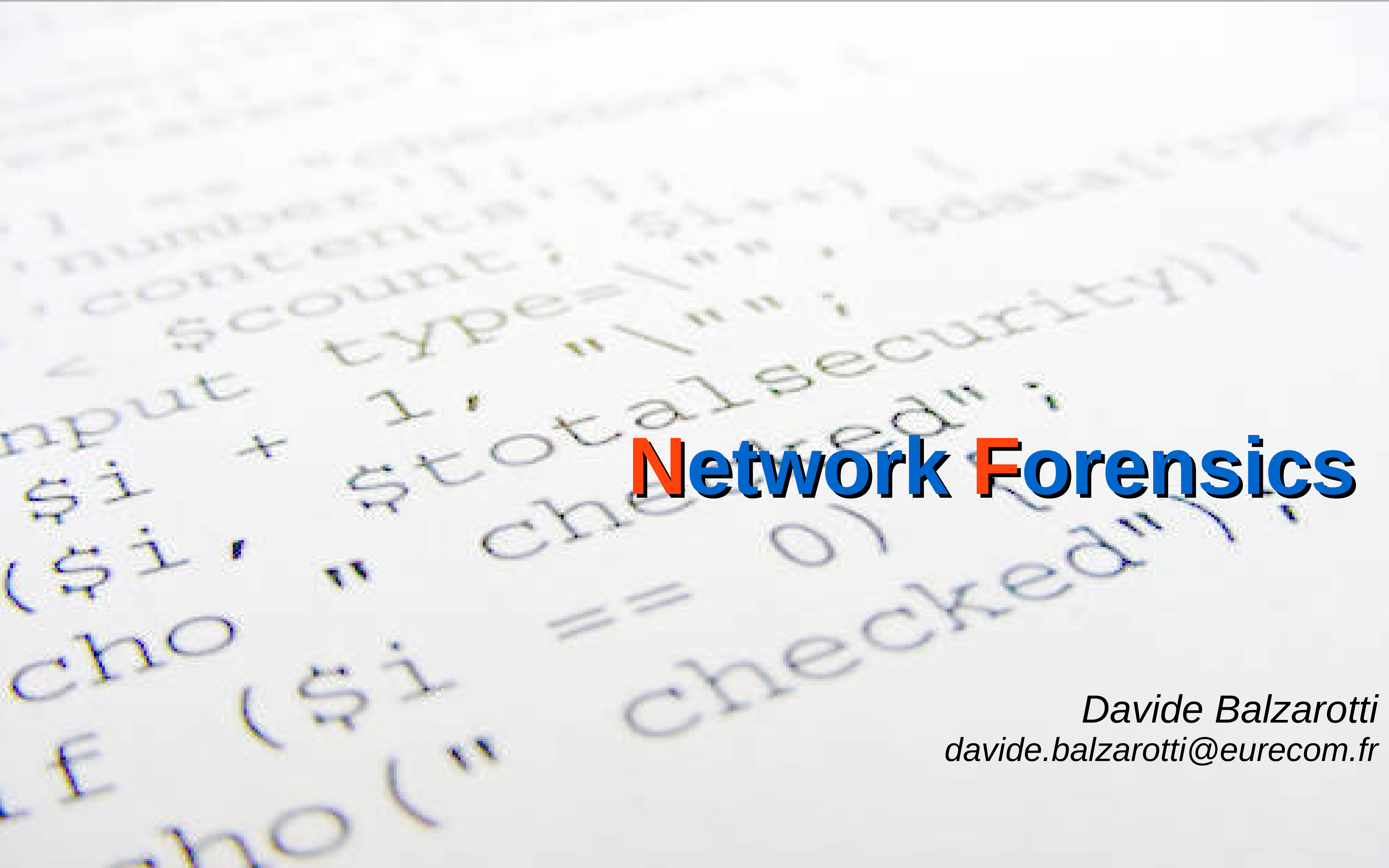
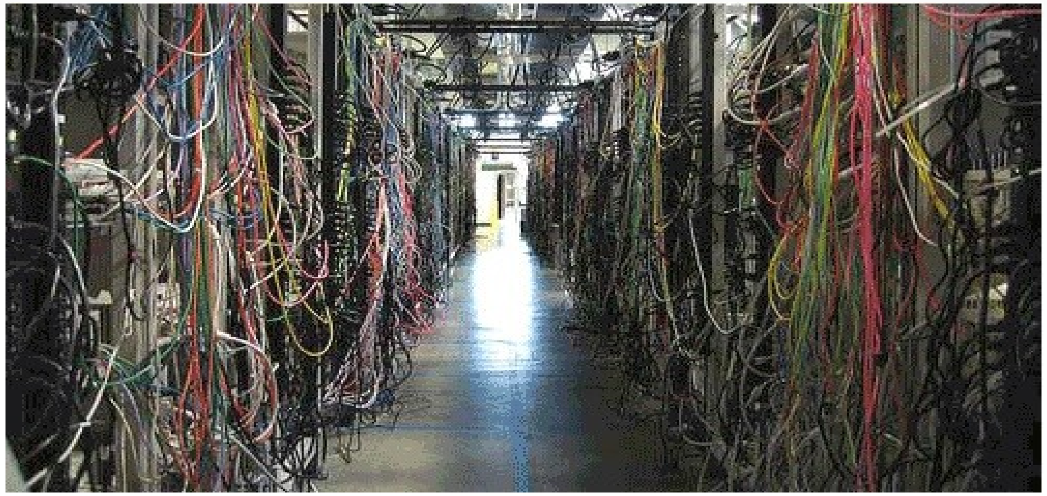


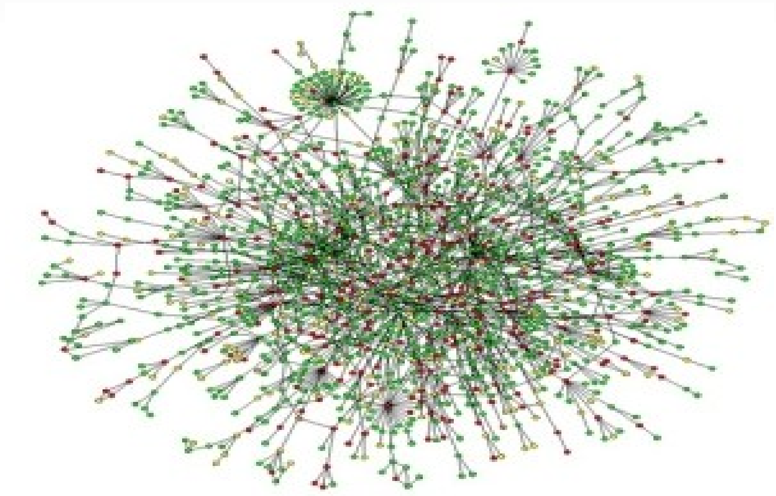
Network Forensics

Davide Balzarotti
davide.balzarotti@eurecom.fr

Network Forensics

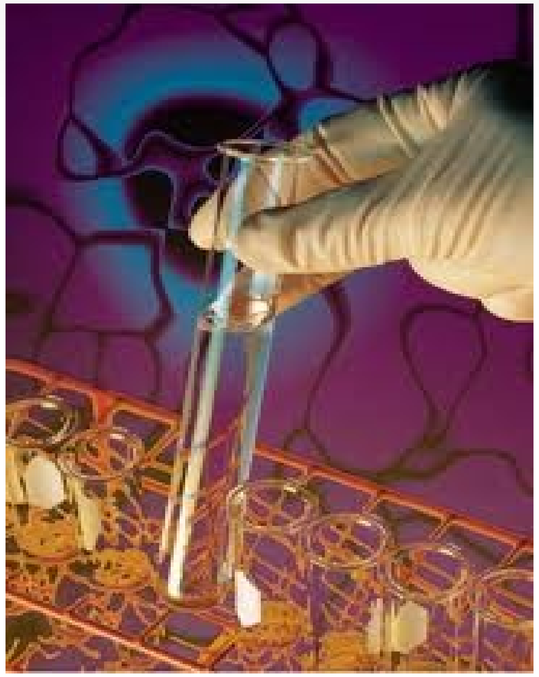
Capture, Recording, and Analysis of network traffic for the purposes of information gathering, legal evidence, or intrusion detection

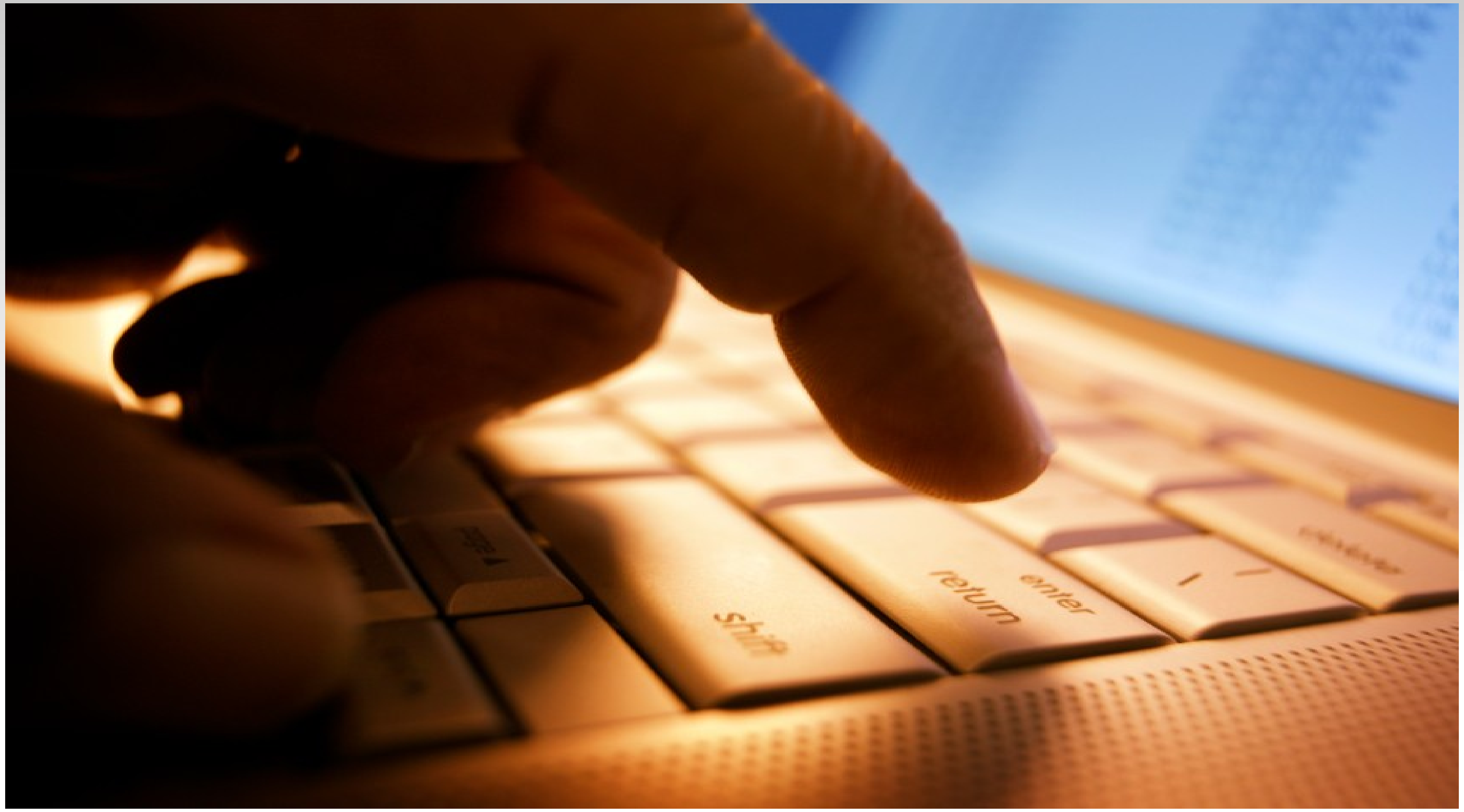




Summary

- Introduction
- Data Collection
- Traffic Analysis
- Enrichment and Fingerprinting
- Custom Analysis
- The Malware Corner
- Malware Network Patterns





Introduction

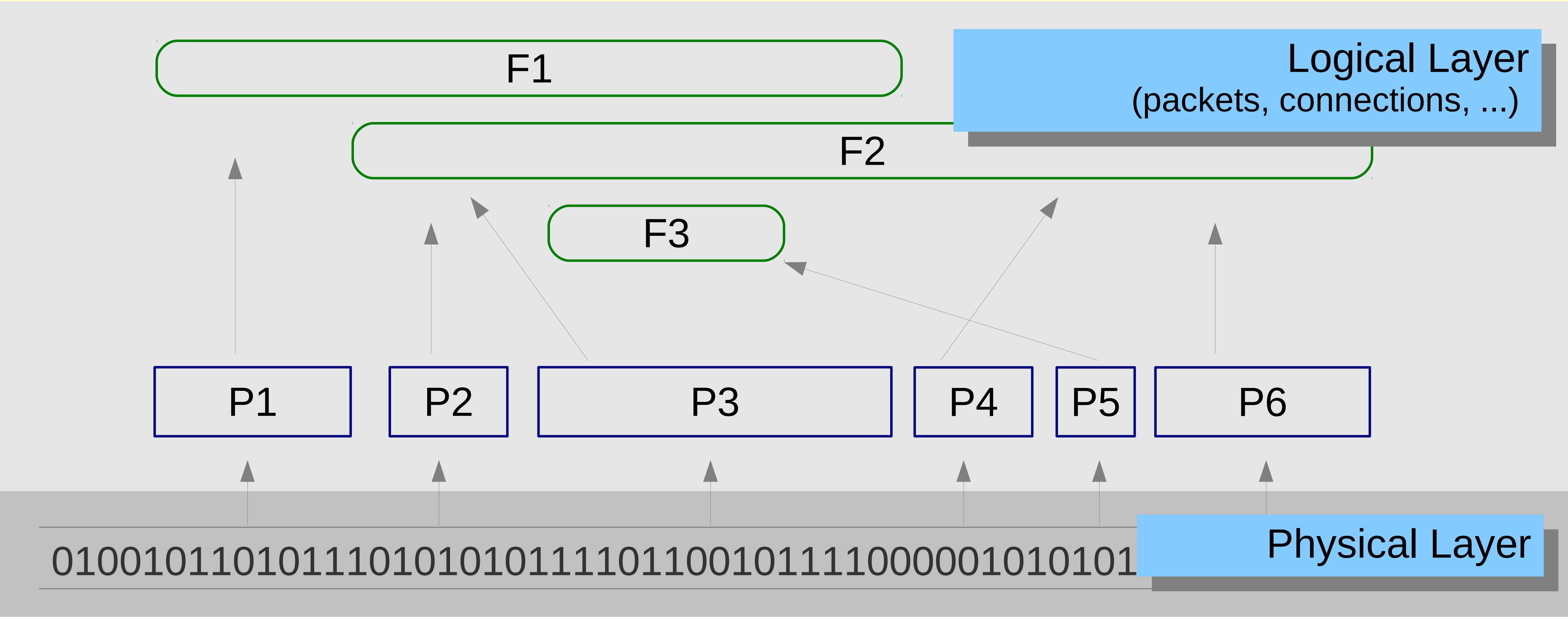
Network Forensics Data

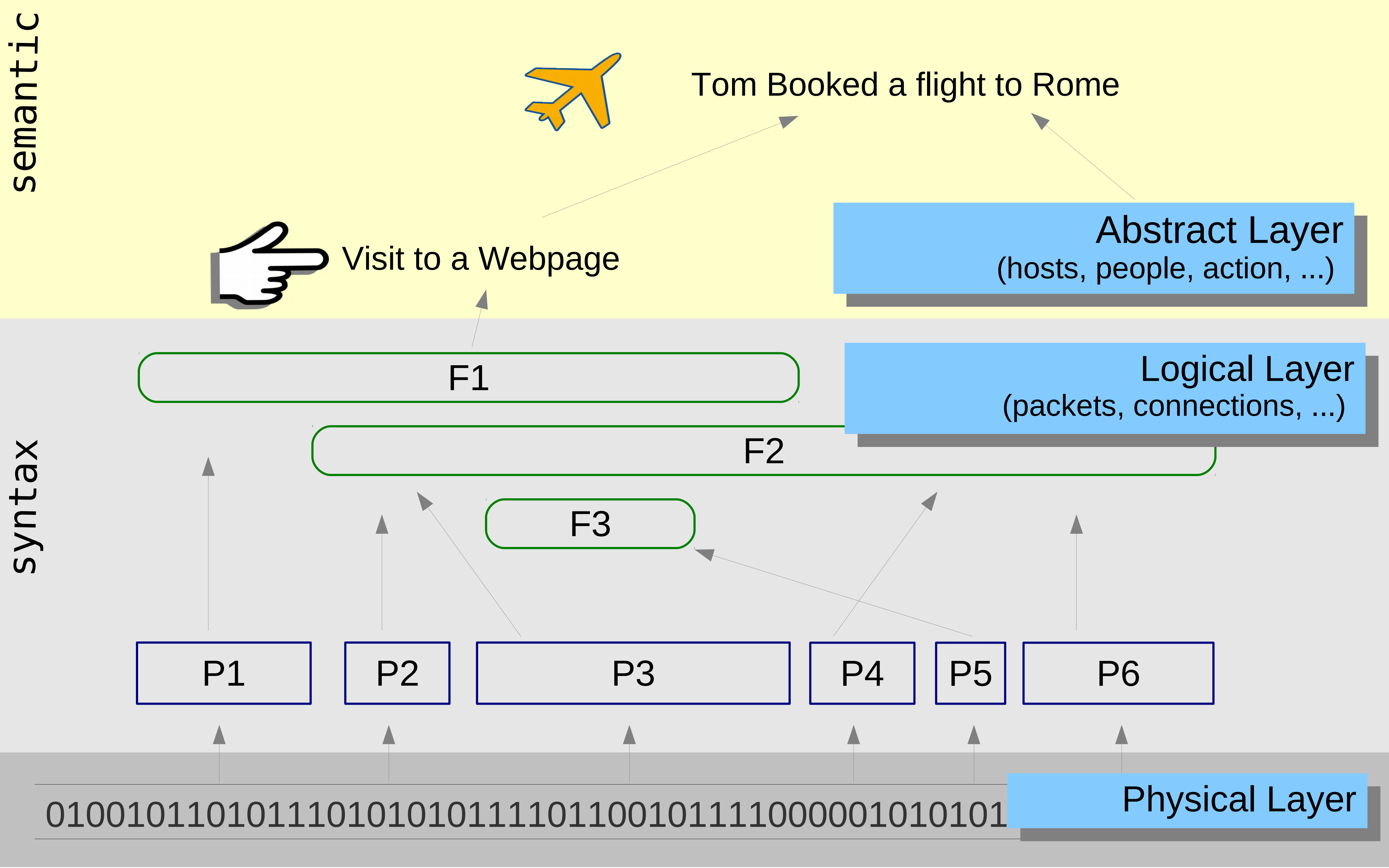
- Full packet dumps (PCAP)
  - → Gives you all the glory details
  - → Good to monitor ongoing activities (typically *case-by-case*)
  - → Really a lot of data, hard to find enough storage
- Summarized flow data (Netflows)
  - → Good to identify patterns and suspicious communication
  - → Sometime the only choice for after-the-fact analysis
- Logs and Sensors Alerts (typically text files)
  - → Good to quickly spot suspicious events

```
18708 903.510922 172.30.1.125 -> 4.2.2.1      DNS 72 Standard query A o.aolcdn.com
18709 903.561054      4.2.2.1 -> 172.30.1.125 DNS 172 Standard query response CNAME o.aolcdn.com.edgesuite.net CNAME a258.g.akamai.net A 198.87.51.27 A
198.87.51.40
18710 903.561925 172.30.1.125 -> 198.87.51.27 TCP 74 34967 > http [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=549562 TSecr=0 WS=64
18711 903.562170 172.30.1.125 -> 198.87.51.27 TCP 74 34968 > http [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=549562 TSecr=0 WS=64
18712 903.620109 198.87.51.27 -> 172.30.1.125 TCP 74 http > 34967 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1380 SACK_PERM=1 TSval=574754589
TSecr=549562 WS=32
18713 903.620402 172.30.1.125 -> 198.87.51.27 TCP 66 34967 > http [ACK] Seq=1 Ack=1 Win=14656 Len=0 TSval=549576 TSecr=574754589
18714 903.620582 172.30.1.125 -> 198.87.51.27 HTTP 616 GET /aim/img/online.gif HTTP/1.1
18715 903.622765 198.87.51.27 -> 172.30.1.125 TCP 74 http > 34968 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1380 SACK_PERM=1 TSval=574754589
TSecr=549562 WS=32
18716 903.622995 172.30.1.125 -> 198.87.51.27 TCP 66 34968 > http [ACK] Seq=1 Ack=1 Win=14656 Len=0 TSval=549577 TSecr=574754589
18717 903.623330 172.30.1.125 -> 198.87.51.27 HTTP 617 GET /aim/img/offline.gif HTTP/1.1
18719 903.680975 198.87.51.27 -> 172.30.1.125 HTTP 297 HTTP/1.1 304 Not Modified
18721 903.686355 198.87.51.27 -> 172.30.1.125 TCP 66 http > 34968 [ACK] Seq=1 Ack=552 Win=6912 Len=0 TSval=574754653 18722 903.687600 198.87.51.27 ->
172.30.1.125 HTTP 297 HTTP/1.1 304 Not Modified
18723 903.687683 172.30.1.125 -> 198.87.51.27 TCP 66 34968 > http [ACK] Seq=552 Ack=232 Win=15680 Len=0 TSval=549593 18724 904.265613 172.30.1.125 ->
64.236.68.246 TCP 74 45015 > http [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 18725 904.321296 64.236.68.246 -> 172.30.1.125 TCP 74 http > 45015
[SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1380 SACK_PERM=1 TSval=2193010808 TSecr=549738 WS=128
18726 904.321379 172.30.1.125 -> 64.236.68.246 TCP 66 45015 > http [ACK] Seq=1 Ack=1 Win=14656 Len=0 TSval=549752 TSecr=2193010808
18727 904.321663 172.30.1.125 -> 64.236.68.246 HTTP 796 GET /adiframe/3.0/5113.1/221794/0/-
1/size=728x90;noperf=1;alias=93225964;cfp=1;noaddonpl=y;artexc=all;artinc=art_image,art_img1x1,art_3pimg,art_text;kvugc=0;kvui=d104b0c6af2111e08bf5b1a4e3de47a2;kv
mn=93225964;target=_blank;aduho=-240;grp=763144798;misc=763144798 HTTP/1.1
18728 904.381376 64.236.68.246 -> 172.30.1.125 TCP 66 http > 45015 [ACK] Seq=1 Ack=731 Win=7296 Len=0 TSval=2193010867 TSecr=549752
18729 904.381998 64.236.68.246 -> 172.30.1.125 HTTP 556 HTTP/1.0 200 OK  (text/html)
18730 904.382022 172.30.1.125 -> 64.236.68.246 TCP 66 45015 > http [ACK] Seq=731 Ack=491 Win=15680 Len=0 TSval=549767 TSecr=2193010867
18731 904.404289 172.30.1.125 -> 205.188.101.136 TCP 60 58116 > http [FIN, ACK] Seq=1574 Ack=253 Win=15544 Len=0
18732 904.404360 172.30.1.125 -> 205.188.101.136 TCP 60 58114 > http [FIN, ACK] Seq=1562 Ack=249 Win=15544 Len=0
18733 904.404373 172.30.1.125 -> 205.188.101.136 TCP 60 58115 > http [FIN, ACK] Seq=1571 Ack=252 Win=15544 Len=0
18749 905.207006 205.188.90.1 -> 172.30.1.125 TCP 60 http > 34867 [ACK] Seq=251569 Ack=487540 Win=25020 Len=0
18750 905.211141 205.188.90.1 -> 172.30.1.125 TCP 60 http > 34867 [ACK] Seq=251569 Ack=487960 Win=64240 Len=0
18751 905.212168 172.30.1.125 -> 69.22.162.160 HTTP 772 GET /b?rn=41838636&C1=2&C2=1000009&C4=http%3A%2F%2Fmail.aol.
%2FBeaconReadMessagePreviewPane.htm&C5=us.memwebmail&C7=http%3A%2F%2Fmail.aol.com%2FBeacons%2Faol%2Fen-us
%2FBeaconReadMessagePreviewPane.htm&C8=Read%20Message%20-%20Preview%20Pane HTTP/1.1
```

0100101101011101010101111011001011110000010101

Physical Layer


F1
F2
F3
Logical Layer
(packets, connections, ...)
P1
P2
P3
P4
P5
P6
0100101101011101010101111011001011110000010101010
Physical Layer


semantic
syntax
Tom Booked a flight to Rome
Visit to a Webpage
Abstract Layer
(hosts, people, action, ...)
F1
Logical Layer
(packets, connections, ...)
F2
F3
P1
P2
P3
P4
P5
P6
010010110101110101010111101100101111000001010101
Physical Layer

Analysis 1: General Purpose Tools

- Packet Collectors (sniffers)
  - Goal: record packets from the network and store them on files
  - Tools: `dumpcap`, `pcapdump`, `netsniff-ng`
- Protocol Analyzers
  - Goal: packet-centric (and sometimes session-centric) inspection
  - Tools: `tcpdump`, `wireshark/tshark`, `tstat`
- Network Forensic Analysis (NFAT) tools
  - Goal: data-centric analysis of the traffic content
  - Tools: `xplico`, `NetworkMiner`

Analysis 2: Tools for Specific Tasks

- Often small programs written to do just one thing
  - Intrusion detection (`snort, suricata, bro`)
  - Match regular expressions (`ngrep`)
  - Extract files (`nfex`) or pictures (`driftnet`)
  - Sniff passwords or HTTP sessions (`dsniff, firesheep, ettercap, creds`)
  - Extract emails (`mailsnarf, smtpcat`)
  - Print network/packet statistics (`ntop, tcpstat, tstat`)
  - Extract ssl information (`ssl_dump`)
  - Reconstruct tcp flows (`tcpflow`, `tcpick`)
  - Fingerprinting (`p0f, prads`)

Analysis 3: Libraries and Frameworks

- Libpcap
  - When you need horsepower
- Scapy
  - Python library
  - Packet-level (suitable for binary protocols in particular)
  - Very flexible and easy to use
  - Painfully slow
- Bro
  - Powerful network analysis framework
  - Possible to add new analysis writing script in bro language
  - Fast and suitable for application layer analysis

Limits of Packet Analysis

- Encrypted communication are replacing plaintext protocols
  - Many tools can decrypt the traffic if the appropriate key is available (which is seldom the case)
  - Although we can not see the content of the data we can sometimes infer some information about it by looking at metadata
    - Traffic volumes
    - Timing
    - Response times

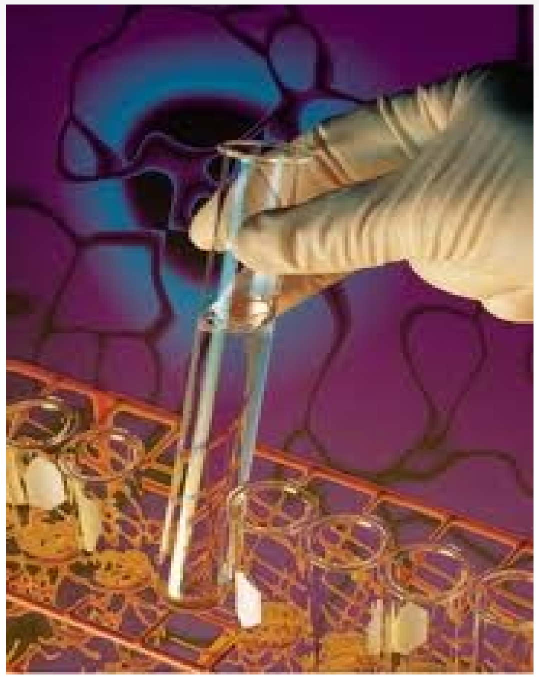
Reconstructing TCP/IP Flows

- Goal: reconstruct the TCP flow as it is *seen* and *reassembled* by the target machine
- Problem – it is hard to know which packet will be received by the target
  - Malformed packets can be discarded
  - Packets with short TTL or large size may not reach the destination
  - The system may discard packets with old timestamps or containing certain options
- Problem – there are multiple possible ways in which the target Operating System can reassemble the data
  - IP packets may arrive fragmented and out of order
  - Fragments may partially or completely overlap

Data Collection

Data Collection

- Physically intercept network traffic
- Acquire a network dump file
- `dumpcap`
- `pcap` data format
- NetFlows



Intercepting Network Traffic

- SPAN Ports (or port mirroring) are used on network switches to receive a copy of the traffic seen on another port (or VLAN)

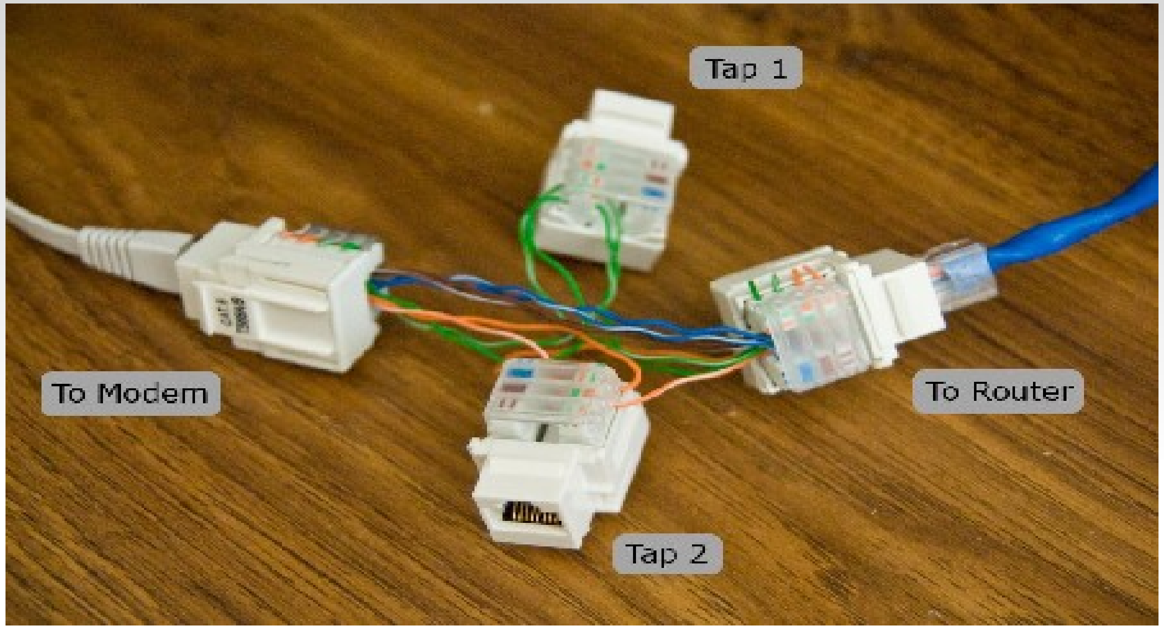
- SPAN ports rely on active packet duplication
  - ✗ Hardware errors are likely dropped
  - ✗ If traffic exceeds the SPAN link capacity, packets are dropped
  - ✔ Both traffic directions will be copied into the same port

Intercepting Network Traffic

- Taps (Test Access Ports) are passive devices that duplicate all traffic on the link and forward it to the monitoring port/s

- Packets are copied electrically or optically to the tap ports
  - ✔ Any monitoring device connected to a tap receives the same traffic as if it were in-line, from layer 1 up, including all errors
  - ✔ Taps do not introduce delay and they do not alter the content or structure of the data
  - ✔ Taps normally fail open – i.e., traffic continues to flow between network devices in the event a monitoring device is removed or power to the device is lost
  - ✗ Taps normally have one port per direction

Garage Solution



http://hackaday.com/2008/09/14/passive-networking-tap/

http://www.instructables.com/id/Make-a-Passive-Network-Tap/

Listen Carefully, and Make No Noise

- The monitoring device should never send any packet
- Two options
  - Use a network cable with disconnected TX wires
  - Configure the network interface with no address and make sure it will not transmit any packets at all

```
$ ifconfig eth0 hw ether 00:00:00:00:00:00 promisc
$ ifconfig eth0 -arp up
$ route -nv  // the output should be empty
```

Sniffing Your Own Traffic

- On recent network cards, checksums and fragmentation can be implemented in hardware
- Using a packet sniffer on a host that is participating in the traffic being sniffed may result in packets with :
  - bad CRC checksums due to CRC off-loading
  - weird packet sizes due to TCP segmentation off-loading
- Always better to use a separate machine
  - In case you cannot, you can check and disable the offloading

```
 $ ethtool -k eth0

 $ ethtool --offload eth0 rx tx sg tso ufo gso gro lro rxvlan txvlan
rxhash off
```

(in my experience, it does not always work)

dumpcap

- Program based on `libpcap` to read packets from a live network and store them on disk
- By default it saves the file in `pcap-ng` format.
  Use the `-P` option to revert to the plain `pcap` format
- Common options:

  `-f “capture filter”` – specify a pcap capture filter

  `-s size` – the max length of each captured packet (default 64K)

  `-w filename` – set the file where the packets are stored

  `-I` – put the wi-fi interface in monitor mode

  `-i interface` – select the interface to use for capture

`dumpcap`

- Common options:
  - `-a condition` – stop sniffing when condition is true
  - `-b condition` – cycle to next file when condition is true
- Conditions:
  - `duration:value` – after a certain number of seconds
  - `filesize:value` – after the file size reach a certain value
  - `files:value` – after a certain number of files have been written (stop with `-a` or restart from the first one with `-b`)
- Example:

```
-b filesize:1024 -b files:5
```

Store the traces in a ring buffer of five files of one megabyte each

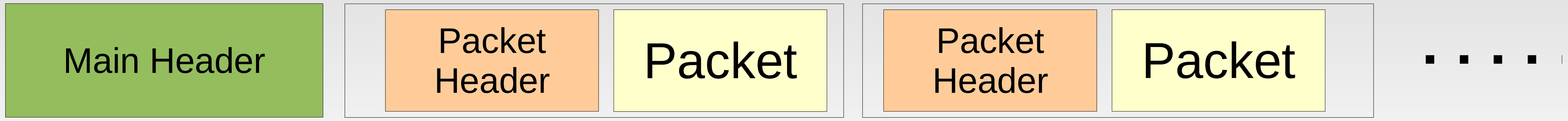
Capture Filters

- Most packet collectors allow users to specify a filter to limit the collection to the packets of interest
- Berkeley Packet Filtering (BPF) language
  - Introduced in 1992 and integrated in `tcpdump`
  - Supported by many command line tools
- Available qualifiers:
  - Type: `host, net, port, portrange`
  - Direction: `src, dst`
  - Protocol: `ether, ip, arp, ..`

```
dumpcap -i eth0 -f "tcp port 80 and dst net 10.1.1"
```

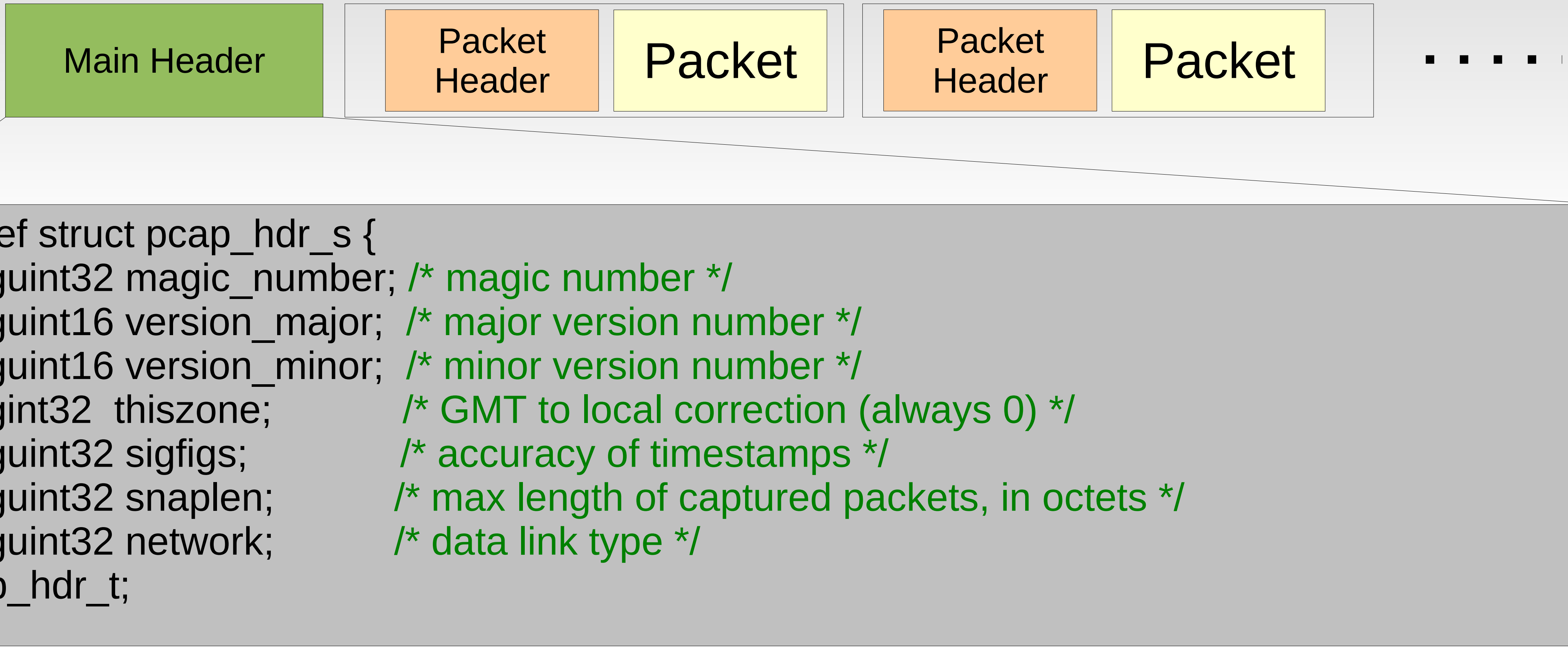
pcap Format

- Very simple format to store network traffic
- De-facto standard for network capturing on Linux



pcap Format

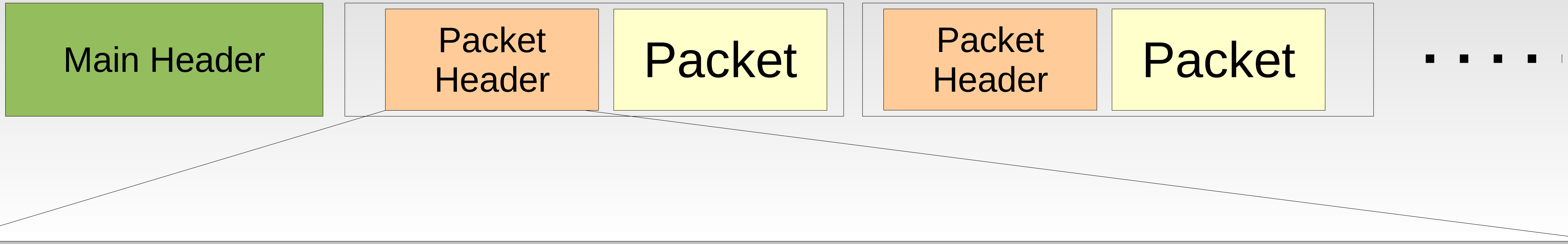
- Very simple format to store network traffic
- De-facto standard for network capturing on Linux



```
typedef struct pcap_hdr_s {
        guint32 magic_number; /* magic number */
        guint16 version_major;  /* major version number */
        guint16 version_minor;  /* minor version number */
        gint32  thiszone;           /* GMT to local correction (always 0) */
        guint32 sigfigs;            /* accuracy of timestamps */
        guint32 snaplen;           /* max length of captured packets, in octets */
        guint32 network;           /* data link type */
} pcap_hdr_t;
```

pcap Format

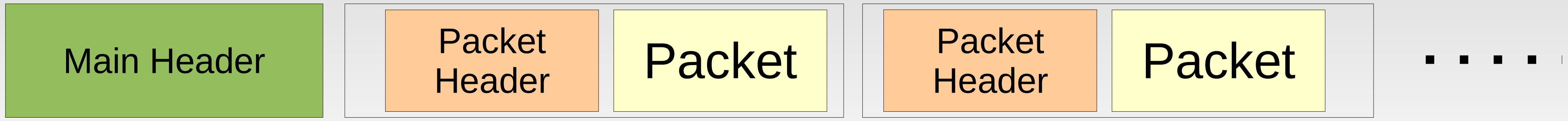
- Very simple format to store network traffic
- De-facto standard for network capturing on Linux



```
typedef struct pcaprec_hdr_s {
        guint32 ts_sec;         /* timestamp seconds */
        guint32 ts_usec;        /* timestamp microseconds */
        guint32 incl_len;       /* number of octets of packet saved in file */
        guint32 orig_len;       /* actual length of packet */
} pcaprec_hdr_t;
```

pcap Format

- Very simple format to store network traffic
- De-facto standard for network capturing on Linux



- Problems
  - ✗ No metadata
  - ✗ No information about the capturing interface
  - ✗ No info about dropped packets
  - ✗ No timezone (timestamps stored in UTC)

    - Protocol analyzers may display it in local time
  - ✗ No way to seek to a certain packet :(

pcapng

- Internet Engineering Task Force (IETF) Internet Draft to specify a next generation `pcap` file format
- Advantages
  - ✓ Supports multiple interface
  - ✓ Better timestamps
  - ✓ Add comments to individual packets
  - ✓ Store metadata
  - ✓ Extensible file format
  - ✓ … and more

NetFlows

- Network protocol developed by CISCO to collect summarized IP traffic information
    - Supported by most of the routers and switches
    - To reduce the burden on the network device, netflows are often sampled (one out of 10, 100, 10.000...)
    - When a flow is completed, the router sends it (in a UDP packet) to a remote netflow collector
- A flow is “completed” when no traffic for it is observed for a certain amount of time
    - TCP session termination in a TCP flow causes the router to expire the flow
- Good to save disk space and still maintain general information about the traffic

NetFlows

- A flow is a (typically) unidirectional sequence of packets that share the same:
  - IP Source and Destination addresses
  - Source and Destination port for UDP and TCP
  - IP protocol
  - Ingress interface

    !! A TCP connection is typically summarized in two separate flows!!

- A netflow record generally contains
  - Timestamp
  - Number of bytes in the flow
  - Union of all TCP flags observed over the life of the flow
  - It does NOT contain the entire payload

NetFlow Collection and Analysis

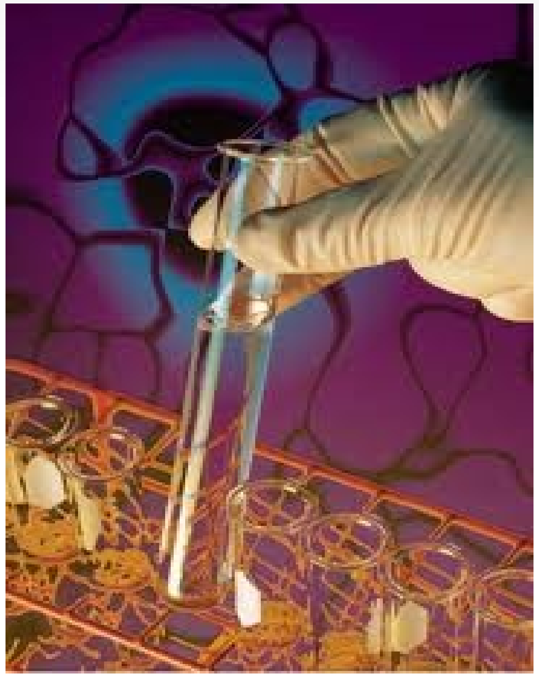
- NetFlows are collected by network devices and then sent (over the network) to one or more listeners
  - The exact collection and transmission setup depends on the device

- Plenty of tools to collect, filter, analyze, query, anonymize, plot, archive, aggregate, … netflows
  - SiLK suite (http://tools.netsa.cert.org/silk/)
  - Argus suite (http://qosient.com/argus/index.shtml)
  - Other tools (http://www.networkuptime.com/tools/netflow/)

Data Analysis

Data Analysis

- General information and metadata
  - General info with `capinfos`
  - Fixing a dump with `editcap`
- Packets and connections analysis
  - `wireshark` / `tshark`
  - Protocol Identification
  - IP address resolution
  - Playing with flows with `tcpick` or `tcpreassembly`



capinfos

```
> capinfos  dump.pcap
File name:               dump.pcap
File type:               Wireshark/tcpdump/... - libpcap
File encapsulation:      Ethernet
Packet size limit:       file hdr: 65535 bytes
Number of packets:       1680
File size:               1161077 bytes
Data size:               1134173 bytes
Capture duration:        14 seconds
Start time:              Thu Feb  9 22:22:03 2012
End time:                Thu Feb  9 22:22:17 2012
Data byte rate:          79074.89 bytes/sec
Data bit rate:           632599.14 bits/sec
Average packet size:     675.10 bytes
Average packet rate:     117.13 packets/sec
SHA1:                    dd90d29d389ff14d87bd897ba46fc5b6b22139fc
RIPEMD160:               03350b1b658715fe2b4196201618c12a58f40e07
MD5:                     c244ccce2e6349d4930ac4b75291e17d
Strict time order:       True
```

Edit a pcap file with `editcap`

- Translate in 27 different formats

```
editcap -F libpcap file.pcapng file.pcap
```

Edit a pcap file with `editcap`

- Translate in 27 different formats

```
editcap -F libpcap file.pcapng file.pcap
```

- Limit to a certain time window

`-A starttime  -B stoptime` (YYYY-MM-DD HH:MM:SS)

Edit a pcap file with `editcap`

- Translate in 27 different formats

```
editcap -F libpcap file.pcapng file.pcap
```

- Limit to a certain time window

`-A starttime  -B stoptime` (YYYY-MM-DD HH:MM:SS)

- Limit to certain packets

`-r p1 p2..` (individual packets)

`-r p1-p2 p3-p4` (ranges)

Without the `-r` it writes all packets EXCLUDED the listed ones

Edit a pcap file with `editcap`

- Translate in 27 different formats

  ```
  editcap -F libpcap file.pcapng file.pcap
  ```

- Limit to a certain time window

  `-A starttime  -B stoptime` (YYYY-MM-DD HH:MM:SS)

- Limit to certain packets

  `-r p1 p2..` (individual packets)

  `-r p1-p2 p3-p4` (ranges)

  Without the `-r` it writes all packets EXCLUDED the listed ones

- Fix time offsets

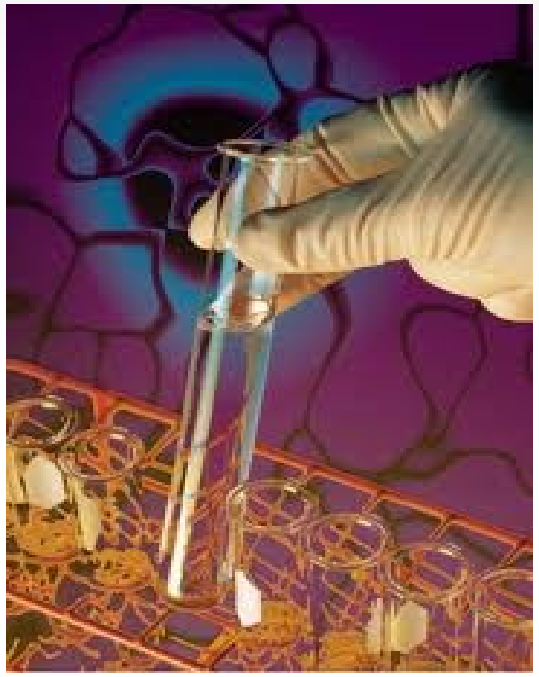
  `-t 3.08` (move all packets forward of 3.08 seconds)

Data Analysis

General information and metadata

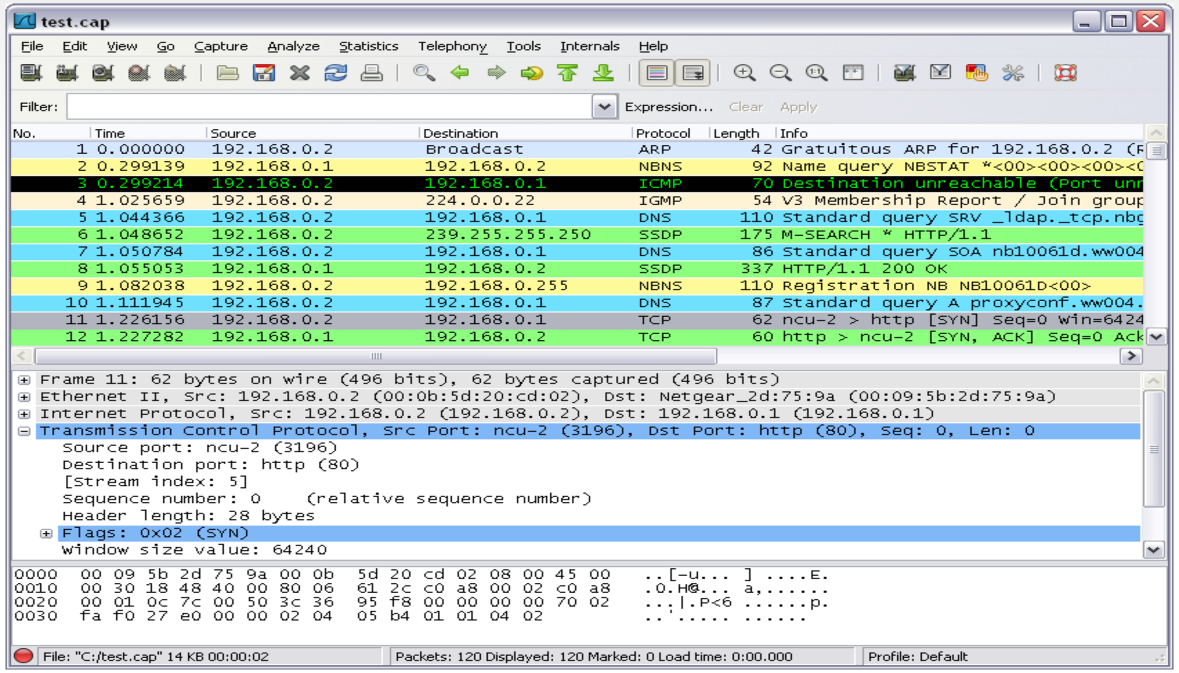
- General info with `capinfos`
- Fixing a dump with `editcap`

- Packets and connections analysis
  - `wireshark` / `tshark`
  - Protocol Identification
  - IP address resolution
  - Playing with flows with `tcpick` or `tcpreassembly`



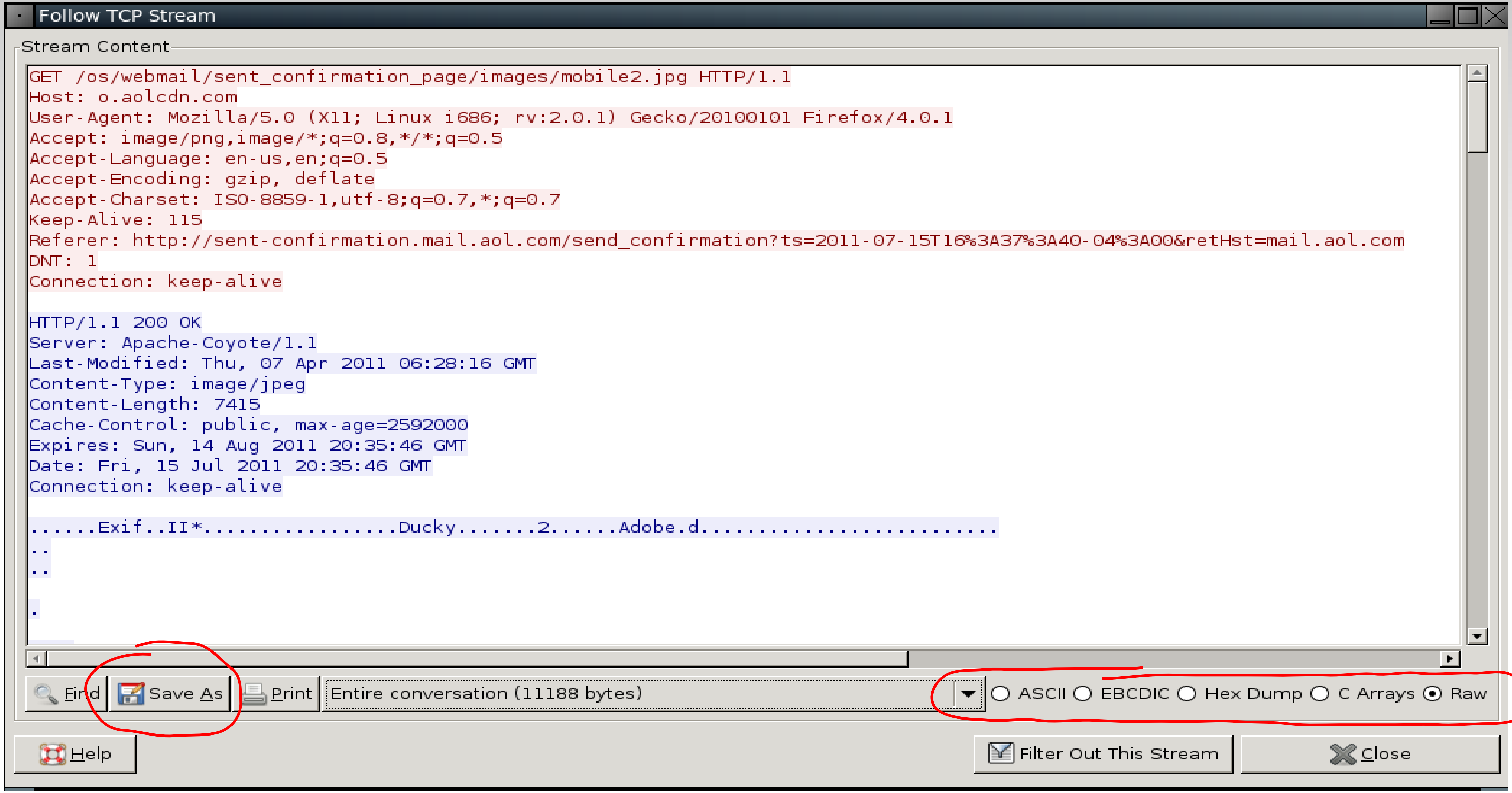
wireshark

- Cross-platform packet analyzer with a nice graphical interface
  - Over 1.100 supported/dissected protocols
    (available protocols and fields: http://www.wireshark.org/docs/dfref/)
- A command-line version is available under the name `tshark`


test.cap
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply

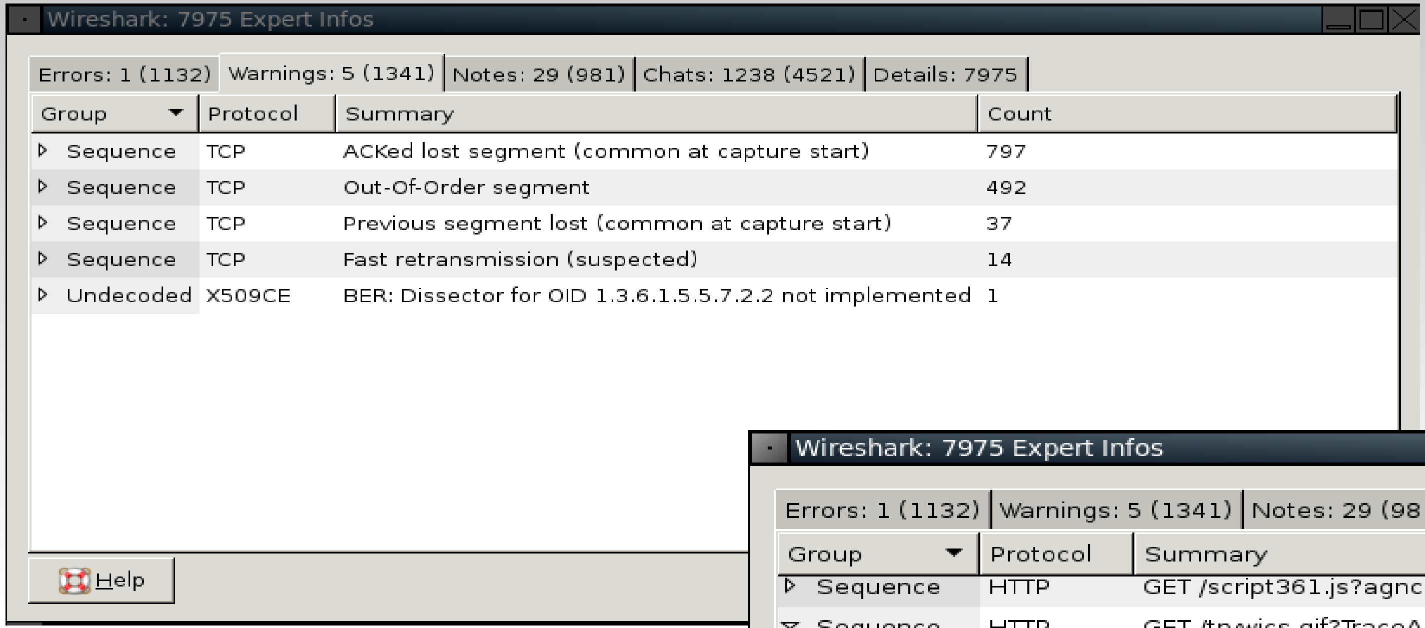
```
No.  Time       Source         Destination      Protocol  Length Info
   1 0.000000   192.168.0.2    Broadcast        ARP          42 Gratuitous ARP for 192.168.0.2 (R
   2 0.299139   192.168.0.1    192.168.0.2      NBNS         92 Name query NBSTAT *<00><00><00><0
   3 0.299214   192.168.0.2    192.168.0.1      ICMP         70 Destination unreachable (Port unr
   4 1.025659   192.168.0.2    224.0.0.22       IGMP         54 V3 Membership Report / Join group
   5 1.044366   192.168.0.2    192.168.0.1      DNS         110 Standard query SRV _ldap._tcp.nbg
   6 1.048652   192.168.0.2    239.255.255.250  SSDP        175 M-SEARCH * HTTP/1.1
   7 1.050784   192.168.0.2    192.168.0.1      DNS          86 Standard query SOA nb10061d.ww004
   8 1.055053   192.168.0.1    192.168.0.2      SSDP        337 HTTP/1.1 200 OK
   9 1.082038   192.168.0.2    192.168.0.255    NBNS        110 Registration NB NB10061D<00>
  10 1.111945   192.168.0.2    192.168.0.1      DNS          87 Standard query A proxyconf.ww004.
  11 1.226156   192.168.0.2    192.168.0.1      TCP          62 ncu-2 > http [SYN] Seq=0 Win=6424
  12 1.227282   192.168.0.1    192.168.0.2      TCP          60 http > ncu-2 [SYN, ACK] Seq=0 Ack

⊞ Frame 11: 62 bytes on wire (496 bits), 62 bytes captured (496 bits)
⊞ Ethernet II, Src: 192.168.0.2 (00:0b:5d:20:cd:02), Dst: Netgear_2d:75:9a (00:09:5b:2d:75:9a)
⊞ Internet Protocol, Src: 192.168.0.2 (192.168.0.2), Dst: 192.168.0.1 (192.168.0.1)
⊟ Transmission Control Protocol, Src Port: ncu-2 (3196), Dst Port: http (80), Seq: 0, Len: 0
    Source port: ncu-2 (3196)
    Destination port: http (80)
    [Stream index: 5]
    Sequence number: 0    (relative sequence number)
    Header length: 28 bytes
  ⊞ Flags: 0x02 (SYN)
    Window size value: 64240

0000  00 09 5b 2d 75 9a 00 0b  5d 20 cd 02 08 00 45 00   ..[-u... ] ....E.
0010  00 30 18 48 40 00 80 06  61 2c c0 a8 00 02 c0 a8   .0.H@... a,......
0020  00 01 0c 7c 00 50 3c 36  95 f8 00 00 00 00 70 02   ...|.P<6 ......p.
0030  fa f0 27 e0 00 00 02 04  05 b4 01 01 04 02         ..'..... ......

File: "C:/test.cap" 14 KB 00:00:02   Packets: 120 Displayed: 120 Marked: 0 Load time: 0:00.000   Profile: Default
```

Follow TCP Stream

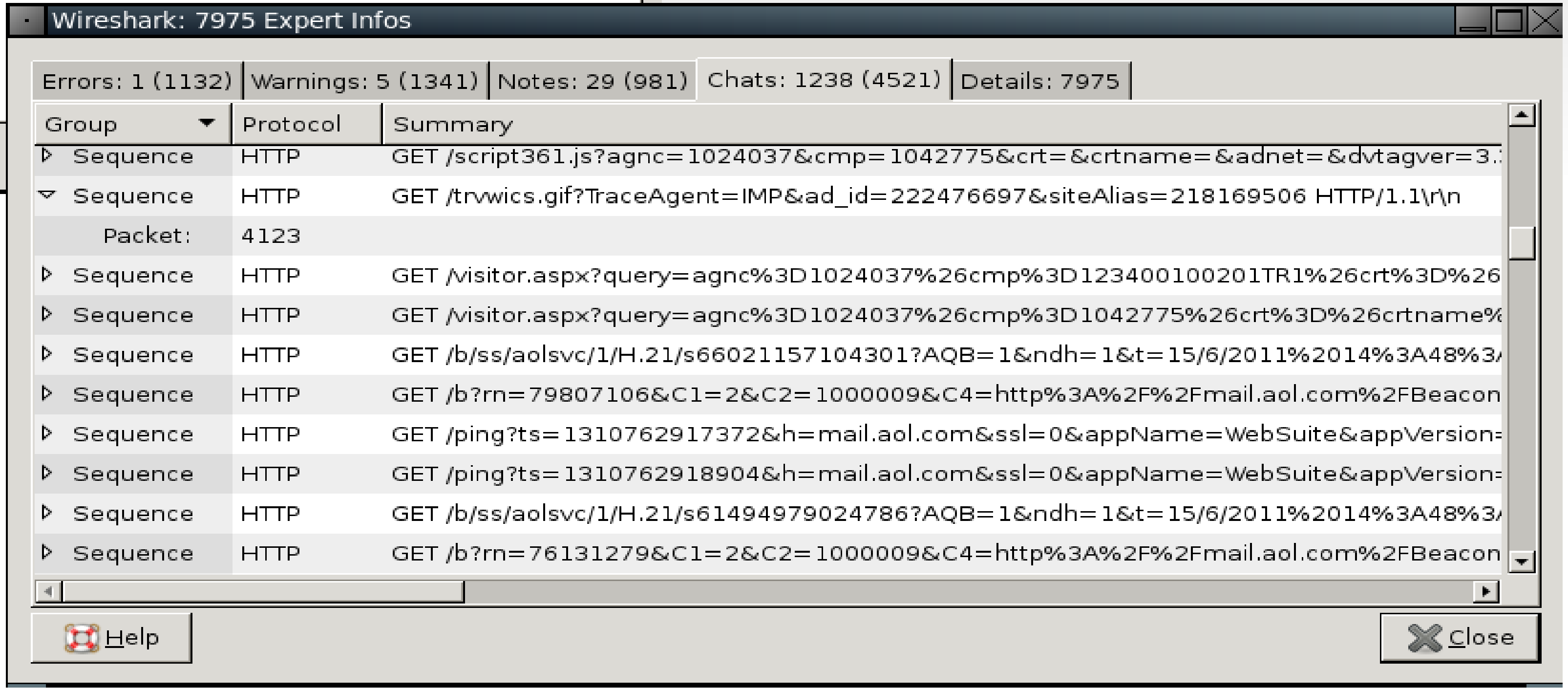


```
GET /os/webmail/sent_confirmation_page/images/mobile2.jpg HTTP/1.1
Host: o.aolcdn.com
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:2.0.1) Gecko/20100101 Firefox/4.0.1
Accept: image/png,image/*;q=0.8,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Referer: http://sent-confirmation.mail.aol.com/send_confirmation?ts=2011-07-15T16%3A37%3A40-04%3A00&retHst=mail.aol.com
DNT: 1
Connection: keep-alive

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Last-Modified: Thu, 07 Apr 2011 06:28:16 GMT
Content-Type: image/jpeg
Content-Length: 7415
Cache-Control: public, max-age=2592000
Expires: Sun, 14 Aug 2011 20:35:46 GMT
Date: Fri, 15 Jul 2011 20:35:46 GMT
Connection: keep-alive

......Exif..II*................Ducky.......2......Adobe.d.........................
..
..

.
```

Expert Infos



Group	Protocol	Summary	Count
Sequence	TCP	ACKed lost segment (common at capture start)	797
Sequence	TCP	Out-Of-Order segment	492
Sequence	TCP	Previous segment lost (common at capture start)	37
Sequence	TCP	Fast retransmission (suspected)	14
Undecoded	X509CE	BER: Dissector for OID 1.3.6.1.5.5.7.2.2 not implemented	1



Group	Protocol	Summary
Sequence	HTTP	GET /script361.js?agnc=1024037&cmp=1042775&crt=&crtname=&adnet=&dvtagver=3.
Sequence	HTTP	GET /trvwics.gif?TraceAgent=IMP&ad_id=222476697&siteAlias=218169506 HTTP/1.1\r\n
Packet:	4123	
Sequence	HTTP	GET /visitor.aspx?query=agnc%3D1024037%26cmp%3D123400100201TR1%26crt%3D%26
Sequence	HTTP	GET /visitor.aspx?query=agnc%3D1024037%26cmp%3D1042775%26crt%3D%26crtname%
Sequence	HTTP	GET /b/ss/aolsvc/1/H.21/s66021157104301?AQB=1&ndh=1&t=15/6/2011%2014%3A48%3
Sequence	HTTP	GET /b?rn=79807106&C1=2&C2=1000009&C4=http%3A%2F%2Fmail.aol.com%2FBeacon
Sequence	HTTP	GET /ping?ts=1310762917372&h=mail.aol.com&ssl=0&appName=WebSuite&appVersion=
Sequence	HTTP	GET /ping?ts=1310762918904&h=mail.aol.com&ssl=0&appName=WebSuite&appVersion=
Sequence	HTTP	GET /b/ss/aolsvc/1/H.21/s61494979024786?AQB=1&ndh=1&t=15/6/2011%2014%3A48%3
Sequence	HTTP	GET /b?rn=76131279&C1=2&C2=1000009&C4=http%3A%2F%2Fmail.aol.com%2FBeacon

tshark

```
> tshark [-n] -r pcapfile <options>
```

`-n` disable name resolution

- Same engine of wireshark
- Excellent to
  - Selectively print only certain fields or information
  - Use in combination with other command line tools
  - Script simple analysis

tshark

- Display (`-Y "…"`) filters

  `ip.addr==128.0.0.1 and (tcp.port==80 || udp.port==80)`

  `not arp` (filter out a protocol)

  `ip.src==192.168.0.0/16` (subnetwork)

  `frame.number==8` (useful with -V)

  `tcp.stream==55` (all packets in that stream)

  `tcp contains "foobar"` ← string

  `http.request.uri matches "gl=se$"` ← regular expression

- Use `tshark -G fields` for a complete list of available fields

`tshark`

- Controlling and customizing the output
  - `-V` – show the entire protocol tree (with all subtree expanded)
  - `-x` – add the packet bytes in hexdump format
  - `-T fields` – customize the output to certain fields

    `-E separator=,` – specify the field separator

    `-e <field1> -e <field2>` … – which field we want to print

    `Example: -T fields -e data |  xxd -r -p`
  - `-w filename` – write to packets back to another file

- Setting a particular protocol decoder (for non-standard ports)

  `-d tcp.port==1234,http`

tshark

```
> tshark -T fields -e dns.qry.name -Y "dns.flags.rcode==3" -r <file>

anologo0094.net
nologo1093.com
nologo1093.com.hsd1.va.comcast.net
nologo0094.net
nologo1093.com
nologo1093.com.hsd1.va.comcast.net
nologo0094.net
nologo1093.com
nologo1093.com.hsd1.va.comcast.net
nologo0094.net
nologo1093.com
nologo1093.com.hsd1.va.comcast.net
nologo0094.net
nologo1093.com
nologo1093.com.hsd1.va.comcast.net
nologo0094.net
nologo1093.com
nologo1093.com.hsd1.va.comcast.net
nologo0094.net
nologo1093.com
nologo1093.com.hsd1.va.comcast.net
```

`tshark` analysis

- Like `wireshark`, `tshark` automatically performs some analysis on the traffic
- Eg., if you are looking for some weird TCP problem:

```
$ tshark -G | grep tcp.analysis

$ tshark -r file.pcap -R tcp.analysis.flags
         -T fields -e tcp.stream
         -e tcp.analysis.out_of_order
         -e tcp.analysis.reused_ports
         -e tcp.analysis.lost_segment
```

- Certain fields can only be computed using a two-pass analysis (`-2`)
- You can access the “expert info” messages in tshark with:

```
$ tshark -T fields -e _ws.expert …

$ tshark -Y "_ws.expert.severity eq Warn" …
```

`tshark` Statistics

- The `-z` option can be used to print different statistics
  - Can be used multiple times in the same command
  - Use with `-q` if you want to see ONLY the statistics
- Conversations: `-z conv,<proto>,<filter>`
  - `<proto>` : eth, ip, tcp, udp (a filter can specify a sub-protocol)
  - Ex: `-q -z conv,ip,tcp.port==80`
- Hosts: `-z ip_hosts,tree`
- Protocol Hierarchy: `-z io,phs[,filter]`
- Html stats: `-z http,tree`

  `-z http_req,tree`

`tshark` Statistics

- IO: `-z io,stat,<time>,[filter],[filter],...`
  - `<time>` : time interval in seconds
  - Each filter is shown in a separate column
  - Filters can also contains aggregation functions

    ```
    <function>(<field>)<filter>
    <function> = COUNT | SUM | MIN | MAX | AVG
    ```
  - Ex:

    ```
    -q -z io,stat,10,tcp.port==80,tcp.port==22

    -q -z io,stat,10, "COUNT(tcp.analysys.retransmission)tcp.analysis.retransmission
                && ip.src==10.0.0.1", "AVG(tcp.analysis.ack_rtt)tcp.analysis.ack_rtt"
    ```
- Hosts: `-z hosts`
  - Export the collected info in hosts format

Packet Timestamps

- Timestamp format
  - `-t  ad | a | r | d | dd | e`
    - `ad` – absolute date and time
    - `a` – absolute time
    - `r` – time relative to the first packet
    - `d` – time delta with respect to the previous *captured* packet
    - `dd` – time delta since the previous *diplayed* packet
    - `e` – epoch (seconds since Jan 1, 1970)
- Example: `-t dd -Y "tcp.stream==18"`

tshark

- Setting options: `-o <option>:<value>`
- Examples:

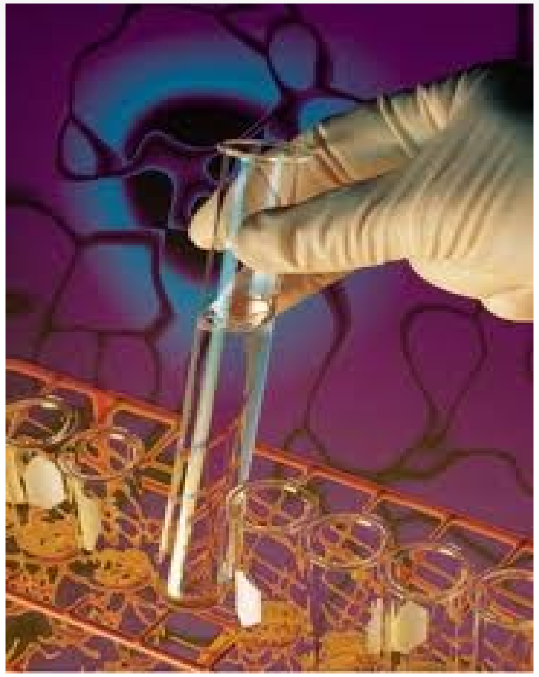
```
tcp.relative_sequence_numbers:FALSE

tcp.desegment_tcp_streams:TRUE

wlan.enable_decryption:TRUE
wlan.wep_key1:wpa-psk: 55f8e415485dd9a272060c...
```

- For a complete list, check the file

```
~/.wireshark/preferences
```

Data Analysis

- General information and metadata
  - General info with `capinfos`
  - Fixing a dump with `editcap`
- Packet and connection analysis
  - `wireshark` / `tshark`
  - Protocol Identification
  - IP address resolution
  - Playing with flows with `tcpick`



Protocol Identification

- The first step in a network trace analysis is often the identification of the protocols
  - Some protocols have fields that contain the type of the inner protocol
  - In wireshark, you can check the `frame.protocols` field
  - For TCP & UDP, most of the tools rely on the destination port number
    - Standard list under `/etc/services/`
    - Sometimes users can “force” a particular non-standard association
- Some tools use deep packet inspection to detect signs of known protocols (aka *Port Independent Protocol Identification*)
  - Library: `nDPI` *(new project part of Google summer of code 2017)*
  - Tools: `tstat, xplico, networkminer professional, bro`

Resolving IP Addresses

- Using reverse DNS name resolution
  - Easy
  - Not always reliable
  - Reverse lookup not always available
- Analyzing the DNS requests in the network trace
  - Very accurate
  - Requires an analysis of the trace
  - Some of the domains could have been resolved before the capture started
- Using online historical services
  - Slow and difficult to scale
  - May provide results also when reverse lookup is not available

tcpick

- Command line packet and flow analyzer
  - Designed to be chained with other commands (grep, sed, awk,...)
  - Colored output is also good for humans

```
$ tcpick -r filename [options] ["filter"]
```

`-C`	: use colors
`-t`	: pre-pend the timestamp
`-yP`	: show the tcp payload (replace non-printable characters with dots)
`-yR`	: show the raw tcp payload

Extracting Flows

- `tcpick` can reorder the tcp packets, discard the duplicate, and rebuild the flows

  `-wR[C|S]` : write the reconstructed flows to files
  [C=client, S=server only]
  (one file per direction per flow)

  `-wR[C|S]u` : as before, but the two directions are merged
  in one file

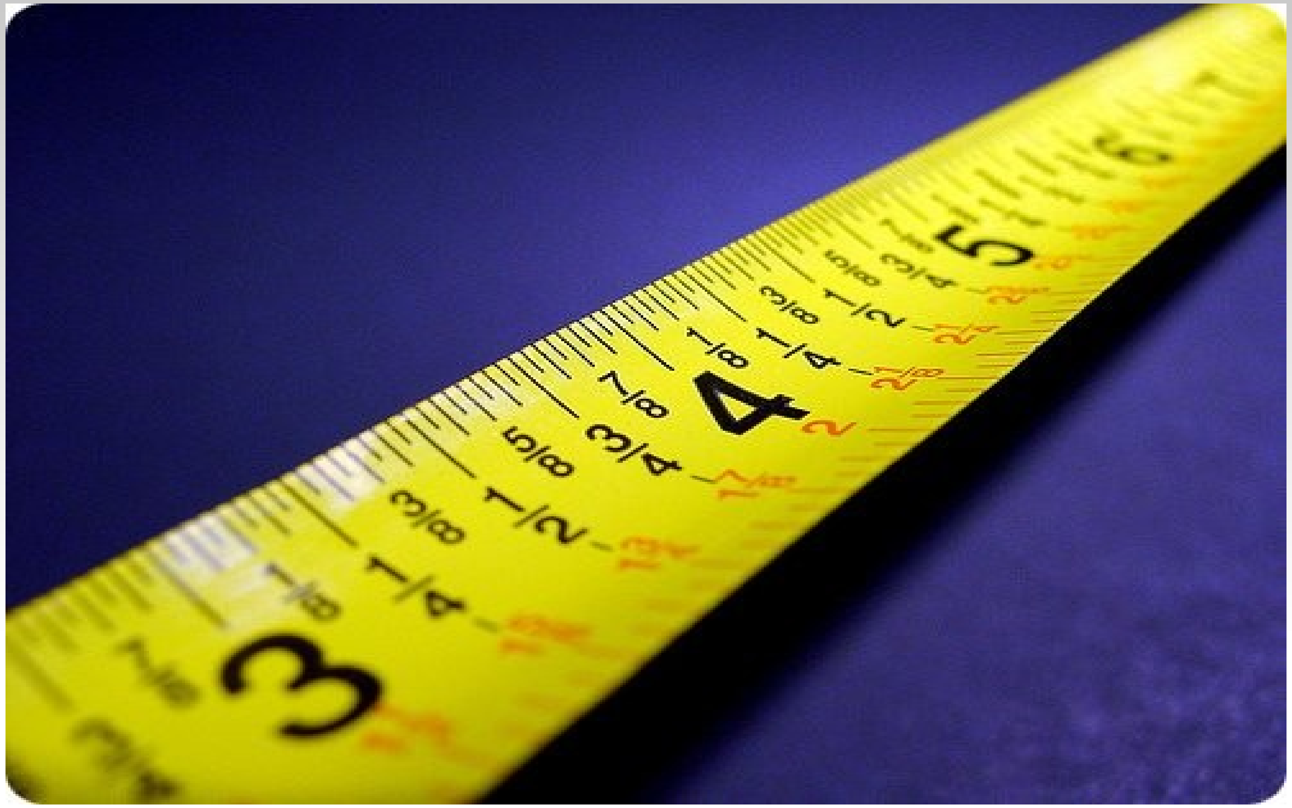
  `-wR[C|S]ub` : add also a banner to separate the two directions

- To pipe to other tools the reassembled streams:

  `-b` : same use and options of -w but print to standard
  output instead of saving to file

TcpReassembly

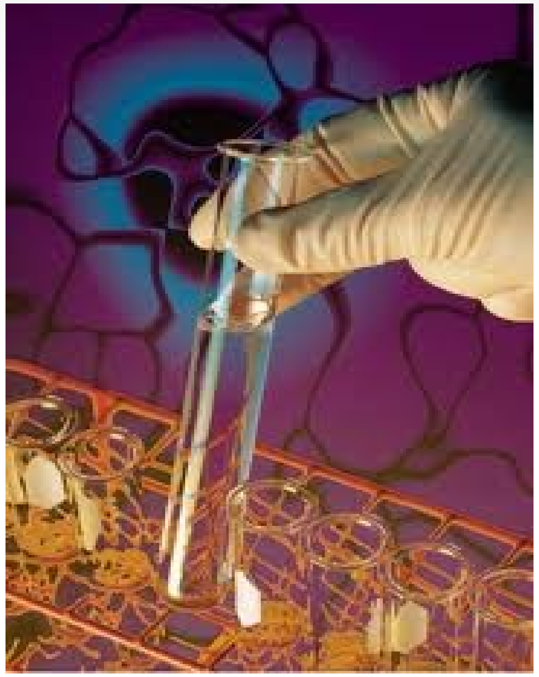
- Comes as an example of the `pcapplusplus` library
  - Application to reconstruct the TCP data streams and stores each connection (or side of a connection) in a separate file(s).
  - Creates little metadata files for each connection
  - Handles TCP retransmission, out-of-order packets and packet loss
  - Just a single C++ file of few hundred lines



Enrichment and Fingerprinting

Summary

- Introduction
- Data Collection
- Traffic Analysis
- Enrichment and Fingerprinting
  - Geolocation
  - OS/Service fingerprinting
  - NAT
- Custom Analysis
- Malware Corner
- Malware Network Patterns



IP GeoLocation

- GeoLocation is the process of mapping network addresses (IPs) back to the real world
  - No black magic, it just requires an up-to-date database
- Many free services on the web
  - Good for few lookups, not for scripting
- Many commercial services and softwares but
   very few free databases

```
$ apt-get install geoip-database geoip-bin
> geoiplookup 193.55.113.231
 GeoIP Country Edition: US, United States
 GeoIP Country Edition: FR, France
```

```
> dpkg -L geoip-database
...
/usr/share/GeoIP
/usr/share/GeoIP/GeoIP.dat
/usr/share/GeoIP/GeoIPv6.dat

> wget http://geolite.maxmind.com/download/geoip/database/GeoLiteCity.dat.gz
> gunzip GeoIPCity.dat.gz
$ cp GeoLiteCity.dat /usr/share/GeoIP/GeoIPCity.dat
> geoiplookup 193.55.113.231
GeoIP Country Edition: FR, France
GeoIP City Edition, Rev 1: FR, B8, Juan-les-pins, N/A, 43.566700,
7.100000, 0, 0
GeoIP City Edition, Rev 0: FR, B8, Juan-les-pins, N/A, 43.566700,
7.100000
```

OS Fingerprinting

- The process of identifying the operating system running on a machine by analyzing various network information
- It can be done in active mode
  - By directly talking to the machine to identify and see how it answers to particularly crafted packets
- … or in a passive mode
  - by just analyzing the captured network traffic

    → More interesting for forensics

- Useful to identify user machines, or to detect multiple computers behind a NAT

Passive Fingerprinting

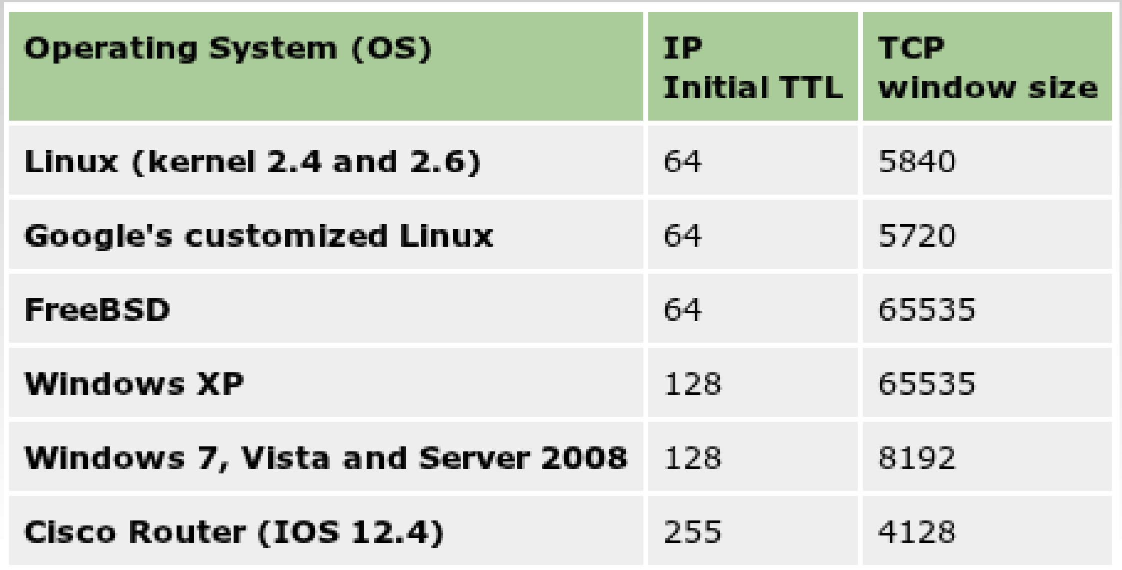
-- It's like listening to other people's accents in a Cafe' --

- TCP/IP fingerprinting
  - Based on the information leaked by the `tcp/ip` stack implementation
- Different responses to ICMP messages
- DCHP messages
- Application Layer OS Fingerprinting
  - Based on application layer protocols
  - Service banners (HTTP, SSH, FTP, ...)
  - Browser User-Agent strings

TCP/IP Fingerprinting

- The initial value of certain TCP parameters are chosen in a different way by different implementations
- Different operating systems (and different versions of the same operating system) use different defaults for several fields:
  - Initial packet size (16 bits)
  - Initial TTL (8 bits)
  - Window size (16 bits)
  - Max segment size (16 bits)
  - Window scaling value (8 bits)
  - "don't fragment" flag (1 bit)
  - "sackOK" flag (1 bit)
  - "nop" flag (1 bit)

TCP/IP Fingerprinting



Operating System (OS)	IP Initial TTL	TCP window size
Linux (kernel 2.4 and 2.6)	64	5840
Google's customized Linux	64	5720
FreeBSD	64	65535
Windows XP	128	65535
Windows 7, Vista and Server 2008	128	8192
Cisco Router (IOS 12.4)	255	4128

```
> tshark -r case09.pcap -Y "tcp.flags.syn eq 1" -T fields -e ip.src
-e ip.ttl -e tcp.window_size
...
192.168.230.1   128    65535
192.168.230.4   128    16384
10.1.1.6         63    65535
...
```

TCP/IP Fingerprinting

- `p0f` – written by Michael Zalewski in 2001
  - Version 3 released in January 2012
  - http://lcamtuf.coredump.cx/p0f3/

```
> p0f -r case09.pcap
.-[ 192.168.230.4/1027 -> 192.168.230.1/139 (syn) ]-
|
| client   = 192.168.230.4/1027
| os       = Windows XP
| dist     = 0
| params   = none
| raw_sig  = 4:128+0:0:1460:16384,0:mss,nop,nop,sok:df,id+:0
`----
.-[ 10.1.1.6/65067 -> 192.168.230.4/445 (syn) ]-
|
| client   = 10.1.1.6/65067
| os       = FreeBSD 8.x
| dist     = 1
| params   = none
| raw_sig  = 4:63+1:0:1460:65535,3:mss,nop,ws,sok,ts:df,id+:0
`----
```

Application Fingerprinting

```
...
-[ 192.168.230.4/1036 -> 64.233.169.147/80 (http response) ]-
|
| server    = 64.233.169.147/80
| app       = Google Web Server
...
```

- Browser User-Agent strings
  - http://www.zytrax.com/tech/web/browser_ids.htm
  - http://www.useragentstring.com/pages/useragentstring.php

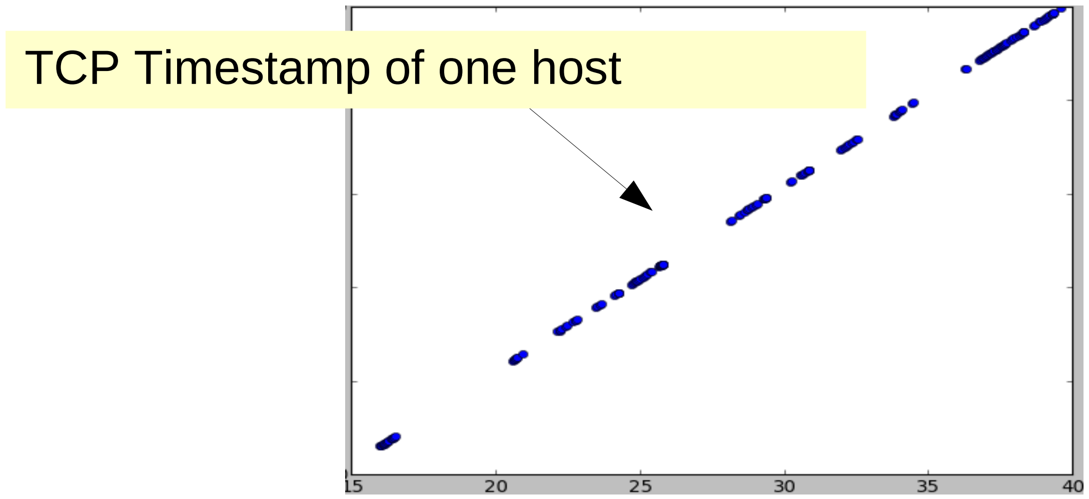
Fingerprinting Accuracy

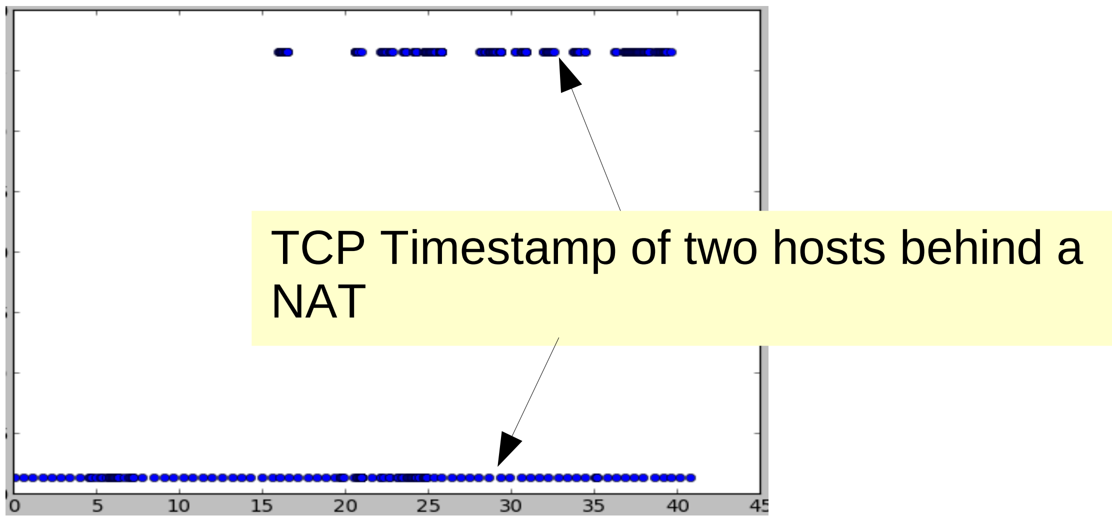
- OS fingerprinting is not very accurate
  - Often relies on information reported by users to build a comprehensive database
  - False positives are not too rare (especially for new or exotic systems)
- OS fingerprinting can be easily (but not very likely) fooled
  - IP Personality (obsolete Linux 2.4 kernel patch to change the machine personality on the net)
  - Most of the network configuration can be changed at runtime by modifying (as root) files under `/proc/sys/net`

```
echo "68" > /proc/sys/net/ipv4/ip_default_ttl
```

Identifying NAT-ted Machines

- Identifying distinct host behind a NAT can be an important step to avoid wrong traffic attribution
- Many techniques have been proposed
  - Different OS fingerprints (`p0f`)
  - Different TTLs
  - Distinct IPid and TCP timestamp trends:





Custom Analysis

Need more Control?

- Parsing individual packets
  - **`Scapy`** (python) for quick prototypes
  - **`libpcap`** (C) for more horsepower (or **`pcapplusplus`**)
- Working with TCP sessions
  - **`libnids`** (C) or **`pynids`** (python bindings) or **`pcapplusplus`**
- New protocol parser
  - Wireshark/tshark plugins (lua, ~python)
- Intrusion detection, complex network analysis
  - **`Bro`** (custom Bro Language)

Scapy

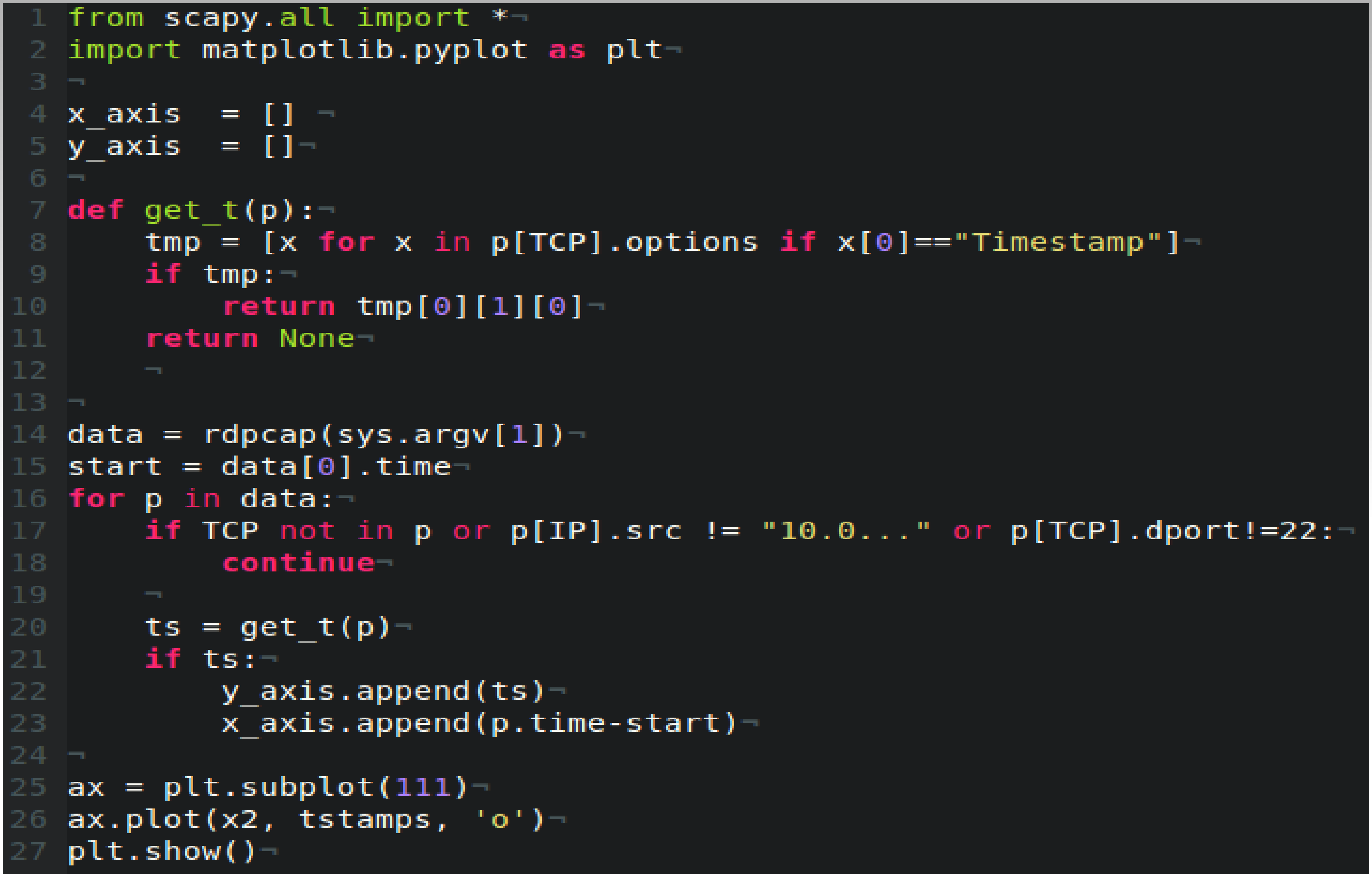
- Scapy is a python library (and a command line interpreter) to play with network packets
- As a library:

```
#!/usr/bin/env python
from scapy.all import *
```

- As a command line interpreter:

```
> scapy
>>> lsc()    # list available commands
>>> ls()     # list supported protocols
>>> ls(TCP)  # list protocol fields
```

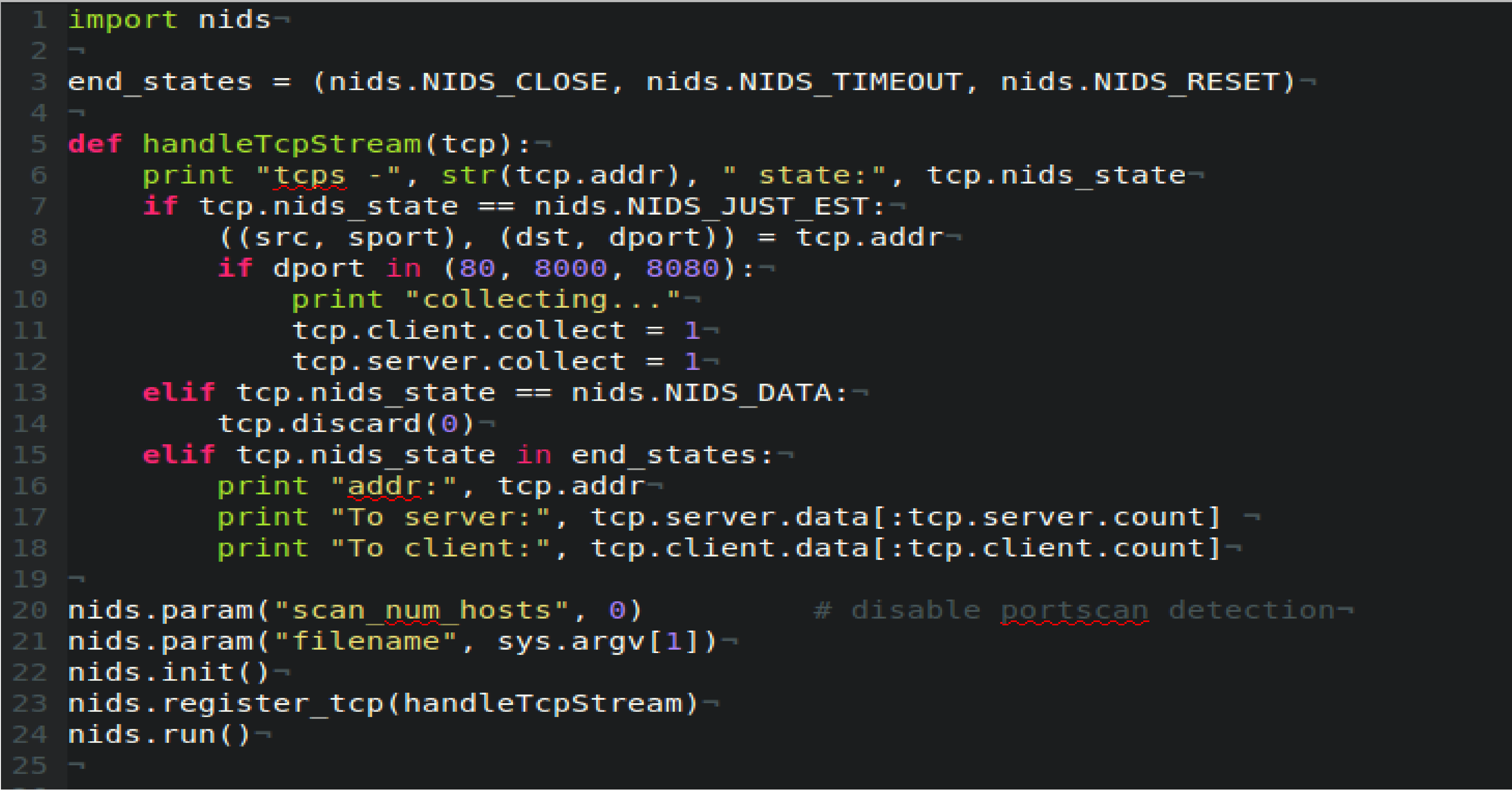
(built around the Python intrepreter + autocompletion)



```python
from scapy.all import *
import matplotlib.pyplot as plt

x_axis  = [] 
y_axis  = []

def get_t(p):
    tmp = [x for x in p[TCP].options if x[0]=="Timestamp"]
    if tmp:
        return tmp[0][1][0]
    return None
    

data = rdpcap(sys.argv[1])
start = data[0].time
for p in data:
    if TCP not in p or p[IP].src != "10.0..." or p[TCP].dport!=22:
        continue
    
    ts = get_t(p)
    if ts:
        y_axis.append(ts)
        x_axis.append(p.time-start)

ax = plt.subplot(111)
ax.plot(x2, tstamps, 'o')
plt.show()
```

(py|lib)nids

- Emulates the IP stack of Linux 2.0.x.
  - IP defragmentation
  - TCP stream reassembly
  - TCP port scan detection
- Convenient way to access data carried by a TCP stream, also if obfuscated by an attacker
- Last modified in 2010



```python
import nids

end_states = (nids.NIDS_CLOSE, nids.NIDS_TIMEOUT, nids.NIDS_RESET)

def handleTcpStream(tcp):
    print "tcps -", str(tcp.addr), " state:", tcp.nids_state
    if tcp.nids_state == nids.NIDS_JUST_EST:
        ((src, sport), (dst, dport)) = tcp.addr
        if dport in (80, 8000, 8080):
            print "collecting..."
            tcp.client.collect = 1
            tcp.server.collect = 1
    elif tcp.nids_state == nids.NIDS_DATA:
        tcp.discard(0)
    elif tcp.nids_state in end_states:
        print "addr:", tcp.addr
        print "To server:", tcp.server.data[:tcp.server.count] 
        print "To client:", tcp.client.data[:tcp.client.count]

nids.param("scan_num_hosts", 0)         # disable portscan detection
nids.param("filename", sys.argv[1])
nids.init()
nids.register_tcp(handleTcpStream)
nids.run()

```

BRO

- A powerful realtime network analysis framework
  - It is not designed to tell you what is wrong, but to tell you what is happening
  - Stateful: keep track of application-layer state
  - Provides comprehensively logs (configurable) of what it observes in the traffic: connections, DNS queries , HTTP requests, SSL certificates, SMTP activity, protocols detected on non-standard ports, transferred files, ….
- Scripts are written in the Bro language
  - Designed to represent network-related artifact and abstractions
- Extensive documentation, with exercises, slides, videos, …
  - https://www.bro.org/documentation/

BRO – Default Analysis

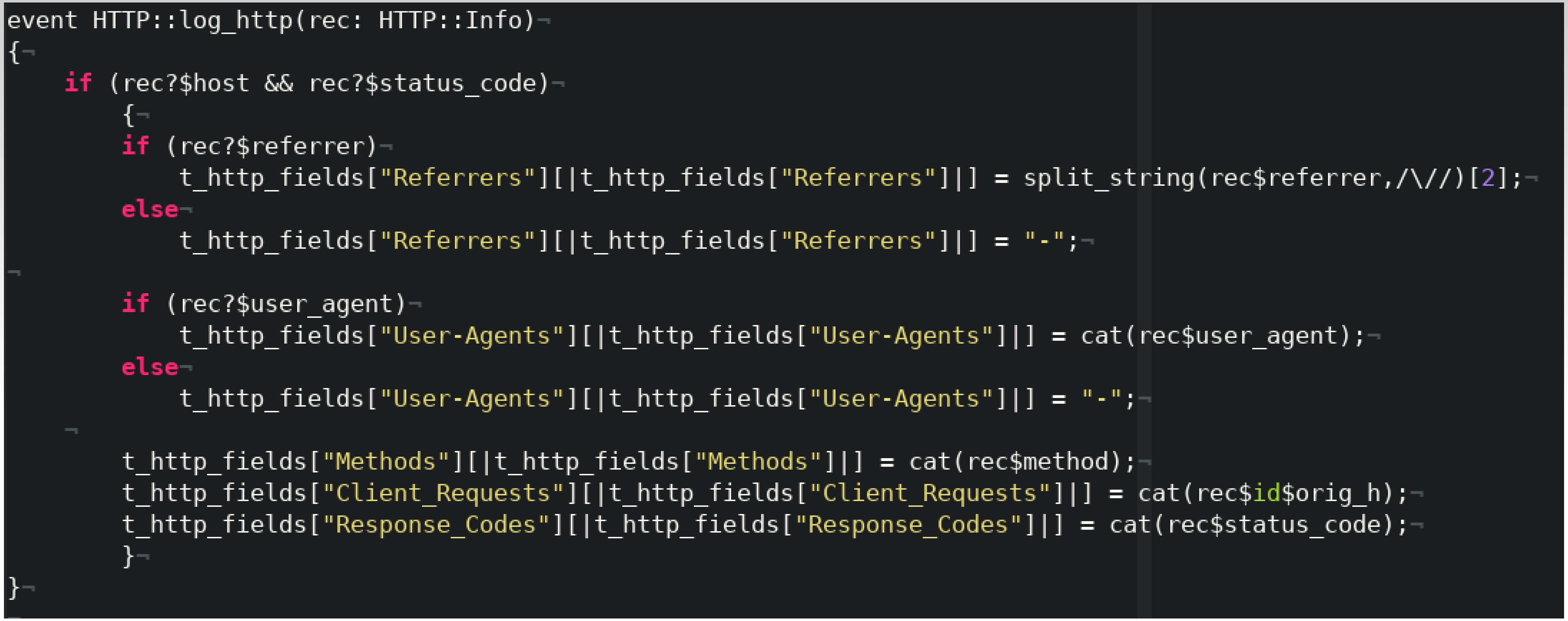
```
> bro -r <pcapfile> local.bro
```

- `local.bro` (typically under `/opt/bro/share/bro/site/`) defines and configures the analysis plugins you want to use
- The analysis generates several log files in the current directory
- Logs typically contains big ASCII tables with many columns.
  You can use `bro-cut` to print only the fields you are interested in:

```
> cat conn.log | bro-cut id.orig_h id.resp_h proto service id.resp_p
```

To also extract files add these lines to your local.bro:

```
# Extract all files
@load frameworks/files/extract-all-files
```



```
event HTTP::log_http(rec: HTTP::Info)
{
    if (rec?$host && rec?$status_code)
        {
        if (rec?$referrer)
            t_http_fields["Referrers"][|t_http_fields["Referrers"]|] = split_string(rec$referrer,/\//)[2];
        else
            t_http_fields["Referrers"][|t_http_fields["Referrers"]|] = "-";

        if (rec?$user_agent)
            t_http_fields["User-Agents"][|t_http_fields["User-Agents"]|] = cat(rec$user_agent);
        else
            t_http_fields["User-Agents"][|t_http_fields["User-Agents"]|] = "-";

        t_http_fields["Methods"][|t_http_fields["Methods"]|] = cat(rec$method);
        t_http_fields["Client_Requests"][|t_http_fields["Client_Requests"]|] = cat(rec$id$orig_h);
        t_http_fields["Response_Codes"][|t_http_fields["Response_Codes"]|] = cat(rec$status_code);
        }
}
```

Pick the Right Tool

```
$ ll file.pcap
-rw-r—r-- 1 balzarot balzarot 850M ….
```

```
$ scapy
>>> for p in PcapReader("file.pcap"):
        pass
  ..............                 → 410 seconds
```

```
$ bro -r file.pcap
   ............                  → 8.7 seconds
```

The Malware Corner

Malware on the Network

- Signs of Infection / Propagation

- Signs of malware behavior
  home ping, Command & Control (C&C), exfiltration, …

- Signs of malicious activity
  spam, scans, DOS, ...

Malware on the Network

- Signs of Infection / Propagation

  (Used for prevention + important step for an investigation)

- Signs of malware behavior

  home ping, Command & Control (C&C), exfiltration, …

  (Common Indicator of Compromise + important step in malware analysis)

  (Typically, it is the infected machine to connect to the attacker, using common ports and protocols)

- Signs of malicious activity

  spam, scans, DOS, ...

  (Often the first thing users notice when a machine has been infected)

Malware Infection

- Click on malicious email attachments
- Download trojan applications
  (e.g., pirated games and software, apps to watch online videos, ...)
- Visit malicious webpages & dry-by downloads

Drive-by Download 101

- The entry point is often an infected website, modified by the attacker to
  - Redirect the victim to another page
  - Load a malicious iframe or javascript library

Drive-by Download 101

- The entry point is often an infected website, modified by the attacker to
  - Redirect the victim to another page
  - Load a malicious iframe or javascript library
- The infected site rarely contains the actual malicious code, which is instead stored on a different page (called Landing Page) which is reached after a chain of redirections
  - Redirections introduce flexibility and allow the attacker to target only certain victims

Drive-by Download 101

- The entry point is often an infected website, modified by the attacker to
  - Redirect the victim to another page
  - Load a malicious iframe or javascript library
- The infected site rarely contains the actual malicious code, which is instead stored on a different page (called Landing Page) which is reached after a chain of redirections
  - Redirections introduce flexibility and allow the attacker to target only certain victims
- The landing page profiles the victim browser and deliver the correct exploit
  - This is typically performed by dedicated Exploit Kits

Drive-by Download 101

- The entry point is often an infected website, modified by the attacker to
  - Redirect the victim to another page
  - Load a malicious iframe or javascript library
- The infected site rarely contains the actual malicious code, which is instead stored on a different page (called Landing Page) which is reached after a chain of redirections
  - Redirections introduce flexibility and allow the attacker to target only certain victims
- The landing page profiles the victim browser and deliver the correct exploit
  - This is typically performed by dedicated Exploit Kits
- The exploit compromises the machine and installs a malware

Network Footprint

- The entry point is often an infected website, modified by the attacker to
  - Redirect the victim to another page
  - Load a malicious iframe or javascript library
- The infected site rarely contains the actual malicious code, which is instead stored on a different page (called Landing Page) which is reached after a chain of redirections
  - The redirection allows the attacker to target only certain victims

blacklists

- The landing page profiles the victim browser and deliver the correct exploit
  - This is typically performed by dedicated Exploit Kits

if not encrypted

- The exploit compromises the machine and installs a malware

Watering Hole Attacks

- Targeted infected pages – especially chosen to reach a given target
  - E.g., infect the pizza delivery website used to order pizza by the employees of a certain company
- Can use zero-day vulnerabilities, chosen for the victim

Like a lion waiting for preys at a watering hole where animals go to drink

Malicious Hosts

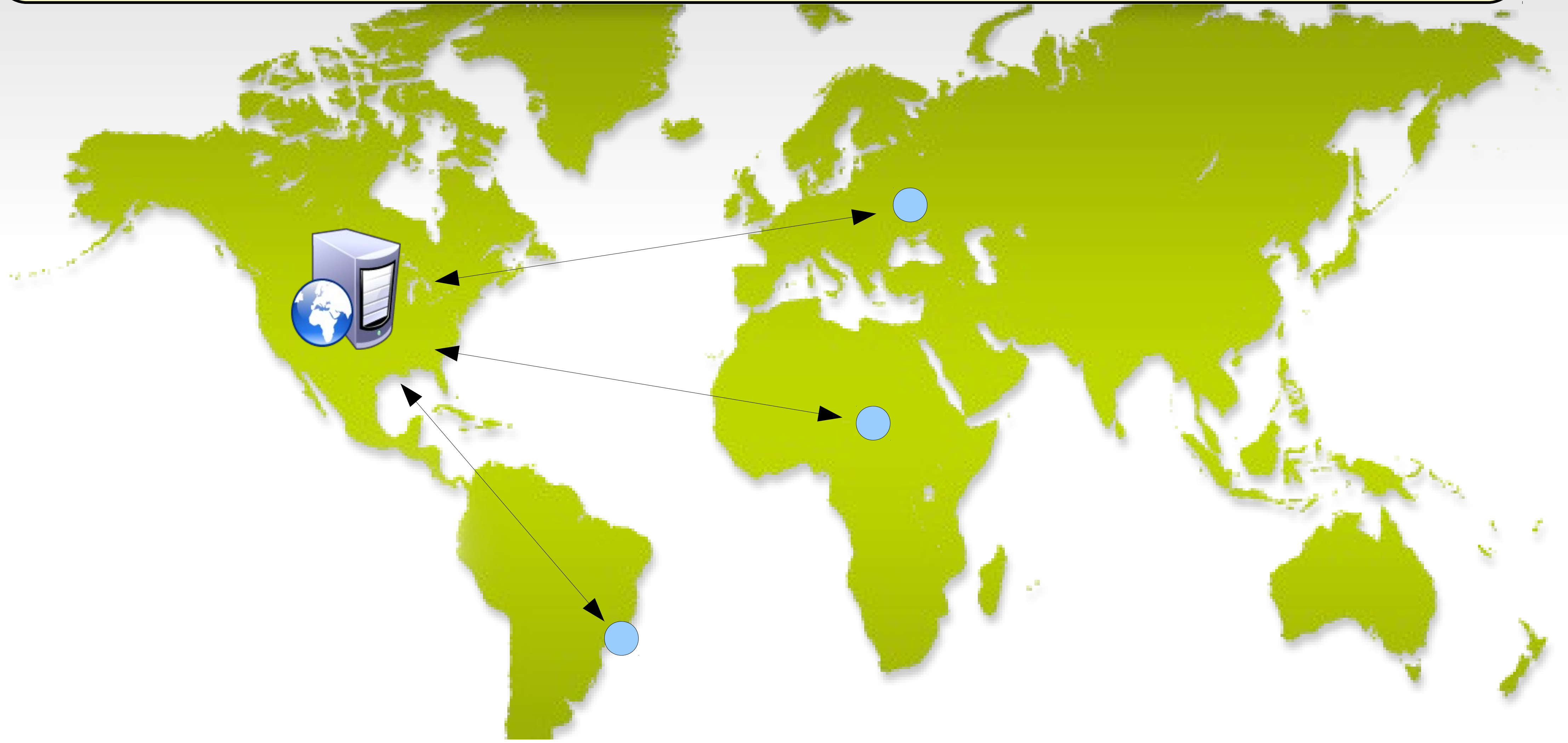
- Most of the online malicious activities involve
  - → Delivering malicious content
    (malware updates, drive-by, phishing sites,...)
  - ← Collecting stolen information
    (uploading credit cards, passwords, bank credentials to dropzones)
  - ⇆ Interacting with compromised machines
    (botnets)
- To make this possible, the attacker has to control some publicly reachable hosts on the Internet
  - Installed by the attacker
  - Compromised machines

Botnets and C&C

- A bot is a compromised machine that can be remotely controlled by an attacker
- Infected machines are incorporated in large networks (botnets) that are controlled by the botmaster like an army

- The Command & Control (C&C) protocol is the main pillar of a botnet (and its main weakness)
- A C&C infrastructure consists of:
  - A protocol (e.g. IRC, HTTP, …)
  - A network topology (star, hierarchical, P2P, …)
  - [Optionally] A lookup resilience mechanism (IP flux and/or Domain flux)

The vanilla approach
(malicious server in plain sight)



The vanilla approach
(malicious server in plain sight)

FAIL

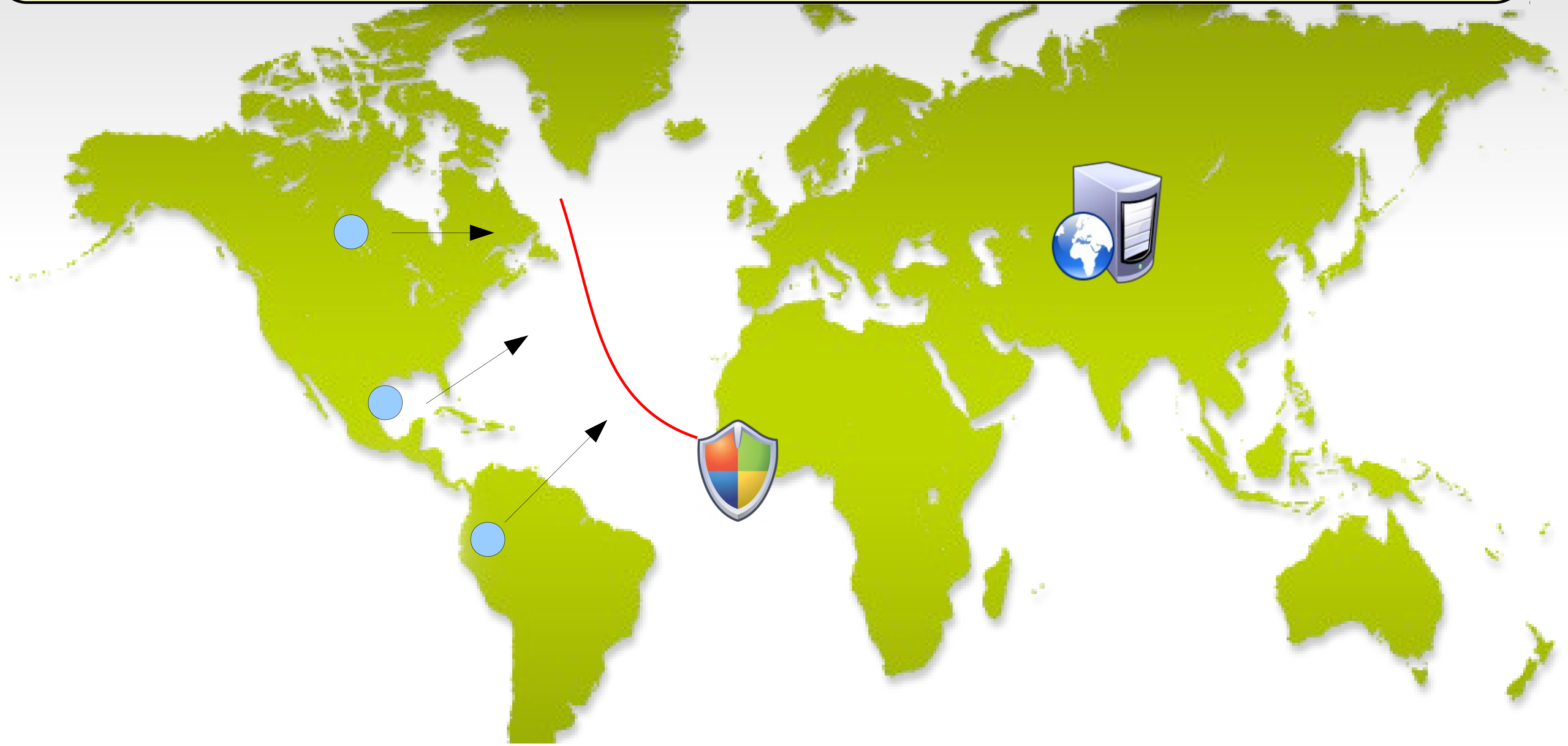
Bulletproof hosting

(service and hosting providers that tolerate illegal customers activities)

Bulletproof hosting

(service and hosting providers that tolerate illegal customer activities)

FAIL



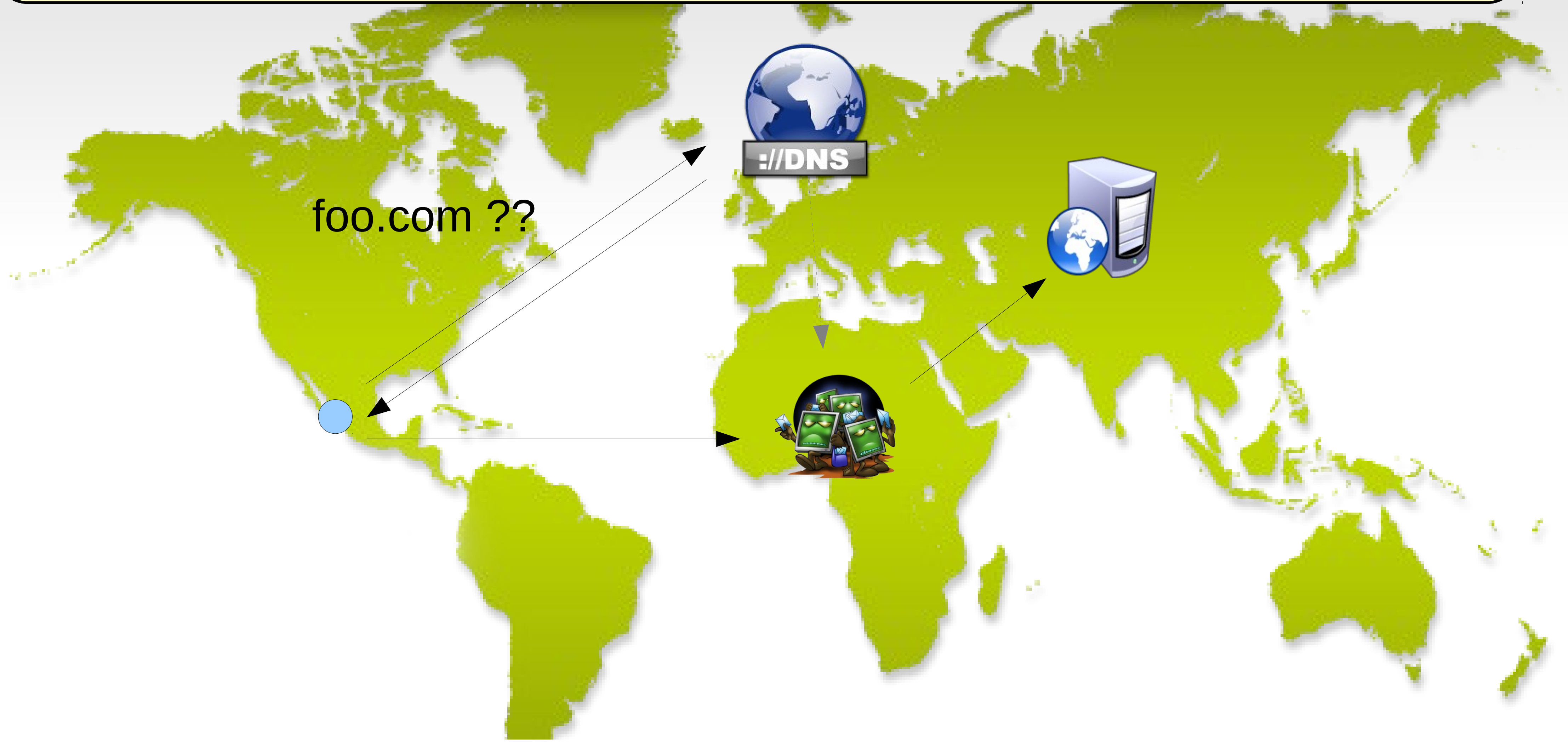
IP Flux

- Consists of constantly changing the IP addresses a particular domain name resolves to. Often, the IPs are changed very rapidly (aka *fast-flux*)

- Single Flux – hundreds or thousands of IP addresses are frequently registered and de-registered using very short Time-to-live (TTL) values
- Double Flux – Also rotates the address of the authoritative DNS server (NS record) that is used to lookup the IP addresses

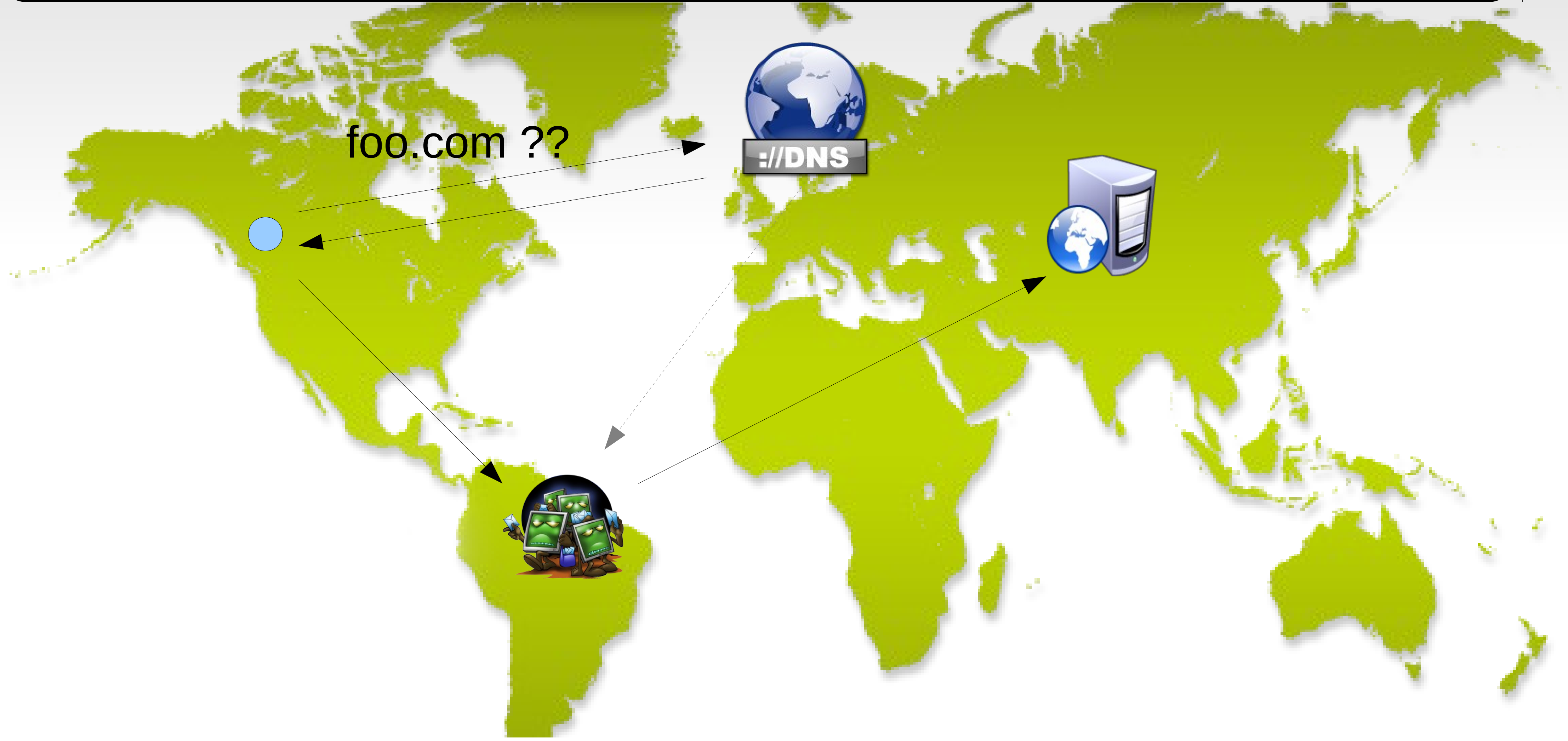
IP Fluxing

(constantly changing the IP address information related to a particular domain)



IP Fluxing

(constantly changing the IP address information related to a particular domain)



IP Fluxing **FAIL (?)**

(constantly changing the IP address information related to a particular domain)

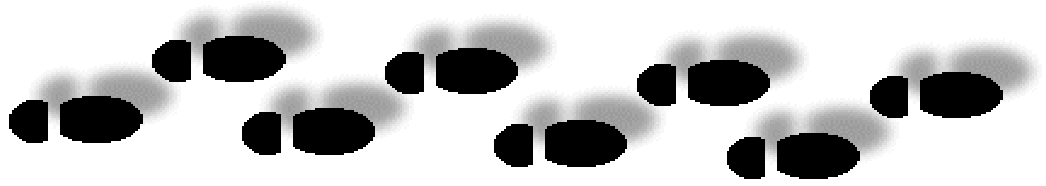
Domain Flux

- Consists of constantly changing the Domain Name an infected machine connects to.
- Typically involve some sort of Domain Generation Algorithm (DGA)
    - Each malware generates a dynamic list of many domains
      (e.g., a different list each day)
    - The malware then tries to connect to each of them, until it finds a valid one
    - The attacker only needs to register one of them, and discard it afterwards

Malware Network Patterns

Before you start
(four things to keep in mind)

- Time is crucial, double-check it to be sure you got the time and the timezone right
  - E.g., compare it with the HTTP response headers
- DNS resolution can be tricky
  - Always use the information in the capture file
- Protocol Identification is often port-based and therefore it may be inaccurate
- TCP retransmissions, out of order and duplicate packets can confuse session reconstruction



Checklist

- Check for known bad
- Check for anomalies and weird connections
- Check the DNS traffic
- Check the HTTP traffic
- Check the certificates

Checklist

- Check for known bad
  1. IDSs signatures
  2. AV signatures
  3. Blacklists
  4. Malware traffic patterns

Check for anomalies and weird connections

Check the DNS traffic

Check the HTTP traffic

Check the certificates

Known Bad - Signatures

- Intrusion Detection
  - Install SNORT and run it on the traffic
  - Emerging threats (http://www.emergingthreats.net/) provides updated rules for most of the popular IDSs
  - SRI maintain a page (http://mtc.sri.com/live_data/signatures/) that shows the most effective rule sets
- Classic AntiVirus signatures
  - Extract all the transmitted files (using `bro` or `nfex`)
  - Run an antivirus on them or query their MD5s on VirusTotal

Known Bad - Blacklists

- mal-dnssearch
  - shell script that compares IP and DNS entries against a number of reputation data sources
  - It supports several formats, including BRO logs, IP or domain lists, `bind` logs, pcap and pcap-ng
  - It can automatically download and match over 10 blacklists

snort	-	http://labs.snort.org/feeds/ip-filter.blf (IP)
et_ips	-	http://rules.emergingthreats.net/open/suricata/rules/compromised-ips.txt (IP)
alienvault	-	http://reputation.alienvault.com/reputation.generic (BIG file) (IP)
botcc	-	http://rules.emergingthreats.net/open/suricata/rules/botcc.rules (IP)
tor	-	http://rules.emergingthreats.net/open/suricata/rules/tor.rules (IP)
rbn	-	http://rules.emergingthreats.net/blockrules/emerging-rbn.rules (IP)
malhosts	-	http://www.malwaredomainlist.com/hostslist/hosts.txt (DNS)
malips	-	http://www.malwaredomainlist.com/hostslist/ip.txt (IP)
ciarmy	-	http://www.ciarmy.com/list/ci-badguys.txt (IP)
mayhemic	-	http://secure.mayhemiclabs.com/malhosts/malhosts.txt (DNS)
mandiant	-	https://raw.github.com/jonschipp/mal-dnssearch/master/mandiant_apt1.dns (DNS)

Known Bad – Other Blacklists

- Malware Traffic Patterns
  - Spreadsheet with "personal notes" with GET and POST requests for different malware families
  - http://data.deependresearch.org/
- SpamHaus DROP (don't route or peer) list
  - Netblocks that are "hijacked" or leased by professional spam or cyber-crime operations

    http://www.spamhaus.org/drop/
- Zeus/SpyEye Tracker

  https://spyeyetracker.abuse.ch/ and https://zeustracker.abuse.ch/
- Google safe browsing
  - Check each URL in the traffic against the list of phishing and malicious pages detected by Google

Checklist

Check for known bad

- Check for anomalies and weird connections

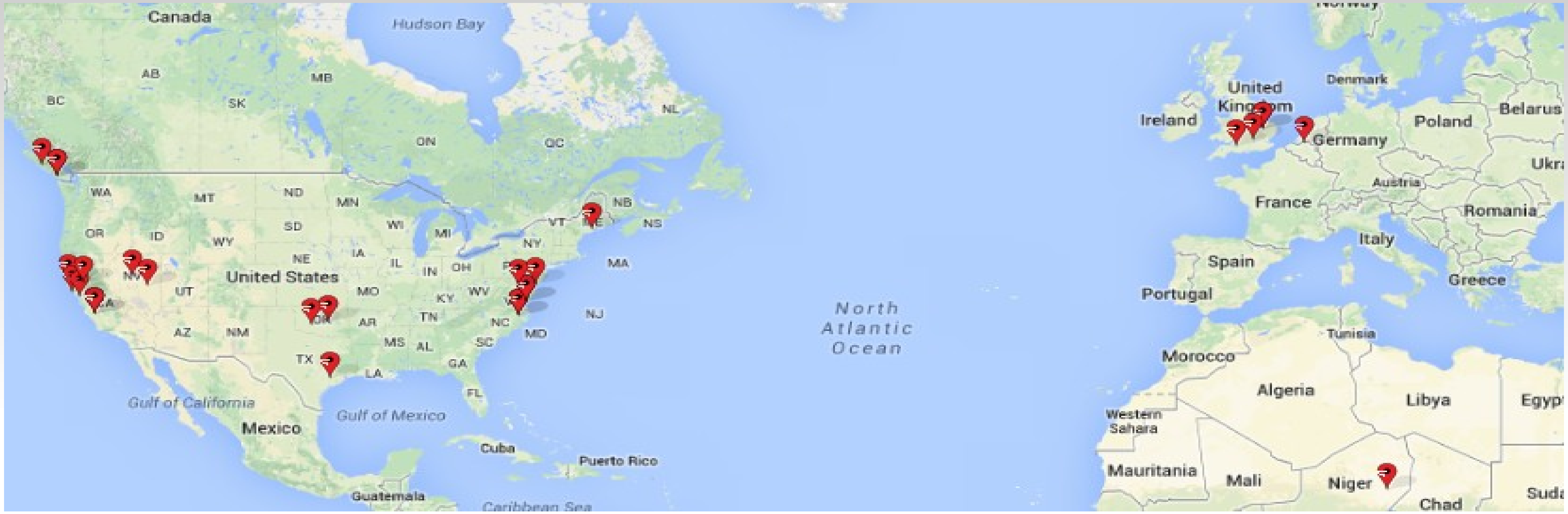
  1. IP geolocation
  2. Weird port numbers
  3. Protocol on non-standard ports
  4. IRC traffic?
  5. Hardcoded IPs
  6. ICMP messages
  7. Packet injection

Check the DNS traffic

Check the HTTP traffic

Check the certificates

Spot the Anomaly



```
> tshark -n -r f.pcap -R "ip.src==”x.x.x.x" -T fields -e "tcp.dstport" | sort | uniq -c
    1886  80
      798  443
        12  445
          8  1335
        85  9001
```

Anomalies

- ICMP Destination Unreachable messages

```
> tshark -n -r malware1.pcap -R "icmp"
```

- Hardcoded IP addresses (without a DNS lookup)

```
> tshark -n -r … -Tfields -e “ip.src” |sort|uniq
> tshark -n -r ... -q -z hosts
```

Packet Injection

- A man-on-the-side hacking techniques in which:
  - The attacker can monitor the traffic and inject new packets into an ongoing connection
  - The attacker is located closer to the target, so that his fake packets arrive before the legitimate responses
- The target receives two overlapping TCP segments with different application data
- Very stealthy attack used by the NSA and other government, e.g., to redirect HTTP connections towards other servers

Checklist

Known bad

Anomalies and weird connections

- DNS
  1. Anomalous TTL values
  2. Dynamic DNS services
  3. DNS errors

HTTP

Certificates

Anomalies

- Weird TTLs

```
> tshark -n -r .. -R dns.resp.ttl -T fields
  -e dns.qry.name -e dns.count.answers
  -e dns.resp.ttl
```

- Dynamic DNS

```
> tshark -n -r … -R 'dns contains "dyndns.org"'
```

- DNS `NXDomain` errors (non existent domain)

```
> tshark -n -r … -R 'dns.flags.rcode==3' -Tfields
  -e dns.qry.name -e ip.dst
```

Compromise Detection

- Known bad
- Anomalies and weird connections
- DNS
- HTTP
  1. Redirects
  2. User agents
  3. File extensions
  4. HTTP on 443
  5. Alexa top
- Certificates

HTTP

- Over 70% of the infections come from the Web
- Malware does not always mimic real user agents
  - Sometimes it leaves it empty:

```
> tshark -r ... -R "http.request == 1 and
   not http.user_agent"
```

  - Sometimes is set to perl, python, …

```
> tshark -r ... -R "http" -Tfields -e http.user_agent
```

- Too many redirections are a clear sign of exploitation

```
> tshark -n -r ... -R http.response -T fields
    -e http.response.code -e http.location
```

HTTP

- Check the file extensions

```
> tshark -r ... -R http.request -T fields -e
  http.request.full_uri |  sed -r 's;\?.*;;' |
  awk 'BEGIN {FS="."} {print $NF}' |
  grep -v "/" | sort | uniq
```

- Alexa keeps a daily-updated list of the top 1million websites
  - Available at: http://s3.amazonaws.com/alexa-static/top-1m.csv.zip
  - You can double-check the web sites in the network trace to list the ones that do not belong to the Alexa list

```
> tshark -n -r … -R http.request -T fields
  -e http.host
```

Checklist

Known bad

Anomalies and weird connections

DNS

HTTP

- Certificates

SSL Certificates

- SSL often used by malware to communicate to the C&C server

```
> tshark -r traffic.pcap -R
  "ssl.handshake.certificates" -T pdml >
   certificates
```

- Looks for empty fields or strange common name values
- Extract signature chains and look for self-signed certificates

Putting it all Together

- Wouldn’t make sense to automatize all these checks?
  - Check out PCAP-Dissector:

    https://github.com/eaam/Bro-PCAP-Dissector
  - Script for BRO that provides an ordered list with the number of connections that satisfy certain suspicious conditions:
    - Upload >3M, last more than 10minutes, connect to strange hostnames, use atypical referrers, query SMB filename,...

Your turn

- Install the tools on your machine
  - You can try a Linux forensic distribution (e.g., Security Onion)...
  - But refreshing some `configure/make/make install` skill is probably better :)

- Get some pcap files to play with:
  - http://contagiodump.blogspot.fr/2013/04/collection-of-pcap-files-from-malware.html
  - http://www.netresec.com/?page=PcapFiles
  - http://chrissanders.org/packet-captures/
  - http://malware-traffic-analysis.net
  - https://www.netresec.com/?page=Blog&month=2016-03&post=Packet-Injection-Attacks-in-the-Wild
  - https://github.com/LiamRandall/BroMalware-Exercise

Research Corner

- “Dynamic Application-Layer Protocol Analysis for Network Intrusion Detection”
  `Dreger et al.  – Usenix Security 2006`

- “Enriching Network Security Analysis with Time Travel”
  `Maier et al. – SigComm 2008`