Exploring Networks

Reliable UDP: Recap

Dominik Kovacs

October 29, 2021

Recap

Assignment 4: Common Pitfalls

Problem 0: Double free

Problem 1: Binary files output and text differ

Problem 2: It works on TCP, but not on UDP

Problem 3: Text files work but not binary/random files

Problem 4: Not correctly initializing structs

Problem 5: Use *stale* memory

Assignment 4: Common Pitfalls

Problem 0: Double free

```
free(): double free detected in tcache 2
Aborted (core dumped)
```

Takeaway 0: Know how to debug those issues

- Rerun with strace

```
$ strace            ./netster -f output
$ strace -k         ./netster -f output
```

- Rerun with ltrace

```
$ ltrace            ./netster -f output
$ ltrace -e fclose ./netster -f output
$ ltrace -e fclose -w 5 ./netster -f output
```

- Rerun with gdb

```
$ gdb -ex run --args ./netster -f output
$ gdb -ex 'b fclose' -ex 'ignore 1 1' -ex r --args ./netster -f output
```

Assignment 4: Common Pitfalls

Problem 1: Binary files output and text differ

```
char recv_buffer[256];

recv(sockfd, recv_buffer, sizeof(recv_buffer), 0);

fwrite(recv_buffer, sizeof(char), sizeof(recv_buffer), fp);
```

Takeaway 1: Analyse binary data with xxd or hexdump -C

Takeaway 2: Only write what you receive

```
char recv_buffer[256];

int read;
read = recv(sockfd, recv_buffer, sizeof(recv_buffer), 0);

fwrite(recv_buffer, sizeof(char), read, fp);
```

Assignment 4: Common Pitfalls

Problem 2: Text files work but not binary/random files

```
char recv_buffer[256];

memset(recv_buffer, 0, BUFLEN);
recv(sockfd, recv_buffer, sizeof(recv_buffer), 0);

fwrite(recv_buffer, sizeof(char), strlen(recv_buffer), fp);
```

Takeaway 3: Do not use string functions on binary data

```
char recv_buffer[256];

int read;
read = recv(sockfd, recv_buffer, sizeof(recv_buffer), 0);

fwrite(recv_buffer, sizeof(char), read, fp);
```

Problem 3: It works on TCP, but not on UDP

```
int read;
int count = 0;
do {
  count++;
  read = recvfrom(sockfd, recv_buffer, sizeof(recv_buffer), 0,
           (struct sockaddr *) &client_addr, &len);
} while (read > 0);

printf("%d\n", count);
```

Takeaway 4: TCPs FIN packet is received by the server

```
IP  CLIENT  >  SERVER:  Flags  [S]   length  0
IP  SERVER  >  CLIENT:  Flags  [S.]  length  0
IP  CLIENT  >  SERVER:  Flags  [.]   length  0

IP  CLIENT  >  SERVER:  Flags  [P.]  length  2
IP  SERVER  >  CLIENT:  Flags  [.]   length  0
...
IP  CLIENT  >  SERVER:  Flags  [P.]  length  2
IP  SERVER  >  CLIENT:  Flags  [.]   length  0

IP  CLIENT  >  SERVER:  Flags  [F.]  length  0
IP  SERVER  >  CLIENT:  Flags  [F.]  length  0
IP  CLIENT  >  SERVER:  Flags  [.]   length  0
```

Problem 4: Not correctly initializing structs

```
struct addrinfo hints;

hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_DGRAM;
hints.ai_protocol = 0;

printf("hints.ai_flags = %d\n", hints.ai_flags);
```

Takeaway 5: Declared variables/structs are never 0

```
struct addrinfo hints;

memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_DGRAM;
hints.ai_protocol = 0;
hints.ai_flags = 0;

printf("hints.ai_flags = %d\n", hints.ai_flags);
```

Problem 5: Use *stale* memory

```
struct addrinfo *result, *rp;

rp = result;
sockfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);

freeaddrinfo(result);

sendto(sockfd, buffer, len, 0, rp->ai_addr, rp->ai_addrlen);
```

Takeaway 6: Never use freed memory

```
struct addrinfo *result, *rp;

rp = result;
sockfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);

sendto(sockfd, buffer, len, 0, rp->ai_addr, rp->ai_addrlen);

freeaddrinfo(result);
```

Assignment 4: Common Pitfalls

Takeaway 0: Know how to debug those issues

Takeaway 1: Analyse binary data with xxd or hexdump -C

Takeaway 2: Only write what you receive

Takeaway 3: Do not use string functions on binary data

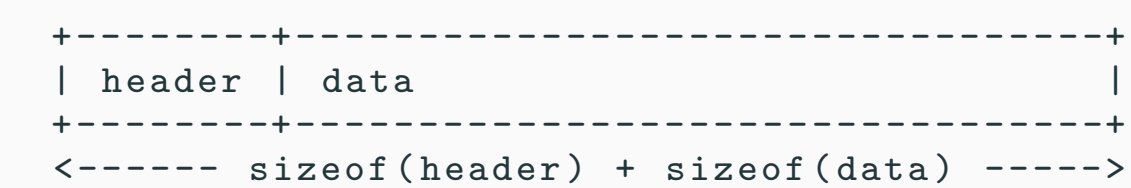
Takeaway 4: TCPs FIN packet is received by the server

Takeaway 5: Declared variables/structs are never 0

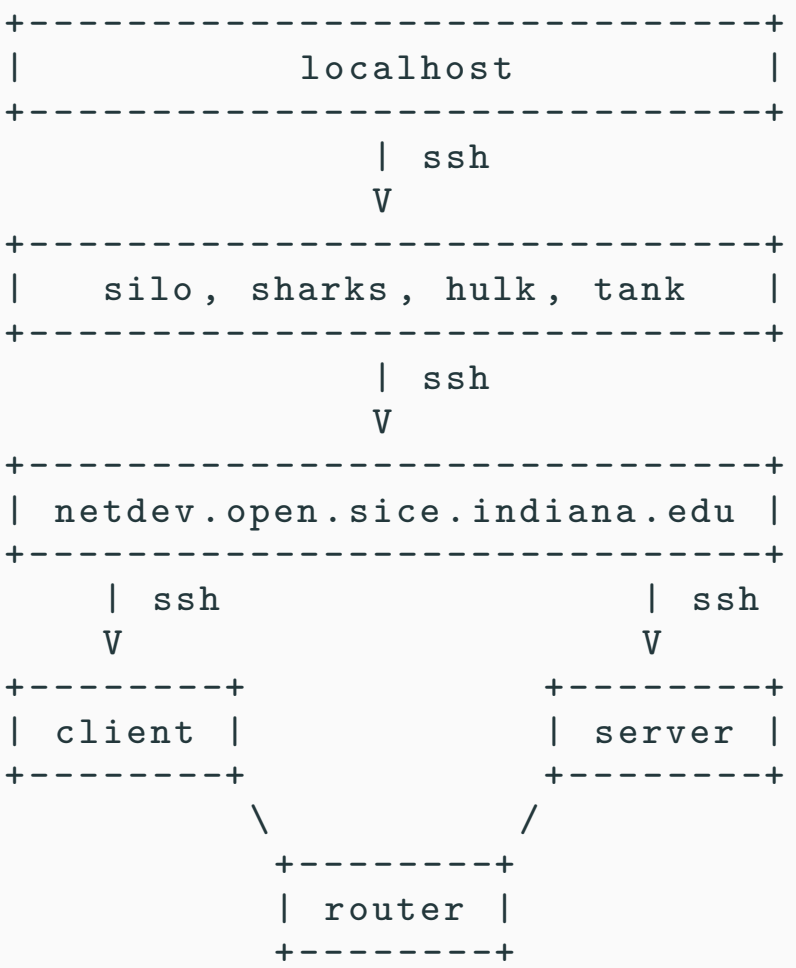
Takeaway 6: Never use freed memory

Assignment 5: How to add a header

```
// Declare header structure
typedef struct {
  int a;
  int b;
} header_t;

// Initialize a header
header_t header = {
  .a = 42,
  .b = 43
};

char buf[256];
char data[32];

// Copy the header and then the data
memcpy(buf, &header, sizeof(header));
memcpy(buf + sizeof(header), data, sizeof(data));

// Send everything as one packet
send(sockfd, buf, sizeof(header) + sizeof(data), 0);
```



Testbed



1) ssh into one of IUs Linux systems

2) ssh into netdev

3) ssh into client/server separately

4) Run iperf3 between client/server

```
baratheon:~$ iperf3 -s
targaryen:~$ iperf3 -c server
```

Tips

- Generate an ssh key with ssh-keygen
- Upload your ssh public key with ssh-copy-id USER@HOST