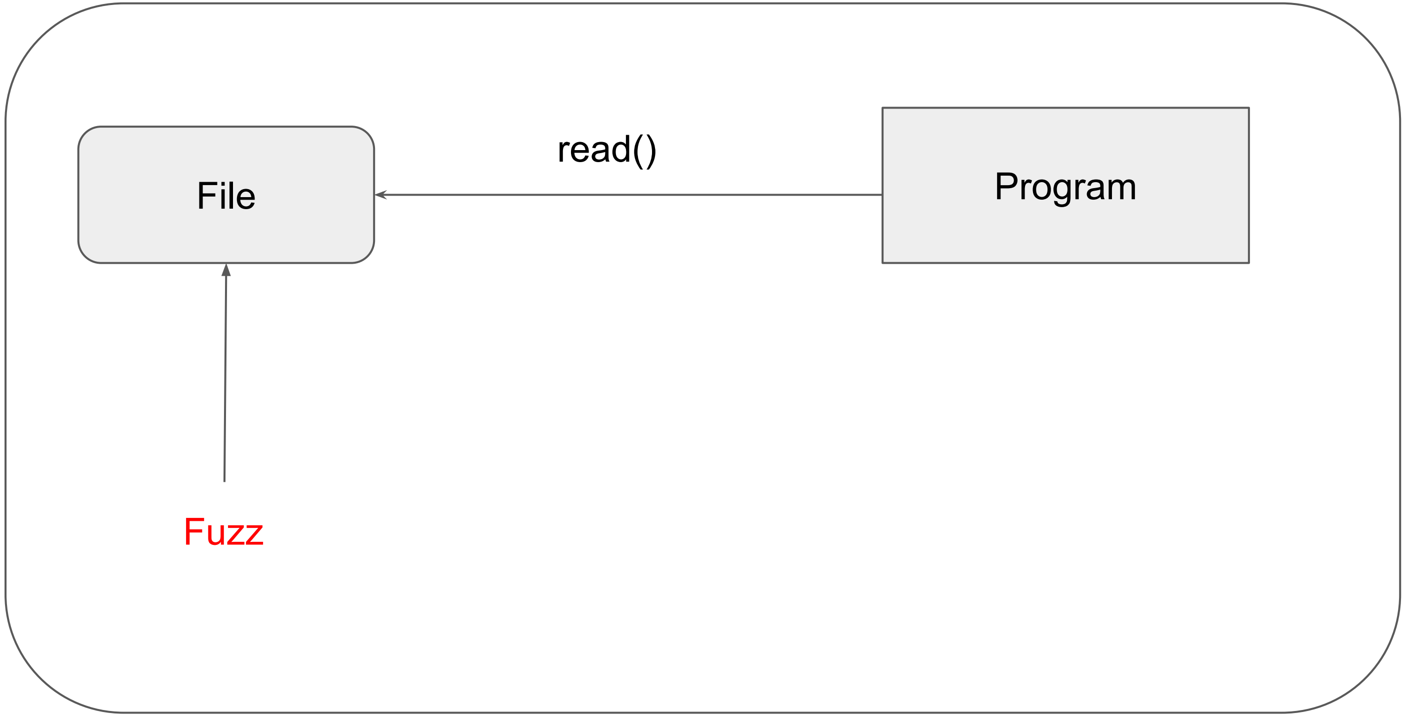
Mongoose AFL Fuzzing

Dobin Rutishauser, 10.07.2017, v1.0

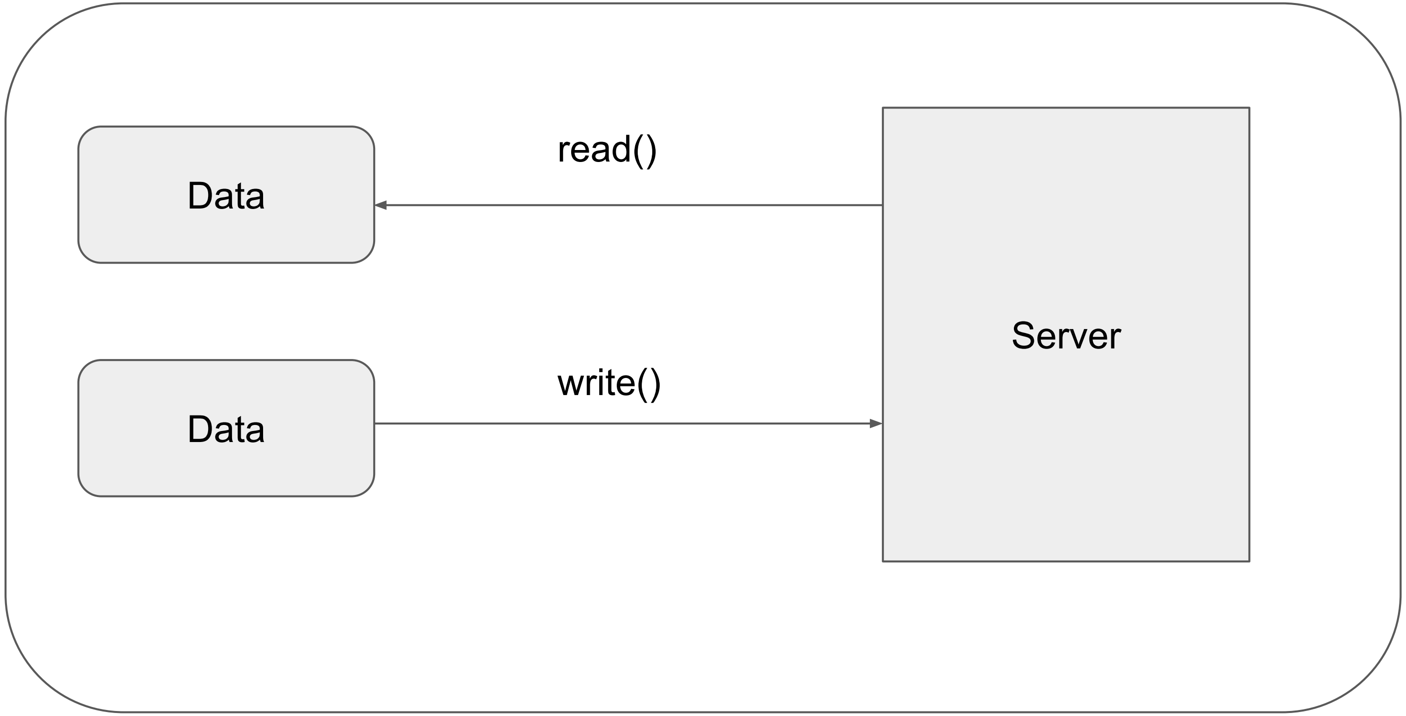
Mongoose

- Cross-platform: works on Linux/UNIX, MacOS, QNX, eCos, Windows, Android, iPhone, FreeRTOS (TI CC3200, ESP8266), etc
- Supported hardware platforms: TI CC3200, TI MSP432, NRF52, STM32, PIC32, ESP8266, ESP32 and more
- Builtin protocols:
    - plain TCP, plain UDP, SSL/TLS (over TCP, one-way or two-way)
    - HTTP client, HTTP server
    - WebSocket client, WebSocket server
    - MQTT client, MQTT broker
    - CoAP client, CoAP server
    - DNS client, DNS server, async DNS resolver
- Single-threaded, asynchronous, non-blocking core with simple event-based API
- Native support for PicoTCP embedded TCP/IP stack, LWIP embedded TCP/IP stack
- Tiny static and run-time footprint
- Source code is both ISO C and ISO C++ compliant
- **Very easy to integrate: just copy mongoose.c and mongoose.h files to your build tree**

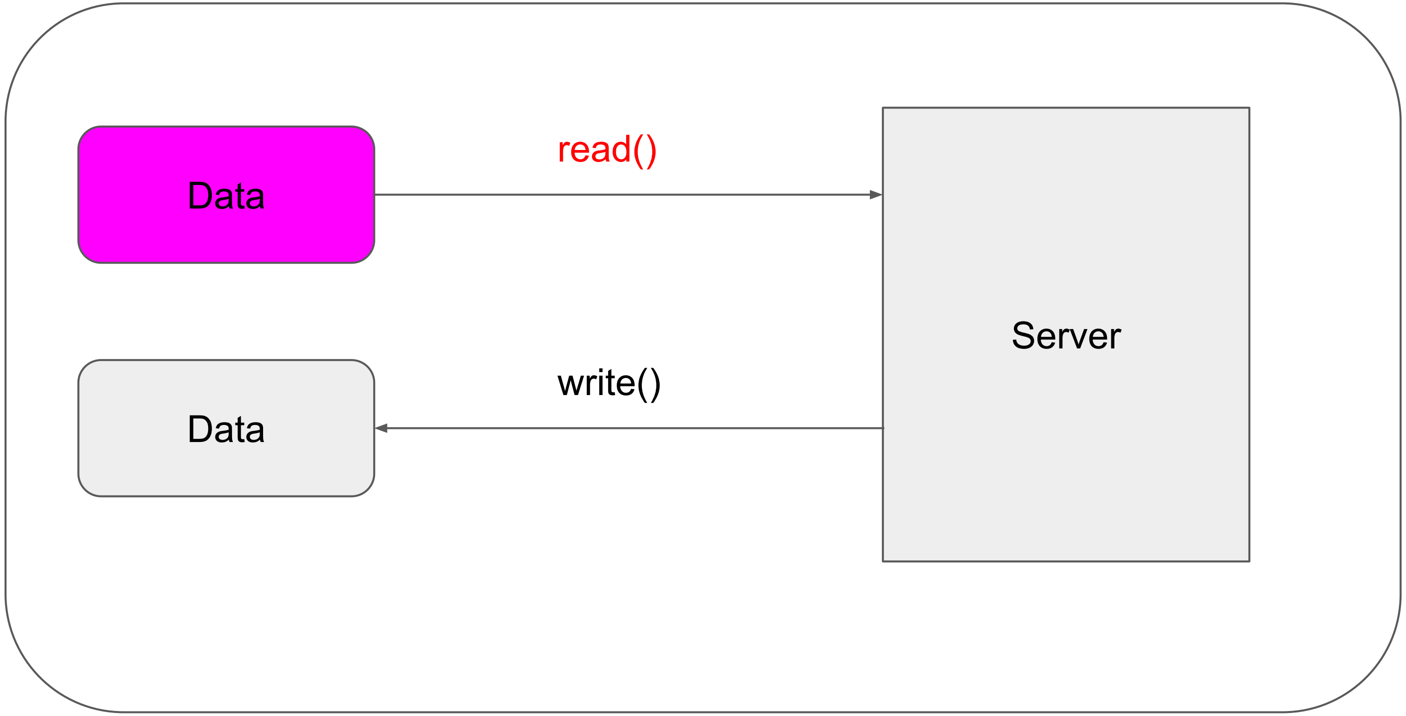
AFL Fuzzing



Unix Networking



Unix Networking AFL Fuzzing



Unix Networking in C

```
serverSocket = socket();
listen(serverSocket);

while( true ) {
    clientSocket = accept( serverSocket );

    packet = read( clientSocket );
    handlePacket( packet );
}
```

Unix Networking in C

```
serverSocket = socket();
listen(serverSocket);
while( true ) {
    clientSocket = accept( serverSocket );

    packet = read( clientSocket );
    handlePacket( packet );
}
```

Mongoose Networking

Mongoose Networking

```
// Periodically called (endless loop, async impl)
mg_socket_if_poll(struct mg_iface *iface, int timeout_ms) {

  // Check which sockets have data
  num_ev = select((int) max_fd + 1, &read_set, &write_set, ..);

  // handle all sockets which have data
  for (nc = mgr->active_connections; nc != NULL; nc = tmp) {
      mg_mgr_handle_conn(nc, fd_flags, now);
```

Mongoose Networking

```
void mg_mgr_handle_conn(struct mg_connection *nc, int fd_flags, double now) {
  if (fd_flags & _MG_F_FD_CAN_READ) {
    if (nc->flags & MG_F_UDP) {
      mg_handle_udp_read(nc);
    } else {
      if (nc->flags & MG_F_LISTENING) {
        mg_accept_conn(nc);
      } else {
        mg_handle_tcp_read(nc);
      }
    }
  }
```

Diff: TCP Poll 1: Remove 1 sec wait time

```
time_t mg_socket_if_poll(struct mg_iface *iface, int
timeout_ms) {

  tv.tv_sec = 0;      // timeout_ms / 1000;
  tv.tv_usec = 0;     // (timeout_ms % 1000) * 1000;
```

Diff: TCP Poll 2: Initialize Connection

```
#ifdef AFL
   static int init = 0;

   if (init != 1) {
       sock_t sock = 0;
       union socket_address sa;
       socklen_t sa_len = sizeof(sa);

       // Artifical new mongoose network connection
       struct mg_connection *nc =
           mg_if_accept_new_conn(mgr->active_connections);
       mg_sock_set(nc, sock);
       mg_if_accept_tcp_cb(nc, &sa, sa_len);
       init = 1;
   }
#endif
```

Diff: TCP Poll 3: Exit

```
   for (nc = mgr->active_connections; nc != NULL; nc = tmp) {
      tmp = nc->next;
      if ((nc->flags & MG_F_CLOSE_IMMEDIATELY) ||
          (nc->send_mbuf.len == 0 && (nc->flags & MG_F_SEND_AND_CLOSE))) {

 #ifdef AFL  /* For AFL persistent mode fuzzing shim  */
         exit(0);
 #endif
```

Diff: TCP Read

```
static void mg_handle_tcp_read(struct mg_connection *conn)
{
  int n = 0;
  char *buf = (char *) MG_MALLOC(MG_TCP_RECV_BUFFER_SIZE);

//    n = (int) MG_RECV_FUNC(conn->sock, buf,
//                           recv_avail_size(conn,
//                           MG_TCP_RECV_BUFFER_SIZE), 0);

  n = (int) read(conn->sock, buf,
          recv_avail_size(conn, MG_TCP_RECV_BUFFER_SIZE));
```

Solution

Current Solution:

- Remove select() timeout
- In poll loop: Fake-initialize a new connection from stdin
  - (internal mongoose structs)
- Change read() so it is able to read from socket (non TCP/IP socket)
- Exit immediatly after handling data

(Alternative Solution)

In tcp_read: Alternative to change read()

1. Read data from stdin into buf
2. send() buf to opened socket
3. (still need to artificially open a socket)

```
buf = read(0);
send(socket, buf);
```

Mongoose with STDIN networking

- Use wireshark to capture example protocol data (HTTP, DNS, MQTT, …)
- Save capture to file
- Put content of file into stdin

```
$ hexdump -C mqtt.dat
00000000  82 0b 00 2a 00 06 2f 73  74 75 66 66 00    |...*../stuff.|

$ cat mqtt.dat | ./mongoose-mqtt
```

Mongoose with AFL

```
$ export CC=afl-clang
$ export CFLAGS_EXTRA="-DAFL"

$ make
afl-clang mqtt_broker.c ../../mongoose.c -o mqtt_broker -g -W -Wall -I../..
-Wno-unused-function -DAFL -DMG_ENABLE_MQTT_BROKER
afl-as 2.41b by <lcamtuf@google.com>
[+] Instrumented 2323 locations (64-bit, non-hardened mode, ratio 100%).

$ cp mqtt.dat afl_in/

$ afl-fuzz -i afl_in -o afl_out ./mongoose
```

Mongoose with AFL

```
# ls afl_out/crashes

id:000027,sig:11,src:000130,op:havoc,rep:32

id:000028,sig:11,src:000081+000077,op:splice,rep:2

id:000029,sig:11,src:000114+000170,op:splice,rep:64

id:000030,sig:11,src:000045+000084,op:splice,rep:128

id:000031,sig:11,src:000099+000170,op:splice,rep:64
```

Mongoose Fuzzing

I fuzzed:

- examples/restful_server
- examples/captive_dns_server
- examples/cookie_auth
- examples/mqtt_broker

Mongoose Fuzzing

Result:

- examples/restful_server
- examples/captive_dns_server
- examples/cookie_auth -> null ptr deref
- examples/mqtt_broker -> stack null bytes overflow

Mongoose Fuzzing: Cookie auth

```
(gdb) r < afl_out/crashes/id\:000001*
Starting web server on port 8000
Program received signal SIGSEGV, Segmentation fault.
0x0000000000402bf8 in ev_handler (nc=0x657650, ev=<optimized out>,
    p=<optimized out>) at cookie_auth.c:217

216       const struct session * s = (const struct session *) nc->user_data;
217       mg_printf_html_escape(nc, "%s",  s->user);
```

```
hexdump -C id\:000001*
00000000  5a 4f 53 54 20 2f 6c 6f  67 69 6e 2e 68 74 6d 6c  |ZOST /login.html|
00000010  54 54 50 2f 2e 2e 20 0d  0a 48 6f 73 74 3a 0a 69  |TTP/.. ..Host:.i|
00000020  6d 0d 0a 43 79 30 0d 0a  52 65 72 3a 20 68 0a 43  |m..Cy0..Rer: h.C|
00000030  6f 3a 20 6d 67 73 3d 0d  0a 43 20 6d 67 73 3d 0d  |o: mgs=..C mgs=.|
00000040  0a 6f 6e 3a 73 65 0d 0a  0d 0a 75 73              |.on:se....us|
0000004c
```

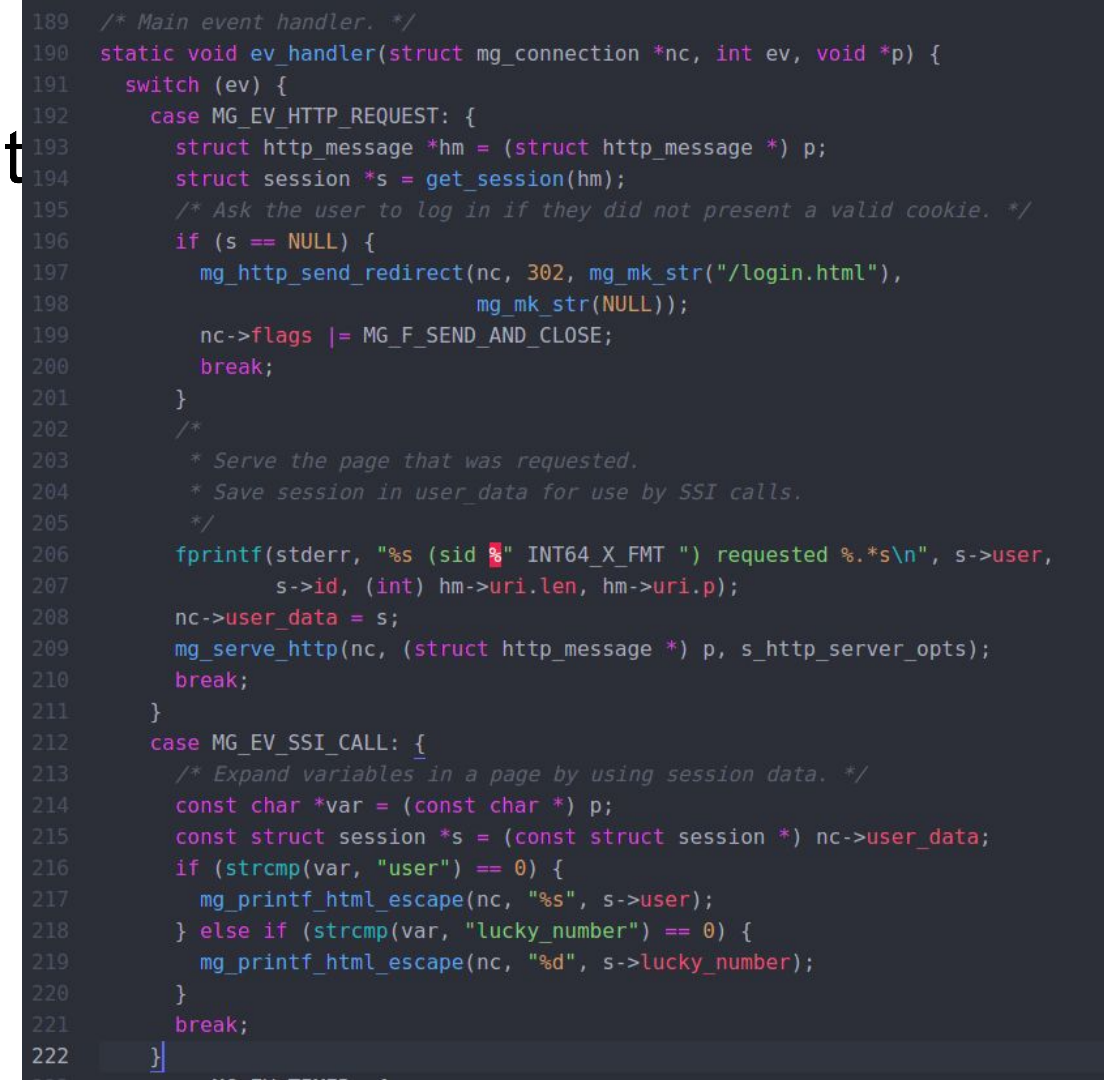
Mongoose: Cookie aut

MG_EV_HTTP_REQUEST:

- Session s is checked

MG_EV_SSI_CALL:

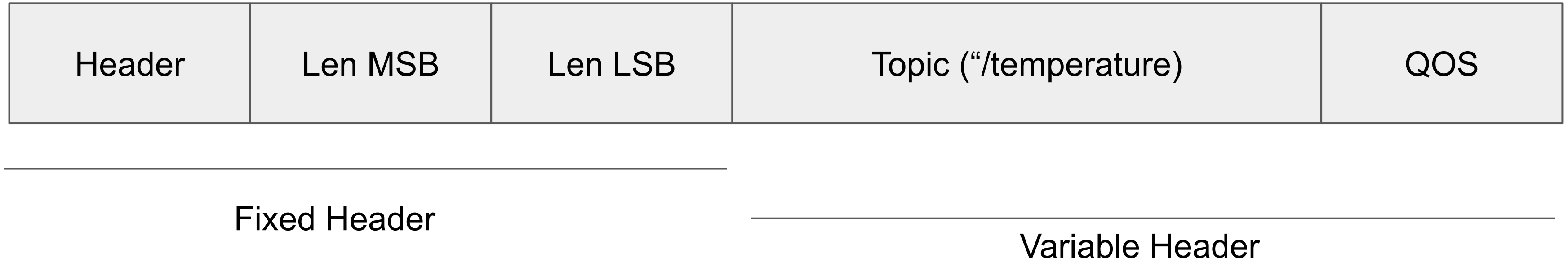
- Session is is not checked



```c
/* Main event handler. */
static void ev_handler(struct mg_connection *nc, int ev, void *p) {
  switch (ev) {
    case MG_EV_HTTP_REQUEST: {
      struct http_message *hm = (struct http_message *) p;
      struct session *s = get_session(hm);
      /* Ask the user to log in if they did not present a valid cookie. */
      if (s == NULL) {
        mg_http_send_redirect(nc, 302, mg_mk_str("/login.html"),
                              mg_mk_str(NULL));
        nc->flags |= MG_F_SEND_AND_CLOSE;
        break;
      }
      /*
       * Serve the page that was requested.
       * Save session in user_data for use by SSI calls.
       */
      fprintf(stderr, "%s (sid %" INT64_X_FMT ") requested %.*s\n", s->user,
              s->id, (int) hm->uri.len, hm->uri.p);
      nc->user_data = s;
      mg_serve_http(nc, (struct http_message *) p, s_http_server_opts);
      break;
    }
    case MG_EV_SSI_CALL: {
      /* Expand variables in a page by using session data. */
      const char *var = (const char *) p;
      const struct session *s = (const struct session *) nc->user_data;
      if (strcmp(var, "user") == 0) {
        mg_printf_html_escape(nc, "%s", s->user);
      } else if (strcmp(var, "lucky_number") == 0) {
        mg_printf_html_escape(nc, "%d", s->lucky_number);
      }
      break;
    }
```

Mongoose: MQTT

“MQTT is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport. It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium. “



Mongoose MQTT: The bug

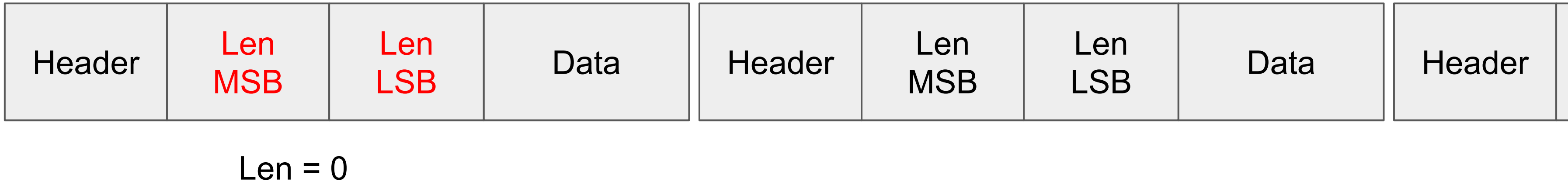
```
Stopped reason: SIGSEGV
0x000000000040ef13 in mg_mqtt_next_subscribe_topic (msg=0x7fffffffd640,
topic=0x7fffffffd3a0, qos=0x7fffffffd383 "\001p\f", pos=0xc70) at
../../mongoose.c:9933

9933  topic->len = buf[0] << 8 | buf[1];
```

```
   0x40ef0f <mg_mqtt_next_subscribe_topic+231>:   mov    rax,QWORD PTR [rbp-0x10]
=> 0x40ef13 <mg_mqtt_next_subscribe_topic+235>:   movzx  eax,BYTE PTR [rax]

gdb-peda$ i r eax rax
eax            0x610d71  0x610d71
rax            0x610d71  0x610d71
gdb-peda$ x/1xw 0x610d71
0x610d71: Cannot access memory at address 0x610d71
```

Mongoose: MQTT

In a MQTT TCP connection:

Multiple MQTT messages



Mongoose: MQTT

```
  /* decode mqtt variable length */
  do {
    len += (*p & 127) << 7 * (p - &io->buf[1]);
  } while ((*p++ & 128) != 0 && ((size_t)(p - io->buf) <= io->len));

  end = p + len;
…
  case MG_MQTT_CMD_SUBSCRIBE:

      mm->message_id = getu16(p);

      p += 2;

      mm->payload.p = p;

      mm->payload.len = end - p;
```

Mongoose: MQTT

```
/* decode mqtt variable length */
do {
  len += (*p & 127) << 7 * (p - &io->buf[1]);
} while ((*p++ & 128) != 0 && ((size_t)(p - io->buf) <= io->len));

end = p + len;  // end = p + 0 = p
…
case MG_MQTT_CMD_SUBSCRIBE:

    mm->message_id = getu16(p);

    p += 2;

    mm->payload.p = p;

    mm->payload.len = end - p;  // len = (p+0) - (p+2)
```

len = -2 (0xfffffffffffffffe)

Mongoose: MQTT

```
  char qoss[512];
  for (pos = 0; (pos = mg_mqtt_next_subscribe_topic(msg, &topic, &qos, pos))
!= -1;) {
    qoss[qoss_len++] = qos;
  }
```

```
int mg_mqtt_next_subscribe_topic(...) {
  unsigned char *buf = (unsigned char *) msg->payload.p + pos;

  if ((size_t) pos >= msg->payload.len) {
    return -1;
  }

  *qos = buf[2 + topic->len];
  return pos + 2 + topic->len + 1;
}
```

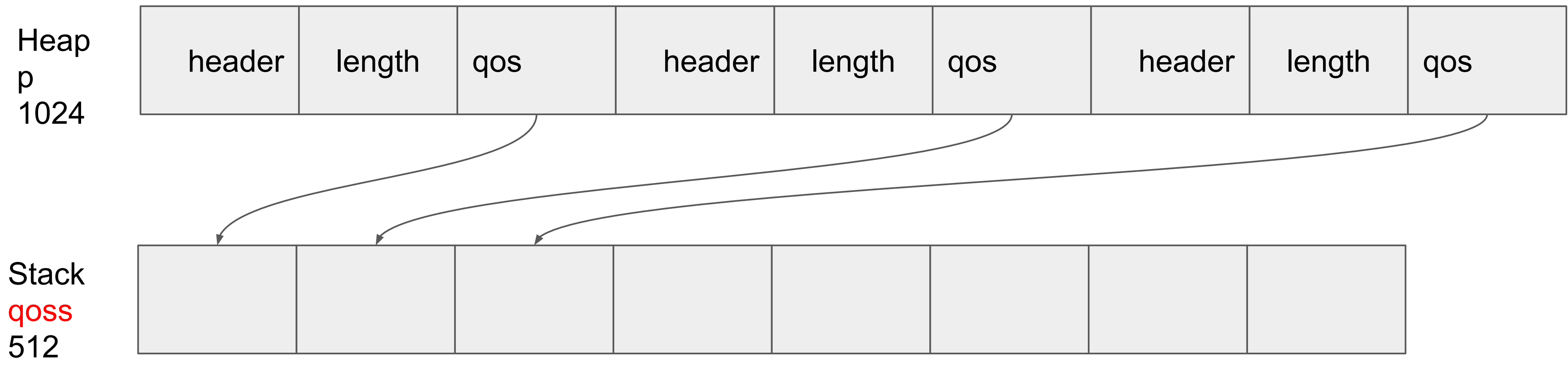
Mongoose: MQTT

```
  char qoss[512];
  for (pos = 0; (pos = mg_mqtt_next_subscribe_topic(msg, &topic, &qos, pos))
!= -1;) {
    qoss[qoss_len++] = qos;
  }
```

Overwrite everything in stack with QOS

```
int mg_mqtt_next_subscribe_topic(...) {
  unsigned char *buf = (unsigned char *) msg->payload.p + pos;

  if ((size_t) pos >= msg->payload.len) {
    return -1;
  }

  *qos = buf[2 + topic->len];
  return pos + 2 + topic->len + 1;
}
```

Mongoose: MQTT



Mongoose: MQTT

```
  char qoss[512];
  for (pos = 0; (pos = mg_mqtt_next_subscribe_topic(msg, &topic, &qos, pos))
!= -1;) {
    qoss[qoss_len++] = qos;
  }
```

Overwrite everything in stack with NULL

```
int mg_mqtt_next_subscribe_topic(...) {
  unsigned char *buf = (unsigned char *) msg->payload.p + pos;

  if ((size_t) pos >= msg->payload.len) {
    return -1;
  }

  *qos = buf[2 + topic->len];
  return pos + 2 + topic->len + 1;
}
```

Null ptr derefence

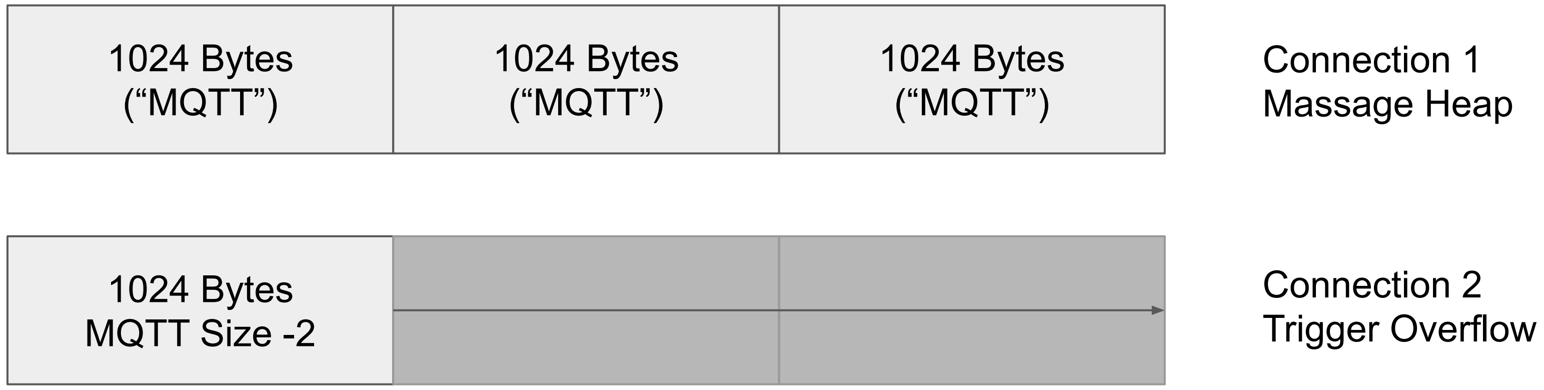
Mongoose MQTT

- Give size 0 in packet len, results in size -2 (0xfffffffffffffffe)
- Iterate through each MQTT message in packet
- Save QOS in static buffer

Problem:

- Loop does not stop
- But max packet len is 1024 bytes
- MQTT message is minimum 3 bytes
- 1024 / 3 < 512 :-(

MQTT Exploit solution



Mongoose: MQTT

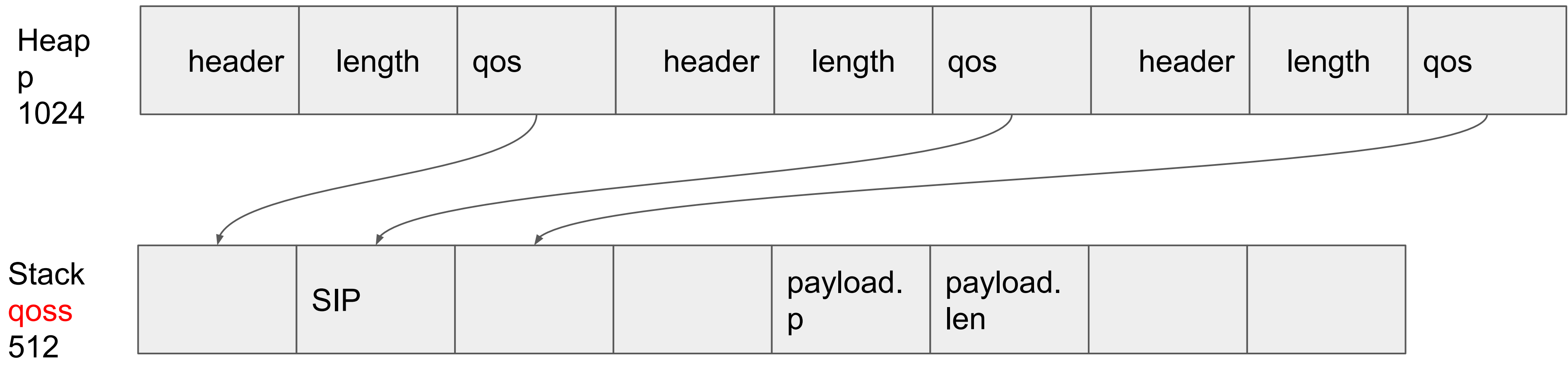
```
  char qoss[512];
  for (pos = 0; (pos = mg_mqtt_next_subscribe_topic(msg, &topic, &qos, pos))
!= -1;) {
    qoss[qoss_len++] = qos;
  }
```

```
int mg_mqtt_next_subscribe_topic(...) {
  unsigned char *buf = (unsigned char *) msg->payload.p + pos;

  if ((size_t) pos >= msg->payload.len) {
    return -1;
  }

  *qos = buf[2 + topic->len];
  return pos + 2 + topic->len + 1;
}
```

1: Overwrite with sane pointer (stop crash)

2: Overwrite with small len (make it return)

Mongoose: MQTT


Heap
p
1024
header
length
qos
header
length
qos
header
length
qos
Stack
qoss
512
SIP
payload.p
payload.len