Netsniff - Design and Implementation Concepts

Urs Keller[1], Jason But
Centre for Advanced Interent Architectures. Technical Report 050204A
Swinburne University of Technology
Melbourne, Australia
urs.keller@epfl.ch, jbut@swin.edu.au

Abstract– This technical report gives an overview of the protocols netsniff currently understands. Also it describes the data netsniff extracts from the parsed protocols. It further gives an overview about the anonymisation schemes currently implemented and about their issues.

Keywords– Netsniff, packet decoding, protocol decoding, anonymisation, Statistics

I. Introduction

Netsniff is a tool designed to capture network traffic. It is designed to parse the captured traffic at the application layer and to gather statistics about different networked applications that are running across the listening point on the network. Originally designed for the ICE^3 project [1], netsniff preforms the task of extracting data at various protocol layers and assigning them to individual application data flows, as well as traffic anonymisation to allow for data publication while preserving individual users privacy.

This technical report documents the characteristics of netsniff and describes the data gathering process in more detail - including the formatting of the output data and why it is captured.

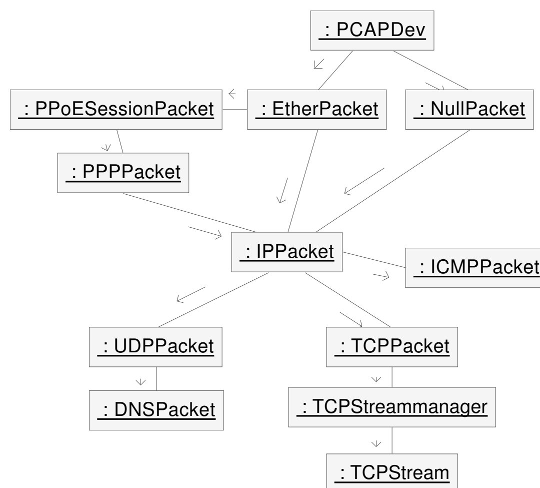

Figure 1: Object layout

II. Basic Netsniff Design

Netsniff is based on a C++ Object Oriented design. Its implementation allows simple expansion to support decoding of new protocols with minimal effort. As an example, if we wish to support decoding of imaginary protocol XYZ which runs over TCP, a module to parse the Application Layer protocol can be designed. TCP layer information is automatically collected and output for each TCP stream, following which the particular Application Layer data is appended. A basic Object Layout is shown in Figure 1.

The details of what protocols are decoded follow in a later section, brief examples for decoding DNS and HTML application data follow:

A. DNS Decoding

All capturing is done using the pcap library to retrieve the entire captured payload. Assuming the DNS packet is captured on an Ethernet link, then:

- The entire data packet is captured and tested to see if it is an Ethernet packet. If so, the header is stripped and the Ethernet Packet passed to the EthernetPacket constructor.
- The EthernetPacket constructor finds an encapsulated IP Packet. An IPPacket is constructed with the Ethernet Payload.
- The IPPacket constructor finds an encapsulated UDP Packet. A UDPPacket is constructed with the IP Payload.
- The UDPPacket constructor determines that the application is DNS. A DNSPacket is constructed with the UDP Payload. Note that RFC1035 [2] states that DNS can use a TCP connection, if the payload exceeds 512 bytes. This isn't currently implemented by Netsniff, but might be added in future releases, depending on the traffic, we will observe.
- The DNSPacket constructor parses the DNS information.
- The packet details are logged, the base layer logs the packet timestamp, the IPPacket logs the source and destination IP addresses, the UDPPacket logs the source and destination Port Numbers, and the DNSPacket logs the DNS information gathered.

[1] Urs Keller worked on Netsniff while visiting CAIA from the Swiss Federal Institute in Lausanne $EPFL$.

B. HTTP Decoding

Again, all capturing is done by the pcap library, in this case:

- The entire data packet is captured and tested to see if it is an Ethernet packet. If so, the header is stripped and the Ethernet Packet passed to the EthernetPacket constructor.
- The EthernetPacket constructor finds an encapsulated IP Packet. An IPPacket is constructed with the Ethernet Payload
- The IPPacket constructor finds an encapsulated TCP Packet. A TCPPacket is constructed with the IP Payload
- The TCPPacket constructor determines which stream the packet belongs to (based on Source-Destination IP Address/Port Number pairs) and passes the TCPPacket to the TCPStream instance collating statistics about the given stream. If no TCPStream instance exists, one is created.
- The TCPStream instance logs information regarding Round Trip Time (RTT), Jitter, Packet Loss Rate (PLR), and transferred bytes as well as reconstructing the actual TCP data stream. The TCPStream instance also determines the application type as HTTP and creates a HTTPParser to parse the reconstructed data stream.
- The HTTPParser collects statistics from the decoded HTTP data exchange.
- When the TCP Stream is concluded, the TCPStream instance initiates logging of information:
  1. The TCPStream instance logs the stored TCP Stream information and calls the Parser to log application layer information.
  2. The HTTPParser logs HTTP level information.

III. Anonymisation

Netsniff supports data anonymisation with the -a command line parameter. This causes:

- IP Addresses to be anonymised in the way tcpdpriv does with the -A50 switch. A more detailed explanation can be found in Appendix A
- User strings extracted from application level protocols are anonymised using a secure hash. A more detailed explanation can be found in Appendix B

Both the above approaches allow for correlation of data. A given IP address will always be anonymised to the same random IP address - this allows determination of server popularity and regularity of visits by a single user, it also allows correlation of DNS queries to later access of the queried host. Further, user strings will be anonymised consistently, we can correlate information like "How often an email is sent to a given (unknown) email address"

IV. Data collection

In accordance with the netsniff design architecture, logged information is layered, with each decoded protocol in the Protocol stack having the opportunity to contribute to the logged information. Of particular interest:

- All packets containing an unrecognised protocol will be logged in tcpdump format. Packet data is shortened to the tcpdump default length of 68 bytes. If anonymisation is enabled, the tcpdump file will have the IP addresses anonymised.
- All TCP information will not log packet-by-packet information, instead collating this information per stream.
- TCP Streams of an unrecognised protocol are logged in tcpstream.log as well as the tcpdump log file.

The following sections describe the output of netsniff for the currently supported protocols. A complete grammar of the output is specified in Appendix C.

A. Packet

- **Timestamp:** time the packet was captured, example: '2004-10-23 03:10:00.732685'. The time stamp is created by the PCAP library [3] and is passed on to netsniff in the pcap_pkthdr struct.
- **Header length:** Summed length of the packet header of each nested protocol. For a DNS packet this would be the sum of Ethernet header, IP header, and the UDP header length.
- **Payload length:** Is the total header length subtracted from the total packet length. For a DNS packet this is the UDP payload.

B. Ethernet Packet

- Outputs no information
- Parses payload of IPPacket and ARPPacket.

C. ARP

Note that some data collected here is redundant, since we will only support ARP with IP/Ethernet. The information is more or less what is found in an ARP packet itself. Source/Destination hardware/protocol address are anonymised, when anonymisation is turned on.

- $\longrightarrow$ **Packet:** All information defined by Packet A
- **Hardware address type:** The type of hardware address ARP is doing address resolution for. We only currently support ARPHRD_ETHER.
- **Protocol address type:** The type of protocol address ARP is used with. Currently there is only support for ETHERTYPE_IP.
- **Hardware address length:** The length of the hardware address, which is 6 bytes for Ethernet.
- **Protocol address length:** Length of the protocol address. This will be 4 in our case.

- **Operation:** What the packet operation is. One of REQUEST, REPLY, REVREQUEST, REVREPLY, INVREQUEST, INVREPLY.
- **Source hardware address:**
- **Source protocol address:**
- **Destination hardware address:**
- **Destination protocol address:** The address contained in an ARP packet. IP addresses are anonymised with IP anonymisation and hardware addresses with string anonymisation described in Appendix A respectively B.

D. IPPacket

- ⟶ **Packet:** All information defined in A
- **IP addresses:** Source and destination IP addresses are captured and logged.

E. UDPPacket

- ⟶ **IPPacket:** All information defined in D
- **Ports:** Source and destination ports are captured and logged.
- Note that UDPPackets are currently only logged in the context of a DNSPacket. All other UDPPackets are logged to the notparsed.dump file.

F. DNSPacket

- ⟶ **UDPPacket:** All information defined in E
- **identification:** 16 bit identification as specified in RFC1035 [2]. It allows us to match queries and their corresponding responses.
- **response code:** RCODE as defined in section 4.1.1 of RFC1035 [2]. This usually indicates if there was an error.
- **queries:** the queries contained in the DNS packet. The host name looked up is replaced by a string of 'x' to anonymise it.
- **responses:** Contains the responses defined in this packet. They are matched up with the queries by means of the identification field collected (described above). The returned (possibly anonymised) IP addresses can later be matched up with protocol exchanges to these servers using other protocols.

G. ICMP

- ⟶ **IPPacket:** All information defined in D
- **Hops:** number of hops this packet passed through.
- **ICMP type:** provides a first information what kind the ICMP packet is.
- **ICMP code:** provides further information on the kind of the ICMP packet
- **Note:** we don't parse the packets further based on their type, for now.

H. TCPStream

- **Timestamp:** time the first packet of the stream was captured. It is created when the SYN packet is captured.
- **Duration:** The duration of the TCP stream. This information is only accurate to a certain degree, since streams are timed out by netsniff, if there is no activity.
- **source IP:** Source address of the TCP stream. This is the address from which the TCP stream was established.
- **source port:** Source port from which the connection was established.
- **destination IP:** IP address to which the connection was made.
- **destination port:** Port to which the connection was established.
- **Client2server data:**
- **Server2client data:** Amount of TCP payload sent from one party to other.
- **Client2server overhead:**
- **Server2client overhead:** Overhead produced by TCP for each direction.
- **Hop count:** Contains the averaged hop count estimation over the duration of the TCP stream. Hop count estimation works on the assumption that Operating Systems set the TTL field of packets to values 64, 128 or 255. It further assumes that hosts are at most 64 hops away from the system netsniff runs on. This field contains the sum of the hops to the source and to the destination of the connection.
- **RTT:** RTT is estimated separately for each portion of the TCP stream - RTT from the client to the measurement device and from the server to the measurement device. The algorithm employed is that devised by But et. al [4]. This algorithm produces a running estimate for the stream RTT, we report the average value of this running estimate.
- **Jitter:** Jitter is estimated based on the RTT samples using the algorithm devised by But et. al [4]. This estimate Jitter as the running absolute difference between witnessed RTT and the current running RTT estimate. This also produces a running estimate for the stream Jitter, we report the the average value.
- **Loss rate:** Netsniff performs a loss rate measurement for both directions, client to server and server to client. Loss rate estimation uses the Benko-Veres algorithm [5]. The running estimate proposed by Favi/Armitage[6] is not yet implemented. Also keep in mind, that the loss rate estimation using this algorithm [5] becomes only reliable after a certain amount of traffic has been seen and this for a given loss rate, as has been demonstrated in [6].
- **Hop Count Histograms:** A histogram of all wit-

nessed hop counts is available. This allows us to see variations in network conditions due to a change in witnessed hop counts over the link.

- **RTT Running Estimates:**
- **Jitter Running Estimates:** The running RTT and Jitter estimates calculated for the duration of the TCP stream are sub-sampled at ten second intervals. This list of running estimates is presented as a means of generating time-based statistics for the stream.

I. HTTP

- ⟶ **TCPStream:** All information described in TCP-Stream H.
- **Request type:** The type of request the client made to the server, like GET, POST, Connection etc.
- **Request URL:** If the data is not anonymised this will contain the requested URL.
- **Request host:** The host name the request went to. Only set when netsniff isn't in anonymised mode.
- **Request referer:** The URI of the document from which the Request URL was obtained. As above this value is only available if not in anonymised mode and it was supplied by the HTTP client.
- **Upload length:** For POST requests the number of bytes uploaded to the server. Note that the Query part of a HTTP request is currently not included in the Upload length.
- **Download length:** For all except POST and CONNECT requests this indicates the number of bytes transferred from the server to the client.
- **Status code:** Corresponds to the HTTP status codes returned by the server.
- **Cacheability:** Indicates the cacheability of the document that is served by the server. This follows the rules for cacheability in RFC2616 section 13.4.
- **Content type:** Is the content type the server returns to the client such as 'text/html'.

J. SMTP

- ⟶ **TCPStream:** All information described in TCP-Stream H.
- **Sender:** All sender addresses used in this particular connection. Most traffic will contain a single sender in its SMTP connections, because we will observe in a home environment, where SMTP server to SMTP server traffic is not common. If anonymisation is turned on, this will be a secure hash of the sender address. This allows us to evaluate data on a per user basis.
- **advertised size:** ESMTP defines a size attribute listed in the EHLO command. This attribute advertises the maximal size the SMTP server is willing to send. This might be a good long term measure of the user demand of sending large amounts of data over e-mail. We will see on how this evolves over let's say 6 month to a year. Just to give an idea here:
  - gsmtp[171,185].google.com announce 20Mb
  - mx[1-4].mail.yahoo.com announce 30Mb
  - mx[1-4].hotmail.com announce 30Mb
- **(unrecognized) / commands:** Keeps track of how many commands the SMTP client sent to the SMTP server. This is mostly there because we want to see, if we need to add further currently unsupported commands.
- **sent mail sizes:** A list of sizes of e-mails sent. More than one e-mail can be sent over an SMTP connection. But as we have seen for the sender, this will most likely only contain a single entry since many mail clients only send one e-mail at a time (except for dial up users, who might still use the 'send later' feature many mail clients have).

K. POP3

- ⟶ **TCPStream:** All information described in TCP-Stream H.
- **User:** User name used to login to the POP3 server. When anonymisation is turned on, this will be an MD4 hash of the user name. This data is collected to do statistics on a user behavior, which will allow us to correlate captured data to user behavior.
- **mails deleted:** The number of mails the user deleted from the server. We don't know yet if this will be useful, since many users don't keep mails on the server when using POP3, so the POP3 client will just delete all mail on the server after download. Basically this would then only help to tell if the user keeps the mails on the server or not.
- **Errors:** Number of errors encountered during the session. This is the number of times the server responded with ERR.
- **protocol errors:** Some implementations of POP3 have some flaws. This is the number of times netsniff encounters behavior, which isn't conform to the RFC.
- **mails received:** A list of the sizes of mails the client downloaded from the server.
- **mails inbox:** A list of mail sizes, which are on the server. This will only contain values, if the client issued a LIST command.

L. IMAP

- ⟶ **TCPStream:** All information described in TCP-Stream H.
- **User:** User name the user logged in with or its anonymisation. This is set to "<UNKNOWN>" if the authentification method is not username/password based.
- **Authentification method:** The method used to authenticate the user. This is currently password or CRAM-MD5.

- **Mails listed:** The sizes of the mails the user downloaded partially. This is usually a listing the user requests from the server. This gives a distribution of the e-mail sizes the user keeps in her inbox.
- **Mails downloaded:** The sizes of the mails the user downloaded completly (for IMAP this usually means looked at). This gives the distribution of mails beeing downloaded as opposed to mails in the inbox.

M. TLS

- ⟶ **TCPStream:** All information described in TCP-Stream.
- **Session ID:** The session identification is created by the TLS server side and sent to the client. Together with the shared secret, it can be used by a client to resume a TLS session or to use additional streams. Session ID is collected, since it uniquely defines the TLS session in a certain time frame.
- **HadError:** This flag is set, when the session wasn't captured completely.
- **Version:** The TLS version number. For TLS this is 0x301, since TLS is the successor of SSL 3.0. Currently this is the only fully implemented TLS/SSL version in netsniff. Most browser nowadays use TLS as their encryption mechanism. Future analysis will show, if the implementation of older SSL versions will be necessary.
- **Cipher:** This is the cipher method used.
- **Compression:** The compression method used. This is usually 0x0 for not compressed content.
- **Payload length:** The number of bytes transported through the TLS session
- **Overhead:** Overhead produced by using TLS.
- **TCP streams:** The number of TCP streams, associated with the TLS session.
- **Session duration:** Session duration is the amount of time elapsed from the creation of the session until the end of the last TCP stream belonging to the TLS session.
- **TCP timestamps:** A list of time stamps, which uniquely identify the TCP streams associated with the TLS session.

N. FTP

- ⟶ **TCPStream:** All information described in TCP-Stream H.
- **(Anonymised) user name:** The user name used to login to the FTP server. If anonymisation is turned on this will be a hashed value of the user name except for names commonly used for anonymous FTP, which are anonymous, guest and ftp.
- **OS string:** String representing the operation system used on the FTP server.
- **File operations:** Counts the number of successful file operations on the FTP server. Examples are: Size, Stat, Checksum
- **Error file operations:** Number of file operations, that produced an error, missing file etc.
- **Unimplemented:** The number of unimplemented commands occurred during the FTP connections. This might be helpful when deciding, if additional commands should be added.
- **Responses:** Number of times the server replied with an error status.
- **Unknown commands:** The number of unknown (by netsniff) commands sent to the server.
- **Anonymous:** Set if the login was an anonymous login. This is useful, to easily select anonymous FTP connections.
- **Login:** If the FTP connection had a successful login. Meaning that a user logged in with a user name and optional password and got a positive reply from the server.
- **List of data connection:** A list of connections opened from the server to the client (active) respectively from the client to the server (passive) and the operation the connection was used for. Currently this is one of LIST, RETR, SEND.

REFERENCES

[1] Centre for Advanced Internet Architecture. "ICE^3 Inverted Capacity Extended Engineering Experiment", January 2005. http://caia.swin.edu.au/ice/.

[2] P.V. Mockapetris. "Domain names - implementation and specification". RFC 1035, IETF, November 1987. http://www.ietf.org/rfc/rfc1035.txt.

[3] "TCPDUMP / PCAP", January 2005. http://www.tcpdump.org.

[4] J. But, U. Keller, D. Kennedy, and G. Armitage. "Passive TCP Stream Estimation of RTT and Jitter Parameters". *Submitted to ACM Computer Communications Review*, January 2005. http://caia.swin.edu.au/cv/jbut/publications.html.

[5] P. Benko and A. Veres. "A Passive Method for Estimating End-to-End TCP Packet Loss". *Proceedings of IEEE Globecom, 2002*, November 2002. http://www.comet.columbia.edu/~veres/globecom02.pdf.

[6] C. Favi and G. Armitage. "Dynamic Performance limits of the Benko-Veres Passive TCP Packet Loss Estimation Algorithm". *Australian Telecommunications Networks & Applications Conference 2004 (ATNAC2004)*, December 2004.

[7] Greg Minshall. "TCPDPRIV", August 1997. http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html.

[8] Jun Xu, Jinliang Fan, Mostafa H. Ammar, and Sue B. Moon. "Prefix-Preserving IP Address Anonymization: Measurement-based Security Evaluation and a New Cryptography-based Scheme". In *Proc. of IEEE ICNP 2002*, November 2002. http://www.cc.gatech.edu/~jx/reprints/ICNP02A.ps.

[9] Greg Minshall. "Thoughts on How to Mount an Attack on tcpdpriv's "-A50" Option", July 2001. http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html.

V. APPENDIX

A. IP anonymisation

IP anonymisation works in the fashion tcpdpriv does [7]. A thorough analysis of prefix-preserving IP address anonymisation is presented in [8]. The tcpdpriv approach is table based. It keeps a lookup table in memory, which

it builds up gradually, when IP addresses are anonymised. [8] defines this nicely in the following way:

Let's suppose that we have a set of $<raw, anonymised>$ binding pairs of IP addresses. For an IP address a, we'd like to anonymise with $a = a_1a_2...a_n$ we find the pair $< x, y >$ with $x = x_1x_2...x_n$ and $y = y_1y_2...y_n$ with longest prefix match k on a and x. Then if a is anonymised to b ($b = b_1b_2...b_n$) we have $b_1b_2...b_kb_{k+1} = y_1y_2...y_k\neg(y_{k+1})$ and $b_{k+2}b_{k+3}...b_n = rand(0, 2^{n-k-1} - 1)$, where $rand(g, f)$ generates a random number between g and f. In netsniff $rand(g, f)$ is an alternating series of 0 and 1. Finally if $a! = x$ (not in the list already), $< a, b >$ will be added to the binding table.

As described in [8] this has some problems, since the anonymisation depends on the traffic sniffed respectively the trace file. Which means, it is inconsistent over multiple netsniff sessions. For instance when netsniff should crash for some reason or is restarted for maintenance, it would loose the lookup table held in memory.

Also there might be a problem that the structure to lookup the anonymised IPs grows to a significant size. From our current data in netsniff we have roughly 5200 distinct IP addresses over about ten weeks. Which is only a small fraction of the possible IP space. Figure 2 shows how the number of distinct IP addresses developed over time for the last 10 weeks for DNS and TCP traffic only. It is natural that the number of distinct IP addresses increases over time, but we don't know how rapidly this increases.

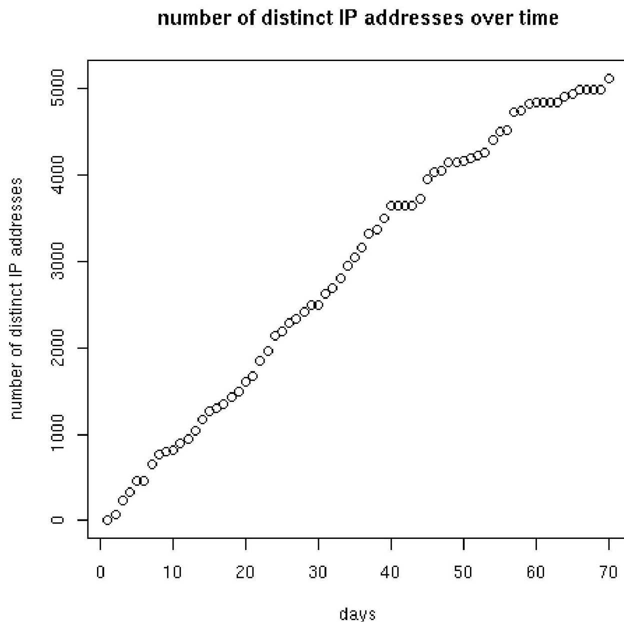

Figure 2: Number of distinct IPs over time

In terms of security [8] proves that the anonymization used in netsniff is as robust as is possible for prefix-preserving IP address anonymisation. It is easy to see, that if no prefix-preserving anonymisation is necessary, anonymisation can be made more robust. Since the guess of a mapping of a popular IP address doesn't necessarily allow to infer other mappings. An interesting article on this topic is described in [9].

B. String anonymisation

We use an anonymisation of strings in ARP, POP3, SMTP, FTP and DNS output of netsniff. DNS host names are anonymised to a string of 'x's with the same length, preserving the dots. In the other protocols a secure hash function is used, which is part of the OpenSSL package. The complete masking of the string as in the DNS part of netsniff is very secure and almost no deduction to the original host name can be made. But there is no way to correlate host names occurring in other protocols to the host name part of the DNS query.

```
http_output                   ::= tcp_output
                                  *(http_request)
                                  *(http_incomplete_request) CRLF
http_incomplete_request       ::= "INCOMPLETE" SP http_request
http_request                  ::= (http_connect_request | http_other_request
                                  | http_post_request) CRLF
http_connect_request          ::= "CONNECT" SP
                                  [http_private_info]
                                  http_return_code
http_post_request             ::= "POST" SP
                                  [http_private_info]
                                  http_post_upload_length SP
                                  http_content_type SP
                                  http_return_code SP
                                  http_download_length SP
                                  [["NOT_"]"cacheable"]
http_other_request            ::= http_request_type SP
                                  [http_private_info]
                                  http_content_type SP
                                  http_return_code SP
                                  http_download_length SP
                                  [["NOT_"]"cacheable"]
http_private_info             ::= http_request_url SP
                                  http_request_host SP
                                  http_request_referer SP
http_request_type             ::= "GET" | "HEAD" | "PUT" | "DELETE"
                                  | "TRACE" | "CONNECT" | "?"
http_request_url              ::= http_url
http_request_host             ::= hostname
http_request_referer          ::= http_url
```

```
mail_stat                     ::= "[" *(number SP) "]" CRLF
mail_num                      ::= number
mail_mean                     ::= decimal
mail_median                   ::= decimal
mail_stddev                   ::= decimal
```

```
imap4_output                  ::= tcp_output
                                  imap4_log_line1
                                  imap4_log_line2
                                  imap4_log_line3
imap4_log_line1               ::= (imap4_anonymized_user | imap4_user) SP
                                  imap4_authentication_method SP
                                  imap4_num_mails_listed SP
                                  imap4_num_mails_received SP
                                  imap4_completly_parsed SP
imap4_log_line2               ::= imap4_mails_listed
imap4_log_line2               ::= imap4_mails_received

imap4_anonymized_user         ::= string
imap4_user                    ::= string
imap4_mails_received          ::= mail_stat
imap4_mails_listed            ::= mail_stat
imap4_mails_deleted           ::= number
imap4_errors                  ::= number
imap4_num_protocol_errors     ::= number
```

```
ftp_output                    ::= tcp_output
                                  ftp_log_line1
                                  ftp_log_line2
ftp_log_line1                 ::= (ftp_username|ftp_anonymized_user) SP
                                  DQUOTE ftp_osstring DQUOTE SP
                                  ftp_num_fileoperations SP
                                  ftp_num_error_fileoperations SP
```

```
                                   ftp_num_unimplemented SP
                                   ftp_num_error_responses SP
                                   ftp_num_unknown_commands SP
                                   ftp_anonymous SP
                                   ftp_login CRLF
ftp_log_line2                  ::= ("LIST"|"RETR"|"SEND") SP
                                   ftp_data_stream_port CRLF
ftp_osstring                   ::= STRING
ftp_data_stream_port           ::= NUMBER
ftp_num_fileoperations         ::= NUMBER
ftp_num_error_fileops          ::= NUMBER
ftp_num_unimplemented          ::= NUMBER
ftp_num_error_responses        ::= NUMBER
ftp_num_unknown commands       ::= NUMBER
DQUOTE                         ::= '"'
```

```
pop3_output                    ::= tcp_output pop3_log_line1
                                   pop3_log_line2
                                   pop3_log_line3
pop3_log_line1                 ::= (pop3_anonymized_user | pop3_user) SP
                                    pop3_mails_deleted SP
                                    pop3_errors SP
                                    pop3_num_protocol_errors CRLF
pop3_log_line2                 ::= pop3_mails_received
pop3_log_line3                 ::= pop3_mails_inbox
pop3_anonymized_user           ::= string
pop3_user                      ::= string
pop3_mails_received            ::= mail_stat
pop3_mails_inbox               ::= mail_stat
pop3_mails_deleted             ::= number
pop3_errors                    ::= number
pop3_num_protocol_errors       ::= number
```

```
tlsstream_output               ::= tcp_output
tlssession_output              ::= tlssession_output_line1
                                   tlssession_output_lines
tlssession_output_line1        ::= tls_session_id SP
                                   tls_hadError SP
                                   tls_version SP
                                   tls_cipher SP
                                   tls_compression SP
                                   tls_payload_length SP
                                   tls_overhead SP
                                   tls_num_tcpstreams SP
                                   tls_session_duration CRLF
tlssession_output_lines        ::= 1*(tcp_timestamp)
tls_session_id                 ::= hex_string
hex_string                     ::= "0x" 1*([0123456789abcdef])
tls_hadError                   ::= bool
tls_version                    ::=
tls_cipher                     ::=
tls_compression                ::=
tls_payload_length             ::= number
tls_overhead                   ::= number
tls_num_tcpstreams             ::= number
bool                           ::= "0" | "1"
```

```
tcphistogram                   ::= number *("," number) CRLF
```

```
tcp_output                     ::= tcp_output_line1
                                   tcp_output_line2
                                   tcp_output_line3
                                   tcp_output_line4
                                   tcp_output_line5
tcp_output_line1               ::= tcp_timestamp SP
                                   tcp_duration SP
```

```
                                  trnsport_ip_output SP
                                  tcp_client2server_data SP
                                  tcp_client2server_overhead SP
                                  tcp_server2client_data SP
                                  tcp_server2client_overhead SP
                                  tcp_path_hop_count SP
                                  tcp_rtt SP
                                  tcp_jitter SP
                                  tcp_client2server_error_rate SP
                                  tcp_server2client_error_rate CRLF
tcp_output_line2               ::= "Hops Client:," tcphistogram CRLF
tcp_output_line3               ::= "Hops Server:," tcphistogram CRLF
tcp_output_line4               ::= "RTT:," tcphistogram CRLF
tcp_output_line5               ::= "Jitter:," tcphistogram CRLF
tcp_rtt                        ::= number
tcp_jitter                     ::= number
tcp_hop_num                    ::= decimal
tcp_timestamp                  ::= timestamp
tcp_duration                   ::= decimal
tcp_client2server_data         ::= number
tcp_client2server_overhead     ::= number
tcp_server2client_data         ::= number
tcp_server2client_overhead     ::= number
tcp_path_hop_count             ::= number
tcp_rtt                        ::= number
tcp_jitter                     ::= number
tcp_client2server_error_rate   ::= decimal
tcp_server2client_error_rate   ::= decimal
```

```
smtp_output                    ::= tcp_output smtp_log_line1 smtp_log_line2
smtp_log_line1                 ::= smtp_sender SP
                                  smtp_advertised_size SP
                                  smtp_num_commands SP
                                  smtp_num_unrecognized_commands CRLF
smtp_log_line2                 ::= mail_stat
smtp_sender                    ::= "["
                                  *(smtp_from_path | smtp_anonymized_path)
                                  "]"
advertises_size                ::= number
smtp_from_path                 ::= smtp_path
smtp_anonymized_path           ::= string
smtp_path                      ::=
smtp_num_commands              ::= number
smtp_num_unrecognized_commands ::= number
```

```
icmp_output                  ::= packet_output SP
                                  ip_output SP
                                  icmp_log_line CRLF
icmp_log_line                    ::= "[" icmp_code | icmp_type "]"
```

```
dns_log_line                   ::= packet_output SP
                                  udp_packet_output SP
                                  dns_output
dns_output                     ::= ("Response" | "Query")
                                  "(" dns_identification ")" SP
                                  dns_resultcode
                                  *(dns_query)
                                  *(dns_response) CRLF
dns_query                      ::= SP
                                  "Q(" (ip | hostname) SP
                                  "-" SP
                                  ("A" | "PTR" | dns_type) "/"
                                  ("IN" | "IN6" | dns_class) ")"
dns_response                   ::= SP
                                  "R(" (dns_response_a
                                  | dns_response_ptr | dns_response_other) SP
```

```
                                 "–" SP
                                 "TTL" SP
                                 ttl SP
                                 "seconds)"
dns_response_a               ::= hostname | ip
dns_response_ptr             ::= hostname | ip
dns_response_other           ::= hostname | ip
dns_identification           ::= number
dns_type                     ::= number
dns_class                    ::= number
dns_resultcode               ::= "NO_ERROR" | "FORMAT_ERROR"
                                 | "SERVER_FAILURE" | "NAME_ERROR"
                                 | "NOT_IMPLEMENTED" | "UNKNOWN"
```

```
packet_output                ::= packet_timestamp SP
                                 "Headers" "(" packet_header_length ")" SP
                                 "Payload" "(" packet_payload_length ")"
packet_timestamp             ::= timestamp
packet_header_length         ::= number
packet_payload_length        ::= number
```

```
udp_packet_output            ::= trnsport_ip_output SP
                                 ip_hops
```

```
trnsport_ip_output           ::= src_ip ":" src_port SP
                                 dst_ip ":" dst_Port
```

```
arp_output                   ::= packet_output SP arp_log_line
arp_log_line                 ::= ar_hrd SP ar_pro SP
                                 ar_hln SP ar_pln SP
                                 ar_op SP
                                 ar_sha SP ar_spa SP
                                 ar_tha SP ar_tpa
```

```
ip_output                    ::= src_ip SP
                                 dst_ip SP
                                 hops
ip_hops                      ::= "Hops" "(" num_hops ")"
src_ip                       ::= ip
dst_ip                       ::= ip
src_port                     ::= port
dst_port                     ::= port
num_hops                     ::= number

port                         ::=
ip                           ::=
hostname                     ::= hostname_part ["." hostname]
hostname_part                ::= 1*(HOSTNAMECHAR)
number                       ::=
url                          ::=
```