Introduction

This document describes the following aspects of the Galore capture engine:

1. Design
2. Implementation
3. Build environment
4. Runtime environment

Design

The capture engine needs to be designed with the following two objectives in mind:

1. Handle any type of traffic (http, smtp, database interactions etc)
2. Ease of deployment (capture engine should not be tightly coupled with any other component)

The following solutions were considered:

1. Web server plugin to capture http traffic – this solution would work quite well for http based applications. The advantage is that capturing the application level data would be trivial and the focus would be on what to do with the captured data. The obvious and major disadvantage is that it works only for http (and quite possibly for a specific web server)
2. Ethernet packet capture – this solution involves capturing the network traffic generated by the interaction between a web client and a web server. The advantage is that since the packet capture happens at a very low level (Ethernet), we would be able to capture traffic related to most of the applications. The disadvantage is that reconstructing the application data from the captured Ethernet frames is non-trivial.

At this point the decision is to go with the "packet capture" solution.

Packet capture

In the field of computer network administration, pcap (packet capture) consists of an application programming interface (API) for capturing network traffic. Unix-like systems implement pcap in the libpcap library; Windows uses a port of libpcap known as WinPcap.

Monitoring software may use libpcap and/or WinPcap to capture packets travelling over a network and, in newer versions, to transmit packets on a network at the link layer, as well as to get a list of network interfaces for possible use with libpcap or WinPcap.

libpcap and WinPcap provide the packet-capture and filtering engines of many open source and commercial network tools, including protocol analyzers (packet sniffers), network monitors, network intrusion detection systems, traffic-generators and network-testers.

Capture engine prototype

Pcap will form the heart of the capture engine. This would involve writing wrappers over the pcap api and building on the basic capture functionality provided by pcap

Build Environment

There are slightly different requirements for Windows and Linux/Unix

Windows

1. Install the Microsoft Netmon utility (this is necessary only if you plan to use the PPP and

loopback interfaces for capture)
2. Install the WinPcap developer pack
3. Install Microsoft Visual Studio 2008
4. Install Wireshark (optional). This is useful for understanding configuration/environment issues with WinPcap.

Linux/Unix

1. Install libpcap (this library is typically available by default on Linux atleast)

Runtime Environment

Windows

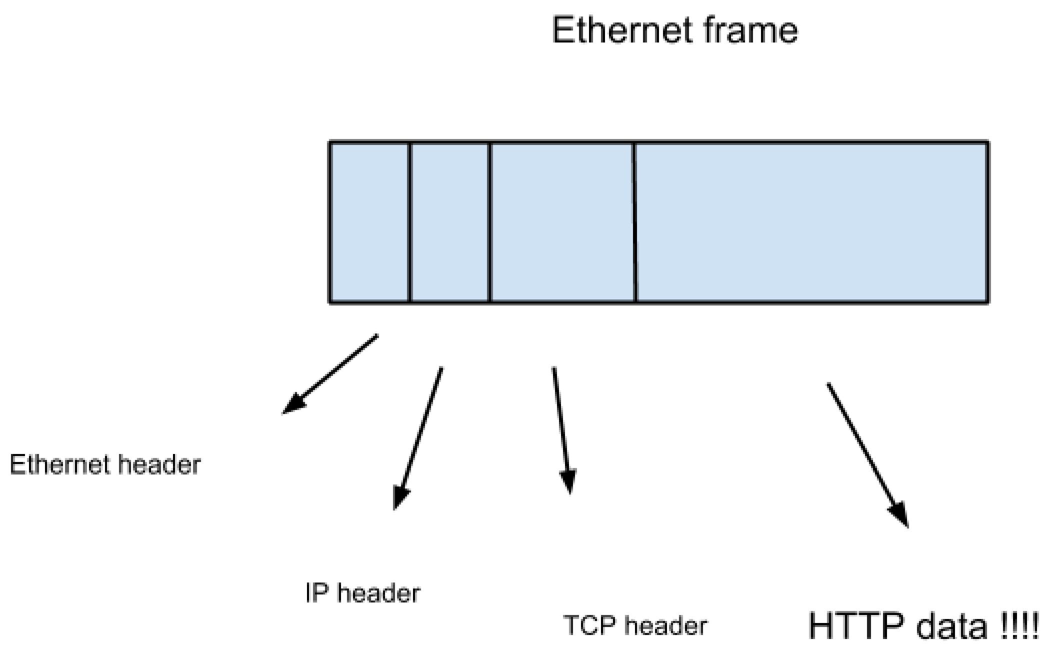
1. Install the Microsoft Netmon utility
2. Install WinPcap (developer pack not required)

Linux/Unix

1. Install libpcap

Implementation challenges

Since the captured packets are basically at the level of ethernet frames, quite a bit of work is involved in reconstructing application data. Since the initial prototype is targeted at capturing HTTP data, let us look at what is involved in the context of HTTP.
1) Extracting HTTP data from Ethernet frames - This is (under simplistic conditions) as simple as extracting the TCP payload.



2) Handling HTTP response spanning multiple TCP segments - Even the simplest of web applications will generate HTTP responses that will span multiple TCP segments. There are two basic ways to handle this scenario:

i) Extract the HTTP response length from the HTTP header. Read in TCP segments until the response length is exceeded.

ii) Use TCP sequence numbers. The segment with sequence number 1 would indicate the start of a HTTP response (multi-segment or uni-segment doesnt matter). Read in TCP segments until you again get a segment with a sequence number 1 (this would indicate reuse of the client port for a new HTTP conversation). Here we dont really track the end of the HTTP conversation. The start of a new conversation signals the end of the previous conversation.

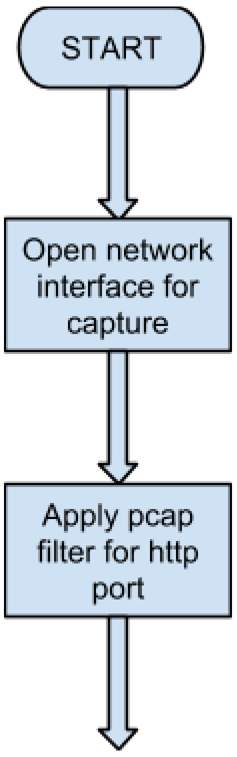
3) Handling out of sequence TCP segments - This is also a common scenario as Ethernet frames are not guaranteed to be delivered in order. This can be handled using TCP sequence numbers. The idea is to keep track of

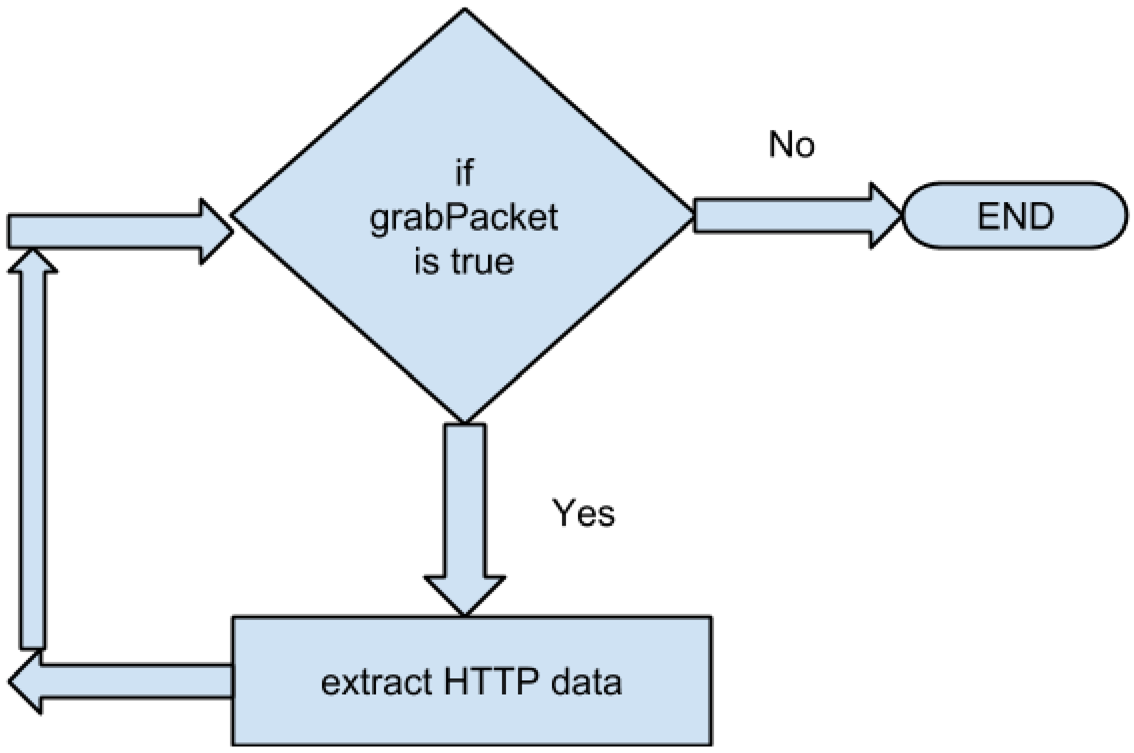
i) The expected sequence number at any point of time

ii) A list of the packets received till now sorted by sequence number

Capture algorithm

This section describes the capture and HTTP reconstruction algorithm used in view of the challenges described in the previous section.


START
Open network interface for capture
Apply pcap filter for http port



As mentioned earlier, extraction of HTTP data is not as simple in view of multiple segments and out of sequence segments. This step would need to use some method for handling multi segments. The algorithm being used currently in the capture engine is described in the following section.

TCP multi segment handling

```
1. Extract the following TCP header info from the segment
   - port
   - seq
   - ack
2. Extract the TCP payload, payload length
3. Identify the direction of the TCP segment
   incoming - capture host is receiving the segment
   OR
   outgoing - capture host is sending the segment
4. if segment is outgoing
   then
     // a http conversation has begun
     search the PACKET_MAP structure by port number to see if it already has a segment list
     if yes then delete the existing structure
     store the sequence and ack numbers of this segment
```

```
      store the pseudo seq and pseudo ack numbers
      // this is nothing but
      // pseudo seq = seq of first segment in conversation - current seq

      store the length of the payload

      set the expected sequence number of next response to 1

      dump capture header
      dump http payload

   else if segment is incoming
   then
      search the PACKET_MAP structure by port
      if an entry is found

        use the info stored in the retrieved structure (pinfo) to compute the pseudo seq and ack numbers for
the current segment

        update “pinfo” with the pseudo seq and ack computed above
     else
       // we should not reach here in normal cases
     end

     if the sequence number of current segment = pinfo.expected_sequence
     then

           if seq no = 1
             // we are at the first segment so we
            // dump the capture header
            dump capture header
           end
           write http payload to file/buffer
           traverse thru all the in sequence segments and dump the payload to file/buffer
           free the above segments
     else

          add the segment to the list
     end
```