SOFSEC SimpleWebServer Lab

by Paul Herz

Questions

3. (Book question 7 a-c, pp. 78-79):
Rewrite the `serveFile()` method such that it imposes a maximum file size limit. If a user attempts to download a file that is larger than the maximum allowed size, write a log entry to a file called `error_log` and return a “403 Forbidden” HTTP response code.

> **See Snippets 5-6 in the Appendix below for implementation of file size limits and error logging.**

3a. What happens if an attacker tries to download `/dev/random` after you have made your modification?

> *They get a "403 Forbidden": I designed my length checker to stop early if the file size read in thus far exceeded the MAX_FILE_SIZE constant I set (to 5MB). This prevents lockups. It would be smart, however, to disallow the server from serving "special files" (aka devices or file sockets).*

3b. What might be some alternative ways in which to implement the maximum file size limit?

> *I could check the size of the buffer less frequently to avoid the large overhead it probably imposes upon the reading process, like checking every kilobyte or so, and then invariably checking at the end of reading to make sure filesize is checked at least once.*
>
> *I could also rely on platform-dependent filesystem metadata for file size information, as filesystems often store file size in their databases. I don't know how to do this in Java, but it is possible, and it avoids having to read the file up to the size limit in order to know whether the file is too big.*
>
> *Checking if a file is special/device like /dev/random would obviate the need to continuously check file size, however, and then we could just check it at the end without worrying about infinite reading.*

3c. [OPTIONAL] Implement multithreading and a mechanism that allows a maximum number of concurrent downloads for a particular IP address.

> *(skipping)*

4. (Book question 9 a-c, p. 79):
Implement basic HTTP authorization for `SimpleWebServer`. Read the HTTP 1.0 specification for more details on how basic HTTP authorization works.

4a. Instrument `SimpleWebServer` to store a username and password as data members. Require that any HTTP request to the web server be authorized by checking for an authorization HTTP header with a base64-encoded username and password. Requests that do not contain an authorization header should receive a `WWW-Authentication` challenge. Requests that do contain an authorization header should be authenticated against the username and password hard-coded in the `SimpleWebServer` class.

> **See Snippets 2-4 and Image 1 in the Appendix below.**
> *For more thoughts about my implementation, read the comments in the full source code (link in the Appendix).*

4b. Pretend that you are an attacker who got ahold of the compiled SimpleWebServer. class file. Run the `strings` utility on the compiled `SimpleWebServer.class` file to reveal the username and password that your modified web server requires. (If you are running a UNIX-based system, the strings utility is most likely preinstalled on your system. If you are running Windows, you can obtain the strings utility from www.sysinternals.com/Utilities/Strings.html).

> **See Snippet 10 in the Appendix below for output.**
> I ran "`strings - SimpleWebServer.class`" (the hyphen is a Mac quirk, I discovered).

4c. Install [WireShark] and a base64 decoder on your system. Make a few HTTP requests to your web server in which you use your username and password to authenticate. Use [WireShark] to capture network traffic that is exchanged between your web client and server. Use the base64 decoder to convert the encoded username and password in the [WireShark] logs to plain text.

> **See Images 2-5 in the Appendix below for screenshots of Wireshark capture and decoded credentials.**

Appendix

This file is available at https://github.com/phrz/SofSecSimpleWebServer/blob/master/SimpleWebServer.java. Snippets have been included in lieu of screenshots, which are less accessible. I refactored a lot of the original code for better understanding (removing one-letter variables) but only include entirely new parts. The program was written to run on a Java 10 compatible installation.

NOTA BENE: There are many comments poring over the details of the HTTP specification and places where I know I am not compliant due to the extreme simplicity of the program (printing directly to the response buffer rather than formatting a response), but some have been removed to fit code on the page.

SNIPPET 1: NEW CONSTANTS

```java
// in bytes (5 MB)
private static final int MAX_FILE_SIZE = 5_000_000;

private static final String USERNAME = "admin";
private static final String PASSWORD = "letmein";
```

SNIPPET 2: HEADER PARSING IN PROCESSREQUEST()

```java
String line = null;
var headers = new HashMap<String, String>();

while((line = inputReader.readLine()) != null) {
	if(line.isEmpty()) { break; }

	int colonLocation = line.indexOf(':');
	if(colonLocation == -1) {
		logError("Malformed header ... [ommited]");
		statusCode(clientConnection, 400); // Bad Request
		endHeaders(clientConnection);
		return;
	}
	String headerName = line.substring(0, colonLocation);
	String headerField = line.substring(colonLocation + 1, line.length()).trim();
	headers.put(headerName, headerField);
}
```

SNIPPET 3: NEW REQUEST HANDLING IN PROCESSREQUEST()

```java
if(!command.equals("GET")) {
	statusCode(clientConnection, 501); // Not Implemented
	endHeaders(clientConnection);
} else if(!checkBasicAuthentication(headers)) {
	statusCode(clientConnection, 401); // Unauthorized
	writeHeader(clientConnection, "WWW-Authenticate", "Basic
realm=\"DefaultRealm\"");
	endHeaders(clientConnection);
} else {
	serveFile(clientConnection, pathName);
}
```

SNIPPET 4: BASIC AUTHENTICATION VALIDATION

```
public Boolean checkBasicAuthentication(Map<String,String> headers) {
        String authorizationField = headers.get("Authorization");
        if(authorizationField == null) { return false; }

        String[] parts = authorizationField.split(" ");
        if(parts.length != 2) { return false; }
        if(!parts[0].equals("Basic")) { return false; }

        byte[] authenticationCookieBytes = null;
        try {
                authenticationCookieBytes = Base64.getDecoder().decode(parts[1]);
        } catch(IllegalArgumentException e) {
                logError("Invalid base64 in Authorization: Basic header.");
                return false;
        }
        String authenticationCookie = new String(authenticationCookieBytes);
                int colonLocation = authenticationCookie.indexOf(":");
        if(colonLocation == -1) {
                logError("Invalid authentication cookie: expected colon.");
                return false;
        }

        String givenUsername = authenticationCookie.substring(0, colonLocation);
        String givenPassword = authenticationCookie.substring(
                colonLocation + 1,
                authenticationCookie.length()
        );
        return givenUsername.equals(USERNAME) && givenPassword.equals(PASSWORD);
}
```

SNIPPET 5: FILE SIZE LIMIT IN SERVEFILE()

```
while(character != endOfFile) {
        buffer.append((char) character);
        character = fileReader.read();

        if(buffer.length() > MAX_FILE_SIZE) {
                statusCode(clientConnection, 403); // Forbidden
                endHeaders(clientConnection);
                logError("Attempted access to file larger ... [omitted]");
                return;
        }
}

statusCode(clientConnection, 200); // OK
endHeaders(clientConnection);
```

SNIPPET 6: ERROR LOGGING

```
public void logError(String message) {
        // ISO 8601 timestamp
        String timestamp = DateTimeFormatter
                .ofPattern("yyyy-MM-dd'T'HH:mmX")
                .withZone(ZoneOffset.UTC)
                .format(Instant.now());

        String logMessage = "[Error] " + timestamp + " - " + message;
        System.out.print(logMessage);

        try(Writer writer = new BufferedWriter(new FileWriter("error_log"))) {
                writer.write(logMessage + "\n");
        } catch(IOException e) {
                System.out.println("... [omitted]");
        }
}
```

SNIPPET 7: COMPILING AND RUNNING

```
bash-3.2$ make
javac SimpleWebServer.java
java SimpleWebServer
Starting web server...
Listening on :8080
GET / HTTP/1.1
GET / HTTP/1.1
```

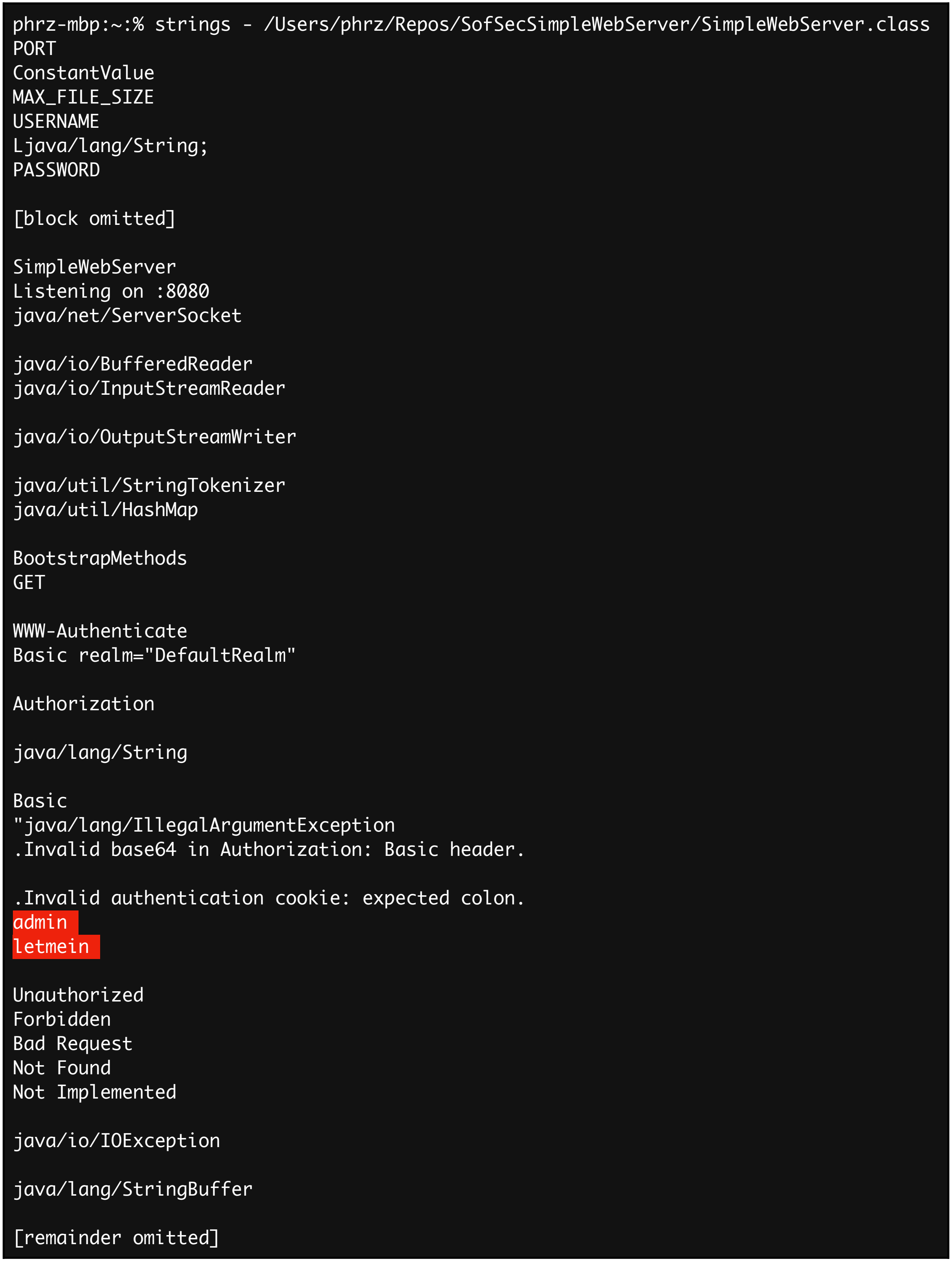
SNIPPET 8: ERROR_LOG FILE CONTENTS

```
[Error] 2018-04-02T04:16Z - Attempted access to file larger than MAX_FILE_SIZE
(largefile)
```

SNIPPET 9: HTTP RESPONSE (USING HTTPIE HTTP CLIENT)

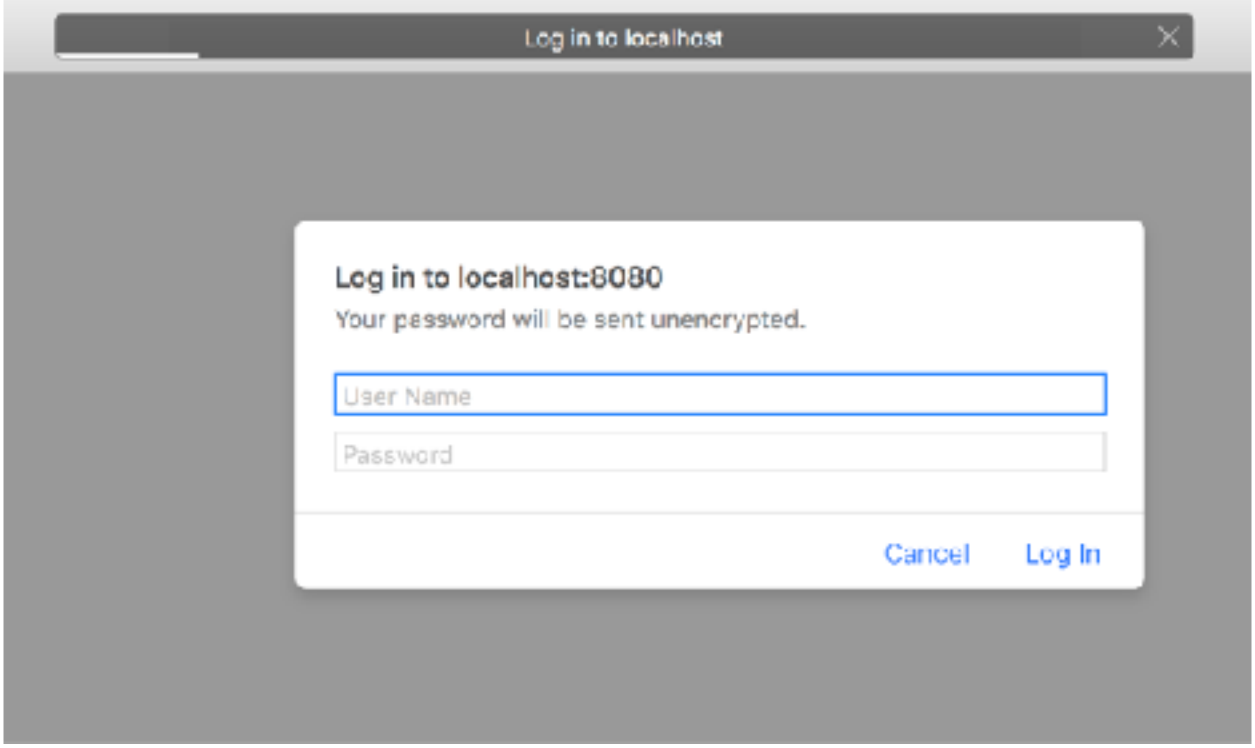
```
bash-3.2$ http GET localhost:8080/
HTTP/1.0 401 Unauthorized
WWW-Authenticate: Basic realm="DefaultRealm"
```

SNIPPET 10: STRINGS UTILITY OUTPUT (CREDENTIALS HIGHLIGHTED)



```
phrz-mbp:~:% strings - /Users/phrz/Repos/SofSecSimpleWebServer/SimpleWebServer.class
PORT
ConstantValue
MAX_FILE_SIZE
USERNAME
Ljava/lang/String;
PASSWORD

[block omitted]

SimpleWebServer
Listening on :8080
java/net/ServerSocket

java/io/BufferedReader
java/io/InputStreamReader

java/io/OutputStreamWriter

java/util/StringTokenizer
java/util/HashMap

BootstrapMethods
GET

WWW-Authenticate
Basic realm="DefaultRealm"

Authorization

java/lang/String

Basic
"java/lang/IllegalArgumentException
.Invalid base64 in Authorization: Basic header.

.Invalid authentication cookie: expected colon.
admin
letmein

Unauthorized
Forbidden
Bad Request
Not Found
Not Implemented

java/io/IOException

java/lang/StringBuffer

[remainder omitted]
```

IMAGE 1: DEMONSTRATION OF HTTP BASIC AUTHENTICATION FROM MY SERVER IN A GRAPHICAL WEB BROWSER

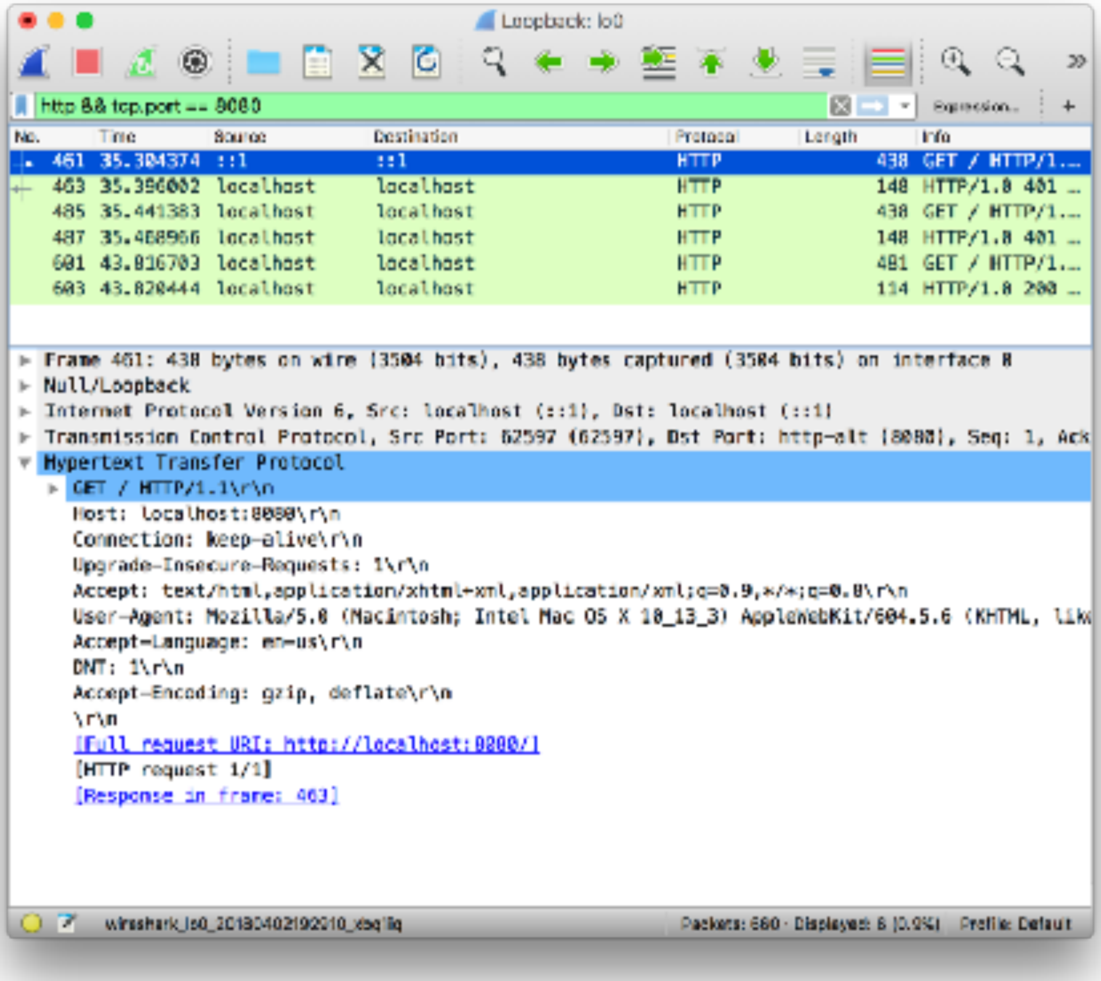


Log in to localhost

Log in to localhost:8080

Your password will be sent unencrypted.

IMAGE 2: WIRESHARK, THE INITIAL BROWSER REQUEST



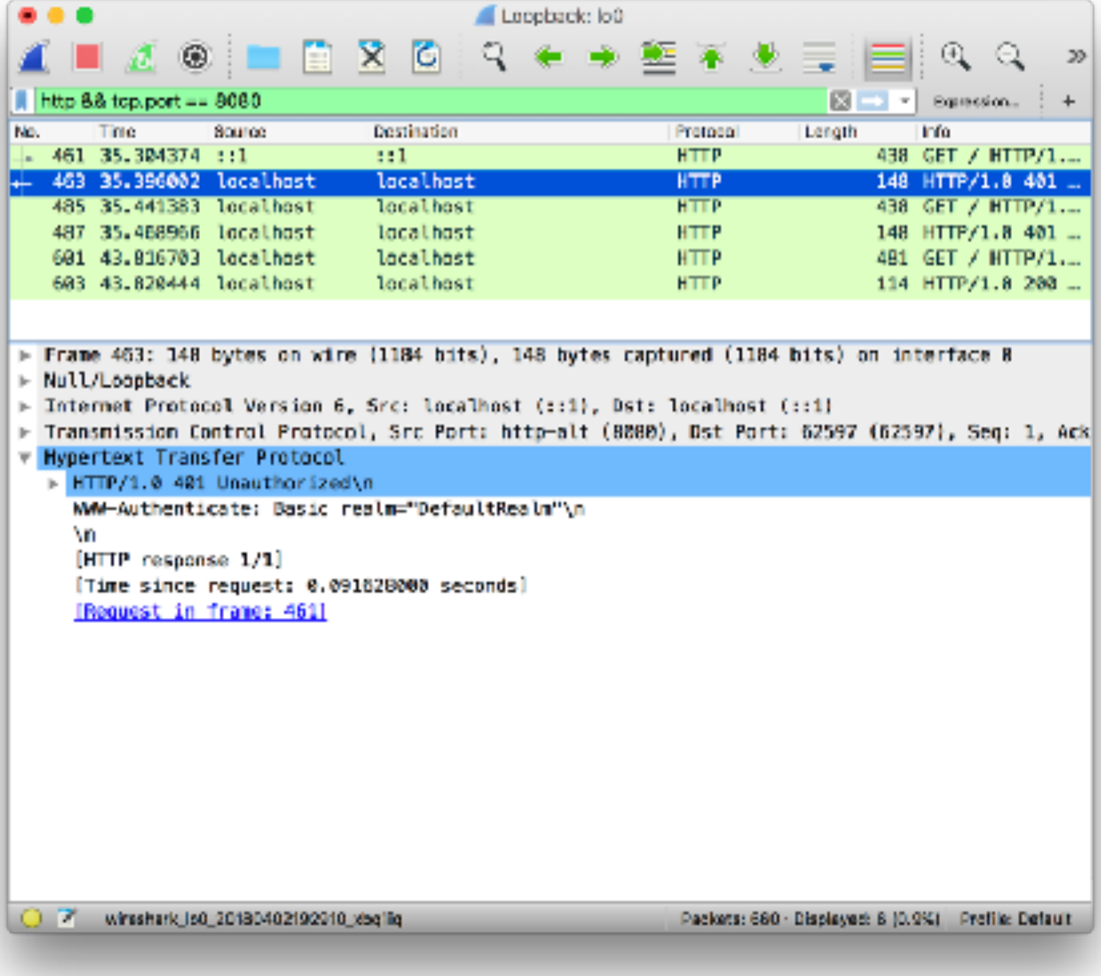
http && tcp.port == 8080

No.	Time	Source	Destination	Protocol	Length	Info
461	35.304374	::1	::1	HTTP	438	GET / HTTP/1.…
463	35.396002	localhost	localhost	HTTP	148	HTTP/1.0 401 …
485	35.441383	localhost	localhost	HTTP	438	GET / HTTP/1.…
487	35.468966	localhost	localhost	HTTP	148	HTTP/1.0 401 …
601	43.016703	localhost	localhost	HTTP	401	GET / HTTP/1.…
603	43.020444	localhost	localhost	HTTP	114	HTTP/1.0 200 …

```
▶ Frame 461: 438 bytes on wire (3504 bits), 438 bytes captured (3504 bits) on interface 0
▶ Null/Loopback
▶ Internet Protocol Version 6, Src: localhost (::1), Dst: localhost (::1)
▶ Transmission Control Protocol, Src Port: 62597 (62597), Dst Port: http-alt (8080), Seq: 1, Ack
▼ Hypertext Transfer Protocol
  ▶ GET / HTTP/1.1\r\n
    Host: localhost:8080\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3) AppleWebKit/604.5.6 (KHTML, like
    Accept-Language: en-us\r\n
    DNT: 1\r\n
    Accept-Encoding: gzip, deflate\r\n
    \r\n
    [Full request URI: http://localhost:8080/]
    [HTTP request 1/1]
    [Response in frame: 463]
```

wireshark_lo0_20180402192010_xbq1lq Packets: 660 · Displayed: 6 (0.9%) Profile: Default

IMAGE 3: WIRESHARK, 401 UNAUTHORIZED RESPONSE FROM SIMPLEWEBSERVER



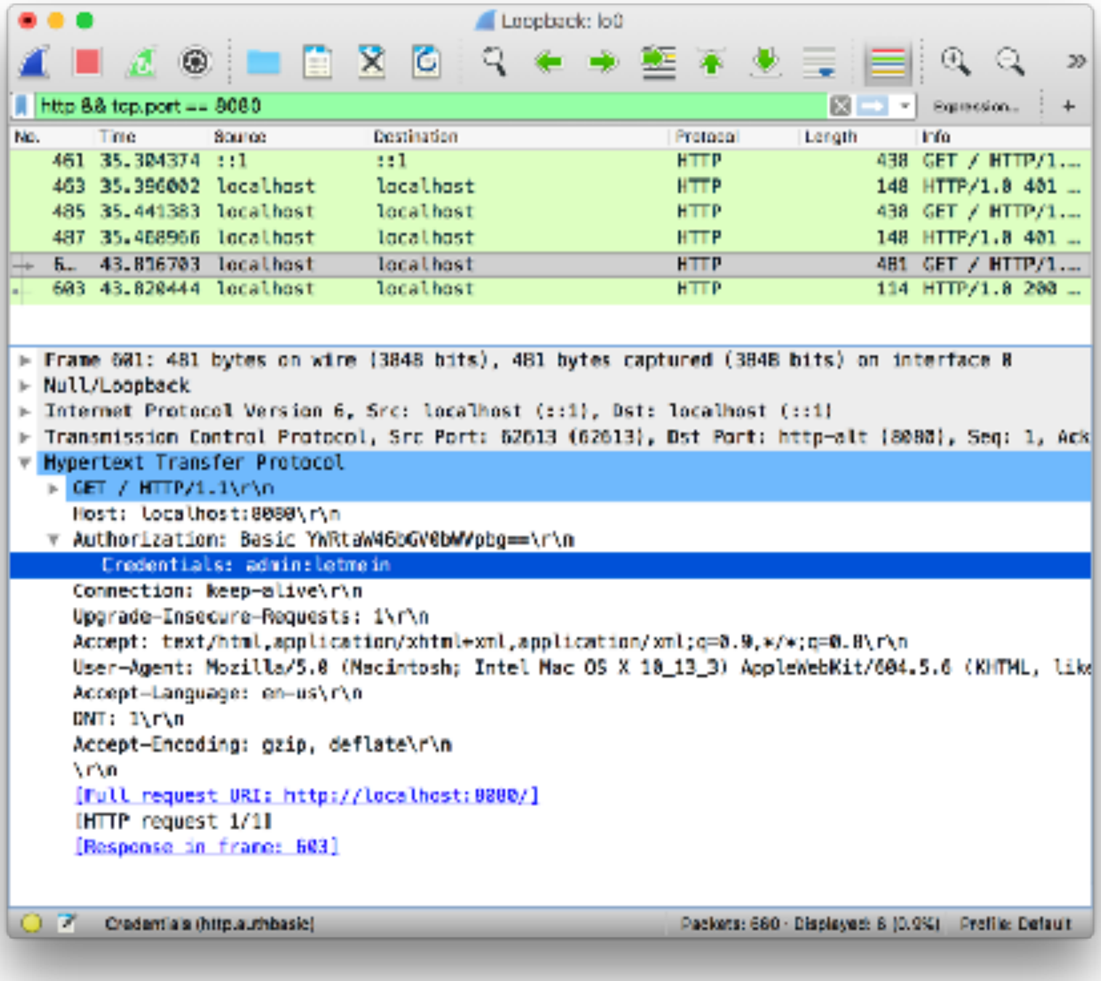
http && tcp.port == 8080

No.	Time	Source	Destination	Protocol	Length	Info
461	35.304374	::1	::1	HTTP	438	GET / HTTP/1.…
463	35.396002	localhost	localhost	HTTP	148	HTTP/1.0 401 …
485	35.441383	localhost	localhost	HTTP	438	GET / HTTP/1.…
487	35.488966	localhost	localhost	HTTP	148	HTTP/1.0 401 …
601	43.816703	localhost	localhost	HTTP	481	GET / HTTP/1.…
603	43.820444	localhost	localhost	HTTP	114	HTTP/1.0 200 …

```
▶ Frame 463: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface 0
▶ Null/Loopback
▶ Internet Protocol Version 6, Src: localhost (::1), Dst: localhost (::1)
▶ Transmission Control Protocol, Src Port: http-alt (8080), Dst Port: 62597 (62597), Seq: 1, Ack
▼ Hypertext Transfer Protocol
  ▶ HTTP/1.0 401 Unauthorized\n
    WWW-Authenticate: Basic realm="DefaultRealm"\n
    \n
    [HTTP response 1/1]
    [Time since request: 0.091820000 seconds]
    [Request in frame: 461]
```

wireshark_lo0_20180402192010_xbq1lq   Packets: 660 · Displayed: 6 (0.9%)   Profile: Default

IMAGE 4: WIRESHARK, HTTP BASIC AUTHORIZATION RESPONSE WITH CREDENTIALS FROM BROWSER



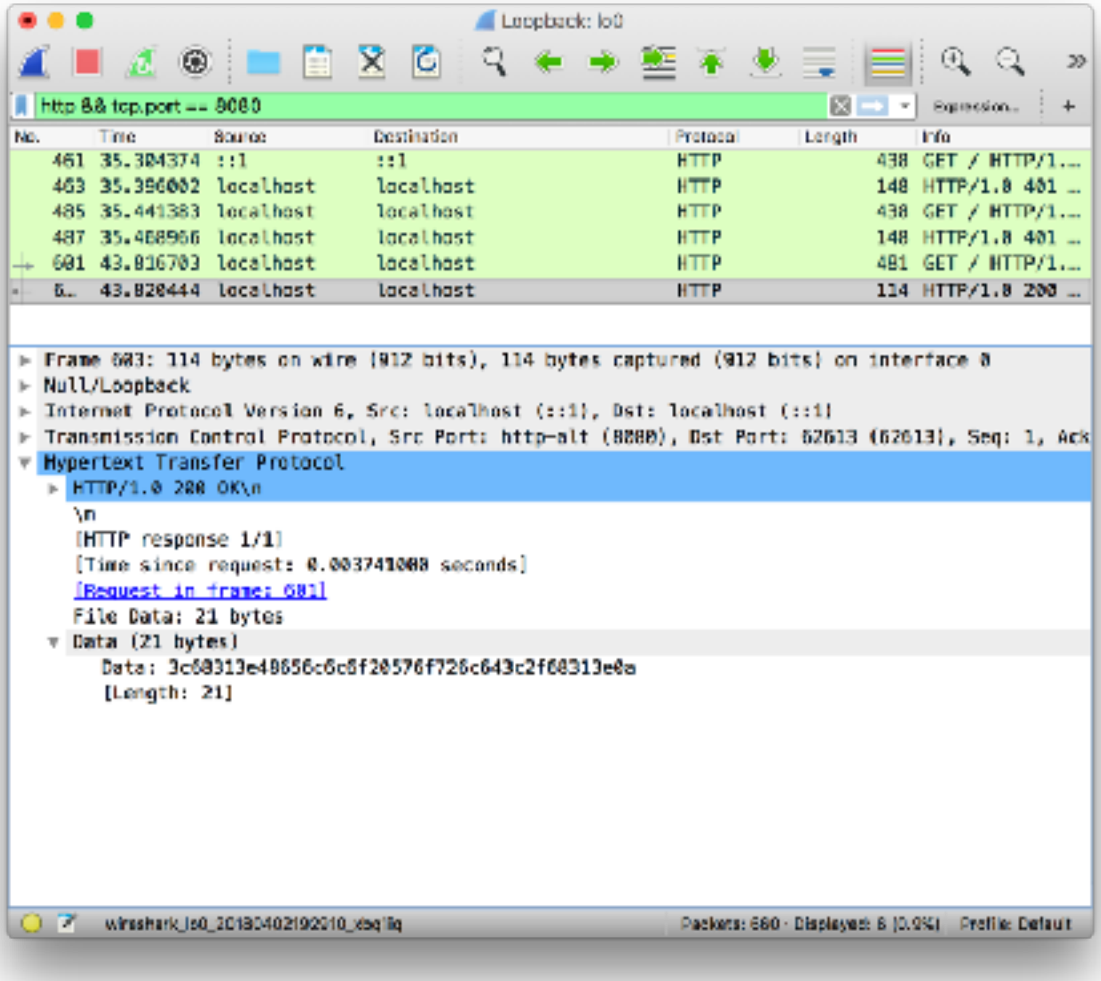
```
http && tcp.port == 8080
```

```
No.  Time       Source     Destination  Protocol  Length Info
 461 35.304374  ::1        ::1          HTTP         438 GET / HTTP/1...
 463 35.396002  localhost  localhost    HTTP         148 HTTP/1.0 401 ...
 485 35.441383  localhost  localhost    HTTP         438 GET / HTTP/1...
 487 35.468966  localhost  localhost    HTTP         148 HTTP/1.0 401 ...
 6.. 43.816703  localhost  localhost    HTTP         481 GET / HTTP/1...
 603 43.820444  localhost  localhost    HTTP         114 HTTP/1.0 200 ...
```

```
▶ Frame 601: 481 bytes on wire (3848 bits), 481 bytes captured (3848 bits) on interface 0
▶ Null/Loopback
▶ Internet Protocol Version 6, Src: localhost (::1), Dst: localhost (::1)
▶ Transmission Control Protocol, Src Port: 62613 (62613), Dst Port: http-alt (8080), Seq: 1, Ack
▼ Hypertext Transfer Protocol
  ▶ GET / HTTP/1.1\r\n
    Host: localhost:8080\r\n
  ▼ Authorization: Basic YWRtaW46bGV0bWVpbg==\r\n
        Credentials: admin:letmein
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3) AppleWebKit/604.5.6 (KHTML, like
    Accept-Language: en-us\r\n
    DNT: 1\r\n
    Accept-Encoding: gzip, deflate\r\n
    \r\n
    [Full request URI: http://localhost:8080/]
    [HTTP request 1/1]
    [Response in frame: 603]
```

```
Credentials (http.authbasic)        Packets: 660 · Displayed: 6 (0.9%)   Profile: Default
```

IMAGE 5: WIRESHARK, FINAL 200 OK RESPONSE FROM SIMPLEWEBSERVER WITH PROTECTED CONTENT



http && tcp.port == 8080

No.	Time	Source	Destination	Protocol	Length	Info
461	35.304374	::1	::1	HTTP	438	GET / HTTP/1.…
463	35.306002	localhost	localhost	HTTP	148	HTTP/1.0 401 …
485	35.441383	localhost	localhost	HTTP	438	GET / HTTP/1.…
487	35.468966	localhost	localhost	HTTP	148	HTTP/1.0 401 …
601	43.816703	localhost	localhost	HTTP	481	GET / HTTP/1.…
6…	43.820444	localhost	localhost	HTTP	114	HTTP/1.0 200 …

```
▶ Frame 603: 114 bytes on wire (912 bits), 114 bytes captured (912 bits) on interface 0
▶ Null/Loopback
▶ Internet Protocol Version 6, Src: localhost (::1), Dst: localhost (::1)
▶ Transmission Control Protocol, Src Port: http-alt (8080), Dst Port: 62613 (62613), Seq: 1, Ack
▼ Hypertext Transfer Protocol
  ▶ HTTP/1.0 200 OK\n
    \n
    [HTTP response 1/1]
    [Time since request: 0.003741000 seconds]
    [Request in frame: 601]
    File Data: 21 bytes
  ▼ Data (21 bytes)
      Data: 3c68313e48656c6c6f20576f726c643c2f68313e0a
      [Length: 21]
```

wireshark_lo0_20180402192010_xbq1iq   Packets: 660 · Displayed: 6 (0.9%)   Profile: Default