# Project 2: IP packet reassembly support

The goal of this project is to add an IP packet reassembly facility to the packet capture tool you built in Project 1. The goal is to reassemble the packets that can later be checked against filtering rules. The final result will be a program that is also capable of detecting any fragmentation attacks (e.g. Teardrop type of attack).

Chapters 7-11 of the Network Intrusion Detection book cover all the material in detail (for those unfamiliar with IP fragmentation and the different problems caused by overlapping fragments).

## IP Packet reassembly

Fragmentation attacks (also known as Teardrop type of attack) are pretty common. The problem arises from the fact that RFC 791 (Internet Protocol specification) does not specify how to reassemble IP fragments (it does specify how to fragment an IP packet). As you might expect, different vendors implement packet reassembly in different ways, leaving them open to certain fragmentation attacks. RFC 815 (IP Datagram Reassembly Algorithms) specifies a simple IP reassembly algorithm. However, this algorithm also remains mute on the issue of overlapping fragments. RFC 1858 (Security Considerations for IP Fragment Filtering) and the Network Intrusion Detection book discuss this problem in detail.

Add to your packet capture program the capability to reassemble IP packets (you may use RFC 815 for ideas). When using the packet reassembly facility, the system must preprocess IP fragments before they are assembled into one datagram. In addition, your program should be able to detect overlapping fragments as well as oversized IP packets (size > 64K, the popular Ping of Death attack uses this approach).

## Implementation notes

To facilitate the transition to a full IDS in Project 3, it is suggested that your packet reassembly facility return a unique identifier (sid, integer) associated with the segment, a reassembled packet (full datagram) and a list of fragments. There are five basic cases to consider:

1. The frame is an ARP packet. Return an sid of zero, the ARP packet and a one packet list consisting only of the ARP packet.
2. The IP fragments assemble correctly with no overlapping or oversize. Return an sid of one, the reassembled packet and a list of the fragments.
3. The IP fragments assemble correctly with overlapping. Return an sid of two, the reassembled packet and the list of fragments.
4. The IP fragments correspond to an IP segment larger than 64K. Return an sid of three, the first segment and a list of all fragments.
5. In the event that a fragment times out, return an sid of four, the first segment and a list of all partially processed segments with the same id.

In the future, the rule checker (signature-based system) will operate over the triple (sid, segment, {segment}).

## Notes:

1. *It is advised that a timestamp be associated with each fragment looked at by the packet reassembly facility. There are DoS attacks that will flood a host with IP fragments that never completely finish an IP packet, exhausting resources. Your IDS should never be a victim of this type of attack (Jolt type of attack).*
2. *A fragment offset of one on a TCP packet, while valid, should be considered a clear indication of an attempt to either break-up the TCP header or overwrite a TCP header sent in a previous fragment.*
3. *It is imperative that your program check IP checksums. Any host will check the IP checksum and discard the packet if invalid. This type of packets may be an indication of a smart attacker trying to evade a naïve IDS (the fragments the IDS sees are not the same that the end-host may see).*

**Testing**

You will be given files with fragmented traffic corresponding to the same sessions we used for Project 1. In addition, the fragmented traffic will be playing continuously on the lab for your on-line tests.

**Submission**

Students will submit a typed report (hard-copy) the day the project is due at class time. In addition, students will submit via email to the instructor all the source code. The report should describe your approach in solving the problem and a description of all the classes used in your implementation. In addition, you must provide in your report instructions on how to use your program and sample output.

**Note**

Under no circumstances are you allowed to use the packet generator outside the environment of the lab.

**Due Date:**

This project is due Wednesday, March 30th 2011 at class time.