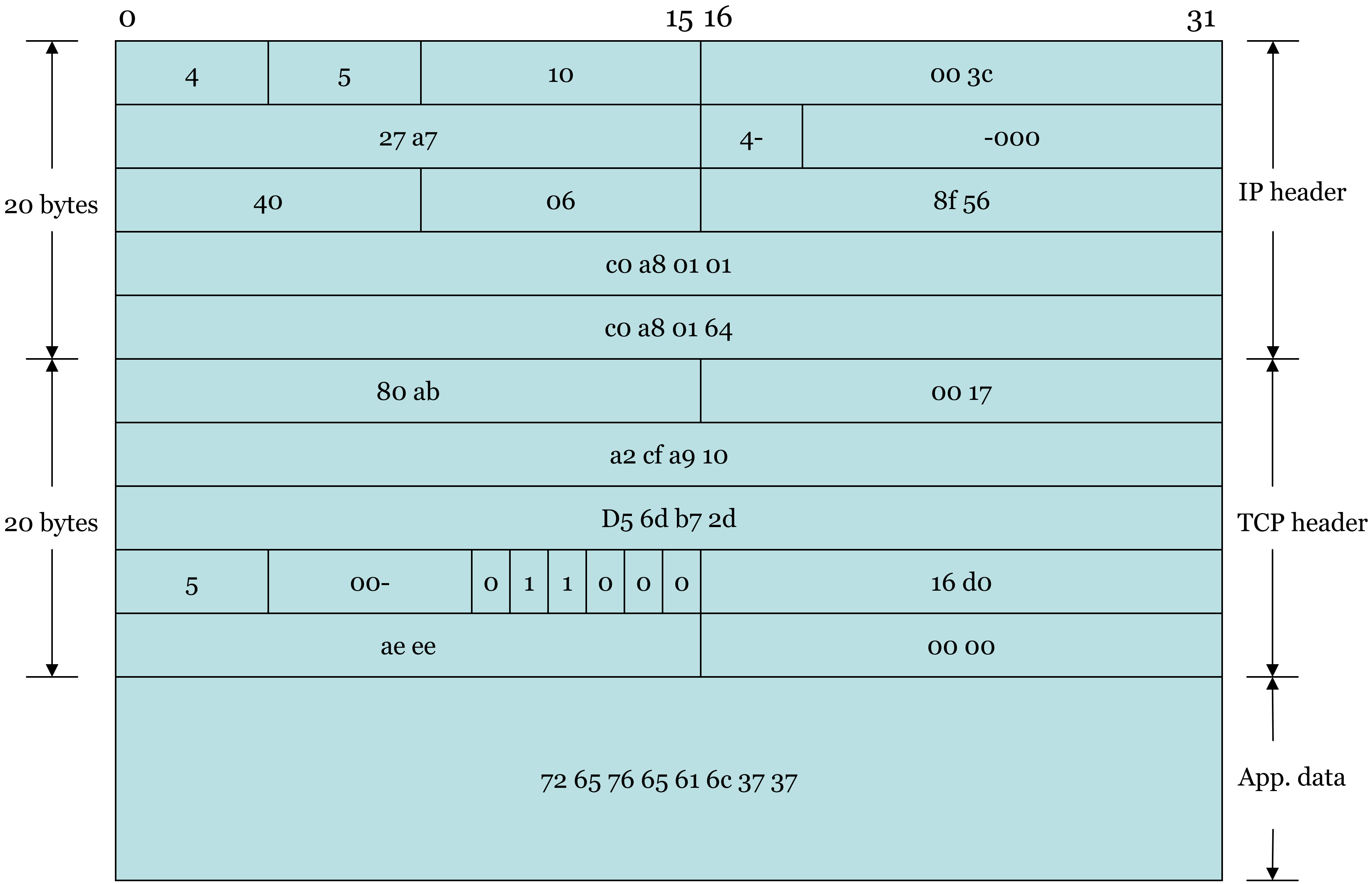
20 bytes

IP header

TCP header

App. data

complete message format: template before



0 15 16 31

4 | 5 | 10 | 00 3c

27 a7 | 4- | -000

40 | 06 | 8f 56

c0 a8 01 01

c0 a8 01 64

80 ab | 00 17

a2 cf a9 10

D5 6d b7 2d

5 | 00- | 0 | 1 | 1 | 0 | 0 | 0 | 16 d0

ae ee | 00 00

72 65 76 65 61 6c 37 37

20 bytes

20 bytes

IP header

TCP header

App. data

complete message format: template after

Interpretation

- IP
  - Version: 4
  - Header length: 20
  - TOS: minimize delay
  - Total length: 60
  - Identification: 10151
  - Flags: DF – Don’t Fragment
  - TTL: 64
  - Protocol: TCP
  - Checksum: 36694
  - Source address: 192.168.1.1
  - Destination address: 192.168.1.100
- TCP
  - Source port: 32939
  - Destination port: 23
  - Sequence number: 2731518224
  - Acknowledgement number: 3580737325
  - Header length: 20
  - Flags: Ack, Psh
  - Window: 5480
  - Checksum: 44782
  - Urgent pointer: 0
- Application data
  - reveal77

```
45 10 00 3c 27 a7 40 00 40 06 8f 56 c0 a8 01 01
c0 a8 01 64 80 ab 00 17 a2 cf a9 10 d5 6d b7 2d
50 18 16 d0 ae ee 00 00 72 65 76 65 61 6c 37 37
```

example message

(Resource starvation DoS): Snork Attack

```
192.168.38.110:135 > 192.168.38.110:135 UDP 46 [tos 0x3]
```

```
45 03 00 4a 96 ac 00 00 40 11 15 c7 c0 a8 26 6e
c0 a8 26 6e 00 87 00 87 00 36 84 33 69 23 61 6d
20 6c 61 6d 65 20 64 6f 73 20 6b 69 64 20 62 75
```

Observations:

the TOS is 0x03 which UDP has no legitimate use for

the source and destination IP are identical (Land Attack)

the source and destination port are identical, creating a socket that loops messages back and forth infinitely.

(Application crash DoS): WinNuke Attack

When a Windows system receives a packet with the URG flag set, it expects data will follow that flag. The exploit consists of setting the URG flag but not following it with data; and then sending a RST to tear down the connection. Not only will it tear down the connection but the victim would experience BSOD.

(Application crash DoS): Small Footprint Attack

```
172.23.133.99 > 172.23.133.4 IP 1204 [ttl 146]

              00 00 04 b4 00 01 00 00 92 04 00 00 ac 17 85 63
              Ac 17 85 04 00 00 00 00 00 00 00 00 00 00 00 00
              00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
              00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
              [snip]
```

Observations:

this indicates an IP version 0 – there was never an IPv0

this indicates a header length of 0 – the minimum is 5

Certain versions of TCPdump cannot process the packet so they crash and dump core.

(Resource starvation DoS): Boink Fragment Attack

```
25.25.25.25:20 > 192.168.38.5:20 udp 28 (frag 1109:36@0+)

              45 00 00 38 04 55 20 00 ff 11 7e 80 19 19 19 19
              c0 a8 26 05 00 14 00 14 00 24 00 00 00 00 00 00
              00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
              00 00 00 00 00 00 00 00

25.25.25.25 > 192.168.38.5 (frag 1109:4@32)

              45 00 00 18 04 55 00 04 ff 11 7e 80 19 19 19 19
              c0 a8 26 05 00 14 00 14
```

Observations:

this is the first fragment because the MF bit is set (0x2) and the offset field is zeroed out (0x000)

the fragment ID (1109) is taken from the IP ID field – all fragments will have the same value

this is the last fragment because neither the DF bit nor the MF bit is set and the offset field is non-zero

IP stack has no concept of negative math – it cannot backspace into memory. Negative numbers are Interpreted as large positive numbers, and thus the data will be written somewhere far away (probably system crash).

(Resource starvation DoS): Teardrop Attack

```
10.10.10.10:53 > 192.168.1.3:53 udp 28 (frag 242:36@0+)

               45 00 00 38 00 f2 20 00 40 11 84 04 0a 0a 0a 0a
               c0 0a 01 03 00 35 00 35 00 24 00 00 00 00 00 00
               00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
               00 00 00 00 00 00 00 00
```

```
10.10.10.10 > 192.168.1.3 (frag 242:4@24)

               45 00 00 18 00 f2 00 03 40 11 a4 21 0a 0a 0a 0a
               c0 a8 01 03 00 35 00 35 00 24 00 00 00 00 00 00
               00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Observations:

this is the first fragment because the MF bit is set (0x2) and the offset field is zeroed out (0x000)

the fragment ID (242) is taken from the IP ID field – all fragments will have the same value

this is the last fragment because neither the DF bit nor the MF bit is set and the offset field is non-zero

The second (and last) fragment is completely contained within the first. A bug in the fragment reassembly code of older TCP/IP stacks cause the system to crash. No room to mention this before – a non-terminal fragment size of 36 is actually illegal, it must be a multiple of 8.

(bandwidth consumption DoS): Smurf Attack

```
179.135.168.43 > 192.168.30.255 icmp: echo request (DF)

              45 00 00 1c c0 14 40 00 1e 01 61 72 b3 87 a8 2b
              c0 a8 1e ff 08 00 f7 ff 00 00 00 00 00 00 00 00
              00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
68.90.226.250 > 192.168.30.255 icmp: echo request (DF)

              45 00 00 1c c0 15 40 00 1e 01 95 cf 44 5a e2 fa
              c0 a8 1e ff 08 00 f7 ff 00 00 00 00 31 36 38 03
              31 33 35 03 31 37 39 07 69 6e 2d 61 64 64
```

Observations:

0xff as the last two digits refers to the broadcast address x.x.x.255

0x01 indicates ICMP protocol, 0x0800 indicates a type 8 code 0 message (better known as echo request)

evidence of forged source IP

The broadcast address is used to amplify a single packet into many.