

LANbeacon

Generated by Doxygen 1.8.13



# Contents

<b>1</b>	<b>Data Structure Index</b>	<b>1</b>
1.1	Data Structures . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Data Structure Documentation</b>	<b>5</b>
3.1	interfaces Struct Reference . . . . .	5
3.1.1	Detailed Description . . . . .	6
3.1.2	Field Documentation . . . . .	6
3.1.2.1	etherType . . . . .	6
3.1.2.2	if_idx . . . . .	6
3.1.2.3	if_mac . . . . .	6
3.1.2.4	maxSockFd . . . . .	6
3.1.2.5	numInterfaces . . . . .	7
3.1.2.6	sendOrReceive . . . . .	7
3.1.2.7	sockfd . . . . .	7
3.1.2.8	sockopt . . . . .	7
3.2	open_ssl_keys Struct Reference . . . . .	8
3.2.1	Detailed Description . . . . .	8
3.2.2	Field Documentation . . . . .	8
3.2.2.1	generate_keys . . . . .	8
3.2.2.2	path_To_Signing_Key . . . . .	9
3.2.2.3	path_To_Verifying_Key . . . . .	9

3.2.2.4	<a href="#">pcszPassphrase</a>	9
3.2.2.5	<a href="#">sender_or_receiver_mode</a>	9
3.3	<a href="#">received_lan_beacon_frame Struct Reference</a>	10
3.3.1	<a href="#">Detailed Description</a>	10
3.3.2	<a href="#">Field Documentation</a>	10
3.3.2.1	<a href="#">challenge</a>	10
3.3.2.2	<a href="#">current_destination_mac</a>	11
3.3.2.3	<a href="#">lan_beacon_ReceivedPayload</a>	11
3.3.2.4	<a href="#">parsedBeaconContents</a>	11
3.3.2.5	<a href="#">payloadSize</a>	11
3.3.2.6	<a href="#">successfullyAuthenticated</a>	11
3.3.2.7	<a href="#">times_left_to_display</a>	12
3.4	<a href="#">receiver_information Struct Reference</a>	12
3.4.1	<a href="#">Detailed Description</a>	13
3.4.2	<a href="#">Field Documentation</a>	13
3.4.2.1	<a href="#">authenticated_mode</a>	13
3.4.2.2	<a href="#">current_lan_beacon_pdu_for_printing</a>	13
3.4.2.3	<a href="#">lanbeacon_keys</a>	13
3.4.2.4	<a href="#">my_receiver_interfaces</a>	13
3.4.2.5	<a href="#">number_of_currently_received_frames</a>	14
3.4.2.6	<a href="#">pointers_to_received_frames</a>	14
3.4.2.7	<a href="#">scroll_speed</a>	14
3.5	<a href="#">sender_information Struct Reference</a>	14
3.5.1	<a href="#">Detailed Description</a>	15
3.5.2	<a href="#">Field Documentation</a>	15
3.5.2.1	<a href="#">interface_to_send_on</a>	16
3.5.2.2	<a href="#">lan_beacon_pdu_len</a>	16
3.5.2.3	<a href="#">lanbeacon_keys</a>	16
3.5.2.4	<a href="#">lanBeacon_PDU</a>	16
3.5.2.5	<a href="#">my_challenge_receiver_interfaces</a>	16
3.5.2.6	<a href="#">send_frequency</a>	16

<b>4 File Documentation</b>	<b>17</b>
4.1 define.h File Reference	17
4.1.1 Detailed Description	18
4.1.2 Macro Definition Documentation	19
4.1.2.1 _	19
4.1.2.2 CHALLENGE_ETHTYPE	19
4.1.2.3 DEFAULT_SCROLLSPEED	19
4.1.2.4 DESCRIPTOR_COMBINED_STRING	19
4.1.2.5 DESCRIPTOR_CUSTOM	19
4.1.2.6 DESCRIPTOR_DHCP	20
4.1.2.7 DESCRIPTOR_EMAIL	20
4.1.2.8 DESCRIPTOR_IPV4	20
4.1.2.9 DESCRIPTOR_IPV6	20
4.1.2.10 DESCRIPTOR_NAME	20
4.1.2.11 DESCRIPTOR_ROUTER	20
4.1.2.12 DESCRIPTOR_SIGNATURE	21
4.1.2.13 DESCRIPTOR_VLAN_ID	21
4.1.2.14 DESCRIPTOR_WIDTH	21
4.1.2.15 KEY_PATHLENGTH_MAX	21
4.1.2.16 LAN_BEACON_BUF_SIZ	21
4.1.2.17 LAN_BEACON_DEST_MAC	21
4.1.2.18 LAN_BEACON_ETHER_TYPE	22
4.1.2.19 LAN_BEACON_SEND_FREQUENCY	22
4.1.2.20 PARSED_TLV_MAX_LENGTH	22
4.1.2.21 PARSED_TLV_MAX_NUMBER	22
4.1.2.22 PRIVATE_KEY_STANDARD_PATH	22
4.1.2.23 PUBLIC_KEY_STANDARD_PATH	22
4.1.2.24 SHOW_FRAMES_X_TIMES	23
4.1.2.25 SUBTYPE_COMBINED_STRING	23
4.1.2.26 SUBTYPE_CUSTOM	23

4.1.2.27	SUBTYPE_DHCP	23
4.1.2.28	SUBTYPE_EMAIL	23
4.1.2.29	SUBTYPE_IPV4	23
4.1.2.30	SUBTYPE_IPV6	24
4.1.2.31	SUBTYPE_NAME	24
4.1.2.32	SUBTYPE_ROUTER	24
4.1.2.33	SUBTYPE_SIGNATURE	24
4.1.2.34	SUBTYPE_VLAN_ID	24
4.2	main.c File Reference	25
4.2.1	Function Documentation	25
4.2.1.1	main()	25
4.2.1.2	printHelp()	26
4.3	main.h File Reference	27
4.3.1	Detailed Description	28
4.3.2	Function Documentation	28
4.3.2.1	main()	28
4.3.2.2	printHelp()	29
4.4	openssl_sign.c File Reference	29
4.4.1	Macro Definition Documentation	30
4.4.1.1	KEY_READ_PROBLEM	30
4.4.1.2	NO_PRIVATE_KEY	30
4.4.1.3	NO_PUBLIC_KEY	30
4.4.1.4	PROBLEM_IN_SIGN_CALL	31
4.4.1.5	PROBLEM_IN_VERIFY_CALL	31
4.4.1.6	SIG_LEN	31
4.4.1.7	VERFIY_PROBLEM	31
4.4.2	Function Documentation	31
4.4.2.1	make_keys()	31
4.4.2.2	passwd_callback()	32
4.4.2.3	print_it()	33

4.4.2.4	<a href="#">read_keys()</a>	33
4.4.2.5	<a href="#">signlanbeacon()</a>	34
4.4.2.6	<a href="#">verifylanbeacon()</a>	35
4.4.3	<a href="#">Variable Documentation</a>	36
4.4.3.1	<a href="#">hn</a>	36
4.5	<a href="#">openssl_sign.h File Reference</a>	36
4.5.1	<a href="#">Detailed Description</a>	37
4.5.2	<a href="#">Macro Definition Documentation</a>	37
4.5.2.1	<a href="#">RECEIVER_MODE</a>	37
4.5.2.2	<a href="#">SENDER_MODE</a>	38
4.5.3	<a href="#">Function Documentation</a>	38
4.5.3.1	<a href="#">make_keys()</a>	38
4.5.3.2	<a href="#">passwd_callback()</a>	39
4.5.3.3	<a href="#">print_it()</a>	40
4.5.3.4	<a href="#">read_keys()</a>	40
4.5.3.5	<a href="#">signlanbeacon()</a>	41
4.5.3.6	<a href="#">verifylanbeacon()</a>	42
4.6	<a href="#">rawsocket_LAN_Beacon.c File Reference</a>	43
4.6.1	<a href="#">Macro Definition Documentation</a>	44
4.6.1.1	<a href="#">_GNU_SOURCE</a>	45
4.6.2	<a href="#">Function Documentation</a>	45
4.6.2.1	<a href="#">getInterfaces()</a>	45
4.6.2.2	<a href="#">lan_beacon_receiver()</a>	45
4.6.2.3	<a href="#">receiveChallenge()</a>	46
4.6.2.4	<a href="#">send_lan_beacon_rawSock()</a>	47
4.6.2.5	<a href="#">sendRawSocket()</a>	48
4.7	<a href="#">rawsocket_LAN_Beacon.h File Reference</a>	49
4.7.1	<a href="#">Detailed Description</a>	51
4.7.2	<a href="#">Macro Definition Documentation</a>	51
4.7.2.1	<a href="#">REC_SOCKET</a>	51

4.7.2.2	SEND_SOCKET	51
4.7.3	Function Documentation	51
4.7.3.1	getInterfaces()	51
4.7.3.2	lan_beacon_receiver()	52
4.7.3.3	receiveChallenge()	53
4.7.3.4	send_lan_beacon_rawSock()	54
4.7.3.5	sendRawSocket()	55
4.8	receiver.c File Reference	56
4.8.1	Macro Definition Documentation	57
4.8.1.1	TLV_CUSTOM_COPY	57
4.8.1.2	TLV_STRING_COPY	57
4.8.2	Function Documentation	57
4.8.2.1	bananaPIprint()	57
4.8.2.2	evaluatelanbeacon()	58
4.8.2.3	receiver()	59
4.9	receiver.h File Reference	60
4.9.1	Detailed Description	61
4.9.2	Function Documentation	62
4.9.2.1	bananaPIprint()	62
4.9.2.2	evaluatelanbeacon()	62
4.9.2.3	receiver()	63
4.10	sender.c File Reference	64
4.10.1	Function Documentation	65
4.10.1.1	ipParser()	66
4.10.1.2	lanbeacon_creator()	67
4.10.1.3	sender()	68
4.10.1.4	transfer_to_pdu()	69
4.10.1.5	transfer_to_pdu_and_string()	70
4.10.1.6	transfer_to_string()	71
4.11	sender.h File Reference	71
4.11.1	Detailed Description	73
4.11.2	Function Documentation	73
4.11.2.1	ipParser()	73
4.11.2.2	lanbeacon_creator()	74
4.11.2.3	sender()	75
4.11.2.4	transfer_to_pdu()	76
4.11.2.5	transfer_to_pdu_and_string()	77
4.11.2.6	transfer_to_string()	78



# Chapter 1

## Data Structure Index

### 1.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">interfaces</a>	Contains all variables, that are needed to access sockets on interfaces . . . . .	5
<a href="#">open_ssl_keys</a>	Key locations, password and further configurations . . . . .	8
<a href="#">received_lan_beacon_frame</a>	Contains all the information related to one received frame . . . . .	10
<a href="#">receiver_information</a>	Receiver configurations . . . . .	12
<a href="#">sender_information</a>	Sender configurations . . . . .	14



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">define.h</a>	Contains application-wide includes with information such as addresses and TLV types . . . . .	17
<a href="#">main.c</a>		25
<a href="#">main.h</a>	Main function and help function . . . . .	27
<a href="#">openssl_sign.c</a>		29
<a href="#">openssl_sign.h</a>	Signing, verifying and key I/O . . . . .	36
<a href="#">rawsocket_LAN_Beacon.c</a>		43
<a href="#">rawsocket_LAN_Beacon.h</a>	Raw-socket sending and receiving . . . . .	49
<a href="#">receiver.c</a>		56
<a href="#">receiver.h</a>	Receiver-specific functions and structures . . . . .	60
<a href="#">sender.c</a>		64
<a href="#">sender.h</a>	Sender-specific functions and structures . . . . .	71



## Chapter 3

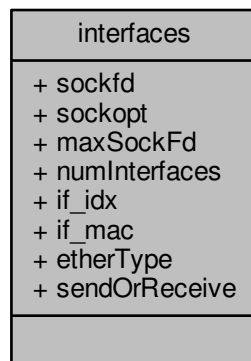
# Data Structure Documentation

### 3.1 interfaces Struct Reference

Contains all variables, that are needed to access sockets on interfaces.

```
#include <receiver.h>
```

Collaboration diagram for interfaces:



#### Data Fields

- int [sockfd](#) [20]
- int [sockopt](#) [20]
- int [maxSockFd](#)
- int [numInterfaces](#)
- struct ifreq [if\\_idx](#) [20]
- struct ifreq [if\\_mac](#) [20]
- unsigned short [etherType](#)
- unsigned short [sendOrReceive](#)

### 3.1.1 Detailed Description

Contains all variables, that are needed to access sockets on interfaces.

### 3.1.2 Field Documentation

#### 3.1.2.1 etherType

```
unsigned short interfaces::etherType
```

EtherType to send or receive on interface.

Referenced by `getInterfaces()`, and `sendRawSocket()`.

#### 3.1.2.2 if\_idx

```
struct ifreq interfaces::if_idx[20]
```

Interface IDs.

Referenced by `getInterfaces()`, and `sendRawSocket()`.

#### 3.1.2.3 if\_mac

```
struct ifreq interfaces::if_mac[20]
```

Interface MACs.

Referenced by `getInterfaces()`, and `sendRawSocket()`.

#### 3.1.2.4 maxSockFd

```
int interfaces::maxSockFd
```

Needed for select function.

Referenced by `getInterfaces()`, `lan_beacon_receiver()`, and `receiveChallenge()`.

### 3.1.2.5 numInterfaces

```
int interfaces::numInterfaces
```

Number of used interfaces.

Referenced by `getInterfaces()`, `lan_beacon_receiver()`, `receiveChallenge()`, and `sendRawSocket()`.

### 3.1.2.6 sendOrReceive

```
unsigned short interfaces::sendOrReceive
```

Switch for send or receive mode.

Referenced by `getInterfaces()`.

### 3.1.2.7 sockfd

```
int interfaces::sockfd[20]
```

File descriptors of raw sockets.

Referenced by `getInterfaces()`, `lan_beacon_receiver()`, `receiveChallenge()`, and `sendRawSocket()`.

### 3.1.2.8 sockopt

```
int interfaces::sockopt[20]
```

. Options for each raw socket.

Referenced by `getInterfaces()`.

The documentation for this struct was generated from the following file:

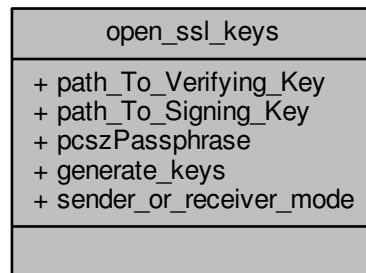
- [receiver.h](#)

## 3.2 open\_ssl\_keys Struct Reference

Key locations, password and further configurations.

```
#include <openssl_sign.h>
```

Collaboration diagram for open\_ssl\_keys:



### Data Fields

- char [path\\_To\\_Verifying\\_Key](#) [KEY\_PATHLENGTH\_MAX+1]
- char [path\\_To\\_Signing\\_Key](#) [KEY\_PATHLENGTH\_MAX+1]
- char [pcszPassphrase](#) [1024]
- int [generate\\_keys](#)
- int [sender\\_or\\_receiver\\_mode](#)

### 3.2.1 Detailed Description

Key locations, password and further configurations.

### 3.2.2 Field Documentation

#### 3.2.2.1 generate\_keys

```
int open_ssl_keys::generate_keys
```

Flag that determines, if keys should be generated.

Referenced by [lanbeacon\\_creator\(\)](#), and [signlanbeacon\(\)](#).



### 3.2.2.2 path\_To\_Signing\_Key

```
char open_ssl_keys::path_To_Signing_Key[KEY_PATHLENGTH_MAX+1]
```

Specified path of private key location.

Referenced by lanbeacon\_creator(), make\_keys(), and read\_keys().

### 3.2.2.3 path\_To\_Verifying\_Key

```
char open_ssl_keys::path_To_Verifying_Key[KEY_PATHLENGTH_MAX+1]
```

Specified path of public key location.

Referenced by lanbeacon\_creator(), make\_keys(), read\_keys(), receiver(), signlanbeacon(), and verifylanbeacon().

### 3.2.2.4 pcszPassphrase

```
char open_ssl_keys::pcszPassphrase[1024]
```

Specified password for private key.

Referenced by lanbeacon\_creator(), make\_keys(), and read\_keys().

### 3.2.2.5 sender\_or\_receiver\_mode

```
int open_ssl_keys::sender_or_receiver_mode
```

Flag for corresponding client mode.

Referenced by read\_keys(), and receiver().

The documentation for this struct was generated from the following file:

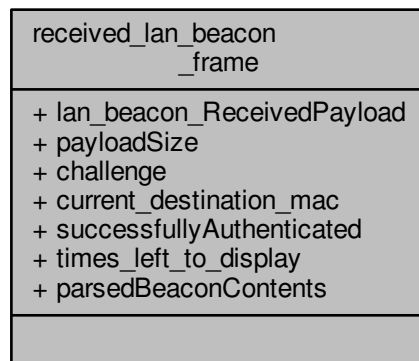
- [openssl\\_sign.h](#)

### 3.3 received\_lan\_beacon\_frame Struct Reference

Contains all the information related to one received frame.

```
#include <receiver.h>
```

Collaboration diagram for received\_lan\_beacon\_frame:



#### Data Fields

- unsigned char [lan\\_beacon\\_ReceivedPayload](#) [[LAN\\_BEACON\\_BUF\\_SIZ](#)]
- ssize\_t [payloadSize](#)
- unsigned long [challenge](#)
- unsigned char [current\\_destination\\_mac](#) [6]
- int [successfullyAuthenticated](#)
- int [times\\_left\\_to\\_display](#)
- char \*\* [parsedBeaconContents](#)

#### 3.3.1 Detailed Description

Contains all the information related to one received frame.

#### 3.3.2 Field Documentation

##### 3.3.2.1 challenge

```
unsigned long received_lan_beacon_frame::challenge
```

The challenge, that has been sent to the server.

Referenced by [evaluatelanbeacon\(\)](#), and [lan\\_beacon\\_receiver\(\)](#).

### 3.3.2.2 current\_destination\_mac

```
unsigned char received_lan_beacon_frame::current_destination_mac[6]
```

The MAC address of the server, which the frame was received from.

Referenced by `lan_beacon_receiver()`.

### 3.3.2.3 lan\_beacon\_ReceivedPayload

```
unsigned char received_lan_beacon_frame::lan_beacon_ReceivedPayload[LAN_BEACON_BUF_SIZ]
```

Contains the raw received payload from a LAN-Beacon frame.

Referenced by `evaluatelanbeacon()`, and `lan_beacon_receiver()`.

### 3.3.2.4 parsedBeaconContents

```
char** received_lan_beacon_frame::parsedBeaconContents
```

Contains the parsed contents, that will be used to print something to the display.

Referenced by `bananaPIprint()`, `lan_beacon_receiver()`, and `receiver()`.

### 3.3.2.5 payloadSize

```
ssize_t received_lan_beacon_frame::payloadSize
```

The size of the raw payload.

Referenced by `evaluatelanbeacon()`, and `lan_beacon_receiver()`.

### 3.3.2.6 successfullyAuthenticated

```
int received_lan_beacon_frame::successfullyAuthenticated
```

Has frame already been authenticated?

Referenced by `evaluatelanbeacon()`, and `lan_beacon_receiver()`.

### 3.3.2.7 times\_left\_to\_display

```
int received_lan_beacon_frame::times_left_to_display
```

Countdown, how many more times the frame will be displayed. Is updated, if frame with same content is received again.

Referenced by `bananaPlprint()`, and `lan_beacon_receiver()`.

The documentation for this struct was generated from the following file:

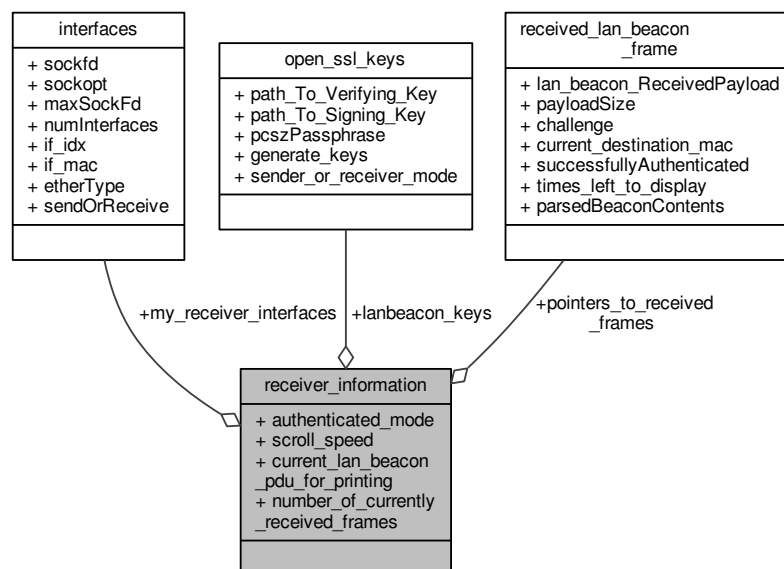
- [receiver.h](#)

## 3.4 receiver\_information Struct Reference

Receiver configurations.

```
#include <receiver.h>
```

Collaboration diagram for `receiver_information`:



### Data Fields

- int [authenticated\\_mode](#)
- int [scroll\\_speed](#)
- int [current\\_lan\\_beacon\\_pdu\\_for\\_printing](#)
- struct [received\\_lan\\_beacon\\_frame](#) \* [pointers\\_to\\_received\\_frames](#) [20]
- int [number\\_of\\_currently\\_received\\_frames](#)
- struct [open\\_ssl\\_keys](#) [lanbeacon\\_keys](#)
- struct [interfaces](#) [my\\_receiver\\_interfaces](#)

### 3.4.1 Detailed Description

Receiver configurations.

### 3.4.2 Field Documentation

#### 3.4.2.1 authenticated\_mode

```
int receiver_information::authenticated_mode
```

Has user specified using the authenticated mode?

Referenced by `lan_beacon_receiver()`, and `receiver()`.

#### 3.4.2.2 current\_lan\_beacon\_pdu\_for\_printing

```
int receiver_information::current_lan_beacon_pdu_for_printing
```

The currently printed PDU.

Referenced by `bananaPIprint()`, and `receiver()`.

#### 3.4.2.3 lanbeacon\_keys

```
struct open_ssl_keys receiver_information::lanbeacon_keys
```

The paths to the keys.

Referenced by `lan_beacon_receiver()`, and `receiver()`.

#### 3.4.2.4 my\_receiver\_interfaces

```
struct interfaces receiver_information::my_receiver_interfaces
```

Interfaces, that are used for LAN-Beacon reception.

Referenced by `lan_beacon_receiver()`, and `receiver()`.

#### 3.4.2.5 number\_of\_currently\_received\_frames

```
int receiver_information::number_of_currently_received_frames
```

How many frames are currently stored for displaying.

Referenced by `bananaPIprint()`, `lan_beacon_receiver()`, and `receiver()`.

#### 3.4.2.6 pointers\_to\_received\_frames

```
struct received\_lan\_beacon\_frame* receiver_information::pointers_to_received_frames[20]
```

Frames, that currently are stored for displaying.

Referenced by `bananaPIprint()`, `lan_beacon_receiver()`, and `receiver()`.

#### 3.4.2.7 scroll\_speed

```
int receiver_information::scroll_speed
```

How fast the display should switch to the next display page.

Referenced by `bananaPIprint()`, and `receiver()`.

The documentation for this struct was generated from the following file:

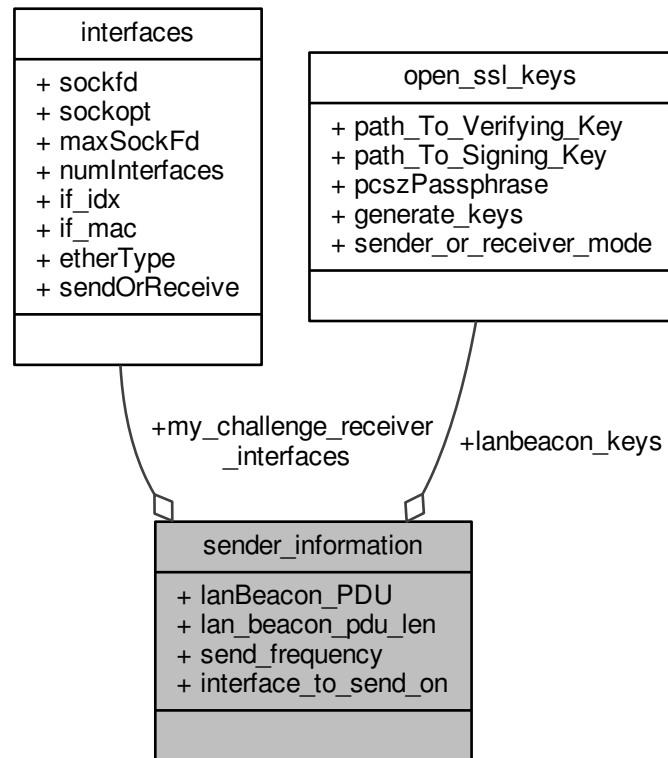
- [receiver.h](#)

### 3.5 sender\_information Struct Reference

Sender configurations.

```
#include <sender.h>
```

Collaboration diagram for sender\_information:



## Data Fields

- `char *` [lanBeacon\\_PDU](#)
- `int` [lan\\_beacon\\_pdu\\_len](#)
- `int` [send\\_frequency](#)
- `char *` [interface\\_to\\_send\\_on](#)
- `struct` [interfaces](#) [my\\_challenge\\_receiver\\_interfaces](#)
- `struct` [open\\_ssl\\_keys](#) [lanbeacon\\_keys](#)

### 3.5.1 Detailed Description

Sender configurations.

### 3.5.2 Field Documentation

### 3.5.2.1 interface\_to\_send\_on

```
char* sender_information::interface_to_send_on
```

If specified by start parameters, interface that is used for sending.

Referenced by lanbeacon\_creator(), sender(), and sendRawSocket().

### 3.5.2.2 lan\_beacon\_pdu\_len

```
int sender_information::lan_beacon_pdu_len
```

Length of the combined PDU.

Referenced by lanbeacon\_creator(), and send\_lan\_beacon\_rawSock().

### 3.5.2.3 lanbeacon\_keys

```
struct open_ssl_keys sender_information::lanbeacon_keys
```

Keys configuration.

Referenced by lanbeacon\_creator(), and sendRawSocket().

### 3.5.2.4 lanBeacon\_PDU

```
char* sender_information::lanBeacon_PDU
```

The combined payload of a PDU, that is being sent.

Referenced by send\_lan\_beacon\_rawSock(), and sender().

### 3.5.2.5 my\_challenge\_receiver\_interfaces

```
struct interfaces sender_information::my_challenge_receiver_interfaces
```

Interfaces that are used for receiving challenges.

Referenced by sender(), and sendRawSocket().

### 3.5.2.6 send\_frequency

```
int sender_information::send_frequency
```

Number of seconds between each sent PDU.

Referenced by lanbeacon\_creator(), receiveChallenge(), and sendRawSocket().

The documentation for this struct was generated from the following file:

- [sender.h](#)



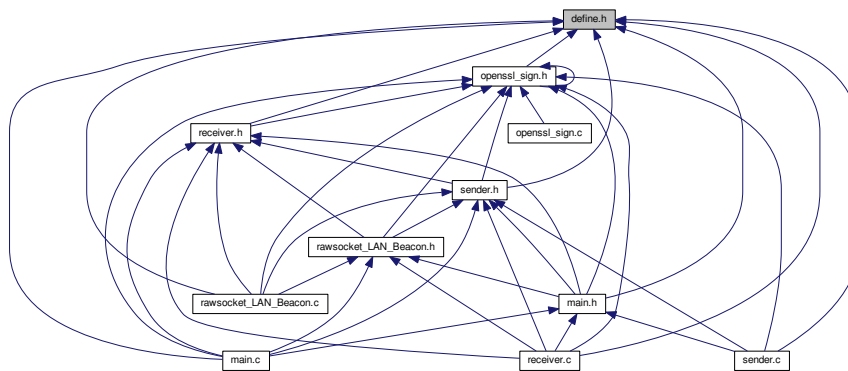
## Chapter 4

# File Documentation

### 4.1 define.h File Reference

Contains application-wide includes with information such as addresses and TLV types.

This graph shows which files directly or indirectly include this file:



### Macros

#### Macro for gettext localization support

- `#define _(STRING) gettext(STRING)`

#### Protocol options such send frequency

- `#define LAN_BEACON_SEND_FREQUENCY 5`

#### LAN-Beacon Multicast addresses and EtherTypes

- `#define LAN_BEACON_DEST_MAC 0xff, 0xff, 0xff, 0xff, 0xff, 0xff`
- `#define LAN_BEACON_ETHER_TYPE 0x88B5`
- `#define CHALLENGE_ETHERTYPE 0x88B6`

### Buffer sizes

- #define `PARSED_TLV_MAX_NUMBER` 25
- #define `PARSED_TLV_MAX_LENGTH` 510
- #define `LAN_BEACON_BUF_SIZ` 2000
- #define `KEY_PATHLENGTH_MAX` 800

### Standard paths

- #define `PRIVATE_KEY_STANDARD_PATH` "private\_key.pem"
- #define `PUBLIC_KEY_STANDARD_PATH` "public\_key.pem"

### Display options

- #define `DEFAULT_SCROLLSPEED` 5
- #define `SHOW_FRAMES_X_TIMES` 2
- #define `DESCRIPTOR_WIDTH` 10

### Subtype numbers lanbeacon

- #define `SUBTYPE_VLAN_ID` 200
- #define `SUBTYPE_NAME` 201
- #define `SUBTYPE_CUSTOM` 202
- #define `SUBTYPE_IPV4` 203
- #define `SUBTYPE_IPV6` 204
- #define `SUBTYPE_EMAIL` 205
- #define `SUBTYPE_DHCP` 206
- #define `SUBTYPE_ROUTER` 207
- #define `SUBTYPE_SIGNATURE` 216
- #define `SUBTYPE_COMBINED_STRING` 217

### Descriptor strings lanbeacon

- #define `DESCRIPTOR_VLAN_ID` gettext("VLAN-ID:")
- #define `DESCRIPTOR_NAME` gettext("VLAN-Name:")
- #define `DESCRIPTOR_CUSTOM` gettext("Freetext:")
- #define `DESCRIPTOR_IPV4` gettext("IPv4:")
- #define `DESCRIPTOR_IPV6` gettext("IPv6:")
- #define `DESCRIPTOR_EMAIL` gettext("Email:")
- #define `DESCRIPTOR_DHCP` gettext("DHCP:")
- #define `DESCRIPTOR_ROUTER` gettext("Router:")
- #define `DESCRIPTOR_SIGNATURE` gettext("Authentication:")
- #define `DESCRIPTOR_COMBINED_STRING` gettext("Combined String:")

## 4.1.1 Detailed Description

Contains application-wide includes with information such as addresses and TLV types.

### Author

Dominik Bitzer

### Date

2017

## 4.1.2 Macro Definition Documentation

### 4.1.2.1 `_`

```
#define _(  
    STRING ) gettext(STRING)
```

Referenced by `bananaPIprint()`, `evaluatelanbeacon()`, `getInterfaces()`, `ipParser()`, `lan_beacon_receiver()`, `lanbeacon_creator()`, `make_keys()`, `printHelp()`, `read_keys()`, `receiveChallenge()`, `receiver()`, `sendRawSocket()`, `signlanbeacon()`, `transfer_to_pdu()`, `transfer_to_string()`, and `verifylanbeacon()`.

### 4.1.2.2 `CHALLENGE_ETHTYPE`

```
#define CHALLENGE_ETHTYPE 0x88B6
```

Referenced by `lan_beacon_receiver()`, `sender()`, and `sendRawSocket()`.

### 4.1.2.3 `DEFAULT_SCROLLSPEED`

```
#define DEFAULT_SCROLLSPEED 5
```

Referenced by `receiver()`.

### 4.1.2.4 `DESCRIPTOR_COMBINED_STRING`

```
#define DESCRIPTOR_COMBINED_STRING gettext("Combined String:")
```

Referenced by `evaluatelanbeacon()`.

### 4.1.2.5 `DESCRIPTOR_CUSTOM`

```
#define DESCRIPTOR_CUSTOM gettext("Freetext:")
```

Referenced by `evaluatelanbeacon()`, and `lanbeacon_creator()`.

#### 4.1.2.6 DESCRIPTOR\_DHCP

```
#define DESCRIPTOR_DHCP gettext("DHCP:")
```

Referenced by `evaluatelanbeacon()`, and `lanbeacon_creator()`.

#### 4.1.2.7 DESCRIPTOR\_EMAIL

```
#define DESCRIPTOR_EMAIL gettext("Email:")
```

Referenced by `evaluatelanbeacon()`, and `lanbeacon_creator()`.

#### 4.1.2.8 DESCRIPTOR\_IPV4

```
#define DESCRIPTOR_IPV4 gettext("IPv4:")
```

Referenced by `evaluatelanbeacon()`, and `ipParser()`.

#### 4.1.2.9 DESCRIPTOR\_IPV6

```
#define DESCRIPTOR_IPV6 gettext("IPv6:")
```

Referenced by `evaluatelanbeacon()`, and `ipParser()`.

#### 4.1.2.10 DESCRIPTOR\_NAME

```
#define DESCRIPTOR_NAME gettext("VLAN-Name:")
```

Referenced by `evaluatelanbeacon()`, and `lanbeacon_creator()`.

#### 4.1.2.11 DESCRIPTOR\_ROUTER

```
#define DESCRIPTOR_ROUTER gettext("Router:")
```

Referenced by `evaluatelanbeacon()`, and `lanbeacon_creator()`.

#### 4.1.2.12 DESCRIPTOR\_SIGNATURE

```
#define DESCRIPTOR_SIGNATURE gettext("Authentication:")
```

Referenced by `evaluatelanbeacon()`.

#### 4.1.2.13 DESCRIPTOR\_VLAN\_ID

```
#define DESCRIPTOR_VLAN_ID gettext("VLAN-ID:")
```

Referenced by `evaluatelanbeacon()`, and `lanbeacon_creator()`.

#### 4.1.2.14 DESCRIPTOR\_WIDTH

```
#define DESCRIPTOR_WIDTH 10
```

Referenced by `bananaPlprint()`.

#### 4.1.2.15 KEY\_PATHLENGTH\_MAX

```
#define KEY_PATHLENGTH_MAX 800
```

Referenced by `lanbeacon_creator()`, and `receiver()`.

#### 4.1.2.16 LAN\_BEACON\_BUF\_SIZ

```
#define LAN_BEACON_BUF_SIZ 2000
```

Referenced by `lan_beacon_receiver()`, and `sendRawSocket()`.

#### 4.1.2.17 LAN\_BEACON\_DEST\_MAC

```
#define LAN_BEACON_DEST_MAC 0xff, 0xff, 0xff, 0xff, 0xff, 0xff
```

Referenced by `lan_beacon_receiver()`, and `send_lan_beacon_rawSock()`.

#### 4.1.2.18 LAN\_BEACON\_ETHER\_TYPE

```
#define LAN_BEACON_ETHER_TYPE 0x88B5
```

Referenced by receiver(), send\_lan\_beacon\_rawSock(), and sendRawSocket().

#### 4.1.2.19 LAN\_BEACON\_SEND\_FREQUENCY

```
#define LAN_BEACON_SEND_FREQUENCY 5
```

Referenced by sender().

#### 4.1.2.20 PARSED\_TLVS\_MAX\_LENGTH

```
#define PARSED_TLVS_MAX_LENGTH 510
```

Referenced by evaluatelanbeacon().

#### 4.1.2.21 PARSED\_TLVS\_MAX\_NUMBER

```
#define PARSED_TLVS_MAX_NUMBER 25
```

Referenced by bananaPlprint(), evaluatelanbeacon(), and receiver().

#### 4.1.2.22 PRIVATE\_KEY\_STANDARD\_PATH

```
#define PRIVATE_KEY_STANDARD_PATH "private_key.pem"
```

Referenced by sender().

#### 4.1.2.23 PUBLIC\_KEY\_STANDARD\_PATH

```
#define PUBLIC_KEY_STANDARD_PATH "public_key.pem"
```

Referenced by receiver(), and sender().

#### 4.1.2.24 SHOW\_FRAMES\_X\_TIMES

```
#define SHOW_FRAMES_X_TIMES 2
```

Referenced by `lan_beacon_receiver()`.

#### 4.1.2.25 SUBTYPE\_COMBINED\_STRING

```
#define SUBTYPE_COMBINED_STRING 217
```

Referenced by `evaluatelanbeacon()`, and `lanbeacon_creator()`.

#### 4.1.2.26 SUBTYPE\_CUSTOM

```
#define SUBTYPE_CUSTOM 202
```

Referenced by `evaluatelanbeacon()`, and `lanbeacon_creator()`.

#### 4.1.2.27 SUBTYPE\_DHCP

```
#define SUBTYPE_DHCP 206
```

Referenced by `evaluatelanbeacon()`, and `lanbeacon_creator()`.

#### 4.1.2.28 SUBTYPE\_EMAIL

```
#define SUBTYPE_EMAIL 205
```

Referenced by `evaluatelanbeacon()`, and `lanbeacon_creator()`.

#### 4.1.2.29 SUBTYPE\_IPV4

```
#define SUBTYPE_IPV4 203
```

Referenced by `evaluatelanbeacon()`, and `ipParser()`.

#### 4.1.2.30 SUBTYPE\_IPV6

```
#define SUBTYPE_IPV6 204
```

Referenced by `evaluatelanbeacon()`, and `ipParser()`.

#### 4.1.2.31 SUBTYPE\_NAME

```
#define SUBTYPE_NAME 201
```

Referenced by `evaluatelanbeacon()`, and `lanbeacon_creator()`.

#### 4.1.2.32 SUBTYPE\_ROUTER

```
#define SUBTYPE_ROUTER 207
```

Referenced by `evaluatelanbeacon()`, and `lanbeacon_creator()`.

#### 4.1.2.33 SUBTYPE\_SIGNATURE

```
#define SUBTYPE_SIGNATURE 216
```

Referenced by `evaluatelanbeacon()`, and `sendRawSocket()`.

#### 4.1.2.34 SUBTYPE\_VLAN\_ID

```
#define SUBTYPE_VLAN_ID 200
```

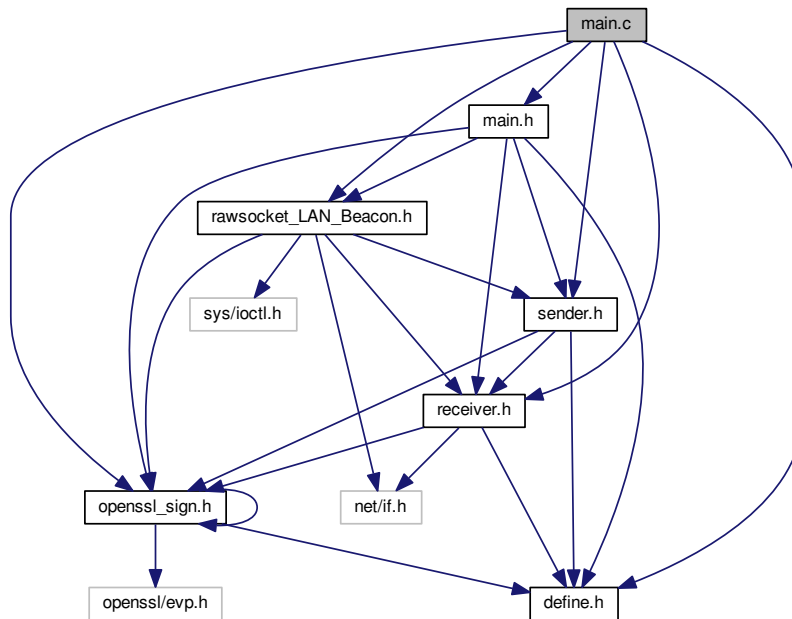
Referenced by `evaluatelanbeacon()`, and `lanbeacon_creator()`.



## 4.2 main.c File Reference

```
#include "openssl_sign.h"
#include "sender.h"
#include "rawsocket_LAN_Beacon.h"
#include "receiver.h"
#include "define.h"
#include "main.h"
```

Include dependency graph for main.c:



### Functions

- int [main](#) (int argc, char \*\*argv)  
*Separates receiver from sender mode and some setup.*
- void [printHelp](#) ()  
*Help function, executed if unknown parameters have been received or user specifically asks for help.*

### 4.2.1 Function Documentation

#### 4.2.1.1 main()

```
int main (
    int argc,
    char ** argv )
```

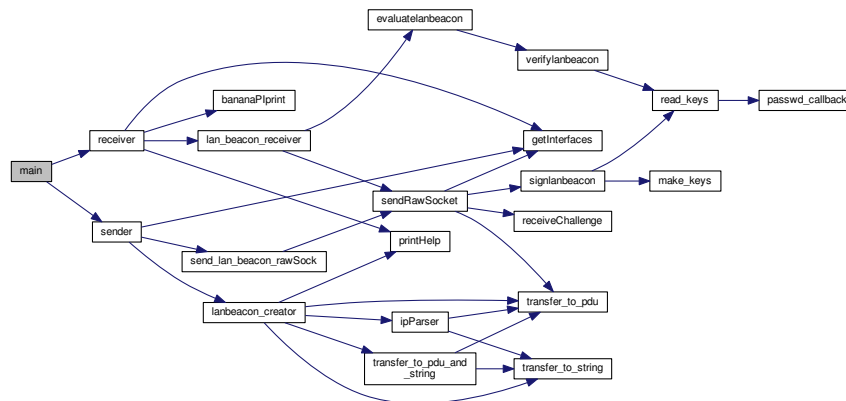
Separates receiver from sender mode and some setup.

**Returns**

Success or failure code.

References receiver(), and sender().

Here is the call graph for this function:

**4.2.1.2 printHelp()**

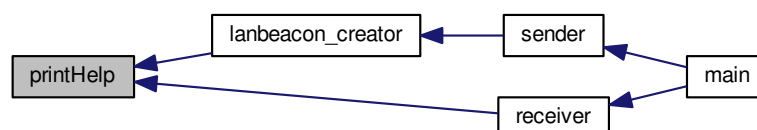
```
void printHelp ( )
```

Help function, executed if unknown parameters have been received or user specifically asks for help.

References \_.

Referenced by lanbeacon\_creator(), and receiver().

Here is the caller graph for this function:

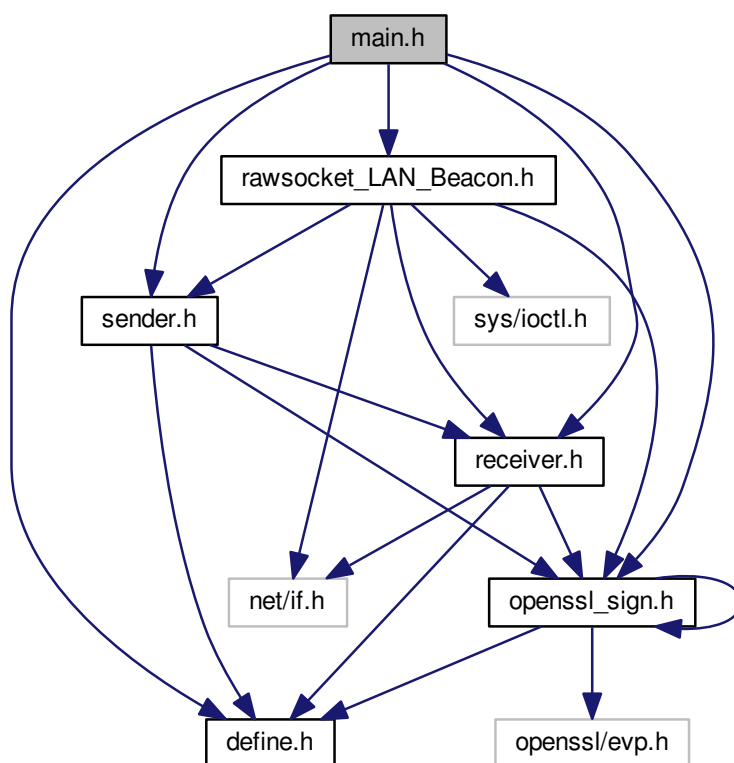


## 4.3 main.h File Reference

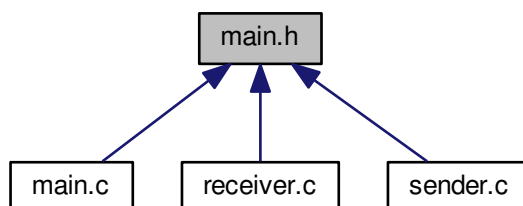
Main function and help function.

```
#include "openssl_sign.h"  
#include "sender.h"  
#include "rawsocket_LAN_Beacon.h"  
#include "receiver.h"  
#include "define.h"
```

Include dependency graph for main.h:



This graph shows which files directly or indirectly include this file:



## Functions

- int `main` (int argc, char \*\*argv)  
*Separates receiver from sender mode and some setup.*
- void `printHelp` ()  
*Help function, executed if unknown parameters have been received or user specifically asks for help.*

### 4.3.1 Detailed Description

Main function and help function.

#### Author

Dominik Bitzer

#### Date

2017

### 4.3.2 Function Documentation

#### 4.3.2.1 main()

```
int main (
    int argc,
    char ** argv )
```

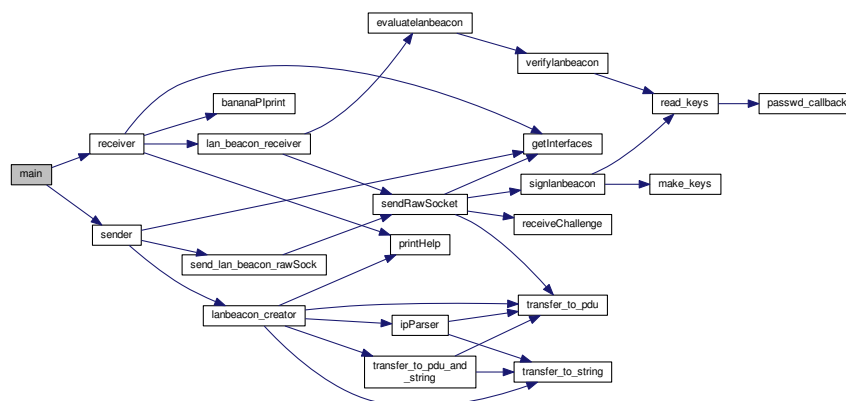
Separates receiver from sender mode and some setup.

#### Returns

Success or failure code.

References receiver(), and sender().

Here is the call graph for this function:



## 4.3.2.2 printHelp()

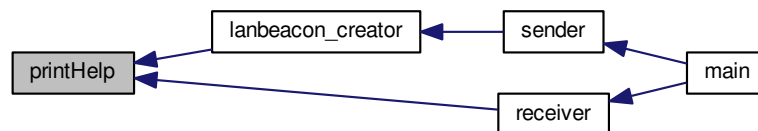
```
void printHelp ( )
```

Help function, executed if unknown parameters have been received or user specifically asks for help.

References [\\_](#).

Referenced by `lanbeacon_creator()`, and `receiver()`.

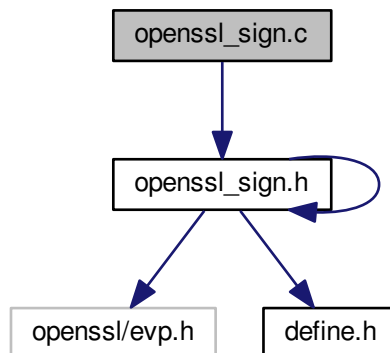
Here is the caller graph for this function:



## 4.4 openssl\_sign.c File Reference

```
#include "openssl_sign.h"
```

Include dependency graph for `openssl_sign.c`:



## Macros

- `#define KEY_READ_PROBLEM 0b1`
- `#define VERFIY_PROBLEM 0b01`
- `#define NO_PRIVATE_KEY 0b001`
- `#define NO_PUBLIC_KEY 0b0001`
- `#define PROBLEM_IN_SIGN_CALL 0b00001`
- `#define PROBLEM_IN_VERIFY_CALL 0b000001`
- `#define SIG_LEN 256`

## Functions

- int [verifylanbeacon](#) (const unsigned char \*msg, size\_t mlen, struct [open\\_ssl\\_keys](#) \*lanbeacon\_keys)  
*Verify the signature for LAN-Beacon PDUs.*
- int [signlanbeacon](#) (unsigned char \*\*sig, size\_t \*slen, const unsigned char \*msg, size\_t mlen, struct [open\\_ssl\\_keys](#) \*lanbeacon\_keys)  
*Create signature for LAN-Beacon PDU.*
- void [print\\_it](#) (const char \*label, const unsigned char \*buff, size\_t len)  
*Prints a buffer to stdout. Label is optional.*
- int [read\\_keys](#) (EVP\_PKEY \*\*skey, EVP\_PKEY \*\*vkey, struct [open\\_ssl\\_keys](#) \*lanbeacon\_keys)  
*Read stored pem files into memory.*
- int [make\\_keys](#) (EVP\_PKEY \*\*skey, EVP\_PKEY \*\*vkey, struct [open\\_ssl\\_keys](#) \*lanbeacon\_keys)  
*Generate and save keys to specified paths.*
- int [passwd\\_callback](#) (char \*pcszBuff, int size, int rwflag, void \*pPass)  
*Password callback function to retrieve password from configuration.*

## Variables

- const char [hn](#) [] = "SHA256"

### 4.4.1 Macro Definition Documentation

#### 4.4.1.1 KEY\_READ\_PROBLEM

```
#define KEY_READ_PROBLEM 0b1
```

#### 4.4.1.2 NO\_PRIVATE\_KEY

```
#define NO_PRIVATE_KEY 0b001
```

Referenced by [read\\_keys\(\)](#), and [signlanbeacon\(\)](#).

#### 4.4.1.3 NO\_PUBLIC\_KEY

```
#define NO_PUBLIC_KEY 0b0001
```

Referenced by [read\\_keys\(\)](#), and [verifylanbeacon\(\)](#).

#### 4.4.1.4 PROBLEM\_IN\_SIGN\_CALL

```
#define PROBLEM_IN_SIGN_CALL 0b00001
```

Referenced by `signlanbeacon()`.

#### 4.4.1.5 PROBLEM\_IN\_VERIFY\_CALL

```
#define PROBLEM_IN_VERIFY_CALL 0b000001
```

Referenced by `verifylanbeacon()`.

#### 4.4.1.6 SIG\_LEN

```
#define SIG_LEN 256
```

#### 4.4.1.7 VERFIY\_PROBLEM

```
#define VERFIY_PROBLEM 0b01
```

### 4.4.2 Function Documentation

#### 4.4.2.1 make\_keys()

```
int make_keys (  
    EVP_PKEY ** skey,  
    EVP_PKEY ** vkey,  
    struct open\_ssl\_keys * lanbeacon_keys )
```

Generate and save keys to specified paths.

##### Parameters

<i>skey</i>	pointer, where private key should be stored
<i>vkey</i>	pointer, where public key should be stored
<i>lanbeacon_keys</i>	configuration for file paths and password

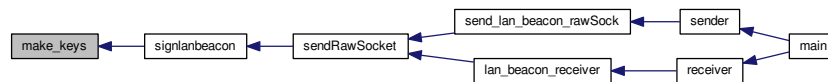
**Returns**

Returns 0 for success, non-0 otherwise

References `_`, `open_ssl_keys::path_To_Signing_Key`, `open_ssl_keys::path_To_Verifying_Key`, and `open_ssl_keys::pcszPassphrase`.

Referenced by `signlanbeacon()`.

Here is the caller graph for this function:

**4.4.2.2 passwd\_callback()**

```

int passwd_callback (
    char * pcszBuff,
    int size,
    int rwflag,
    void * pPass )
  
```

Password callback function to retrieve password from configuration.

**Parameters**

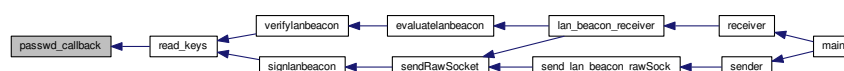
<i>pcszBuff</i>	Buffer for password
<i>size</i>	Size of buffer
<i>rwflag</i>	Read/write flag
<i>pPass</i>	Password

**Returns**

Success or error codes

Referenced by `read_keys()`.

Here is the caller graph for this function:





#### 4.4.2.3 print\_it()

```
void print_it (
    const char * label,
    const unsigned char * buff,
    size_t len )
```

Prints a buffer to stdout. Label is optional.

##### Parameters

<i>label</i>	Descriptor that will be put with contents
<i>buff</i>	Buffer for printing
<i>len</i>	Length of the buffer

#### 4.4.2.4 read\_keys()

```
int read_keys (
    EVP_PKEY ** skey,
    EVP_PKEY ** vkey,
    struct open_ssl_keys * lanbeacon_keys )
```

Read stored pem files into memory.

##### Parameters

<i>skey</i>	Memory address for the private key
<i>vkey</i>	Memory address for the public key
<i>lanbeacon_keys</i>	Configurations of the keys

##### Returns

Success or error codes

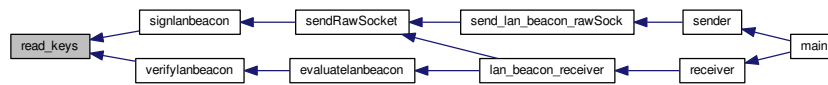
References `_`, `NO_PRIVATE_KEY`, `NO_PUBLIC_KEY`, `passwd_callback()`, `open_ssl_keys::path_To_Signing_Key`, `open_ssl_keys::path_To_Verifying_Key`, `open_ssl_keys::pcszPassphrase`, `RECEIVER_MODE`, `SENDER_MODE`, and `open_ssl_keys::sender_or_receiver_mode`.

Referenced by `signlanbeacon()`, and `verifylanbeacon()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.4.2.5 signlanbeacon()

```

int signlanbeacon (
    unsigned char ** sig,
    size_t * slen,
    const unsigned char * msg,
    size_t qqlen,
    struct open_ssl_keys * lanbeacon_keys )

```

Create signature for LAN-Beacon PDU.

##### Parameters

<i>sig</i>	Memory pointer for signature
<i>slen</i>	Length of the created signature
<i>msg</i>	LAN-Beacon PDU that should be signed
<i>qqlen</i>	Size of the passed LAN-Beacon PDU
<i>lanbeacon_keys</i>	Configurations of the keys

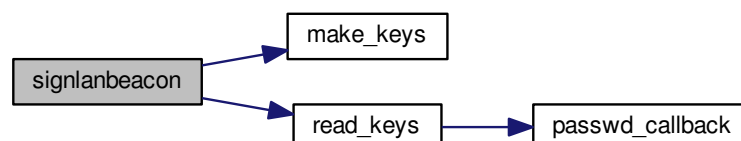
##### Returns

Success or error codes

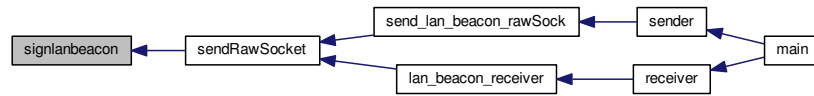
References `_`, `open_ssl_keys::generate_keys`, `hn`, `make_keys()`, `NO_PRIVATE_KEY`, `open_ssl_keys::path_To_↔` `Verifying_Key`, `PROBLEM_IN_SIGN_CALL`, and `read_keys()`.

Referenced by `sendRawSocket()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.4.2.6 verifylanbeacon()

```

int verifylanbeacon (
    const unsigned char * msg,
    size_t mlen,
    struct open_ssl_keys * lanbeacon_keys )

```

Verify the signature for LAN-Beacon PDUs.

##### Parameters

<i>msg</i>	Message, that should be verified
<i>mlen</i>	Length of the message, that should be verified
<i>lanbeacon_keys</i>	Configurations of the keys

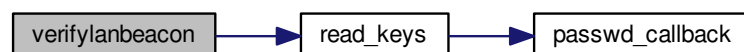
##### Returns

Success or error codes

References `_`, `hn`, `NO_PUBLIC_KEY`, `open_ssl_keys::path_To_Verifying_Key`, `PROBLEM_IN_VERIFY_CALL`, and `read_keys()`.

Referenced by `evaluatelanbeacon()`.

Here is the call graph for this function:



Here is the caller graph for this function:



### 4.4.3 Variable Documentation

#### 4.4.3.1 hn

```
const char hn[] = "SHA256"
```

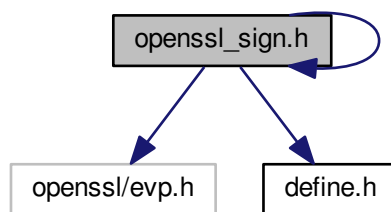
Referenced by `signlanbeacon()`, and `verifylanbeacon()`.

## 4.5 openssl\_sign.h File Reference

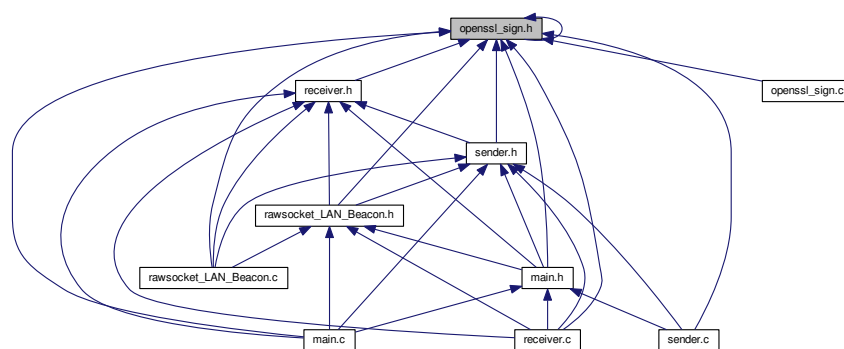
signing, verifying and key I/O

```
#include <openssl/evp.h>
#include "openssl_sign.h"
#include "define.h"
```

Include dependency graph for `openssl_sign.h`:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct [open\\_ssl\\_keys](#)  
*Key locations, password and further configurations.*

## Macros

- #define [SENDER\\_MODE](#) 0
- #define [RECEIVER\\_MODE](#) 1

## Functions

- int [make\\_keys](#) (EVP\_PKEY \*\*skey, EVP\_PKEY \*\*vkey, struct [open\\_ssl\\_keys](#) \*lanbeacon\_keys)  
*Generate and save keys to specified paths.*
- void [print\\_it](#) (const char \*label, const unsigned char \*buff, size\_t len)  
*Prints a buffer to stdout. Label is optional.*
- int [passwd\\_callback](#) (char \*pcszBuff, int size, int rwflag, void \*pPass)  
*Password callback function to retrieve password from configuration.*
- int [signlanbeacon](#) (unsigned char \*\*sig, size\_t \*slen, const unsigned char \*msg, size\_t qlen, struct [open\\_ssl\\_keys](#) \*lanbeacon\_keys)  
*Create signature for LAN-Beacon PDU.*
- int [read\\_keys](#) (EVP\_PKEY \*\*skey, EVP\_PKEY \*\*vkey, struct [open\\_ssl\\_keys](#) \*lanbeacon\_keys)  
*Read stored pem files into memory.*
- int [verifylanbeacon](#) (const unsigned char \*msg, size\_t mlen, struct [open\\_ssl\\_keys](#) \*lanbeacon\_keys)  
*Verify the signature for LAN-Beacon PDUs.*

### 4.5.1 Detailed Description

signing, verifying and key I/O

#### Author

Dominik Bitzer

#### Date

2017

### 4.5.2 Macro Definition Documentation

#### 4.5.2.1 RECEIVER\_MODE

```
#define RECEIVER_MODE 1
```

Referenced by [read\\_keys\(\)](#), and [receiver\(\)](#).

#### 4.5.2.2 SENDER\_MODE

```
#define SENDER_MODE 0
```

Referenced by `read_keys()`, and `sender()`.

### 4.5.3 Function Documentation

#### 4.5.3.1 make\_keys()

```
int make_keys (
    EVP_PKEY ** skey,
    EVP_PKEY ** vkey,
    struct open_ssl_keys * lanbeacon_keys )
```

Generate and save keys to specified paths.

##### Parameters

<i>skey</i>	pointer, where private key should be stored
<i>vkey</i>	pointer, where public key should be stored
<i>lanbeacon_keys</i>	configuration for file paths and password

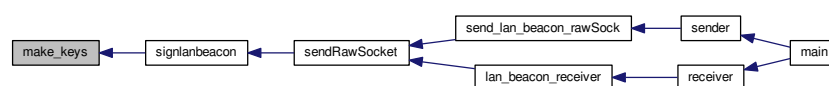
##### Returns

Returns 0 for success, non-0 otherwise

References `_`, `open_ssl_keys::path_To_Signing_Key`, `open_ssl_keys::path_To_Verifying_Key`, and `open_ssl_keys::pcszPassphrase`.

Referenced by `signlanbeacon()`.

Here is the caller graph for this function:



#### 4.5.3.2 passwd\_callback()

```
int passwd_callback (
    char * pcszBuff,
    int size,
    int rwflag,
    void * pPass )
```

Password callback function to retrieve password from configuration.

**Parameters**

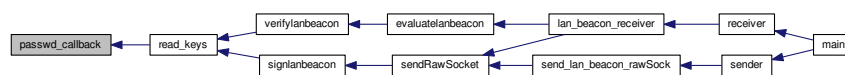
<i>pcszBuff</i>	Buffer for password
<i>size</i>	Size of buffer
<i>rwflag</i>	Read/write flag
<i>pPass</i>	Password

**Returns**

Success or error codes

Referenced by `read_keys()`.

Here is the caller graph for this function:

**4.5.3.3 print\_it()**

```

void print_it (
    const char * label,
    const unsigned char * buff,
    size_t len )

```

Prints a buffer to stdout. Label is optional.

**Parameters**

<i>label</i>	Descriptor that will be put with contents
<i>buff</i>	Buffer for printing
<i>len</i>	Length of the buffer

**4.5.3.4 read\_keys()**

```

int read_keys (
    EVP_PKEY ** skey,
    EVP_PKEY ** vkey,
    struct open_ssl_keys * lanbeacon_keys )

```

Read stored pem files into memory.



## Parameters

<i>skey</i>	Memory address for the private key
<i>vkey</i>	Memory address for the public key
<i>lanbeacon_keys</i>	Configurations of the keys

## Returns

Success or error codes

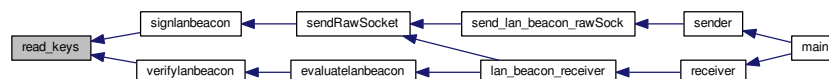
References `_`, `NO_PRIVATE_KEY`, `NO_PUBLIC_KEY`, `passwd_callback()`, `open_ssl_keys::path_To_Signing_Key`, `open_ssl_keys::path_To_Verifying_Key`, `open_ssl_keys::pcszPassphrase`, `RECEIVER_MODE`, `SENDER_MODE`, and `open_ssl_keys::sender_or_receiver_mode`.

Referenced by `signlanbeacon()`, and `verifylanbeacon()`.

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.5.3.5 signlanbeacon()

```

int signlanbeacon (
    unsigned char ** sig,
    size_t * slen,
    const unsigned char * msg,
    size_t qlen,
    struct open_ssl_keys * lanbeacon_keys )
  
```

Create signature for LAN-Beacon PDU.

## Parameters

<i>sig</i>	Memory pointer for signature
<i>slen</i>	Length of the created signature
<i>msg</i>	LAN-Beacon PDU that should be signed
<i>qqlen</i>	Size of the passed LAN-Beacon PDU
<i>lanbeacon_keys</i>	Configurations of the keys

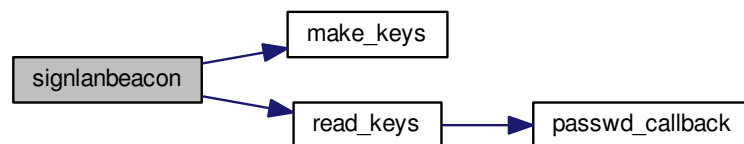
## Returns

Success or error codes

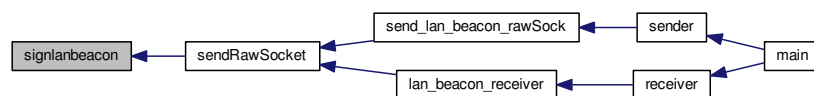
References `_`, `open_ssl_keys::generate_keys`, `hn`, `make_keys()`, `NO_PRIVATE_KEY`, `open_ssl_keys::path_To_`, `Verifying_Key`, `PROBLEM_IN_SIGN_CALL`, and `read_keys()`.

Referenced by `sendRawSocket()`.

Here is the call graph for this function:



Here is the caller graph for this function:

4.5.3.6 `verifylanbeacon()`

```

int verifylanbeacon (
    const unsigned char * msg,
    size_t mlen,
    struct open_ssl_keys * lanbeacon_keys )
  
```

Verify the signature for LAN-Beacon PDUs.

## Parameters

<i>msg</i>	Message, that should be verified
<i>mlen</i>	Length of the message, that should be verified
<i>lanbeacon_keys</i>	Configurations of the keys

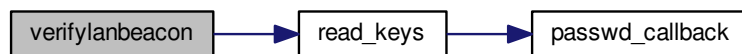
## Returns

Success or error codes

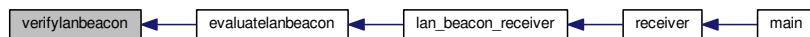
References `_`, `hn`, `NO_PUBLIC_KEY`, `open_ssl_keys::path_To_Verifying_Key`, `PROBLEM_IN_VERIFY_CALL`, and `read_keys()`.

Referenced by `evaluatelanbeacon()`.

Here is the call graph for this function:



Here is the caller graph for this function:

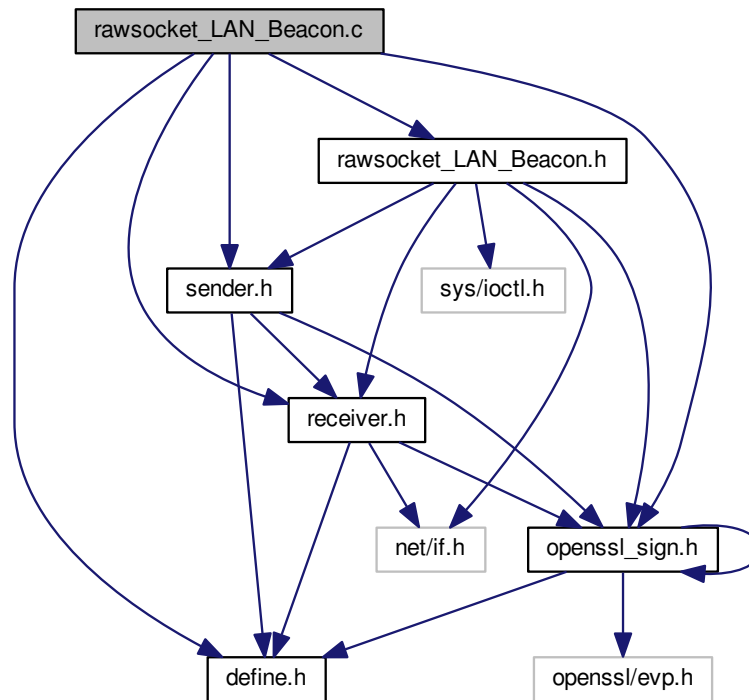


## 4.6 rawsocket\_LAN\_Beacon.c File Reference

```
#include "rawsocket_LAN_Beacon.h"
#include "receiver.h"
#include "openssl_sign.h"
#include "define.h"
```

```
#include "sender.h"
```

Include dependency graph for rawsocket\_LAN\_Beacon.c:



## Macros

- `#define GNU_SOURCE`

## Functions

- void [sendRawSocket](#) (unsigned char \*destination\_mac, void \*payload, int payloadLen, unsigned short etherType, struct [sender\\_information](#) \*my\_sender\_information)  
*Generic function to send on raw sockets, both handles sending of LAN-Beacon and challenges.*
- int [send\\_lan\\_beacon\\_rawSock](#) (struct [sender\\_information](#) \*my\_sender\_information)  
*Shortcut that can be used for sending LAN-Beacons, provides some configuration already.*
- void [lan\\_beacon\\_receiver](#) (struct [receiver\\_information](#) \*my\_receiver\_information)  
*Receives LAN-Beacons and adds them to the structure of received beacons.*
- unsigned long [receiveChallenge](#) (struct [interfaces](#) \*my\_challenge\_receiver\_interfaces, char \*challenge\_↔ dest\_mac, struct [sender\\_information](#) \*my\_sender\_information)  
*Listen for any and eventually receive challenges, that clients have sent in response to LAN-Beacon frames.*
- void [getInterfaces](#) (struct [interfaces](#) \*my\_interfaces\_struct, char \*interface\_to\_send\_on)  
*Get raw sockets for interfaces.*

### 4.6.1 Macro Definition Documentation

#### 4.6.1.1 \_GNU\_SOURCE

```
#define _GNU_SOURCE
```

### 4.6.2 Function Documentation

#### 4.6.2.1 getInterfaces()

```
void getInterfaces (
    struct interfaces * my_interfaces_struct,
    char * interface_to_send_on )
```

Get raw sockets for interfaces.

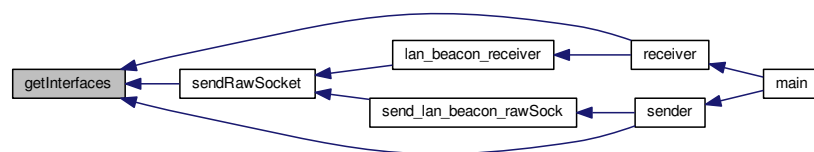
##### Parameters

<i>my_interfaces_struct</i>	Struct that contains interfaces information and configuration
<i>interface_to_send_on</i>	Specified interfaces for sending

References `_`, `interfaces::etherType`, `interfaces::if_idx`, `interfaces::if_mac`, `interfaces::maxSockFd`, `interfaces::numInterfaces`, `REC_SOCKET`, `SEND_SOCKET`, `interfaces::sendOrReceive`, `interfaces::sockfd`, and `interfaces::sockopt`.

Referenced by `receiver()`, `sender()`, and `sendRawSocket()`.

Here is the caller graph for this function:



#### 4.6.2.2 lan\_beacon\_receiver()

```
void lan_beacon_receiver (
    struct receiver_information * my_receiver_information )
```

Receives LAN-Beacons and adds them to the structure of received beacons.

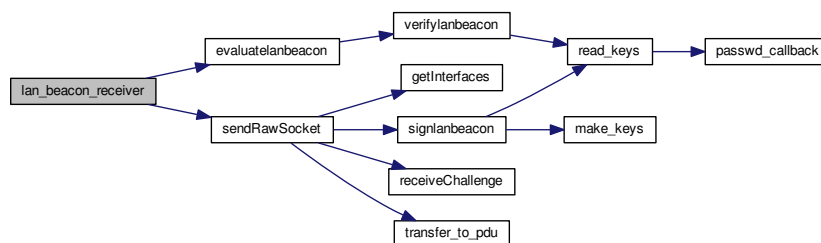
## Parameters

<code>my_receiver_information</code>	Receiver configuration and structs for storing the received beacons
--------------------------------------	---

References `_`, `receiver_information::authenticated_mode`, `received_lan_beacon_frame::challenge`, `CHALLENGE_ETHTYPE`, `received_lan_beacon_frame::current_destination_mac`, `evaluatelanbeacon()`, `LAN_BEACON_BUF_SIZ`, `LAN_BEACON_DEST_MAC`, `received_lan_beacon_frame::lan_beacon_ReceivedPayload`, `receiver_information::lanbeacon_keys`, `interfaces::maxSockFd`, `receiver_information::my_receiver_interfaces`, `receiver_information::number_of_currently_received_frames`, `interfaces::numInterfaces`, `received_lan_beacon_frame::parsedBeaconContents`, `received_lan_beacon_frame::payloadSize`, `receiver_information::pointers_to_received_frames`, `sendRawSocket()`, `SHOW_FRAMES_X_TIMES`, `interfaces::sockfd`, `received_lan_beacon_frame::successfullyAuthenticated`, and `received_lan_beacon_frame::times_left_to_display`.

Referenced by `receiver()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.6.2.3 receiveChallenge()

```

unsigned long receiveChallenge (
    struct interfaces * my_challenge_interfaces,
    char * challenge_dest_mac,
    struct sender_information * my_sender_information )

```

Listen for any and eventually receive challenges, that clients have sent in response to LAN-Beacon frames.

## Parameters

<i>my_challenge_interfaces</i>	Struct with the sockets for receiving challenges
<i>challenge_dest_mac</i>	States the destination to send the authenticated LAN-Beacon
<i>my_sender_information</i>	Sender configurations

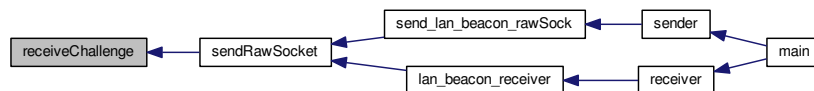
## Returns

Returns the value of the received challenge

References `_`, `interfaces::maxSockFd`, `interfaces::numInterfaces`, `sender_information::send_frequency`, and `interfaces::sockfd`.

Referenced by `sendRawSocket()`.

Here is the caller graph for this function:



## 4.6.2.4 send\_lan\_beacon\_rawSock()

```
int send_lan_beacon_rawSock (
    struct sender_information * my_sender_information )
```

Shortcut that can be used for sending LAN-Beacons, provides some configuration already.

## Parameters

<i>my_sender_information</i>	Struct that contains everything needed for sending
------------------------------	--

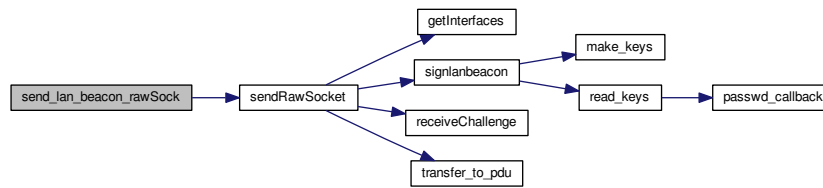
## Returns

Success or failure code, which is passed on from called function

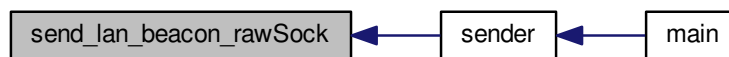
References `LAN_BEACON_DEST_MAC`, `LAN_BEACON_ETHER_TYPE`, `sender_information::lan_beacon_pdu_len`, `sender_information::lanBeacon_PDU`, and `sendRawSocket()`.

Referenced by `sender()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.6.2.5 sendRawSocket()

```

void sendRawSocket (
    unsigned char * destination_mac,
    void * payload,
    int payloadLen,
    unsigned short etherType,
    struct sender_information * my_sender_information )
  
```

Generic function to send on raw sockets, both handles sending of LAN-Beacon and challenges.

##### Parameters

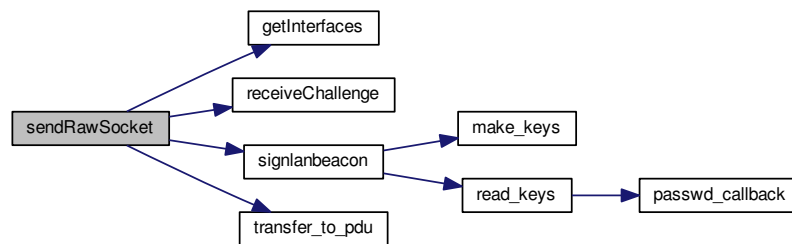
<i>destination_mac</i>	Destination MAC address
<i>payload</i>	Payload that should be sent
<i>payloadLen</i>	Length of payload
<i>etherType</i>	EtherType of payload
<i>my_sender_information</i>	Sender configurations

References `_`, `CHALLENGE_ETHTYPE`, `interfaces::etherType`, `getInterfaces()`, `interfaces::if_idx`, `interfaces::if_mac`, `sender_information::interface_to_send_on`, `LAN_BEACON_BUF_SIZ`, `LAN_BEACON_ETHER_TYPE`, `sender_information::lanbeacon_keys`, `sender_information::my_challenge_receiver_interfaces`, `interfaces::numInterfaces`, `receiveChallenge()`, `sender_information::send_frequency`, `SEND_SOCKET`, `signalanbeacon()`, `interfaces::sockfd`, `SUBTYPE_SIGNATURE`, and `transfer_to_pdu()`.

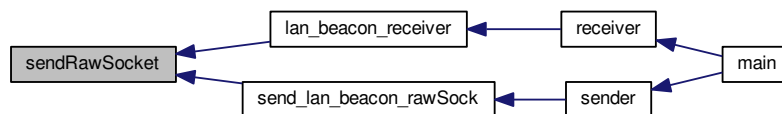
Referenced by `lan_beacon_receiver()`, and `send_lan_beacon_rawSock()`.



Here is the call graph for this function:



Here is the caller graph for this function:



## 4.7 rawsocket\_LAN\_Beacon.h File Reference

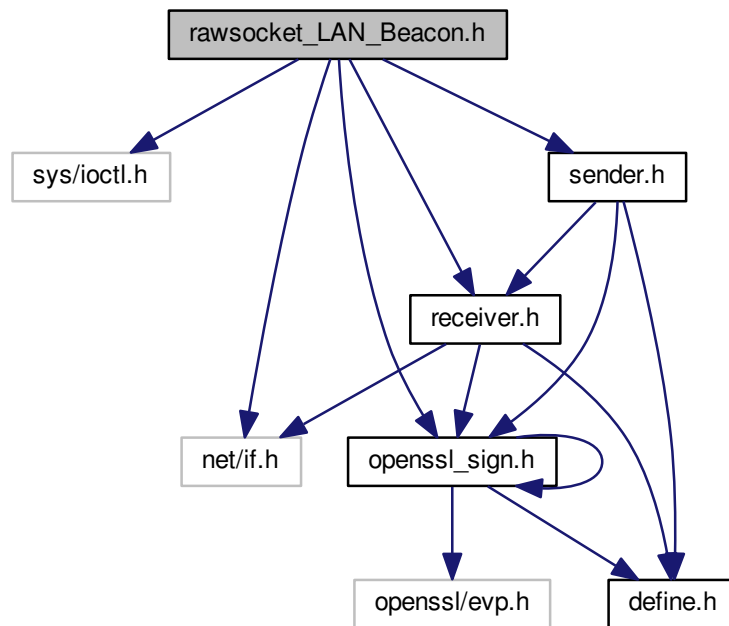
raw-socket sending and receiving

```

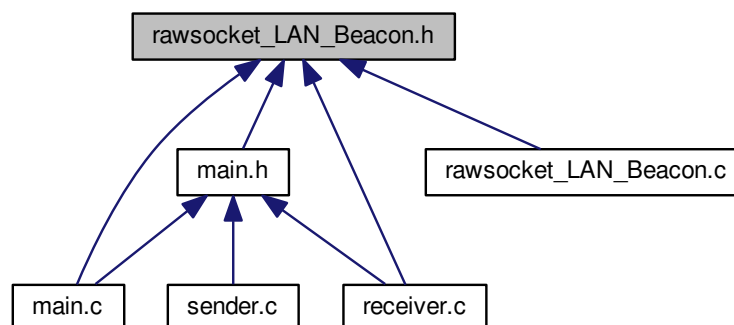
#include <sys/ioctl.h>
#include <net/if.h>
#include "openssl_sign.h"
#include "receiver.h"
#include "sender.h"

```

Include dependency graph for rawsocket\_LAN\_Beacon.h:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define SEND_SOCKET 0`
- `#define REC_SOCKET 1`

## Functions

- void [lan\\_beacon\\_receiver](#) (struct [receiver\\_information](#) \*my\_receiver\_information)  
*Receives LAN-Beacons and adds them to the structure of received beacons.*
- int [send\\_lan\\_beacon\\_rawSock](#) (struct [sender\\_information](#) \*my\_sender\_information)  
*Shortcut that can be used for sending LAN-Beacons, provides some configuration already.*
- unsigned long [receiveChallenge](#) (struct [interfaces](#) \*my\_challenge\_interfaces, char \*challenge\_dest\_mac, struct [sender\\_information](#) \*my\_sender\_information)  
*Listen for any and eventually receive challenges, that clients have sent in response to LAN-Beacon frames.*
- void [getInterfaces](#) (struct [interfaces](#) \*my\_interfaces\_struct, char \*interface\_to\_send\_on)  
*Get raw sockets for interfaces.*
- void [sendRawSocket](#) (unsigned char \*destination\_mac, void \*payload, int payloadLen, unsigned short etherType, struct [sender\\_information](#) \*my\_sender\_information)  
*Generic function to send on raw sockets, both handles sending of LAN-Beacon and challenges.*

### 4.7.1 Detailed Description

raw-socket sending and receiving

#### Author

Dominik Bitzer

#### Date

2017

### 4.7.2 Macro Definition Documentation

#### 4.7.2.1 REC\_SOCKET

```
#define REC_SOCKET 1
```

Referenced by [getInterfaces\(\)](#), [receiver\(\)](#), and [sender\(\)](#).

#### 4.7.2.2 SEND\_SOCKET

```
#define SEND_SOCKET 0
```

Referenced by [getInterfaces\(\)](#), and [sendRawSocket\(\)](#).

### 4.7.3 Function Documentation

#### 4.7.3.1 getInterfaces()

```
void getInterfaces (
    struct interfaces * my_interfaces_struct,
    char * interface_to_send_on )
```

Get raw sockets for interfaces.

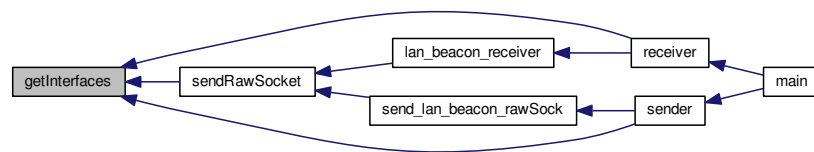
## Parameters

<i>my_interfaces_struct</i>	Struct that contains interfaces information and configuration
<i>interface_to_send_on</i>	Specified interfaces for sending

References `_`, `interfaces::etherType`, `interfaces::if_idx`, `interfaces::if_mac`, `interfaces::maxSockFd`, `interfaces::numInterfaces`, `REC_SOCKET`, `SEND_SOCKET`, `interfaces::sendOrReceive`, `interfaces::sockfd`, and `interfaces::sockopt`.

Referenced by `receiver()`, `sender()`, and `sendRawSocket()`.

Here is the caller graph for this function:



## 4.7.3.2 lan\_beacon\_receiver()

```
void lan_beacon_receiver (
    struct receiver_information * my_receiver_information )
```

Receives LAN-Beacons and adds them to the structure of received beacons.

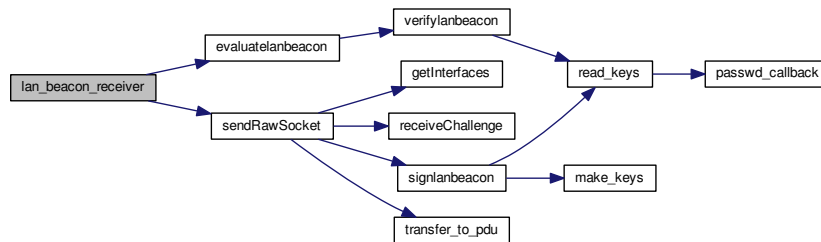
## Parameters

<i>my_receiver_information</i>	Receiver configuration and structs for storing the received beacons
--------------------------------	---

References `_`, `receiver_information::authenticated_mode`, `received_lan_beacon_frame::challenge`, `CHALLENGE_ETHTYPE`, `received_lan_beacon_frame::current_destination_mac`, `evaluateLanBeacon()`, `LAN_BEACON_BUFFER_SIZE`, `LAN_BEACON_DEST_MAC`, `received_lan_beacon_frame::lan_beacon_ReceivedPayload`, `receiver_information::lanbeacon_keys`, `interfaces::maxSockFd`, `receiver_information::my_receiver_interfaces`, `receiver_information::number_of_currently_received_frames`, `interfaces::numInterfaces`, `received_lan_beacon_frame::parsedBeaconContents`, `received_lan_beacon_frame::payloadSize`, `receiver_information::pointers_to_received_frames`, `sendRawSocket()`, `SHOW_FRAMES_X_TIMES`, `interfaces::sockfd`, `received_lan_beacon_frame::successfullyAuthenticated`, and `received_lan_beacon_frame::times_left_to_display`.

Referenced by `receiver()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.7.3.3 receiveChallenge()

```

unsigned long receiveChallenge (
    struct interfaces * my_challenge_interfaces,
    char * challenge_dest_mac,
    struct sender_information * my_sender_information )

```

Listen for any and eventually receive challenges, that clients have sent in response to LAN-Beacon frames.

##### Parameters

<i>my_challenge_interfaces</i>	Struct with the sockets for receiving challenges
<i>challenge_dest_mac</i>	States the destination to send the authenticated LAN-Beacon
<i>my_sender_information</i>	Sender configurations

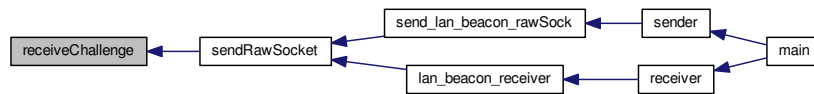
##### Returns

Returns the value of the received challenge

References `_`, `interfaces::maxSockFd`, `interfaces::numInterfaces`, `sender_information::send_frequency`, and `interfaces::sockfd`.

Referenced by `sendRawSocket()`.

Here is the caller graph for this function:



#### 4.7.3.4 send\_lan\_beacon\_rawSock()

```
int send_lan_beacon_rawSock (
    struct sender_information * my_sender_information )
```

Shortcut that can be used for sending LAN-Beacons, provides some configuration already.

##### Parameters

<i>my_sender_information</i>	Struct that contains everything needed for sending
------------------------------	--

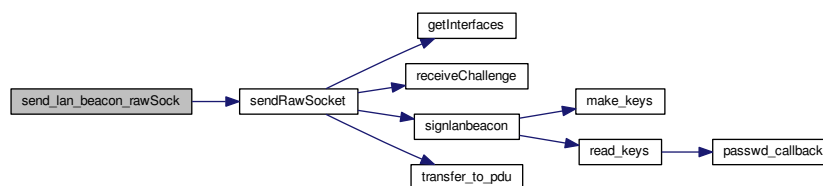
##### Returns

Success or failure code, which is passed on from called function

References LAN\_BEACON\_DEST\_MAC, LAN\_BEACON\_ETHER\_TYPE, sender\_information::lan\_beacon\_pdu↔\_len, sender\_information::lanBeacon\_PDU, and sendRawSocket().

Referenced by sender().

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.7.3.5 sendRawSocket()

```
void sendRawSocket (
    unsigned char * destination_mac,
    void * payload,
    int payloadLen,
    unsigned short etherType,
    struct sender_information * my_sender_information )
```

Generic function to send on raw sockets, both handles sending of LAN-Beacon and challenges.

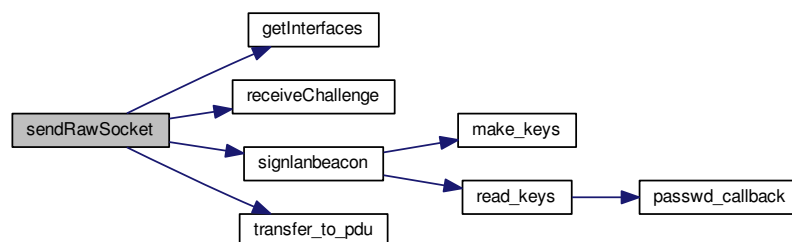
## Parameters

<i>destination_mac</i>	Destination MAC address
<i>payload</i>	Payload that should be sent
<i>payloadLen</i>	Length of payload
<i>etherType</i>	EtherType of payload
<i>my_sender_information</i>	Sender configurations

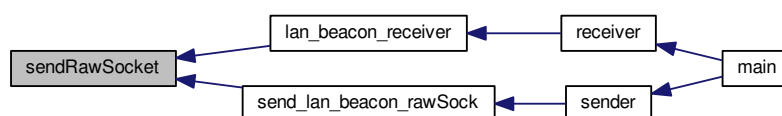
References `_`, `CHALLENGE_ETHERTYPE`, `interfaces::etherType`, `getInterfaces()`, `interfaces::if_idx`, `interfaces::if_mac`, `sender_information::interface_to_send_on`, `LAN_BEACON_BUF_SIZ`, `LAN_BEACON_ETHER_TYPE`, `sender_information::lanbeacon_keys`, `sender_information::my_challenge_receiver_interfaces`, `interfaces::numInterfaces`, `receiveChallenge()`, `sender_information::send_frequency`, `SEND_SOCKET`, `signlanbeacon()`, `interfaces::sockfd`, `SUBTYPE_SIGNATURE`, and `transfer_to_pdu()`.

Referenced by `lan_beacon_receiver()`, and `send_lan_beacon_rawSock()`.

Here is the call graph for this function:



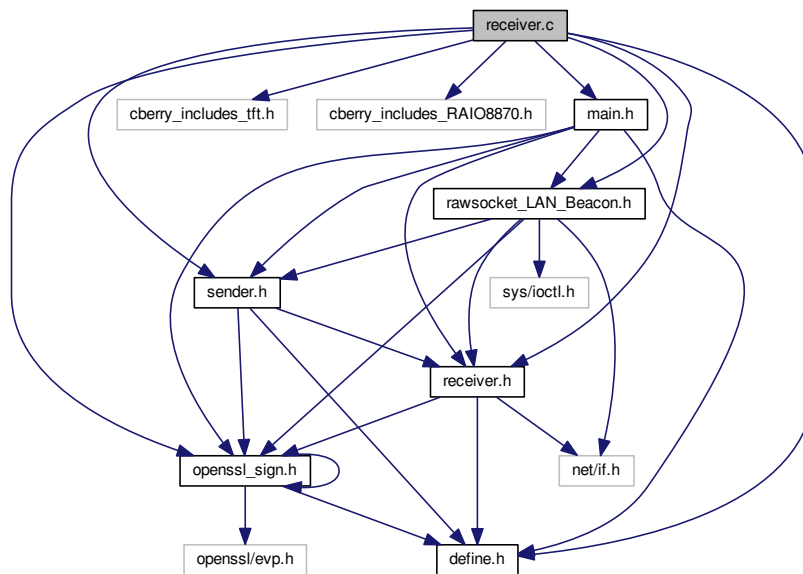
Here is the caller graph for this function:



## 4.8 receiver.c File Reference

```
#include "openssl_sign.h"
#include "cberry_includes_tft.h"
#include "cberry_includes_RAIO8870.h"
#include "rawsocket_LAN_Beacon.h"
#include "receiver.h"
#include "sender.h"
#include "define.h"
#include "main.h"
```

Include dependency graph for receiver.c:



### Macros

- `#define TLV_CUSTOM_COPY(descriptor, TLV_parsed_content, makro_currentTLVcontentSize)`
- `#define TLV_STRING_COPY(descriptor)`

### Functions

- `int receiver(int argc, char **argv)`  
*This function has the main receiver logic and starts all other receiver functions.*
- `char ** evaluatelanbeacon(struct received_lan_beacon_frame *my_received_lan_beacon_frame, struct open_ssl_keys *lanbeacon_keys)`  
*This function takes raw received LAN-Beacon frames and creates strings from them, that can be used for printing or further processing.*
- `void bananaP!print(struct receiver_information *my_receiver_information)`  
*This function prints the received content on the standard output and, if compiler flags are set, also on a C-Berry display.*



## 4.8.1 Macro Definition Documentation

### 4.8.1.1 TLV\_CUSTOM\_COPY

```
#define TLV_CUSTOM_COPY(  
    descriptor,  
    TLV_parsed_content,  
    makro_currentTLVcontentSize )
```

**Value:**

```
snprintf(parsedTLVs [numberParsedTLVs++], PARSED\_TLVs\_MAX\_LENGTH, "%-10s%.s", \  
    descriptor, (int) makro_currentTLVcontentSize, TLV_parsed_content); \  
    break;
```

Referenced by `evaluatelanbeacon()`.

### 4.8.1.2 TLV\_STRING\_COPY

```
#define TLV_STRING_COPY(  
    descriptor )
```

**Value:**

```
TLV\_CUSTOM\_COPY(descriptor, \  
    (char*) &my_received_lan_beacon_frame->lan_beacon_ReceivedPayload[currentPayloadByte+6], \  
    currentTLVsize-4);
```

Referenced by `evaluatelanbeacon()`.

## 4.8.2 Function Documentation

### 4.8.2.1 bananaPIprint()

```
void bananaPIprint (  
    struct receiver\_information * my_receiver_information )
```

This function prints the received content on the standard output and, if compiler flags are set, also on a C-Berry display.

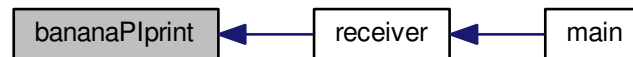
## Parameters

<i>my_receiver_information</i>	receiver information struct, that contains display settings and contents that should be printed
--------------------------------	---

References `_`, `receiver_information::current_lan_beacon_pdu_for_printing`, `DESCRIPTOR_WIDTH`, `receiver_information::number_of_currently_received_frames`, `PARSED_TLV_MAX_NUMBER`, `received_lan_beacon_frame::parsedBeaconContents`, `receiver_information::pointers_to_received_frames`, `receiver_information::scroll_speed`, and `received_lan_beacon_frame::times_left_to_display`.

Referenced by `receiver()`.

Here is the caller graph for this function:

4.8.2.2 `evaluatelanbeacon()`

```

char** evaluatelanbeacon (
    struct received_lan_beacon_frame * my_received_lan_beacon_frame,
    struct open_ssl_keys * lanbeacon_keys )

```

This function takes raw received LAN-Beacon frames and creates strings from them, that can be used for printing or further processing.

## Parameters

<i>my_received_lan_beacon_frame</i>	Pointer to one single received LAN-Beacon frame, that should be evaluated
<i>lanbeacon_keys</i>	Pointer to struct for keys, needed in order to verify authentication information

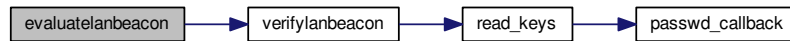
## Returns

Returns parsed content as an array of TLV-descriptor and TLV-content pairs

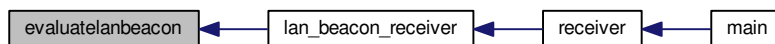
References `_`, `received_lan_beacon_frame::challenge`, `DESCRIPTOR_COMBINED_STRING`, `DESCRIPTOR_CUSTOM`, `DESCRIPTOR_DHCP`, `DESCRIPTOR_EMAIL`, `DESCRIPTOR_IPV4`, `DESCRIPTOR_IPV6`, `DESCRIPTOR_NAME`, `DESCRIPTOR_ROUTER`, `DESCRIPTOR_SIGNATURE`, `DESCRIPTOR_VLAN_ID`, `received_lan_beacon_frame::lan_beacon_ReceivedPayload`, `PARSED_TLV_MAX_LENGTH`, `PARSED_TLV_MAX_NUMBER`, `received_lan_beacon_frame::payloadSize`, `SUBTYPE_COMBINED_STRING`, `SUBTYPE_CUSTOM`, `SUBTYPE_DHCP`, `SUBTYPE_EMAIL`, `SUBTYPE_IPV4`, `SUBTYPE_IPV6`, `SUBTYPE_NAME`, `SUBTYPE_ROUTER`, `SUBTYPE_SIGNATURE`, `SUBTYPE_VLAN_ID`, `received_lan_beacon_frame::successfullyAuthenticated`, `TLV_CUSTOM_COPY`, `TLV_STRING_COPY`, and `verifylanbeacon()`.

Referenced by `lan_beacon_receiver()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.8.2.3 receiver()

```

int receiver (
    int argc,
    char ** argv )
  
```

This function has the main receiver logic and starts all other receiver functions.

##### Parameters

<i>argc</i>	Number of command line arguments.
<i>argv</i>	Contents of command line arguments.

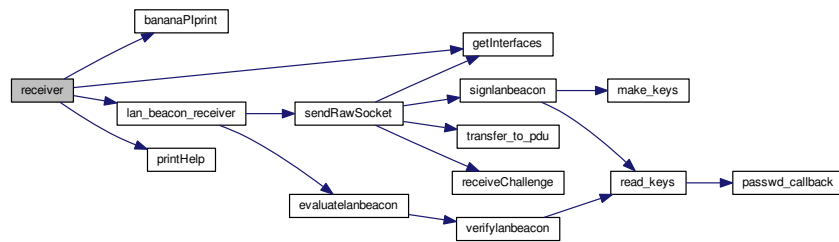
##### Returns

Error or failure code

References `_`, `receiver_information::authenticated_mode`, `bananaPlprint()`, `receiver_information::current_lan_beacon_pdu_for_printing`, `DEFAULT_SCROLLSPEED`, `getInterfaces()`, `KEY_PATHLENGTH_MAX`, `LAN_BEACON_ETHER_TYPE`, `lan_beacon_receiver()`, `receiver_information::lanbeacon_keys`, `receiver_information::my_receiver_interfaces`, `receiver_information::number_of_currently_received_frames`, `PARSED_TLV_MAX_NUMBER`, `received_lan_beacon_frame::parsedBeaconContents`, `open_ssl_keys::path_To_Verifying_Key`, `receiver_information::pointers_to_received_frames`, `printHelp()`, `PUBLIC_KEY_STANDARD_PATH`, `REC_SOCKET`, `RECEIVER_MODE`, `receiver_information::scroll_speed`, and `open_ssl_keys::sender_or_receiver_mode`.

Referenced by `main()`.

Here is the call graph for this function:



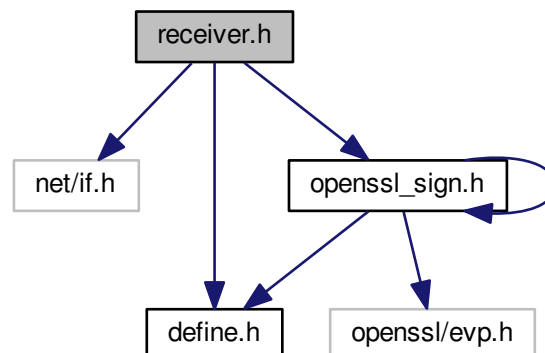
Here is the caller graph for this function:



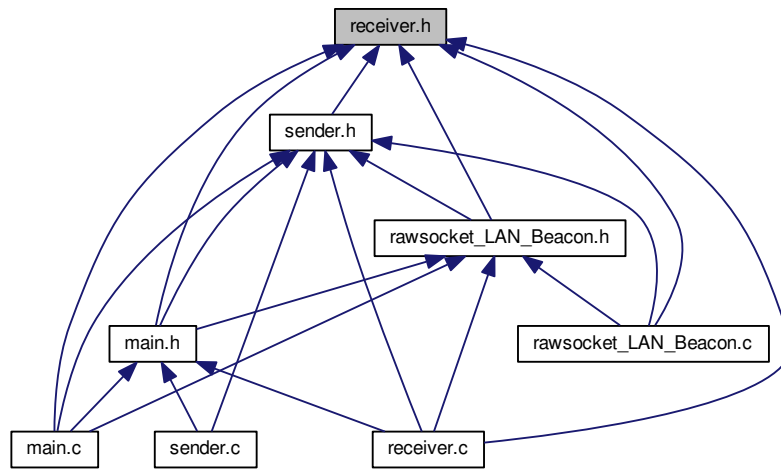
## 4.9 receiver.h File Reference

Receiver-specific functions and structures.

```
#include <net/if.h>
#include "define.h"
#include "openssl_sign.h"
Include dependency graph for receiver.h:
```



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct [received\\_lan\\_beacon\\_frame](#)  
*Contains all the information related to one received frame.*
- struct [interfaces](#)  
*Contains all variables, that are needed to access sockets on interfaces.*
- struct [receiver\\_information](#)  
*Receiver configurations.*

## Functions

- int [receiver](#) (int argc, char \*\*argv)  
*This function has the main receiver logic and starts all other receiver functions.*
- char \*\* [evaluatelanbeacon](#) (struct [received\\_lan\\_beacon\\_frame](#) \*my\_received\_lan\_beacon\_frame, struct [open\\_ssl\\_keys](#) \*lanbeacon\_keys)  
*This function takes raw received LAN-Beacon frames and creates strings from them, that can be used for printing or further processing.*
- void [bananaPlprint](#) (struct [receiver\\_information](#) \*my\_receiver\_information)  
*This function prints the received content on the standard output and, if compiler flags are set, also on a C-Berry display.*

### 4.9.1 Detailed Description

Receiver-specific functions and structures.

#### Author

Dominik Bitzer

#### Date

2017

## 4.9.2 Function Documentation

### 4.9.2.1 bananaPIprint()

```
void bananaPIprint (
    struct receiver_information * my_receiver_information )
```

This function prints the received content on the standard output and, if compiler flags are set, also on a C-Berry display.

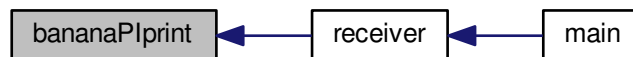
#### Parameters

<i>my_receiver_information</i>	receiver information struct, that contains display settings and contents that should be printed
--------------------------------	---

References `_`, `receiver_information::current_lan_beacon_pdu_for_printing`, `DESCRIPTOR_WIDTH`, `receiver_information::number_of_currently_received_frames`, `PARSED_TLV_MAX_NUMBER`, `received_lan_beacon_frame::parsedBeaconContents`, `receiver_information::pointers_to_received_frames`, `receiver_information::scroll_speed`, and `received_lan_beacon_frame::times_left_to_display`.

Referenced by `receiver()`.

Here is the caller graph for this function:



### 4.9.2.2 evaluatelanbeacon()

```
char** evaluatelanbeacon (
    struct received_lan_beacon_frame * my_received_lan_beacon_frame,
    struct open_ssl_keys * lanbeacon_keys )
```

This function takes raw received LAN-Beacon frames and creates strings from them, that can be used for printing or further processing.

#### Parameters

<i>my_received_lan_beacon_frame</i>	Pointer to one single received LAN-Beacon frame, that should be evaluated
<i>lanbeacon_keys</i>	Pointer to struct for keys, needed in order to verify authentication information

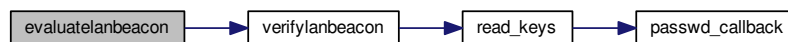
**Returns**

Returns parsed content as an array of TLV-descriptor and TLV-content pairs

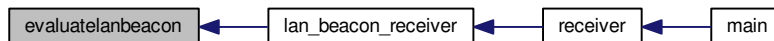
References `_`, `received_lan_beacon_frame::challenge`, `DESCRIPTOR_COMBINED_STRING`, `DESCRIPTOR_CUSTOM`, `DESCRIPTOR_DHCP`, `DESCRIPTOR_EMAIL`, `DESCRIPTOR_IPV4`, `DESCRIPTOR_IPV6`, `DESCRIPTOR_NAME`, `DESCRIPTOR_ROUTER`, `DESCRIPTOR_SIGNATURE`, `DESCRIPTOR_VLAN_ID`, `received_lan_beacon_frame::lan_beacon_ReceivedPayload`, `PARSED_TLVs_MAX_LENGTH`, `PARSED_TLVs_MAX_NUMBER`, `received_lan_beacon_frame::payloadSize`, `SUBTYPE_COMBINED_STRING`, `SUBTYPE_CUSTOM`, `SUBTYPE_DHCP`, `SUBTYPE_EMAIL`, `SUBTYPE_IPV4`, `SUBTYPE_IPV6`, `SUBTYPE_NAME`, `SUBTYPE_ROUTER`, `SUBTYPE_SIGNATURE`, `SUBTYPE_VLAN_ID`, `received_lan_beacon_frame::successfullyAuthenticated`, `TLV_CUSTOM_COPY`, `TLV_STRING_COPY`, and `verifylanbeacon()`.

Referenced by `lan_beacon_receiver()`.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.9.2.3 receiver()**

```

int receiver (
    int argc,
    char ** argv )
  
```

This function has the main receiver logic and starts all other receiver functions.

**Parameters**

<i>argc</i>	Number of command line arguments.
<i>argv</i>	Contents of command line arguments.

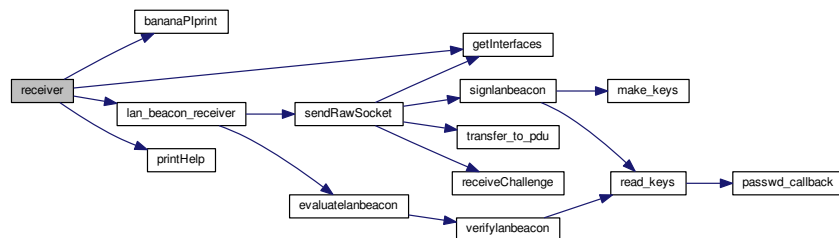
**Returns**

Error or failure code

References `_`, `receiver_information::authenticated_mode`, `bananaPIprint()`, `receiver_information::current_lan_beacon_pdu_for_printing`, `DEFAULT_SCROLLSPEED`, `getInterfaces()`, `KEY_PATHLENGTH_MAX`, `LAN_BEACON_ETHER_TYPE`, `lan_beacon_receiver()`, `receiver_information::lanbeacon_keys`, `receiver_information::my_receiver_interfaces`, `receiver_information::number_of_currently_received_frames`, `PARSED_TLV_MAX_NUMBER`, `received_lan_beacon_frame::parsedBeaconContents`, `open_ssl_keys::path_To_Verifying_Key`, `receiver_information::pointers_to_received_frames`, `printHelp()`, `PUBLIC_KEY_STANDARD_PATH`, `REC_SOCKET`, `RECEIVER_MODE`, `receiver_information::scroll_speed`, and `open_ssl_keys::sender_or_receiver_mode`.

Referenced by `main()`.

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.10 sender.c File Reference

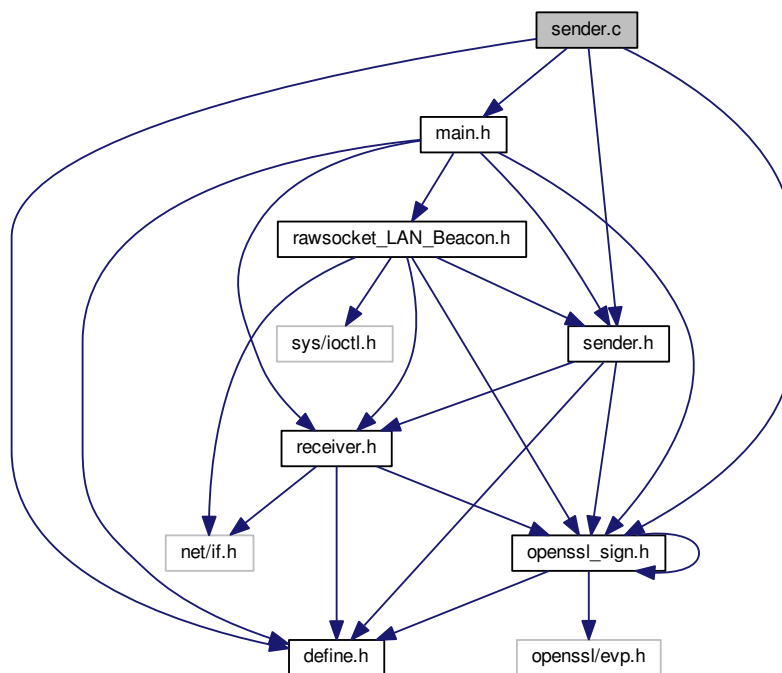
```

#include "openssl_sign.h"
#include "define.h"
#include "sender.h"
#include "main.h"

```



Include dependency graph for sender.c:



## Functions

- int [sender](#) (int argc, char \*\*argv)  
*This function has the main receiver logic and starts all other receiver functions.*
- char \* [lanbeacon\\_creator](#) (int \*argc, char \*\*argv, struct [sender\\_information](#) \*my\_sender\_information)  
*Creates a LAN-Beacon PDU from the command line arguments.*
- void [transfer\\_to\\_pdu\\_and\\_string](#) (unsigned char subtype, char \*TLVdescription, char \*\*combinedString, char \*source, char \*combinedBeacon, int \*currentByte)  
*Shortcut function for cases in which only a string is transferred, no binary format TLVs.*
- void [transfer\\_to\\_pdu](#) (unsigned char subtype, void \*source, char \*combinedBeacon, int \*currentByte, unsigned short int currentTLVlength)  
*Transferring the content of the field to the combined lanbeacon in binary format.*
- void [transfer\\_to\\_string](#) (char \*TLVdescription, char \*\*combinedString, char \*TLVcontents)  
*Transfer human-readable information to combined string.*
- void [ipParser](#) (int ip\_V4or6, char \*optarg, char \*\*combinedString, char \*combinedBeacon, int \*currentByte)  
*Parse IPv4 or IPv6 subnets to binary format.*

### 4.10.1 Function Documentation

#### 4.10.1.1 ipParser()

```
void ipParser (
    int ip_V4or6,
    char * optarg,
    char ** combinedString,
    char * combinedBeacon,
    int * currentByte )
```

Parse IPv4 or IPv6 subnets to binary format.

Using regex to get IP-addresses from string input, then convert them to binary representation for transport

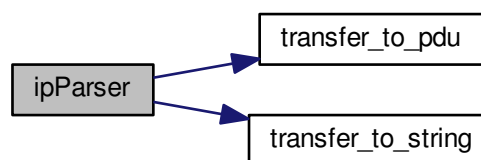
##### Parameters

<i>ip_V4or6</i>	Switch between IPv4 and IPv6 mode
<i>optarg</i>	String, which should be parsed
<i>combinedString</i>	Pointer to the string, that contains text representation of all contents
<i>combinedBeacon</i>	PDU of beacon, that TLVs should be added to
<i>currentByte</i>	current position in the Beacon-PDU

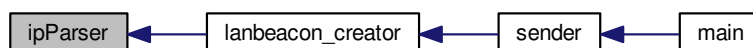
References `_`, `DESCRIPTOR_IPV4`, `DESCRIPTOR_IPV6`, `SUBTYPE_IPV4`, `SUBTYPE_IPV6`, `transfer_to_pdu()`, and `transfer_to_string()`.

Referenced by `lanbeacon_creator()`.

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.10.1.2 lanbeacon\_creator()

```
char* lanbeacon_creator (
    int * argc,
    char ** argv,
    struct sender_information * my_sender_information )
```

Creates a LAN-Beacon PDU from the command line arguments.

Howto for adding new fields:

1. Add defines for desired new field in [define.h](#)
2. Add desired options in [lanbeacon\\_creator\(\)](#)

## Parameters

<i>argc</i>	Number of command line arguments.
<i>argv</i>	Contents of command line arguments.

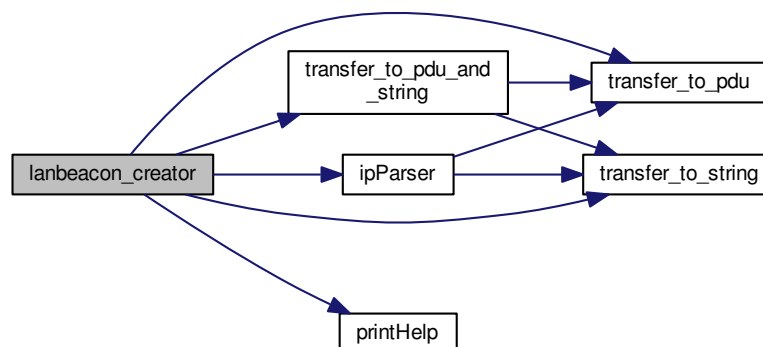
## Returns

Returns an array, that contains the payload of a lanBeacon\_PDU

References `_`, `DESCRIPTOR_CUSTOM`, `DESCRIPTOR_DHCP`, `DESCRIPTOR_EMAIL`, `DESCRIPTOR_NAME`, `DESCRIPTOR_ROUTER`, `DESCRIPTOR_VLAN_ID`, `open_ssl_keys::generate_keys`, `sender_information::interface_to_send_on`, `ipParser()`, `KEY_PATHLENGTH_MAX`, `sender_information::lan_beacon_pdu_len`, `sender_information::lanbeacon_keys`, `open_ssl_keys::path_To_Signing_Key`, `open_ssl_keys::path_To_Verifying_Key`, `open_ssl_keys::pcszPassphrase`, `printHelp()`, `sender_information::send_frequency`, `SUBTYPE_COMBINED_STRING`, `SUBTYPE_CUSTOM`, `SUBTYPE_DHCP`, `SUBTYPE_EMAIL`, `SUBTYPE_NAME`, `SUBTYPE_ROUTER`, `SUBTYPE_VLAN_ID`, `transfer_to_pdu()`, `transfer_to_pdu_and_string()`, and `transfer_to_string()`.

Referenced by `sender()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.10.1.3 sender()

```

int sender (
    int argc,
    char ** argv )
  
```

This function has the main receiver logic and starts all other receiver functions.

##### Parameters

<i>argc</i>	Number of command line arguments.
<i>argv</i>	Contents of command line arguments.

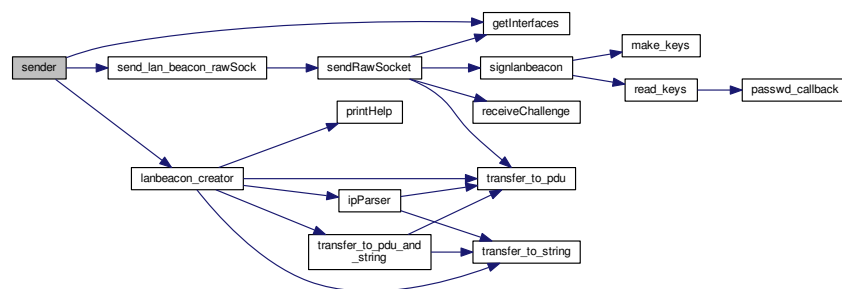
##### Returns

Error or failure code

References CHALLENGE\_ETHTYPE, getInterfaces(), sender\_information::interface\_to\_send\_on, LAN\_BEACON\_SEND\_FREQUENCY, lanbeacon\_creator(), sender\_information::lanBeacon\_PDU, sender\_information::my\_challenge\_receiver\_interfaces, PRIVATE\_KEY\_STANDARD\_PATH, PUBLIC\_KEY\_STANDARD\_PATH, RECEIVE\_SOCKET, send\_lan\_beacon\_rawSock(), and SENDER\_MODE.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.10.1.4 transfer\_to\_pdu()

```

void transfer_to_pdu (
    unsigned char subtype,
    void * source,
    char * combinedBeacon,
    int * currentByte,
    unsigned short int currentTLVlength )
  
```

Transferring the content of the field to the combined lanbeacon in binary format.

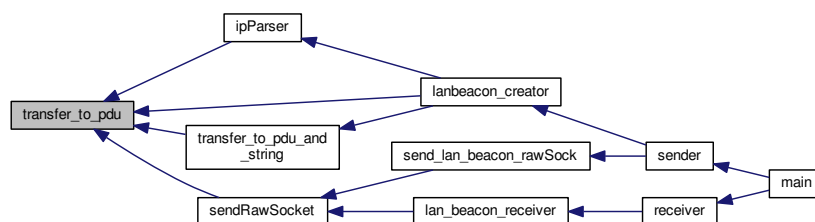
##### Parameters

<i>subtype</i>	Subtype of the TLV
<i>source</i>	String contents, that should be included to the PDU
<i>combinedBeacon</i>	PDU of beacon, that TLVs should be added to
<i>currentByte</i>	current position in the Beacon-PDU
<i>currentTLVlength</i>	Length of the passed TLV

##### References

Referenced by ipParser(), lanbeacon\_creator(), sendRawSocket(), and transfer\_to\_pdu\_and\_string().

Here is the caller graph for this function:



## 4.10.1.5 transfer\_to\_pdu\_and\_string()

```

void transfer_to_pdu_and_string (
    unsigned char subtype,
    char * TLVdescription,
    char ** combinedString,
    char * source,
    char * combinedBeacon,
    int * currentByte )

```

Shortcut function for cases in which only a string is transferred, no binary format TLVs.

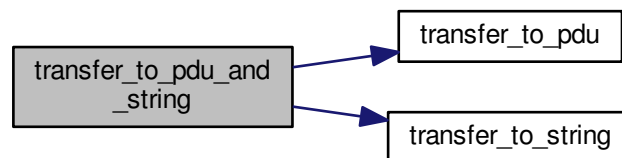
## Parameters

<i>subtype</i>	Subtype of the TLV
<i>TLVdescription</i>	Descriptor string of the TLV
<i>combinedString</i>	Pointer to the string, that contains text representation of all contents
<i>source</i>	String contents, that should be included to the PDU
<i>combinedBeacon</i>	PDU of beacon, that TLVs should be added to
<i>currentByte</i>	current position in the Beacon-PDU

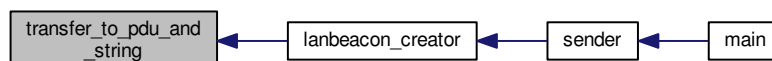
References transfer\_to\_pdu(), and transfer\_to\_string().

Referenced by lanbeacon\_creator().

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.10.1.6 transfer\_to\_string()

```
void transfer_to_string (
    char * TLVdescription,
    char ** combinedString,
    char * source )
```

Transfer human-readable information to combined string.

Transferring the content of the field to the combined string in human-readable format. If one combined string exceeds 507 byte limit of TLV it is put to the next combined string TLV

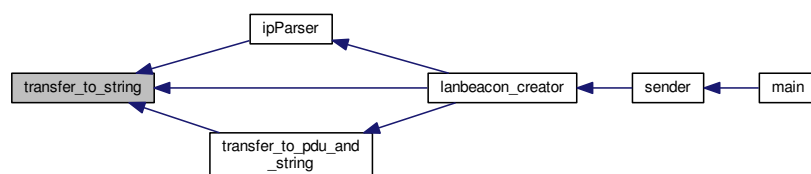
## Parameters

<i>TLVdescription</i>	Descriptor string of the TLV
<i>combinedString</i>	Pointer to the string, that contains text representation of all contents
<i>source</i>	String contents, that should be included to the PDU

References [\\_](#).

Referenced by `ipParser()`, `lanbeacon_creator()`, and `transfer_to_pdu_and_string()`.

Here is the caller graph for this function:

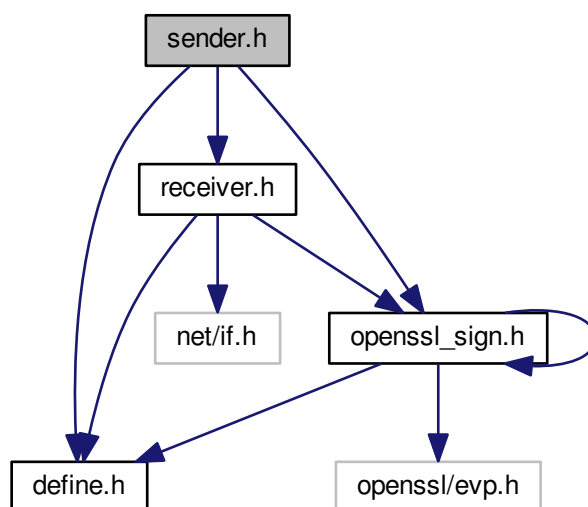


## 4.11 sender.h File Reference

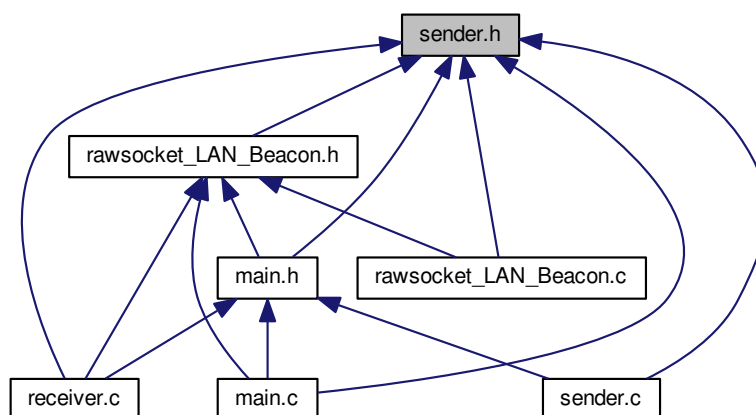
Sender-specific functions and structures.

```
#include "define.h"
#include "openssl_sign.h"
#include "receiver.h"
```

Include dependency graph for sender.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct [sender\\_information](#)

*Sender configurations.*



## Functions

- int [sender](#) (int argc, char \*\*argv)  
*This function has the main receiver logic and starts all other receiver functions.*
- char \* [lanbeacon\\_creator](#) (int \*argc, char \*\*argv, struct [sender\\_information](#) \*my\_sender\_information)  
*Creates a LAN-Beacon PDU from the command line arguments.*
- void [transfer\\_to\\_pdu\\_and\\_string](#) (unsigned char subtype, char \*TLVdescription, char \*\*combinedString, char \*source, char \*combinedBeacon, int \*currentByte)  
*Shortcut function for cases in which only a string is transferred, no binary format TLVs.*
- void [transfer\\_to\\_pdu](#) (unsigned char subtype, void \*source, char \*combinedBeacon, int \*currentByte, unsigned short int currentTLVlength)  
*Transferring the content of the field to the combined lanbeacon in binary format.*
- void [transfer\\_to\\_string](#) (char \*TLVdescription, char \*\*combinedString, char \*source)  
*Transfer human-readable information to combined string.*
- void [ipParser](#) (int ip\_V4or6, char \*optarg, char \*\*combinedString, char \*combinedBeacon, int \*currentByte)  
*Parse IPv4 or IPv6 subnets to binary format.*

### 4.11.1 Detailed Description

Sender-specific functions and structures.

#### Author

Dominik Bitzer

#### Date

2017

### 4.11.2 Function Documentation

#### 4.11.2.1 ipParser()

```
void ipParser (
    int ip_V4or6,
    char * optarg,
    char ** combinedString,
    char * combinedBeacon,
    int * currentByte )
```

Parse IPv4 or IPv6 subnets to binary format.

Using regex to get IP-addresses from string input, then convert them to binary representation for transport

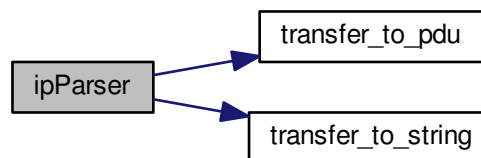
#### Parameters

<i>ip_V4or6</i>	Switch between IPv4 and IPv6 mode
<i>optarg</i>	String, which should be parsed
<i>combinedString</i>	Pointer to the string, that contains text representation of all contents
<i>combinedBeacon</i>	PDU of beacon, that TLVs should be added to
<i>currentByte</i>	current position in the Beacon-PDU

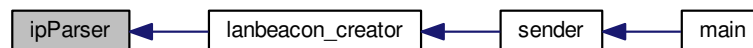
References `_`, `DESCRIPTOR_IPV4`, `DESCRIPTOR_IPV6`, `SUBTYPE_IPV4`, `SUBTYPE_IPV6`, `transfer_to_pdu()`, and `transfer_to_string()`.

Referenced by `lanbeacon_creator()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.11.2.2 lanbeacon\_creator()

```

char* lanbeacon_creator (
    int * argc,
    char ** argv,
    struct sender_information * my_sender_information )
  
```

Creates a LAN-Beacon PDU from the command line arguments.

Howto for adding new fields:

1. Add defines for desired new field in [define.h](#)
2. Add desired options in [lanbeacon\\_creator\(\)](#)

#### Parameters

<code>argc</code>	Number of command line arguments.
<code>argv</code>	Contents of command line arguments.

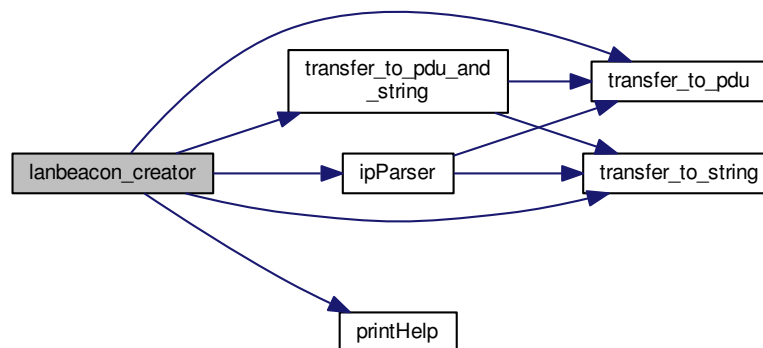
## Returns

Returns an array, that contains the payload of a lanBeacon\_PDU

References `_`, `DESCRIPTOR_CUSTOM`, `DESCRIPTOR_DHCP`, `DESCRIPTOR_EMAIL`, `DESCRIPTOR_NAME`, `DESCRIPTOR_ROUTER`, `DESCRIPTOR_VLAN_ID`, `open_ssl_keys::generate_keys`, `sender_information::interface_to_send_on`, `ipParser()`, `KEY_PATHLENGTH_MAX`, `sender_information::lan_beacon_pdu_len`, `sender_information::lanbeacon_keys`, `open_ssl_keys::path_To_Signing_Key`, `open_ssl_keys::path_To_Verifying_Key`, `open_ssl_keys::pcszPassphrase`, `printHelp()`, `sender_information::send_frequency`, `SUBTYPE_COMBINED_STRING`, `SUBTYPE_CUSTOM`, `SUBTYPE_DHCP`, `SUBTYPE_EMAIL`, `SUBTYPE_NAME`, `SUBTYPE_ROUTER`, `SUBTYPE_VLAN_ID`, `transfer_to_pdu()`, `transfer_to_pdu_and_string()`, and `transfer_to_string()`.

Referenced by `sender()`.

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.11.2.3 sender()

```

int sender (
    int argc,
    char ** argv )

```

This function has the main receiver logic and starts all other receiver functions.

## Parameters

<i>argc</i>	Number of command line arguments.
<i>argv</i>	Contents of command line arguments.

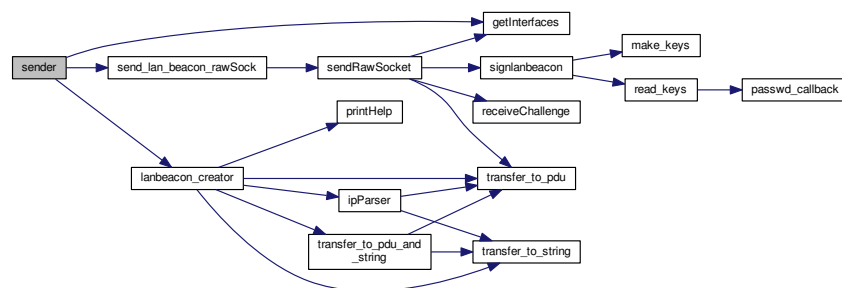
## Returns

Error or failure code

References CHALLENGE\_ETHTYPE, getInterfaces(), sender\_information::interface\_to\_send\_on, LAN\_BEACON\_SEND\_FREQUENCY, lanbeacon\_creator(), sender\_information::lanBeacon\_PDU, sender\_information::my\_challenge\_receiver\_interfaces, PRIVATE\_KEY\_STANDARD\_PATH, PUBLIC\_KEY\_STANDARD\_PATH, RECEIVE\_SOCKET, send\_lan\_beacon\_rawSock(), and SENDER\_MODE.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.11.2.4 transfer\_to\_pdu()

```

void transfer_to_pdu (
    unsigned char subtype,
    void * source,
    char * combinedBeacon,
    int * currentByte,
    unsigned short int currentTLVlength )
  
```

Transferring the content of the field to the combined lanbeacon in binary format.

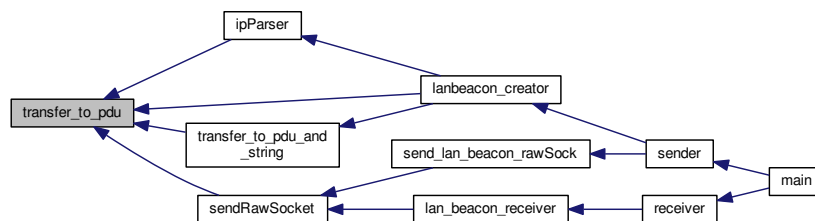
## Parameters

<i>subtype</i>	Subtype of the TLV
<i>source</i>	String contents, that should be included to the PDU
<i>combinedBeacon</i>	PDU of beacon, that TLVs should be added to
<i>currentByte</i>	current position in the Beacon-PDU
<i>currentTLVlength</i>	Length of the passed TLV

## References \_.

Referenced by ipParser(), lanbeacon\_creator(), sendRawSocket(), and transfer\_to\_pdu\_and\_string().

Here is the caller graph for this function:



## 4.11.2.5 transfer\_to\_pdu\_and\_string()

```

void transfer_to_pdu_and_string (
    unsigned char subtype,
    char * TLVdescription,
    char ** combinedString,
    char * source,
    char * combinedBeacon,
    int * currentByte )

```

Shortcut function for cases in which only a string is transferred, no binary format TLVs.

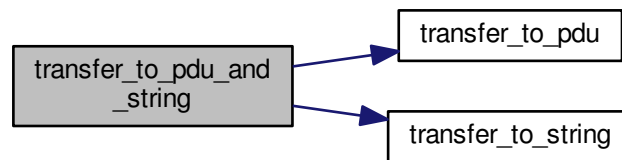
## Parameters

<i>subtype</i>	Subtype of the TLV
<i>TLVdescription</i>	Descriptor string of the TLV
<i>combinedString</i>	Pointer to the string, that contains text representation of all contents
<i>source</i>	String contents, that should be included to the PDU
<i>combinedBeacon</i>	PDU of beacon, that TLVs should be added to
<i>currentByte</i>	current position in the Beacon-PDU

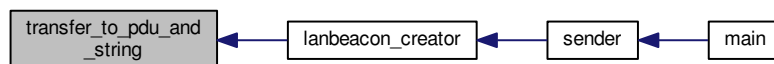
References `transfer_to_pdu()`, and `transfer_to_string()`.

Referenced by `lanbeacon_creator()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.11.2.6 `transfer_to_string()`

```

void transfer_to_string (
    char * TLVdescription,
    char ** combinedString,
    char * source )
  
```

Transfer human-readable information to combined string.

Transferring the content of the field to the combined string in human-readable format. If one combined string exceeds 507 byte limit of TLV it is put to the next combined string TLV

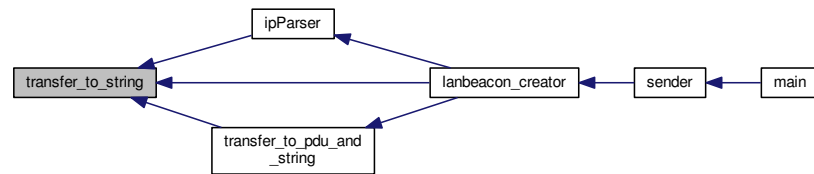
##### Parameters

<i>TLVdescription</i>	Descriptor string of the TLV
<i>combinedString</i>	Pointer to the string, that contains text representation of all contents
<i>source</i>	String contents, that should be included to the PDU

##### References

Referenced by `ipParser()`, `lanbeacon_creator()`, and `transfer_to_pdu_and_string()`.

Here is the caller graph for this function:







# Index

- - define.h, [19](#)
- \_GNU\_SOURCE
  - rawsocket\_LAN\_Beacon.c, [44](#)
- authenticated\_mode
  - receiver\_information, [13](#)
- bananaPlprint
  - receiver.c, [57](#)
  - receiver.h, [62](#)
- CHALLENGE\_ETHTYPE
  - define.h, [19](#)
- challenge
  - received\_lan\_beacon\_frame, [10](#)
- current\_destination\_mac
  - received\_lan\_beacon\_frame, [10](#)
- current\_lan\_beacon\_pdu\_for\_printing
  - receiver\_information, [13](#)
- DEFAULT\_SCROLLSPEED
  - define.h, [19](#)
- DESCRIPTOR\_COMBINED\_STRING
  - define.h, [19](#)
- DESCRIPTOR\_CUSTOM
  - define.h, [19](#)
- DESCRIPTOR\_DHCP
  - define.h, [19](#)
- DESCRIPTOR\_EMAIL
  - define.h, [20](#)
- DESCRIPTOR\_IPV4
  - define.h, [20](#)
- DESCRIPTOR\_IPV6
  - define.h, [20](#)
- DESCRIPTOR\_NAME
  - define.h, [20](#)
- DESCRIPTOR\_ROUTER
  - define.h, [20](#)
- DESCRIPTOR\_SIGNATURE
  - define.h, [20](#)
- DESCRIPTOR\_VLAN\_ID
  - define.h, [21](#)
- DESCRIPTOR\_WIDTH
  - define.h, [21](#)
- define.h, [17](#)
- \_, [19](#)
- CHALLENGE\_ETHTYPE, [19](#)
- DEFAULT\_SCROLLSPEED, [19](#)
- DESCRIPTOR\_COMBINED\_STRING, [19](#)
- DESCRIPTOR\_CUSTOM, [19](#)
- DESCRIPTOR\_DHCP, [19](#)
- DESCRIPTOR\_EMAIL, [20](#)
- DESCRIPTOR\_IPV4, [20](#)
- DESCRIPTOR\_IPV6, [20](#)
- DESCRIPTOR\_NAME, [20](#)
- DESCRIPTOR\_ROUTER, [20](#)
- DESCRIPTOR\_SIGNATURE, [20](#)
- DESCRIPTOR\_VLAN\_ID, [21](#)
- DESCRIPTOR\_WIDTH, [21](#)
- KEY\_PATHLENGTH\_MAX, [21](#)
- LAN\_BEACON\_BUF\_SIZ, [21](#)
- LAN\_BEACON\_DEST\_MAC, [21](#)
- LAN\_BEACON\_ETHER\_TYPE, [21](#)
- LAN\_BEACON\_SEND\_FREQUENCY, [22](#)
- PARSED\_TLVs\_MAX\_LENGTH, [22](#)
- PARSED\_TLVs\_MAX\_NUMBER, [22](#)
- PRIVATE\_KEY\_STANDARD\_PATH, [22](#)
- PUBLIC\_KEY\_STANDARD\_PATH, [22](#)
- SHOW\_FRAMES\_X\_TIMES, [22](#)
- SUBTYPE\_COMBINED\_STRING, [23](#)
- SUBTYPE\_CUSTOM, [23](#)
- SUBTYPE\_DHCP, [23](#)
- SUBTYPE\_EMAIL, [23](#)
- SUBTYPE\_IPV4, [23](#)
- SUBTYPE\_IPV6, [23](#)
- SUBTYPE\_NAME, [24](#)
- SUBTYPE\_ROUTER, [24](#)
- SUBTYPE\_SIGNATURE, [24](#)
- SUBTYPE\_VLAN\_ID, [24](#)
- etherType
  - interfaces, [6](#)
- evaluatelanbeacon
  - receiver.c, [58](#)
  - receiver.h, [62](#)
- generate\_keys
  - open\_ssl\_keys, [8](#)
- getInterfaces
  - rawsocket\_LAN\_Beacon.c, [45](#)
  - rawsocket\_LAN\_Beacon.h, [51](#)
- hn
  - openssl\_sign.c, [36](#)
- if\_idx
  - interfaces, [6](#)
- if\_mac
  - interfaces, [6](#)

- interface\_to\_send\_on
  - sender\_information, 15
- interfaces, 5
  - etherType, 6
  - if\_idx, 6
  - if\_mac, 6
  - maxSockFd, 6
  - numInterfaces, 6
  - sendOrReceive, 7
  - sockfd, 7
  - sockopt, 7
- ipParser
  - sender.c, 65
  - sender.h, 73
- KEY\_PATHLENGTH\_MAX
  - define.h, 21
- KEY\_READ\_PROBLEM
  - openssl\_sign.c, 30
- LAN\_BEACON\_BUF\_SIZ
  - define.h, 21
- LAN\_BEACON\_DEST\_MAC
  - define.h, 21
- LAN\_BEACON\_ETHER\_TYPE
  - define.h, 21
- LAN\_BEACON\_SEND\_FREQUENCY
  - define.h, 22
- lan\_beacon\_ReceivedPayload
  - received\_lan\_beacon\_frame, 11
- lan\_beacon\_pdu\_len
  - sender\_information, 16
- lan\_beacon\_receiver
  - rawsocket\_LAN\_Beacon.c, 45
  - rawsocket\_LAN\_Beacon.h, 52
- lanBeacon\_PDU
  - sender\_information, 16
- lanbeacon\_creator
  - sender.c, 66
  - sender.h, 74
- lanbeacon\_keys
  - receiver\_information, 13
  - sender\_information, 16
- main
  - main.c, 25
  - main.h, 28
- main.c, 25
  - main, 25
  - printHelp, 26
- main.h, 27
  - main, 28
  - printHelp, 28
- make\_keys
  - openssl\_sign.c, 31
  - openssl\_sign.h, 38
- maxSockFd
  - interfaces, 6
- my\_challenge\_receiver\_interfaces
  - sender\_information, 16
- my\_receiver\_interfaces
  - receiver\_information, 13
- NO\_PRIVATE\_KEY
  - openssl\_sign.c, 30
- NO\_PUBLIC\_KEY
  - openssl\_sign.c, 30
- numInterfaces
  - interfaces, 6
- number\_of\_currently\_received\_frames
  - receiver\_information, 13
- open\_ssl\_keys, 8
  - generate\_keys, 8
  - path\_To\_Signing\_Key, 8
  - path\_To\_Verifying\_Key, 9
  - pcszPassphrase, 9
  - sender\_or\_receiver\_mode, 9
- openssl\_sign.c, 29
  - hn, 36
  - KEY\_READ\_PROBLEM, 30
  - make\_keys, 31
  - NO\_PRIVATE\_KEY, 30
  - NO\_PUBLIC\_KEY, 30
  - PROBLEM\_IN\_SIGN\_CALL, 30
  - PROBLEM\_IN\_VERIFY\_CALL, 31
  - passwd\_callback, 32
  - print\_it, 32
  - read\_keys, 33
  - SIG\_LEN, 31
  - signlanbeacon, 34
  - VERFIY\_PROBLEM, 31
  - verifylanbeacon, 35
- openssl\_sign.h, 36
  - make\_keys, 38
  - passwd\_callback, 38
  - print\_it, 40
  - RECEIVER\_MODE, 37
  - read\_keys, 40
  - SENDER\_MODE, 37
  - signlanbeacon, 41
  - verifylanbeacon, 42
- PARSED\_TLVs\_MAX\_LENGTH
  - define.h, 22
- PARSED\_TLVs\_MAX\_NUMBER
  - define.h, 22
- PRIVATE\_KEY\_STANDARD\_PATH
  - define.h, 22
- PROBLEM\_IN\_SIGN\_CALL
  - openssl\_sign.c, 30
- PROBLEM\_IN\_VERIFY\_CALL
  - openssl\_sign.c, 31
- PUBLIC\_KEY\_STANDARD\_PATH
  - define.h, 22
- parsedBeaconContents
  - received\_lan\_beacon\_frame, 11
- passwd\_callback

- openssl\_sign.c, 32
  - openssl\_sign.h, 38
- path\_To\_Signing\_Key
  - open\_ssl\_keys, 8
- path\_To\_Verifying\_Key
  - open\_ssl\_keys, 9
- payloadSize
  - received\_lan\_beacon\_frame, 11
- pcszPassphrase
  - open\_ssl\_keys, 9
- pointers\_to\_received\_frames
  - receiver\_information, 14
- print\_it
  - openssl\_sign.c, 32
  - openssl\_sign.h, 40
- printHelp
  - main.c, 26
  - main.h, 28
- REC\_SOCKET
  - rawsocket\_LAN\_Beacon.h, 51
- RECEIVER\_MODE
  - openssl\_sign.h, 37
- rawsocket\_LAN\_Beacon.c, 43
  - \_GNU\_SOURCE, 44
  - getInterfaces, 45
  - lan\_beacon\_receiver, 45
  - receiveChallenge, 46
  - send\_lan\_beacon\_rawSock, 47
  - sendRawSocket, 48
- rawsocket\_LAN\_Beacon.h, 49
  - getInterfaces, 51
  - lan\_beacon\_receiver, 52
  - REC\_SOCKET, 51
  - receiveChallenge, 53
  - SEND\_SOCKET, 51
  - send\_lan\_beacon\_rawSock, 54
  - sendRawSocket, 55
- read\_keys
  - openssl\_sign.c, 33
  - openssl\_sign.h, 40
- receiveChallenge
  - rawsocket\_LAN\_Beacon.c, 46
  - rawsocket\_LAN\_Beacon.h, 53
- received\_lan\_beacon\_frame, 10
  - challenge, 10
  - current\_destination\_mac, 10
  - lan\_beacon\_ReceivedPayload, 11
  - parsedBeaconContents, 11
  - payloadSize, 11
  - successfullyAuthenticated, 11
  - times\_left\_to\_display, 11
- receiver
  - receiver.c, 59
  - receiver.h, 63
- receiver.c, 56
  - bananaPIprint, 57
  - evaluatelanbeacon, 58
  - receiver, 59
- TLV\_CUSTOM\_COPY, 57
  - TLV\_STRING\_COPY, 57
- receiver.h, 60
  - bananaPIprint, 62
  - evaluatelanbeacon, 62
  - receiver, 63
- receiver\_information, 12
  - authenticated\_mode, 13
  - current\_lan\_beacon\_pdu\_for\_printing, 13
  - lanbeacon\_keys, 13
  - my\_receiver\_interfaces, 13
  - number\_of\_currently\_received\_frames, 13
  - pointers\_to\_received\_frames, 14
  - scroll\_speed, 14
- SEND\_SOCKET
  - rawsocket\_LAN\_Beacon.h, 51
- SENDER\_MODE
  - openssl\_sign.h, 37
- SHOW\_FRAMES\_X\_TIMES
  - define.h, 22
- SIG\_LEN
  - openssl\_sign.c, 31
- SUBTYPE\_COMBINED\_STRING
  - define.h, 23
- SUBTYPE\_CUSTOM
  - define.h, 23
- SUBTYPE\_DHCP
  - define.h, 23
- SUBTYPE\_EMAIL
  - define.h, 23
- SUBTYPE\_IPV4
  - define.h, 23
- SUBTYPE\_IPV6
  - define.h, 23
- SUBTYPE\_NAME
  - define.h, 24
- SUBTYPE\_ROUTER
  - define.h, 24
- SUBTYPE\_SIGNATURE
  - define.h, 24
- SUBTYPE\_VLAN\_ID
  - define.h, 24
- scroll\_speed
  - receiver\_information, 14
- send\_frequency
  - sender\_information, 16
- send\_lan\_beacon\_rawSock
  - rawsocket\_LAN\_Beacon.c, 47
  - rawsocket\_LAN\_Beacon.h, 54
- sendOrReceive
  - interfaces, 7
- sendRawSocket
  - rawsocket\_LAN\_Beacon.c, 48
  - rawsocket\_LAN\_Beacon.h, 55
- sender
  - sender.c, 68
  - sender.h, 75
- sender.c, 64

- ipParser, [65](#)
- lanbeacon\_creator, [66](#)
- sender, [68](#)
- transfer\_to\_pdu, [69](#)
- transfer\_to\_pdu\_and\_string, [69](#)
- transfer\_to\_string, [70](#)
- sender.h, [71](#)
  - ipParser, [73](#)
  - lanbeacon\_creator, [74](#)
  - sender, [75](#)
  - transfer\_to\_pdu, [76](#)
  - transfer\_to\_pdu\_and\_string, [77](#)
  - transfer\_to\_string, [78](#)
- sender\_information, [14](#)
  - interface\_to\_send\_on, [15](#)
  - lan\_beacon\_pdu\_len, [16](#)
  - lanBeacon\_PDU, [16](#)
  - lanbeacon\_keys, [16](#)
  - my\_challenge\_receiver\_interfaces, [16](#)
  - send\_frequency, [16](#)
- sender\_or\_receiver\_mode
  - open\_ssl\_keys, [9](#)
- signlanbeacon
  - openssl\_sign.c, [34](#)
  - openssl\_sign.h, [41](#)
- sockfd
  - interfaces, [7](#)
- sockopt
  - interfaces, [7](#)
- successfullyAuthenticated
  - received\_lan\_beacon\_frame, [11](#)
- TLV\_CUSTOM\_COPY
  - receiver.c, [57](#)
- TLV\_STRING\_COPY
  - receiver.c, [57](#)
- times\_left\_to\_display
  - received\_lan\_beacon\_frame, [11](#)
- transfer\_to\_pdu
  - sender.c, [69](#)
  - sender.h, [76](#)
- transfer\_to\_pdu\_and\_string
  - sender.c, [69](#)
  - sender.h, [77](#)
- transfer\_to\_string
  - sender.c, [70](#)
  - sender.h, [78](#)
- VERFIY\_PROBLEM
  - openssl\_sign.c, [31](#)
- verifylanbeacon
  - openssl\_sign.c, [35](#)
  - openssl\_sign.h, [42](#)