Passive Network Analysis Using Libtrace

Shane Alcock

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

25 November 2008

Network Research Group

Outline

- Introduction and Basics
- The Libtrace Tools
- Simple Libtrace Programming
- Advanced Topics

Part Three

- Simple Libtrace Programming
    - Opening and closing traces
    - Error handling
    - Compiling and running libtrace programs
    - Reading packets
    - Timestamps
    - Packet lengths
    - Simple protocol analysis
    - Writing packets
    - Packet filtering
    - Trace configuration options

Supplementary Material

- Source code examples
  - http://www.wand.net.nz/~salcock/tutorial/codedemo
- Examples are already available on the Live CD
- Will also be included in upcoming libtrace releases

Opening a Trace

```
libtrace_t *trace_create(char *uri);
```

- Opens a trace file or live capture for reading
- Location and format specified using the `uri` parameter
- Returns a pointer to the trace structure (libtrace_t)
- Returns NULL if an error occurs
- Created trace is not yet available for reading!

Preparing a Trace

```
int trace_start(libtrace_t *trace);
```

- Prepares a trace file or live capture for reading
- Applies any configuration options
- `trace` must have been previously created using trace_create()
- Returns 0 if successful, -1 if an error occurs
- Packets can now be read from the trace

Checking for Errors

```
bool trace_is_err(libtrace_t *trace);
```

- Returns true if the error state is set for the given trace
- Does not reset the error state

```
void trace_perror(libtrace_t *trace, const char *msg...);
```

- Very similar to perror() in standard C
- Prints a (hopefully useful) error message to stderr
- `msg` is prepended to the error message
- Clears the error status for the trace

Destroying a Trace

```
void trace_pause(libtrace_t *trace);
```

- Opposite of trace_start
- Current configuration options remain in effect

```
void trace_destroy(libtrace_t *trace);
```

- Closes a trace and frees up any resources it was using
- Will pause the trace prior to destruction if not already paused

Our First Libtrace Program

- Example - createdemo.c
    - Let's look at some actual libtrace code

Building our Program

- We now have a program – time to build it

- Compiling and linking against the libtrace library

```
gcc -ltrace -o createdemo createdemo.c
```

- If libtrace is installed to a non-default location

```
gcc -L/home/install/lib -I/home/install/include -ltrace -o
  createdemo createdemo.c
```

Libtrace Packets

```
libtrace_packet_t *trace_create_packet();
```

- Creates a structure for reading packets into
- Returns a pointer to an initialised libtrace packet
- Returns NULL in the event of an error

```
void trace_destroy_packet(libtrace_packet_t *packet);
```

- Fairly self-explanatory
- Frees all resources associated with the packet

Using Libtrace Packets

- Basic tricks for using libtrace packets effectively
  - Libtrace packets can (and should) be re-used
    - Most applications only need to create one libtrace packet
  - Do not touch the contents of the packet directly
    - Use libtrace functions to access the data you want
    - This also applies to the libtrace_t trace structure

Reading a Packet

```
int trace_read_packet(libtrace_t *trace, libtrace_packet_t
 *packet);
```

- Reads the next available packet from `trace` into `packet`
- `trace` must have been successfully started
- If `packet` already contains a packet, it will be replaced
- Returns 0 on EOF, -1 on error, otherwise the number of bytes read
  - Remember to handle errors appropriately!

Reading a Packet

- Example - readdemo.c
  - Expand our skeleton to read and count packets

Timestamps

```
uint64_t trace_get_erf_timestamp(libtrace_packet_t *packet);
struct timeval trace_get_timeval(libtrace_packet_t *packet);
double trace_get_seconds(libtrace_packet_t *packet);
```

- Returns the time that the provided packet was captured
- If capture timestamp is in a different format, it will be converted
  - e.g. calling `trace_get_erf_timestamp` on a pcap trace
- Time formats vary in accuracy and resolution

Timestamps

- Example - timedemo.c
  - Using timestamps to print counts every 10 seconds of trace time

Packet Length

```
size_t trace_get_capture_length(libtrace_packet_t *packet);
```

- Returns the current size of the packet
- Does not include the capture format header

```
size_t trace_get_wire_length(libtrace_packet_t *packet);
```

- Returns the size of the packet when it was first captured
- Does not include the capture format header
- Can include the Frame Check Sequence on Ethernet packets
  - e.g. DAG captures retain FCS, PCAP does not

Packet Length

```
size_t trace_get_framing_length(libtrace_packet_t *packet);
```

- Returns the size of the capture format framing header

```
size_t trace_set_capture_length(libtrace_packet_t *packet,
 size_t size);
```

- Truncates the packet to the suggested length
- If `size` is larger than the current capture length, the packet is unchanged
- Returns the new capture length

Packet Length

- Example - lengthdemo.c
  - Instead of just counting packets, let's try counting bytes

Helper Functions

```
uint16_t trace_get_source_port(libtrace_packet_t *packet);

uint16_t trace_get_destination_port(libtrace_packet_t *packet);
```

- Returns the requested port number from the transport header
- The port number is returned in HOST byte order
- Returns 0 if no port number is available

Helper Functions

```
struct sockaddr *trace_get_source_address(libtrace_packet_t
  *packet, struct sockaddr *addr);

struct sockaddr *trace_get_destination_address(libtrace_packet_t
  *packet, struct sockaddr *addr);
```

- Returns the requested IP address inside the provided sockaddr
- If `addr` is NULL, static storage is used to store the result
- Returns NULL, if no IP address is present (i.e. not an IP packet)
- Works for v4 or v6
- Some knowledge of sockaddr conventions in C is required

Helper Functions

```
uint8_t *trace_get_source_mac(libtrace_packet_t *packet);

uint8_t *trace_get_destination_mac(libtrace_packet_t *packet);
```

- Returns a pointer to the requested MAC address
- Works for both Ethernet and 802.11 frames
- Returns NULL if no MAC address available

Helper Functions

- Example - sourcedemo.c
  - Printing source MAC, IP and port

Helper Functions

```
libtrace_ip_t *trace_get_ip(libtrace_packet_t *packet);

libtrace_ip6_t *trace_get_ip6(libtrace_packet_t *packet);

libtrace_tcp_t *trace_get_tcp(libtrace_packet_t *packet);

libtrace_udp_t *trace_get_udp(libtrace_packet_t *packet);

libtrace_icmp_t *trace_get_icmp(libtrace_packet_t *packet);
```

- Easy direct access to the header for a particular protocol
- No need to worry about casting the returned header
- All functions return NULL if the required header is not present

Helper Functions

- Example - gettcpdemo.c
    - A better version of our TCP port 80 counting program

Writing Packets

```
libtrace_out_t *trace_create_output(char *uri);
```

- Opens a trace file for writing
- Location and format specified using the `uri` parameter
- Returns a pointer to the trace structure (libtrace_out_t)
- Returns NULL if an error occurs
- As with trace_create, trace is not yet ready for writing

Writing Packets

```
int trace_start_output(libtrace_out_t *trace);
```

- Prepares a trace file for writing
- Applies any configuration options
- Returns 0 if successful, -1 if an error occurs
- Can now write packets to the trace

Writing Packets

```
bool trace_is_err_output(libtrace_out_t *trace);
```

- Returns true if the error state is set for the given trace
- Does not reset the error state

```
void trace_perror_output(libtrace_out_t *trace, const char
  *msg...);
```

- Very similar to perror() in standard C
- Prints a (hopefully useful) error message to stderr
- msg is prepended to the error message
- Clears the error status for the trace

Writing Packets

```
void trace_destroy_output(libtrace_out_t *trace);
```

- Closes an output trace and frees up any resources it was using

Writing Packets

```
int trace_write_packet(libtrace_out_t *trace,
 libtrace_packet_t *packet);
```

- Writes the given packet to the output trace
- Returns -1 if an error occurs, otherwise the number of bytes written

Writing Packets

- Example - writedemo.c
  - Create a trace containing only TCP port 25 traffic

Filtering Packets

```
libtrace_filter_t *trace_create_filter(char *filterstring);
```

- Creates a libtrace filter object
- Will always return a valid filter – not compiled until first applied

```
int trace_apply_filter(libtrace_filter_t *filter,
 libtrace_packet_t *packet);
```

- Applies a libtrace filter to an individual packet
- Returns 0 if the filter does not match, >0 if it does
- Returns -1 if an error occurs

Filtering Packets

```
void trace_destroy_filter(libtrace_filter_t *filter);
```

- Deallocates all resources associated with a libtrace filter

Filtering Packets

- Example - filterdemo.c
    - Write our own version of tracefilter

Trace Configuration

```
int trace_config(libtrace_t *trace, trace_option_t option,
 void *value);
```

- Set a configuration option for a trace
- Configuration changes are applied when `trace_start` is called
- Returns -1 if configuration failed, 0 otherwise
- Some possible options for input traces
  - `TRACE_OPTION_SNAPLEN`
  - `TRACE_OPTION_PROMISC`
  - `TRACE_OPTION_FILTER`

Trace Configuration

```
int trace_config_output(libtrace_out_t *trace,
 trace_option_output_t option, void *value);
```

- Set a configuration option for an output trace
- Configuration changes are applied when `trace_start_output` is called
- Returns -1 if configuration failed, 0 otherwise
- Possible options for output traces
  - `TRACE_OPTION_OUTPUT_FILEFLAGS`
  - `TRACE_OPTION_OUTPUT_COMPRESS`

Trace Configuration

- Example - configdemo.c
  - Tracefilter Mark II (including output compression)

More Information

- API documentation via Doxygen
  - http://research.wand.net.nz/software/libtrace-docs/html/libtrace_8h.html
- Libtrace coding conventions
  - http://wand.net.nz/trac/libtrace/wiki/CodingConventions
- Libtrace Wiki
  - http://wand.net.nz/trac/libtrace/wiki/UserDocumentation

WAND Network Research Group
Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

www.crc.net.nz
www.wand.net.nz
www.waikato.ac.nz

WAND
Network Research Group

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato