IP Checksum, Fragmentation

Internet Networking recitation #2
# IP Header Diagram

<table>
<thead>
<tr>
<th>Ver.</th>
<th>IHL</th>
<th>TOS</th>
<th>Total Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Identification</th>
<th>Flags</th>
<th>Fragment Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TTL</th>
<th>Protocol</th>
<th>Checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Source Address

Destination Address

Options | Padding
IP Header Checksum

- We want to ensure the integrity of the IP header.
  - Why?
    - Does not check data integrity.
    - Allowing higher level protocols to choose their own verification scheme for the data.
    - Reducing processing time at routers.
1’s complement

• Negation is done by flipping bits:
  if 1(decimal) = 001 then -1 = 110.

• 1’s complement sum is done by adding the bit overflow to the sum.
Checksum Calculation

• Adjacent octets to be checksummed are paired to form 16-bit words. The checksum field is cleared.
• The 16-bit 1's complement sum is computed over the 16-bit words
• Any overflows are added to the sum.
• The 1's complement of this sum is placed in the checksum field.
Checksum Verification

• To verify a checksum, the 1's complement sum is computed over the same set of octets, including the checksum field.

• If the result is all 1 bits, the check succeeds (how do we represent 0 in 1’s complement?).
Is It any good?

- Original Message:
  - 0x00 0x00 0x00 0x02 (Checksum = 0xfd, ENGINE_SHUTDOWN)

- Corrupted Message:
  - 0x00 0x00 0x01 0x01 (Checksum = 0xfd, SELF_DESTRUCT)

- OOPS...
IP Checksum Effectiveness

• What is the distance of the code?
  2

• What about larger errors?
  16-bit Checksum supports up to 16 bits error burst detection.

• What about random errors?
  Probability of $\frac{1}{2^{16}}$ to receive the same checksum.

**Hamming Distance:**
The minimum number of errors, which can be unrecognized by the error detection code.
IP Checksum VS CRC

• Cyclic Redundancy Check – A Method based on polynomial division.
• Ethernet (layer-2) uses CRC to check frames integrity.
• Much better error detecting capabilities.

• So why do we need checksum at all??
Incremental Checksum Update

• Updating part of the IP header data doesn’t require recalculating the entire Checksum field. (For example: when a router changes the TTL field.)
Incremental Checksum Update

• Calculating the new 1’s complement sum:

\[ C' = C + (-m) + m' = C + (m' - m) \]

• The checksum is the negative of \( C' \):

\[ \sim C' = \sim(C + (-m) + m') = \sim C + (m - m') = \sim C + m + \sim m' \]

• Note, the second equality does not work if the sum in the brackets is 0xFFFF (see RFC 1624).

Notation:

- \( C \) - Old 1’s complement sum.
- \( m \) - Old data.
- \( C' \) - New 1’s complement sum.
- \( m' \) - New data.
IP Fragmentation

• An IP packet travels over different networks, each of which may have a different MTU (Maximum Transmission Unit), which is the largest packet size that can be carried by a network.

• In particular every technology has its own MTU.
  • In Ethernet the MTU is 1500 bytes.

• Large IP packets may traverse networks in which the MTU is smaller than the packet size - These packets must be fragmented.
How is it done?

• IP fragmentation is done in the entry point of the (physical) network that requires the fragmentation.
  • IP layer automatically performs fragmentation of a datagram when it is too large to be sent on the physical network.
  • A datagram can be fragmented more than one time.
  • Fragments are counted in units of 8 octets.

• Reassembly is done at the IP layer of the destination.
  • Fragmentation is transparent to upper protocols.
### IP Header - Reminder

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ver.</td>
<td>Version</td>
</tr>
<tr>
<td>IHL</td>
<td>Internet Header Length</td>
</tr>
<tr>
<td>TOS</td>
<td>Type of Service</td>
</tr>
<tr>
<td>Total Length</td>
<td>Total Length of the IP Header</td>
</tr>
<tr>
<td>Identification</td>
<td>Identification</td>
</tr>
<tr>
<td>Flags</td>
<td>Flags</td>
</tr>
<tr>
<td>Fragment Offset</td>
<td>Fragment Offset</td>
</tr>
<tr>
<td>TTL</td>
<td>Time To Live</td>
</tr>
<tr>
<td>Protocol</td>
<td>Protocol</td>
</tr>
<tr>
<td>Checksum</td>
<td>Checksum</td>
</tr>
<tr>
<td>Source Address</td>
<td>Source Address</td>
</tr>
<tr>
<td>Destination Address</td>
<td>Destination Address</td>
</tr>
<tr>
<td>Options</td>
<td>Options</td>
</tr>
<tr>
<td>Padding</td>
<td>Padding</td>
</tr>
</tbody>
</table>

**Question:** Which fields are relevant to fragmentation?
Identification Field

• A unique integer which identifies the packet.
• The originator of the IP packet (i.e. the source) sets the identification field to a value that must be unique for that source-destination pair and protocol for the time the datagram will be active in the internet system.
  • Implemented by a counter, which increments by one.
• Ensures that fragments of different packets are not mixed.
Fragment Offset Field

• Tells the receiver the position of a fragment in the original packet.
  • Identifies the fragment location, relative to the beginning of the original unfragmented packet.

• The fragment offset and total length fields determine the portion of the original packet covered by this fragment.

• The fragment offset is measured in units of 8 octets (64 bits).
  • Enables Additional fragmentation.
  • It does not include the length of the IP header.
  • Reason: 13 bits are used to map 16 bits (IP length).

• The first fragment has offset zero.
Flags Field

- Bit 1 (DF): 0 = May Fragment.
  1 = Don't Fragment.
  - If is set, then fragmentation of this datagram is NOT permitted
  - If fragmentation required, but this bit is set then the packet is discarded and ICMP is returned.
- Bit 2 (MF): 0 = Last Fragment.
  1 = More Fragments.
  - Set if the datagram is not the last fragment.
The Evil Bit

Network Working Group
Request for Comments: 3514
Category: Informational

1 April 2003

The Security Flag in the IPv4 Header

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (2003). All Rights Reserved.

Abstract

Firewalls, packet filters, intrusion detection systems, and the like often have difficulty distinguishing between packets that have malicious intent and those that are merely unusual. We define a security flag in the IPv4 header as a means of distinguishing the two cases.

1. Introduction

Firewalls [CBBR80], packet filters, intrusion detection systems, and the like often have difficulty distinguishing between packets that have malicious intent and those that are merely unusual. The problem is that making such determinations is hard. To solve this problem, we define a security flag, known as the "evil" bit, in the IPv4 [RFC789] header. Benign packets have this bit set to 0; those that are used for an attack will have the bit set to 1.

Fragmentation Example

Original Packet
Size = 6000 Bytes

First Fragmentation
MTU = 3300 Bytes

<table>
<thead>
<tr>
<th>MF</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Data
5980 Bytes

<table>
<thead>
<tr>
<th>MF</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Data
3280 Bytes

<table>
<thead>
<tr>
<th>MF</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>410</td>
</tr>
</tbody>
</table>

Data
2700 Bytes
What if another layer of fragmentation is needed?

SONET
MTU = 4470 Bytes

<table>
<thead>
<tr>
<th>MF</th>
<th>Offset</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>3280 Bytes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MF</th>
<th>Offset</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>410</td>
<td>2700 Bytes</td>
</tr>
</tbody>
</table>

Ethernet
MTU = 1500 Bytes

<table>
<thead>
<tr>
<th>MF</th>
<th>Offset</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1480 Bytes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MF</th>
<th>Offset</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>185</td>
<td>1480 Bytes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MF</th>
<th>Offset</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>370</td>
<td>320 Bytes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MF</th>
<th>Offset</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>410</td>
<td>1480 Bytes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MF</th>
<th>Offset</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>595</td>
<td>1220 Bytes</td>
</tr>
</tbody>
</table>

Why is this 1 important?
Fragmentation should be avoided!

- Loss of one fragment requires retransmission of the entire packet.
- Fragmentation must be supported by every IP entity (routers, hosts, etc.).

- MTU discovery protocol (RFC 1191) that uses the DF bit, is used to avoid the necessity of IP fragmentation (wait for the ICMP tutorial).
Wireshark
What is Wireshark?

• “Wireshark is a free and open source packet analyzer. It is used for network troubleshooting, analysis, software and communication protocol development, and education.”

• Basically a great tool for looking on the “inside” of network protocols.

• Built upon the libpcap/WinPcap library (Like tcpdump/windump cmd line tools).
# Wireshark Interface

Wireshark is a powerful network protocol analyzer. It allows you to capture and analyze network traffic. Here are some key features:

- **Display filters**
- **Captured packets**
- **Analyze a specific packet**
- **Packet content in HEX and ASCII**

## Screenshot of Wireshark Interface

![Wireshark Interface Screenshot](image.png)

### Table Example

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>297</td>
<td>00:00:000000</td>
<td>1.2.3.4</td>
<td>2.3.4.5</td>
<td>Ethernet</td>
<td>60</td>
<td>TCP 80, RTT: 123 ms</td>
</tr>
</tbody>
</table>

## Example Packet Analysis

```
Ethernet (0x0000) [Wireshark 1.2.3.4 (v1.1.2.3-0-gb486f1d0 from master-1.12)]

Frame: 299, 612 bytes on wire (4896 bytes), 612 bytes captured (4896 bytes) on Interface 0


Internet Protocol Version 4, Src: 10.0.0.1, Dst: 127.0.0.1, Dst Port: 80

Transmission Control Protocol, Src Port: 5555, Dst Port: 80
```

## Packet Content

```
GET /glitz-a-zixief/glitz-a-zixief.mub?user=HTTP_Proxy_Interstream_8072809&session=Mhsid=MLS_4608128 HTTP/1.1

Content-Length: 1239

Connection: keep-alive

User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.107 Safari/537.36

```

### Further Reading

- [Wireshark Official Documentation](https://www.wireshark.org)
- [Wireshark Tutorial](https://www.tutorialspoint.com/wireshark/wireshark_tutorial.htm)
- [Wireshark Advanced Features](https://www.gowidgetsoftware.com/wireshark-advanced-features.html)
Capture Options

• Found under Capture – Options…
• We can define from which Interface packets will be captured.
• We can choose *promiscuous mode in order to see all traffic going through the network (will not work if we are connected to a switch, why?).
• We can define Name resolution and how to save captured data.
• We can define Capture Filters

*WARNING – Use with caution! Some Net Admins might not like it.
Filters

• Most of the time we don’t want to see all traffic going through the network, so we can use filters. There are 2 types of filters:

  • Capture filters – select which packets will be saved to disk, filtering done in kernel mode => much faster.
  • Display filters – change the view of a capture file, they can deeply dissect all packets => can do very complex & advanced filtering.
Defining capture filters

• Wireshark uses the same syntax for capture filters as tcpdump any other program that uses the libpcap/WinPcap library.

• *Examples:
  • Capture only traffic to or from IP address 172.18.5.4: host 172.18.5.4
  • Capture only DNS (port 53) traffic: port 53
  • Capture except all ARP and DNS traffic: Port not 53 and not arp

*More examples can be found at https://wiki.wireshark.org/CaptureFilters.
Defining display filters

• The basics and the syntax of the display filters are described in the User's Guide.

• Examples:
  • Show only SMTP (port 25) and ICMP traffic:
    tcp.port eq 25 or icmp
  • Or more complicated ones like, TCP buffer full -- Source is instructing Destination to stop sending data
    tcp.window_size == 0 && tcp.flags.reset != 1

*More examples can be found at https://wiki.wireshark.org/DisplayFilters.
Demo
Questions?

ARE THERE ANY

'QUESTIONS?'

memegenerator.net