Stream Parser (strparser)

Introduction

The stream parser (strparser) is a utility that parses messages of an application layer protocol running over a data stream. The stream parser works in conjunction with an upper layer in the kernel to provide kernel support for application layer messages. For instance, Kernel Connection Multiplexor (KCM) uses the Stream Parser to parse messages using a BPF program.

The strparser works in one of two modes: receive callback or general mode.

In receive callback mode, the strparser is called from the data_ready callback of a TCP socket. Messages are parsed and delivered as they are received on the socket.

In general mode, a sequence of skbs are fed to strparser from an outside source. Message are parsed and delivered as the sequence is processed. This modes allows strparser to be applied to arbitrary streams of data.

Interface

The API includes a context structure, a set of callbacks, utility functions, and a data_ready function for receive callback mode. The callbacks include a parse_msg function that is called to perform parsing (e.g. BPF parsing in case of KCM), and a rcv_msg function that is called when a full message has been completed.

Functions

```
strp_init(struct strparser *strp, struct sock *sk,
        const struct strp_callbacks *cb)
```

Called to initialize a stream parser. strp is a struct of type strparser that is allocated by the upper layer. sk is the TCP socket associated with the stream parser for use with receive callback mode; in general mode this is set to NULL. Callbacks are called by the stream parser (the callbacks are listed below).

```
void strp_pause(struct strparser *strp)
```

Temporarily pause a stream parser. Message parsing is suspended and no new messages are delivered to the upper layer.

```
void strp_unpause(struct strparser *strp)
```

Unpause a paused stream parser.

```
void strp_stop(struct strparser *strp);
```

strp_stop is called to completely stop stream parser operations. This is called internally when the stream parser encounters an error, and it is called from the upper layer to stop parsing operations.

```
void strp_done(struct strparser *strp);
```

strp_done is called to release any resources held by the stream parser instance. This must be called after the stream processor has been stopped.

```
int strp_process(struct strparser *strp, struct sk_buff *orig_skb,
                 unsigned int orig_offset, size_t orig_len,
                 size_t max_msg_size, long timeo)
```

strp_process is called in general mode for a stream parser to parse an sk_buff. The number of bytes processed or a negative error number is returned. Note that strp_process does not consume the sk_buff. max_msg_size is maximum size the stream parser will parse. timeo is timeout for completing a message.

```
void strp_data_ready(struct strparser *strp);
```

The upper layer calls strp_tcp_data_ready when data is ready on the lower socket for strparser to process. This should be called from a data_ready callback that is set on the socket. Note that maximum messages size is the limit of the receive socket buffer and message timeout is the receive timeout for the socket.

```
void strp_check_rcv(struct strparser *strp);
```

strp_check_rcv is called to check for new messages on the socket. This is normally called at initialization of a stream parser instance or after strp_unpause.

Callbacks

There are six callbacks:

```
int (*parse_msg)(struct strparser *strp, struct sk_buff *skb);
```

parse_msg is called to determine the length of the next message in the stream. The upper layer must implement this function. It should parse the sk_buff as containing the headers for the next application layer message in the stream.

The skb->cb in the input skb is a struct strp_msg. Only the offset field is relevant in parse_msg and gives the offset where the message starts in the skb.

The return values of this function are:

>0	indicates length of successfully parsed message
0	indicates more data must be received to parse the message
-ESTRPIPE	current message should not be processed by the kernel, return control of the socket to userspace which can proceed to read the messages itself
other < 0	Error in parsing, give control back to userspace assuming that synchronization is lost and the stream is unrecoverable (application expected to close TCP socket)

In the case that an error is returned (return value is less than zero) and the parser is in receive callback mode, then it will set the error on TCP socket and wake it up. If parse_msg returned -ESTRPIPE and the stream parser had previously read some bytes for the current message, then the error set on the attached socket is ENODATA since the stream is unrecoverable in that case.

```
void (*lock)(struct strparser *strp)
```

The lock callback is called to lock the strp structure when the strparser is performing an asynchronous operation (such as processing a timeout). In receive callback mode the default function is to lock_sock for the associated socket. In general mode the callback must be set appropriately.

```
void (*unlock)(struct strparser *strp)
```

The unlock callback is called to release the lock obtained by the lock callback. In receive callback mode the default function is release_sock for the associated socket. In general mode the callback must be set appropriately.

```
void (*rcv_msg)(struct strparser *strp, struct sk_buff *skb);
```

rcv_msg is called when a full message has been received and is queued. The callee must consume the sk_buff; it can call strp_pause to prevent any further messages from being received in rcv_msg (see strp_pause above). This callback must be set.

The skb->cb in the input skb is a struct strp_msg. This struct contains two fields: offset and full_len. Offset is where the message starts in the skb, and full_len is the the length of the message. skb->len - offset may be greater then full_len since strparser does not trim the skb.

```
   int (*read_sock_done)(struct strparser *strp, int err);

read_sock_done is called when the stream parser is done reading
the TCP socket in receive callback mode. The stream parser may
read multiple messages in a loop and this function allows cleanup
to occur when exiting the loop. If the callback is not set (NULL
in strp_init) a default function is used.

::

   void (*abort_parser)(struct strparser *strp, int err);

This function is called when stream parser encounters an error
in parsing. The default function stops the stream parser and
sets the error in the socket if the parser is in receive callback
mode. The default function can be changed by setting the callback
to non-NULL in strp_init.
```

Statistics

Various counters are kept for each stream parser instance. These are in the strp_stats structure. strp_aggr_stats is a convenience structure for accumulating statistics for multiple stream parser instances. save_strp_stats and aggregate_strp_stats are helper functions to save and aggregate statistics.

Message assembly limits

The stream parser provide mechanisms to limit the resources consumed by message assembly.

A timer is set when assembly starts for a new message. In receive callback mode the message timeout is taken from rcvtime for the associated TCP socket. In general mode, the timeout is passed as an argument in strp_process. If the timer fires before assembly completes the stream parser is aborted and the ETIMEDOUT error is set on the TCP socket if in receive callback mode.

In receive callback mode, message length is limited to the receive buffer size of the associated TCP socket. If the length returned by parse_msg is greater than the socket buffer size then the stream parser is aborted with EMSGSIZE error set on the TCP socket. Note that this makes the maximum size of receive skbuffs for a socket with a stream parser to be 2*sk_rcvbuf of the TCP socket.

In general mode the message length limit is passed in as an argument to strp_process.

Author

Tom Herbert (tom@quantonium.net)