Recitation 12: ProxyLab Part 1

Instructor: TA(s)
Outline

- Proxies
- Networking
- PXYDRIVE Demo
Proxy Lab

- Checkpoint is worth 1%, due Thursday, Nov. 29th
- Final is worth 7%, due Thursday, Dec 6th
- You may use at most one grace / late day for each phase
  - Last day to submit checkpoint: Friday, Nov. 30th
  - Last day to submit final: Friday, Dec 7th
  - There will be no extensions!

- You are submitting an entire project
  - Modify the makefile
  - Split source file into separate pieces

- Submit regularly to verify proxy builds on Autolab

- Your proxy is a server, it should not crash!
Why Proxies?

- Proxies are both clients and servers
- Can perform useful functions as requests and responses pass by
  - Examples: Caching, logging, anonymization, filtering, transcoding
2. Start client

Client

open_clientfd

Connection request

Await connection request from client

1. Start server

Server

open_listenfd

accept

3. Exchange data

fgets
rio_write
fputs
rio_readlineb
rio_writen
rif_readlineb

close
EOF

4. Disconnect client

5. Drop client
Transferring HTTP Data

If something requests a file from a web server, how does it know that the transfer is complete?

A) It reads a NULL byte.
B) The connection closes.
C) It reads a blank line.
D) The HTTP header specifies the number of bytes to receive.
E) The reading function receives EOF.
Introducing **PXYDRIVE**\(^1\)

- **A REPL for testing your proxy implementation**
  - We also grade using this
- **Typical pre-f18 proxy debugging experience:**
  - Open up three terminals:
    - for Tiny server, *gdb proxy* and *curl*
  - Can make multiple requests, but need more terminals for multiple instances of the Tiny server
  - If the data is corrupted, need to manually inspect lines of gibberish binary data to check error
- **Not anymore with PXYDRIVE!**

---

\(^1\) Not typing PXYDRIVE in small-caps is a style violation.
Introducing PXYDRIVE

- **General workflow**
  - Generate text and binary data to test your proxy with
  - Create (multiple) server
  - Make **transactions**
  - Trace transactions to inspect headers and response data

- **Transaction**
Some practice

- Get the tarball

- $ wget http://www.cs.cmu.edu/~213/activities/pxydrive-tutorial.tar
- $ tar -xvf pxydrive-tutorial.tar
- $ cd pxydrive-tutorial
Trying out PXYDRIVE

- It’s a REPL: the user can run commands

- $ ./pxy/pxydrive
  - Just start PXYDRIVE
  - Try entering commands:
    - >help
    - >help help help help help help help...
    - >quit

- $ ./pxy/pxydrive -p ./proxy-ref
  - Starts PXYDRIVE and specifies a proxy to run
  - Proxy set up at <someshark>:30104
  - Picks the right port and starts the proxy
  - ./proxy-ref is the reference proxy
PXYDRIVE Tutorial 1

- Introducing basic procedures:
  generate data, create server, fetch / request file from server, trace transaction

- Open s01-basic-fetch.cmd
PxyDrive Tutorial 1

- >generate data1.txt 1K
  - Generates a 1K text file called data1.txt
- >serve s1
  - Launches a server called s1
- >fetch f1 data1.txt s1
  - Fetches data1.txt from server s1, in a transaction called f1
- >wait *
  - Waits for all transactions to finish
  - Needed in the trace, not in the command-line
- >trace f1
  - Traces the transaction f1
- >check f1
  - Checks the transaction f1
PXYDRIVE Tutorial 1

- Run trace with \(-f\) option:

- $ ./pxy/pxydrive \(-p\) ./proxy-ref
  \(-f\) s01-basic-fetch.cmd
Look at the trace of the transaction!

- Identify:
  - GET command
  - Host header
  - Other headers
  - Request from client to proxy
  - Request from proxy to server
  - Response by server to proxy
  - Response by proxy to client
PXYDRIVE Tutorial 1

- Run a different trace

- $ ./pxy/pxydrive -p ./proxy-ref -f s02-basic-request.cmd

- You should get a different output from the first trace
- Why? Let’s look at this trace...
PxyDrive Tutorial 1

- >generate data1.txt 1K
- >serve s1
- >request r1 data1.txt s1
  - Requests data1.txt from server s1, in a transaction called r1
- >wait *
- >trace r1
- >respond r1
  - Allow server to respond to the transaction r1
- >wait *
- >trace r1
- >check f1
  - Checks the transaction f1
PxyDrive Tutorial 1

- The fetch command makes the server immediately respond to a request.
- All steps of a transaction is complete after a fetch.

- The request command does not complete a transaction.
- A request needs a respond to complete its transaction.
PXYDrive Tutorial 2

- Debugging a proxy that clobbers responses
- Run the same trace but with a faulty proxy

$ ./pxy/pxydrive -f s01-basic-fetch.cmd -p ./proxy-corrupt
What went wrong?

```
Response status: ok
Source file in ./source_files/random/data1.txt
Request status: error (Mismatch between source file ./source_files/random/data1.txt and response file ./response_files/f1-data1.txt starting at position 447: 'F' (hex 0x46) ≠ 'G' (hex 0x47))
Result file in ./response_files/f1-data1.txt
>
> Make sure it was retrieved properly
> check f1

ERROR: Request f1 generated status 'error'. Expecting 'ok' (Mismatch between source file ./source_files/random/data1.txt and response file ./response_files/f1-data1.txt starting at position 447: 'F' (hex 0x46) ≠ 'G' (hex 0x47))

>quit
ERROR COUNT = 1
-bash-4.2$
```
PxyDrive Tutorial 3

- Debugging a proxy that clobbers headers
- Run the same trace but with another faulty proxy

$ ./pxy/pxydrive -f s01-basic-fetch.cmd -p ./proxy-strip -S 3

- -S specifies strictness level
What went wrong?

Response status: bad_request (Missing Request-ID header)
Source file in ./source_files/random/data1.txt
Request status: bad_request (Bad request)
Result file in ./response_files/f1-status.html

> Make sure it was retrieved properly
> check f1
ERROR: Request f1 generated status 'bad_request'. Expecting 'ok' (Bad request)
> quit
ERROR COUNT = 1
-bash-4.2$ _
PxyDrive Tutorial 4

- Debugging a proxy that crashes
- Run the same trace but with yet another faulty proxy

$ ./pxy/pxydrive -f s03-overrun.cmd
   -p ./proxy-overrun

- Is the error message helpful?
We resort to multi-window debugging
Set up another window and run GDB in one:

$ gdb ./proxy-overrun
(gdb) run 15213

In the other window, run PXYDRIVE:

$ ./pxy/pxydrive.py -P localhost:XXXXX
   -f s03-overrun.cmd
   -P specifies the host and port the proxy is running on
Reminders

- Read the writeup
- Start early
  - Remember, only one grace / late day per phase
  - Come to office hours this week, before it gets crowded!
- Work incrementally and take breaks
Appendix on echoserver / client
Echoserver, echoclient
Echo Demo

- See the instructions written in the telnet results to set up the echo server. Get someone nearby to connect using the echo client.

- What does echoserver output? (Sample output:)

```bash
$ ./echoserver 10101
Accepted connection from hammerheadshark.ics.cs.cmu.edu:46422
hammerheadshark.ics.cs.cmu.edu:46422 sent 6 bytes
Disconnected from hammerheadshark.ics.cs.cmu.edu:46422
```
Echo Demo

- **Look at echoclient.c**
  - Opens a connection to the server
  - Reads/writes from the server

- **Look at echoserver output**
  - Why is the printed client port different from the server’s listening port?
  - Server opens **one “listening” port**
    - Incoming clients connect to this port
  - Once server **accepts** a connection, it talks to client on a different **“ephemeral” port**
Echo Demo

- Try to connect two clients to the same server.
- What happens?
  - Second client has to wait for first client to finish!
  - Server doesn’t even accept second client’s connection
  - Where/why are we getting stuck?
- Because we’re stuck in echo() talking to the first client, echoserver can’t handle any more clients
- Solution: multi-threading
Echo Server Multithreaded

How might we make this server multithreaded?

(Don’t look at echoserver_t.c)

while (1) {
    // Allocate space on the stack for client info
    client_info client_data;
    client_info *client = &client_data;

    // Initialize the length of the address
    client->addrlen = sizeof(client->addr);

    // Accept() will block until a client connects to the port
    client->connfd = Accept(listenfd,
                            (SA *) &client->addr, &client->addrlen);

    // Connection is established; echo to client
    echo(client);
}

Echo Server Multithreaded

- `echoserver_t.c` isn’t too different from `echoserver.c`
  - To see the changes: `diff echoserver.c echoserver_t.c`
- Making your proxy multithreaded will be very similar
- However, don’t underestimate the difficulty of addressing race conditions between threads!
  - Definitely the hardest part of proxylab
  - More on this next time...