fuzz实战之honggfuzz

Honggfuzz实战

前言

本文介绍 `libfuzzer` 和 `afl` 联合增强版 `honggfuzz` .同时介绍利用 `honggfuzz` 来 `fuzz` 网络应用服务。

介绍

`honggfuzz` 也是 `google` 开发的一款 `fuzz` . 其设计思路 和 `libfuzzer` 和 `afl`类似， 感觉就是 `libfuzzer` + `afl` 的 增强版。

编译

```
git clone https://github.com/google/honggfuzz.git
cd honggfuzz
make
```

`honggfuzz` 的使用文档在

https://github.com/google/honggfuzz/tree/master/docs

对几条命令做个解释

```
honggfuzz -f input_dir -z -s -- /usr/bin/djpeg
```

`-f`：指定初始样本集目录

`-z`：使用编译时的指令插桩信息来 为 样本变异做回馈， 默认选项

`-s`：表示目标程序从 `stdin` 获取输入，即样本数据通过 `stdin` 喂给程序

```
honggfuzz -f input_dir -- /usr/bin/djpeg ___FILE___
```

`___FILE___`：类似于 `afl` 的 `@@`，表示程序通过文件获取输入， `fuzz` 过程会被替换为相应的样本文件名

```
honggfuzz -f input_dir -P -- /usr/bin/djpeg_persistent_mode
```

`-P`：表示使用 `persistent` 模式

简单示例（libFuzzer模式）

这里用 `libfuzzer-workshop` 里面的 `libxml2` 来进行 `fuzz` 测试

https://github.com/Dor1s/libfuzzer-workshop/tree/master/lessons/08

下载那个 `libxml2.tgz` 然后解压，用 `hfuzz-clang` 编译

```
tar xvf libxml2.tgz
cd  libxml2/
CC=/home/haclh/vmdk_kernel/honggfuzz/hfuzz_cc/hfuzz-clang CXX=/home/haclh/vmdk_kernel/honggfuzz/hfuzz_cc/hfuzz-clang++
export CC=/home/haclh/vmdk_kernel/honggfuzz/hfuzz_cc/hfuzz-clang
export CXX=/home/haclh/vmdk_kernel/honggfuzz/hfuzz_cc/hfuzz-clang++
./autogen.sh
CC=/home/haclh/vmdk_kernel/honggfuzz/hfuzz_cc/hfuzz-clang CXX=/home/haclh/vmdk_kernel/honggfuzz/hfuzz_cc/hfuzz-clang++  ./configure
make -j4
```

`/home/haclh/vmdk_kernel/honggfuzz/` 是 `honggfuzz` 所在的目录

然后会生成 `libxml2/.libs/libxml2.a` 。

看看 `fuzzer` 代码

```
#ifdef __cplusplus
extern "C" {
#endif

#include <inttypes.h>
#include "libxml/parser.h"
#include <stdlib.h>

#include <libhfuzz/libhfuzz.h>

FILE* null_file = NULL;

int LLVMFuzzerInitialize(int* argc, char*** argv)
{
    null_file = fopen("/dev/null", "w");
```

```
    return 0;
}
int LLVMFuzzerTestOneInput(const uint8_t* buf, size_t len)
{
    xmlDocPtr p = xmlReadMemory((const char*)buf, len, "http://www.google.com", "UTF-8", XML_PARSE_RECOVER | XML_PARSE_NONET);
    if (!p) {
        return 0;
    }
    xmlDocFormatDump(null_file, p, 1);
    xmlFreeDoc(p);
    return 0;
}

#ifdef __cplusplus
}
#endif
```

和 `libfuzzer` 的代码基本一致，用 `hfuzz-clang` 编译（类似于 `libfuzzer`，需要和 `libhfuzz.a` 链接）。

```
/home/haclh/vmdk_kernel/honggfuzz/hfuzz_cc/hfuzz-clang persistent-xml2.c -Ilibxml2/include  libxml2/.libs/libxml2.a  /usr/lib/x86_64-linux-gnu/liblzma.a /home/haclh/vmdk_kernel/
```

然后运行 `fuzz`

```
/home/haclh/vmdk_kernel/honggfuzz/honggfuzz -W out -f ~/vmdk_kernel/libfuzzer-workshop-master/lessons/08/corpus2/ -- ./persistent-xml2
```

- `-W`：指定输出目录
- `-f`：指定 初始样本集目录

```
------------------------[  0 days 00 hrs 00 mins 03 secs ]----------------------
  Iterations : 1,336 [1.34k]
        Mode : [1/2] Feedback Driven Dry Run
      Target : ./persistent-xml2
     Threads : 1, CPUs: 2, CPU%: 105% [52%/CPU]
       Speed : 165/sec [avg: 445]
     Crashes : 0 [unique: 0, blacklist: 0, verified: 0]
    Timeouts : 0 [10 sec]
 Corpus Size : 406, max size: 8,192 bytes, init dir: 2,624 files
  Cov Update : 0 days 00 hrs 00 mins 00 secs ago
    Coverage : edge: 5,588 pc: 192 cmp: 56,229
---------------------------------- [ LOGS ] ------------------/ honggfuzz 1.6 /-
 ize:49 (i,b,hw,edge,ip,cmp): 0/0/0/6/0/34, Tot:0/0/0/5545/192/55781
Size:21 (i,b,hw,edge,ip,cmp): 0/0/0/5/0/94, Tot:0/0/0/5550/192/55875
Size:62 (i,b,hw,edge,ip,cmp): 0/0/0/1/0/6, Tot:0/0/0/5551/192/55881
Size:27 (i,b,hw,edge,ip,cmp): 0/0/0/1/0/2, Tot:0/0/0/5552/192/55883
Size:49 (i,b,hw,edge,ip,cmp): 0/0/0/1/0/0, Tot:0/0/0/5553/192/55883
Size:62 (i,b,hw,edge,ip,cmp): 0/0/0/2/0/1, Tot:0/0/0/5555/192/55884
Size:63 (i,b,hw,edge,ip,cmp): 0/0/0/1/0/3, Tot:0/0/0/5556/192/55887
Size:64 (i,b,hw,edge,ip,cmp): 0/0/0/2/0/5, Tot:0/0/0/5558/192/55892
Size:64 (i,b,hw,edge,ip,cmp): 0/0/0/0/0/1, Tot:0/0/0/5558/192/55893
```

Persistent Fuzzing

`honggfuzz` 还支持 `persistent` 模式， 而且他这种模式实现的比 `afl` 要 更加容易使用。直接上 `demo`

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <inttypes.h>
extern HF_ITER(uint8_t** buf, size_t* len);
void test(char* buf){
    if (buf[0] == 'f') {
      printf("one\n");
      if (buf[1] == 'o') {
        printf("two\n");
        if (buf[2] == 'o') {
          printf("three\n");
          if (buf[3] == '!') {
            printf("four\n");
            abort();
          }
        }
      }
    }
```

```
}
int main(void) {
  for (;;) {
    size_t len;
    uint8_t *buf;

    HF_ITER(&buf, &len);
    test(buf);

  }
  return 0;
}
```

代码和普通的应用差不多， 需要注意的就是 `HF_ITER` ， **程序可以通过这个宏 来获取** `honggfuzz` **生成的样本数据**，这就非常方便了，我们可以把它插入对数据处理的逻辑，然后循环这段逻辑，就可以不断的进行 fuzz 而不用重新起进程。

对应到上面的代码就是， 一个死循环，循环里面使用 `HF_ITER` 获取数据，然后把数据喂给我们要测试的逻辑 （这里是 `test` 函数）， 在 `test` 函数内部 如果满足条件，会触发 `abort` 模拟一个`crash`， 让 `honggfuzz` 捕获到。

编译

```
/home/haclh/vmdk_kernel/honggfuzz/hfuzz_cc/hfuzz-clang test.c /home/haclh/vmdk_kernel/honggfuzz/libhfuzz/libhfuzz.a -o test
```

然后运行

```
mkdir in
echo 111 > in/1
/home/haclh/vmdk_kernel/honggfuzz/honggfuzz -P -f in/ -W out -- ./test
```

`-P`： 用于开启 `persistent` 模式

马上就能看的 `crash` 了。

```
-----------------------[  0 days 00 hrs 00 mins 04 secs ]----------------------
  Iterations : 3,265 [3.27k]
        Mode : [2/2] Feedback Driven Mode
      Target : ./test
     Threads : 1, CPUs: 2, CPU%: 100% [50%/CPU]
       Speed : 215/sec [avg: 816]
     Crashes : 19 [unique: 0, blacklist: 0, verified: 0]
    Timeouts : 0 [10 sec]
 Corpus Size : 5, max size: 8,192 bytes, init dir: 11 files
  Cov Update : 0 days 00 hrs 00 mins 04 secs ago
    Coverage : edge: 11 pc: 0 cmp: 128
--------------------------------- [ LOGS ] ------------------/ honggfuzz 1.6 /-
 ersistent mode: Launched new persistent PID: 2528
Crash (dup): 'out/SIGABRT.PC.7ffff6efa428.STACK.17326c6e5e.CODE.-6.ADDR.(nil).INSTR.cmp____$0xffffffffffff000,%rax.fuzz' already exists, skipping
[2018-04-23T22:26:16-0700][W][2509] arch_checkWait():309 Persistent mode: PID 2528 exited with status: SIGNALED, signal: 6 (Aborted)
Persistent mode: Launched new persistent PID: 2530
Crash (dup): 'out/SIGABRT.PC.7ffff6efa428.STACK.17326c6e5e.CODE.-6.ADDR.(nil).INSTR.cmp____$0xffffffffffff000,%rax.fuzz' already exists, skipping
[2018-04-23T22:26:16-0700][W][2509] arch_checkWait():309 Persistent mode: PID 2530 exited with status: SIGNALED, signal: 6 (Aborted)
```

然后会在 `out` 目录生成触发 `crash` 的文件

```
22:28 haclh@ubuntu:persistent $ xxd out/SIGABRT.PC.7ffff6efa428.STACK.17326c6e5e.CODE.-6.ADDR.\(nil\).INSTR.cmp____\$0xffffffffffff000\,%rax.fuzz
00000000: 666f 6f21                                foo!
```

内容满足触发 `abort` 的条件

Honggfuzz NetDriver

介绍

这个是 `honggfuzz` 自带的一个库 用于 `fuzz socket` 类程序

https://github.com/google/honggfuzz/tree/master/libhfnetdriver

用它 `fuzz socket` 程序很方便， 只要把 `main` 函数 改成 `HFND_FUZZING_ENTRY_FUNCTION` （: 不确定是不是所有的都这样，具体还是得看代码 ， 以后再研究研究

```
int main(int argc, char *argv[]){
    ......................
    ......................
    ......................
}
```

改成

```
HFND_FUZZING_ENTRY_FUNCTION(int argc, char *argv[]){
    ......................
    ......................
    ......................
}
```

然后用 `hfuzz-clang` 编译， 同时用 `libhfnetdriver.a` 链接即可。

demo

不知道为啥 `libmodbus` 的 `tcp server` 用这种方式跑不起来，这里用一个 有漏洞的 `socket` 程序作为例子，试试这个功能。

vuln.c

```
#include <crypt.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

/* Do nothing with first message */
void handleData0(char *data, int len) {
    printf("Auth success\n");
}

/* Second message is stack based buffer overflow */
void handleData1(char *data, int len) {
    char buff[8];
    bzero(buff, 8);
    memcpy(buff, data, len);
    printf("Handledata1: %s\n", buff);
}

/* Third message is heap overflow */
void handleData2(char *data, int len) {
    char *buff = malloc(8);
    bzero(buff, 8);
    memcpy(buff, data, len);
    printf("Handledata2: %s\n", buff);
    free(buff);
}

void handleData3(char *data, int len) {
    printf("Meh: %i\n", len);
}

void handleData4(char *data, int len) {
    printf("Blah: %i\n", len);
}

void doprocessing(int sock) {
    char data[1024];
    int n = 0;
    int len = 0;

    while (1) {
        bzero(data, sizeof(data));
        len = read(sock, data, 1024);

        if (len == 0 || len <= 1) {
            return;
        }

        printf("Received data with len: %i on state: %i\n", len, n);
        switch (data[0]) {
            case 'A':
                handleData0(data, len);
                write(sock, "ok", 2);
                break;
            case 'B':
                handleData1(data, len);
                write(sock, "ok", 2);
                break;
```

```
            case 'C':
                handleData2(data, len);
                write(sock, "ok", 2);
                break;
            case 'D':
                handleData3(data, len);
                write(sock, "ok", 2);
                break;
            case 'E':
                handleData4(data, len);
                write(sock, "ok", 2);
                break;
            default:
                return;
        }

        n++;
    }
}

HFND_FUZZING_ENTRY_FUNCTION(int argc, char *argv[]) {
    int sockfd, newsockfd, portno, clilen;
    char buffer[256];
    struct sockaddr_in serv_addr, cli_addr;
    int n, pid;

    if (argc == 2) {
        portno = atoi(argv[1]);
    } else {
        portno = 5001;
    }

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        perror("ERROR opening socket");
        exit(1);
    }

    int reuse = 1;
    if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEPORT, (const char *)&reuse, sizeof(reuse)) < 0)
        perror("setsockopt(SO_REUSEPORT) failed");

    bzero((char *)&serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(portno);

    printf("Listening on port: %i\n", portno);

    /* Now bind the host address using bind() call.*/
    if (bind(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        perror("ERROR on binding");
        exit(1);
    }

    listen(sockfd, 5);
    clilen = sizeof(cli_addr);

    while (1) {
        newsockfd = accept(sockfd, (struct sockaddr *)&cli_addr, &clilen);
        if (newsockfd < 0) {
            perror("ERROR on accept");
            exit(1);
        }
        printf("New client connected\n");
        doprocessing(newsockfd);
        printf("Closing...\n");
        shutdown(newsockfd, 2);
```

```
        close(newsockfd);
    }
}
```

就是一个简单的 `socket` 程序，其中有两个漏洞 一个栈溢出 一个堆溢出， 同时把 `int main` 改成了 `HFND_FUZZING_ENTRY_FUNCTION` 。

代码修改自： https://github.com/google/honggfuzz/blob/master/socketfuzzer/vulnserver_cov.c

编译

```
/home/haclh/vmdk_kernel/honggfuzz/hfuzz_cc/hfuzz-clang vuln.c /home/haclh/vmdk_kernel/honggfuzz/libhfnetdriver/libhfnetdriver.a  -o vuln
```

关键是要和 `libhfnetdriver.a` 链接

运行

```
mkdir in
echo A > in/1
echo B > in/2
_HF_TCP_PORT=5001 /home/haclh/vmdk_kernel/honggfuzz/honggfuzz  -f in -- ./vuln
```

前面三条命令用于生成两个简单样本文件

最后一条命令用于启动 `fuzzer` .

`_HF_TCP_PORT` 指定 `server` 监听的端口， `fuzzer` 会和这个端口建立 `tcp` 连接，然后发送数据。

秒出 `crash` 。

```
-----------------------[  0 days 00 hrs 00 mins 08 secs ]----------------------
  Iterations : 237
       Mode : [2/2] Feedback Driven Mode
     Target : ./vuln
    Threads : 1, CPUs: 2, CPU%: 0% [0%/CPU]
      Speed : 0/sec [avg: 29]
    Crashes : 7 [unique: 6, blacklist: 0, verified: 0]
   Timeouts : 0 [10 sec]
 Corpus Size : 7, max size: 8,192 bytes, init dir: 2 files
  Cov Update : 0 days 00 hrs 00 mins 02 secs ago
    Coverage : edge: 19 pc: 0 cmp: 254
--------------------------------- [ LOGS ] -----------------/ honggfuzz 1.6 /-
 ize:1 (i,b,hw,edge,ip,cmp): 0/0/0/1/0/2, Tot:0/0/0/15/0/251
[2018-04-24T00:13:50-0700][W][3283] arch_getProcMem():342 Couldn't PT_READ_D on pid 3285, addr: (nil): Input/output error
Crash: saved as './SIGSEGV.PC.0.STACK.0.CODE.1.ADDR.(nil).INSTR.[NOT_MMAPED].2018-04-24.00:13:50.3285.fuzz'
[2018-04-24T00:13:50-0700][W][3283] arch_checkWait():309 Persistent mode: PID 3284 exited with status: SIGNALED, signal: 11 (Segmentation fault)
Persistent mode: Launched new persistent PID: 3287
[2018-04-24T00:13:50-0700][W][3283] arch_getProcMem():342 Couldn't PT_READ_D on pid 3288, addr: (nil): Input/output error
Crash: saved as './SIGSEGV.PC.0.STACK.0.CODE.1.ADDR.(nil).INSTR.[NOT_MMAPED].2018-04-24.00:13:50.3288.fuzz'
[2018-04-24T00:13:50-0700][W][3283] arch_checkWait():309 Persistent mode: PID 3287 exited with status: SIGNALED, signal: 11 (Segmentation fault)
Persistent mode: Launched new persistent PID: 3290
```

参考

FUZZING TCP SERVERS

实战

本节以 `mongoose` 为例 实战一波

https://github.com/cesanta/mongoose

这是一个用于 嵌入式网络服务的库， 实现了很多 `IOT` 中用到的协议。

本节以 `http` 为例进行 `fuzz` 。

程序的源代码内有很多的示例， 其中 `examples/simplest_web_server` 目录里面是一个很简单的 `http server`。

把代码中的 `int main` 改成 `HFND_FUZZING_ENTRY_FUNCTION` 。

```
// Copyright (c) 2015 Cesanta Software Limited
// All rights reserved

#include "mongoose.h"

static const char *s_http_port = "8000";
static struct mg_serve_http_opts s_http_server_opts;

static void ev_handler(struct mg_connection *nc, int ev, void *p) {
  if (ev == MG_EV_HTTP_REQUEST) {
    mg_serve_http(nc, (struct http_message *) p, s_http_server_opts);
  }
}

HFND_FUZZING_ENTRY_FUNCTION(void) {
  struct mg_mgr mgr;
  struct mg_connection *nc;

  mg_mgr_init(&mgr, NULL);
```

```
  printf("Starting web server on port %s\n", s_http_port);
  nc = mg_bind(&mgr, s_http_port, ev_handler);
  if (nc == NULL) {
    printf("Failed to create listener\n");
    return 1;
  }

  // Set up HTTP server parameters
  mg_set_protocol_http_websocket(nc);
  s_http_server_opts.document_root = ".";  // Serve current directory
  s_http_server_opts.enable_directory_listing = "yes";

  for (;;) {
    mg_mgr_poll(&mgr, 1000);
  }
  mg_mgr_free(&mgr);

  return 0;
}
```

然后用 `hfuzz-clang` 编译， 注意要和 `libhfnetdriver.a` 链接

```
export HONGGFUZZ_HOME=/home/haclh/vmdk_kernel/honggfuzz
$HONGGFUZZ_HOME/hfuzz_cc/hfuzz-clang simplest_web_server.c ../../mongoose.c -o simplest_web_server  -I../.. -DMG_DISABLE_DAV_AUTH -DMG_ENABLE_FAKE_DAVLOCK $HONGGFUZZ_HOME/libhfn
```

然后开始 fuzz

```
_HF_TCP_PORT=8000 /home/haclh/vmdk_kernel/honggfuzz/honggfuzz -W oout -f corpus_httpl/ -w httpd.wordlist -- ./simplest_web_server
```

其中用到的样本集 和 字典文件来自

https://github.com/google/honggfuzz/tree/master/examples/apache-httpd

```
-----------------------[  0 days 13 hrs 10 mins 15 secs ]----------------------
  Iterations : 2,820,996 [2.82M]
        Mode : [2/2] Feedback Driven Mode
      Target : ./simplest_web_server
     Threads : 1, CPUs: 2, CPU%: 125% [62%/CPU]
       Speed : 250/sec [avg: 59]
     Crashes : 3420 [unique: 2, blacklist: 0, verified: 0]
    Timeouts : 1 [10 sec]
 Corpus Size : 94, max size: 996,825 bytes, init dir: 843 files
  Cov Update : 0 days 00 hrs 08 mins 21 secs ago
    Coverage : edge: 1,151 pc: 10 cmp: 12,937
---------------------------------- [ LOGS ] ------------------/ honggfuzz 1.6 /-
 rash (dup): 'oout/SIGSEGV.PC.45f466.STACK.1b09c09382.CODE.128.ADDR.(nil).INSTR.cmpq___$0x0,(%rax).fuzz' already exists, skipping
[2018-04-24T18:25:26-0700][W][5844] arch_checkWait():309 Persistent mode: PID 52596 exited with status: SIGNALED, signal: 11 (Segmentation fault)
Persistent mode: Launched new persistent PID: 53032
Crash (dup): 'oout/SIGSEGV.PC.439112.STACK.1a4cde986d.CODE.1.ADDR.0x10.INSTR.mov____0x10(%rax),%rax.fuzz' already exists, skipping
[2018-04-24T18:25:28-0700][W][5844] arch_checkWait():309 Persistent mode: PID 53032 exited with status: SIGNALED, signal: 11 (Segmentation fault)
Persistent mode: Launched new persistent PID: 53040
Crash (dup): 'oout/SIGSEGV.PC.439112.STACK.1a4cde986d.CODE.1.ADDR.0x10.INSTR.mov____0x10(%rax),%rax.fuzz' already exists, skipping
[2018-04-24T18:25:29-0700][W][5844] arch_checkWait():309 Persistent mode: PID 53040 exited with status: SIGNALED, signal: 11 (Segmentation fault)
Persistent mode: Launched new persistent PID: 53068
Crash (dup): 'oout/SIGSEGV.PC.439112.STACK.1a4cde986d.CODE.1.ADDR.0x10.INSTR.mov____0x10(%rax),%rax.fuzz' already exists, skipping
[2018-04-24T18:25:31-0700][W][5844] arch_checkWait():309 Persistent mode: PID 53068 exited with status: SIGNALED, signal: 11 (Segmentation fault)
```

发现了 `crash` ， 下面定位触发漏洞代码, 使用 `-fsanitize=address` 编译，可以在 触发漏洞时停下来。

```
clang -fsanitize=address simplest_web_server.c ../../mongoose.c -o simplest_web_server -g -W -Wall -Werror -I../.. -Wno-unused-function  -DMG_DISABLE_DAV_AUTH -DMG_ENABLE_FAKE_D
```

然后触发漏洞的样本发送给服务器，就会 `crash` 。

```
01:19 haclh@ubuntu:simplest_web_server $ ./simplest_web_server
Starting web server on port 8000
=================================================================
==16867==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x619000000980 at pc 0x0000004e653d bp 0x7fffb8bab790 sp 0x7fffb8baaf40
READ of size 876 at 0x619000000980 thread T0
    #0 0x4e653c in __asan_memcpy /home/haclh/vmdk_kernel/libfuzzer-workshop-master/src/llvm/projects/compiler-rt/lib/asan/asan_interceptors_memintrinsics.cc:23
    #1 0x53b9d6 in mbuf_insert /tmp/t/mongoose-6.11/examples/simplest_web_server/../../mongoose.c:1477:24
    #2 0x53bafc in mbuf_append /tmp/t/mongoose-6.11/examples/simplest_web_server/../../mongoose.c:1490:10
```

然后可以定位到 `mongoose.c` 的 8925 行

```
 8923   if (mg_start_process(opts->cgi_interpreter, prog, blk.buf, blk.vars, dir,
 8924                        fds[1]) != 0) {
 8925     size_t n = nc->recv_mbuf.len - (hm->message.len - hm->body.len);
 8926     struct mg_connection *cgi_nc =
 8927         mg_add_sock(nc->mgr, fds[0], mg_cgi_ev_handler MG_UD_ARG(nc));
 8928     struct mg_http_proto_data *cgi_pd = mg_http_get_proto_data(nc);
 8929     cgi_pd->cgi.cgi_nc = cgi_nc;
 8930 #if !MG_ENABLE_CALLBACK_USERDATA
 8931     cgi_pd->cgi.cgi_nc->user_data = nc;
 8932 #endif
 8933     nc->flags |= MG_F_HTTP_CGI_PARSE_HEADERS;
 8934     /* Push POST data to the CGI */
```

```
8935    if (n > 0 && n < nc->recv_mbuf.len) {
8936      mg_send(cgi_pd->cgi.cgi_nc, hm->body.p, n);
8937    }
```

来源： https://www.cnblogs.com/hac425/p/9416915.html