# ECE 443 - Homework #6

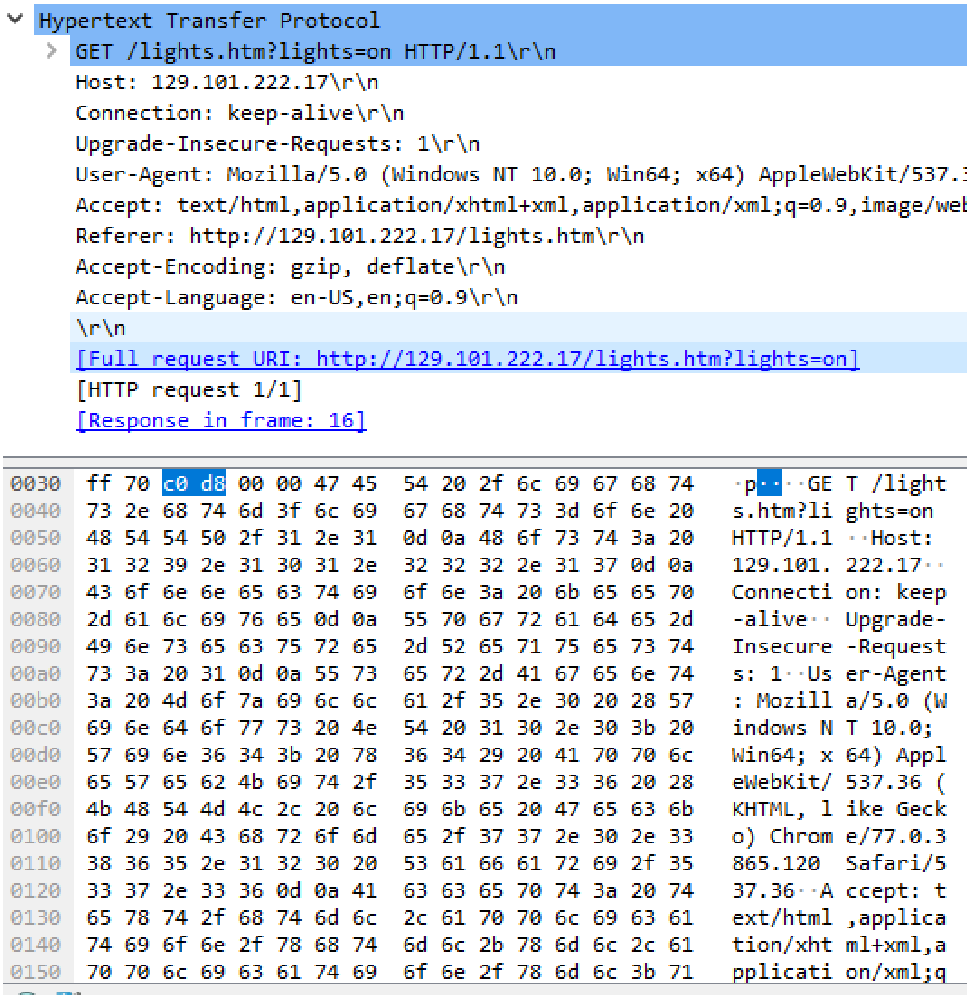
Collin Heist

11th November 2019

# 1 Observations

## 1.1 GET Method

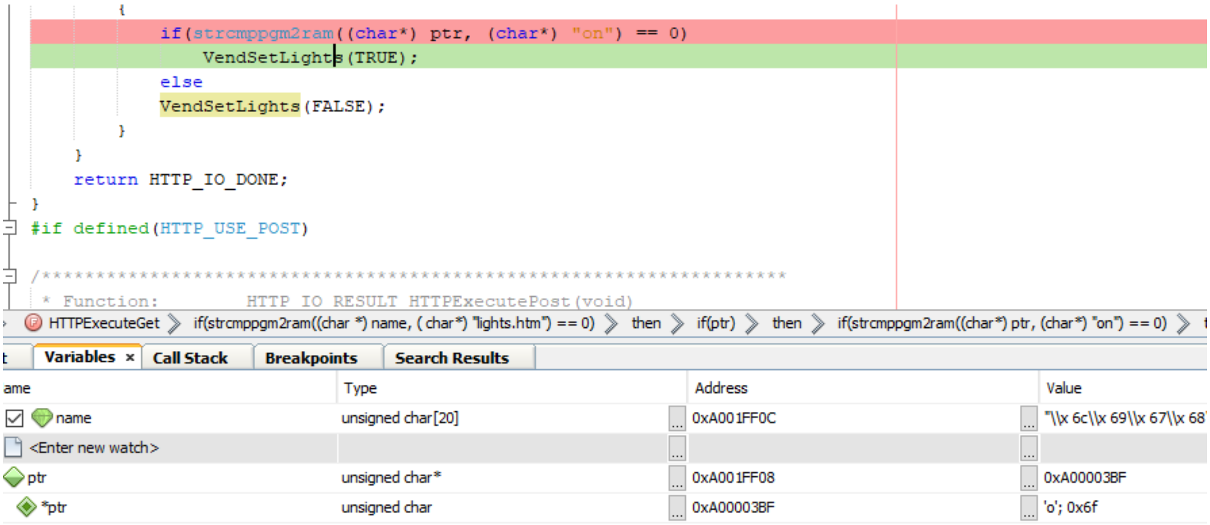
Wireshark shows the TCP packet sent with a new light setting. You can see in the **Full request URI** part of the HTTP message that the name and value pair are appended to the address. In this case, it is **lights=on**.



```
Hypertext Transfer Protocol
  GET /lights.htm?lights=on HTTP/1.1\r\n
  Host: 129.101.222.17\r\n
  Connection: keep-alive\r\n
  Upgrade-Insecure-Requests: 1\r\n
  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/we
  Referer: http://129.101.222.17/lights.htm\r\n
  Accept-Encoding: gzip, deflate\r\n
  Accept-Language: en-US,en;q=0.9\r\n
  \r\n
  [Full request URI: http://129.101.222.17/lights.htm?lights=on]
  [HTTP request 1/1]
  [Response in frame: 16]

0030  ff 70 c0 d8 00 00 47 45  54 20 2f 6c 69 67 68 74   ·p····GE T /light
0040  73 2e 68 74 6d 3f 6c 69  67 68 74 73 3d 6f 6e 20   s.htm?li ghts=on
0050  48 54 54 50 2f 31 2e 31  0d 0a 48 6f 73 74 3a 20   HTTP/1.1 ··Host:
0060  31 32 39 2e 31 30 31 2e  32 32 32 2e 31 37 0d 0a   129.101. 222.17··
0070  43 6f 6e 6e 65 63 74 69  6f 6e 3a 20 6b 65 65 70   Connecti on: keep
0080  2d 61 6c 69 76 65 0d 0a  55 70 67 72 61 64 65 2d   -alive·· Upgrade-
0090  49 6e 73 65 63 75 72 65  2d 52 65 71 75 65 73 74   Insecure -Request
00a0  73 3a 20 31 0d 0a 55 73  65 72 2d 41 67 65 6e 74   s: 1··Us er-Agent
00b0  3a 20 4d 6f 7a 69 6c 6c  61 2f 35 2e 30 20 28 57   : Mozill a/5.0 (W
00c0  69 6e 64 6f 77 73 20 4e  54 20 31 30 2e 30 3b 20   indows N T 10.0;
00d0  57 69 6e 36 34 3b 20 78  36 34 29 20 41 70 70 6c   Win64; x 64) Appl
00e0  65 57 65 62 4b 69 74 2f  35 33 37 2e 33 36 20 28   eWebKit/ 537.36 (
00f0  4b 48 54 4d 4c 2c 20 6c  69 6b 65 20 47 65 63 6b   KHTML, l ike Geck
0100  6f 29 20 43 68 72 6f 6d  65 2f 37 37 2e 30 2e 33   o) Chrom e/77.0.3
0110  38 36 35 2e 31 32 30 20  53 61 66 61 72 69 2f 35   865.120  Safari/5
0120  33 37 2e 33 36 0d 0a 41  63 63 65 70 74 3a 20 74   37.36··A ccept: t
0130  65 78 74 2f 68 74 6d 6c  2c 61 70 70 6c 69 63 61   ext/html ,applica
0140  74 69 6f 6e 2f 78 68 74  6d 6c 2b 78 6d 6c 2c 61   tion/xht ml+xml,a
0150  70 70 6c 69 63 61 74 69  6f 6e 2f 78 6d 6c 3b 71   pplicati on/xml;q
```

Figure 1: Wireshark screenshot of the HTTP portion of the TCP packet.

Inside the **HTTPExecuteGet()** function, the pointer **ptr** parses this TCP packet to get *just* the value of the light setting, and so the pointer specifically points to the first character in the string “on”.



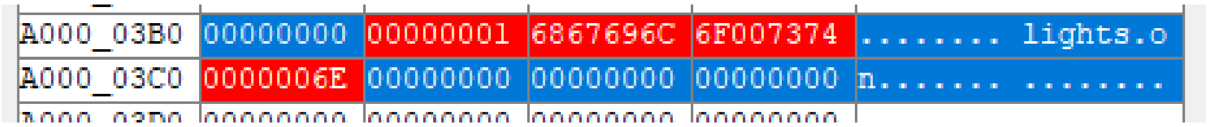
```
        {
            if(strcmppgm2ram((char*) ptr, (char*) "on") == 0)
                VendSetLights(TRUE);
            else
            VendSetLights(FALSE);
        }
    }
    return HTTP_IO_DONE;
}
#if defined(HTTP_USE_POST)

/*********************************************************************
 * Function:        HTTP_IO_RESULT HTTPExecutePost(void)
```

HTTPExecuteGet
if(strcmppgm2ram((char *) name, ( char*) "lights.htm") == 0)
then
if(ptr)
then
if(strcmppgm2ram((char*) ptr, (char*) "on") == 0)
Variables ×
Call Stack
Breakpoints
Search Results

| ame | Type | Address | Value |
|---|---|---|---|
| ☑ name | unsigned char[20] | 0xA001FF0C | "\\x 6c\\x 69\\x 67\\x 68 |
| <Enter new watch> | | | |
| ptr | unsigned char* | 0xA001FF08 | 0xA00003BF |
| *ptr | unsigned char | 0xA00003BF | 'o'; 0x6f |

Figure 2: MPLab Variable Window showing the value of the pointer for the name / value pair.

This pointer itself is stored at **0xA001FF08**, while it points to the memory addres **0xA00003BF**. Looking at that memory location in the MPLAB Memory View shows the rest of that string. It also shows (in space) the ignored name of the TCP pair (lights). This is in **Figure 3**.

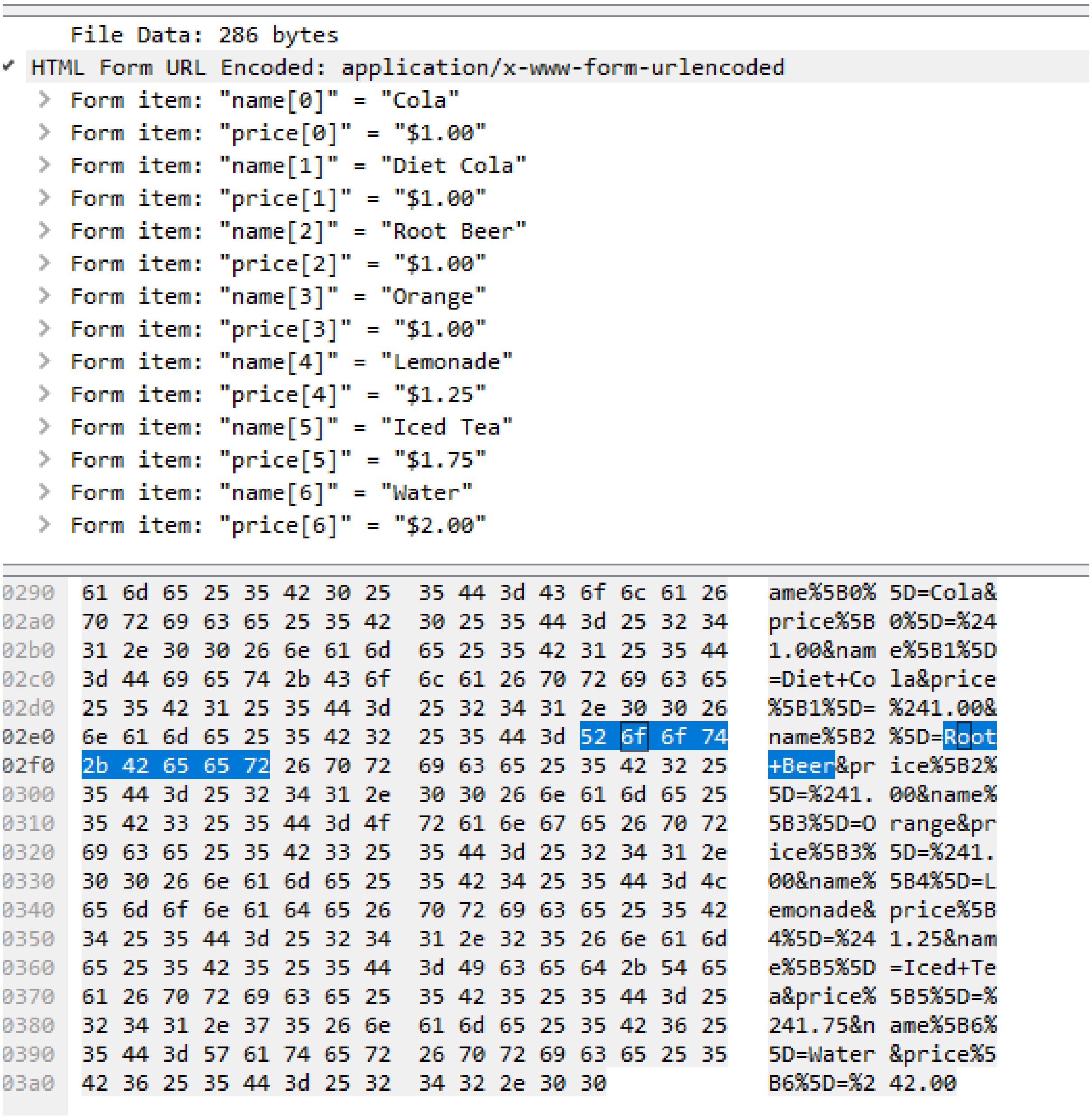


```
A000_03B0  00000000  00000001  6867696C  6F007374  ........ lights.o
A000_03C0  0000006E  00000000  00000000  00000000  n....... ........
A000_03D0  00000000  00000000  00000000  00000000
```

Figure 3: MPLab Memory View of the location to which **ptr** points to – the value of the name sent through the TCP packet.

## 1.2 POST Method

The POST method sends a lot more data in its TCP packet. Part of this data is shown below, however given there is no real limit to the data-length, not all of it is shown.

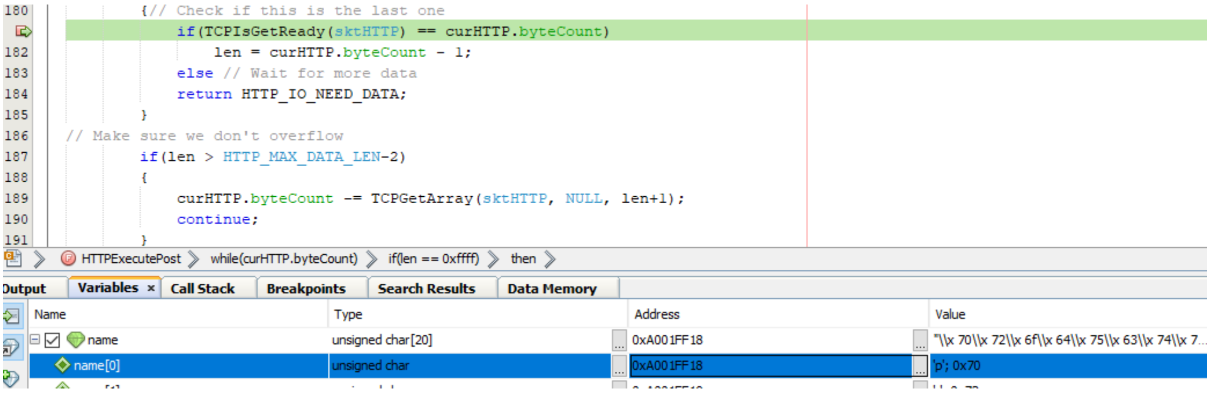


```
    File Data: 286 bytes
✔ HTML Form URL Encoded: application/x-www-form-urlencoded
  › Form item: "name[0]" = "Cola"
  › Form item: "price[0]" = "$1.00"
  › Form item: "name[1]" = "Diet Cola"
  › Form item: "price[1]" = "$1.00"
  › Form item: "name[2]" = "Root Beer"
  › Form item: "price[2]" = "$1.00"
  › Form item: "name[3]" = "Orange"
  › Form item: "price[3]" = "$1.00"
  › Form item: "name[4]" = "Lemonade"
  › Form item: "price[4]" = "$1.25"
  › Form item: "name[5]" = "Iced Tea"
  › Form item: "price[5]" = "$1.75"
  › Form item: "name[6]" = "Water"
  › Form item: "price[6]" = "$2.00"
```

```
0290  61 6d 65 25 35 42 30 25  35 44 3d 43 6f 6c 61 26   ame%5B0% 5D=Cola&
02a0  70 72 69 63 65 25 35 42  30 25 35 44 3d 25 32 34   price%5B 0%5D=%24
02b0  31 2e 30 30 26 6e 61 6d  65 25 35 42 31 25 35 44   1.00&nam e%5B1%5D
02c0  3d 44 69 65 74 2b 43 6f  6c 61 26 70 72 69 63 65   =Diet+Co la&price
02d0  25 35 42 31 25 35 44 3d  25 32 34 31 2e 30 30 26   %5B1%5D= %241.00&
02e0  6e 61 6d 65 25 35 42 32  25 35 44 3d 52 6f 6f 74   name%5B2 %5D=Root
02f0  2b 42 65 65 72 26 70 72  69 63 65 25 35 42 32 25   +Beer&pr ice%5B2%
0300  35 44 3d 25 32 34 31 2e  30 30 26 6e 61 6d 65 25   5D=%241. 00&name%
0310  35 42 33 25 35 44 3d 4f  72 61 6e 67 65 26 70 72   5B3%5D=O range&pr
0320  69 63 65 25 35 42 33 25  35 44 3d 25 32 34 31 2e   ice%5B3% 5D=%241.
0330  30 30 26 6e 61 6d 65 25  35 42 34 25 35 44 3d 4c   00&name% 5B4%5D=L
0340  65 6d 6f 6e 61 64 65 26  70 72 69 63 65 25 35 42   emonade& price%5B
0350  34 25 35 44 3d 25 32 34  31 2e 32 35 26 6e 61 6d   4%5D=%24 1.25&nam
0360  65 25 35 42 35 25 35 44  3d 49 63 65 64 2b 54 65   e%5B5%5D =Iced+Te
0370  61 26 70 72 69 63 65 25  35 42 35 25 35 44 3d 25   a&price% 5B5%5D=%
0380  32 34 31 2e 37 35 26 6e  61 6d 65 25 35 42 36 25   241.75&n ame%5B6%
0390  35 44 3d 57 61 74 65 72  26 70 72 69 63 65 25 35   5D=Water &price%5
03a0  42 36 25 35 44 3d 25 32  34 32 2e 30 30            B6%5D=%2 42.00
```

Figure 4: Wireshark screenshot of the non-encoded HTML form – contains info on each item on the Product page.

In this case, the name array points to the name of the HTML page making the TCP request. Since the POST method is only used on the **products.htm** page, we expect the first element in this array to be ‘p’.

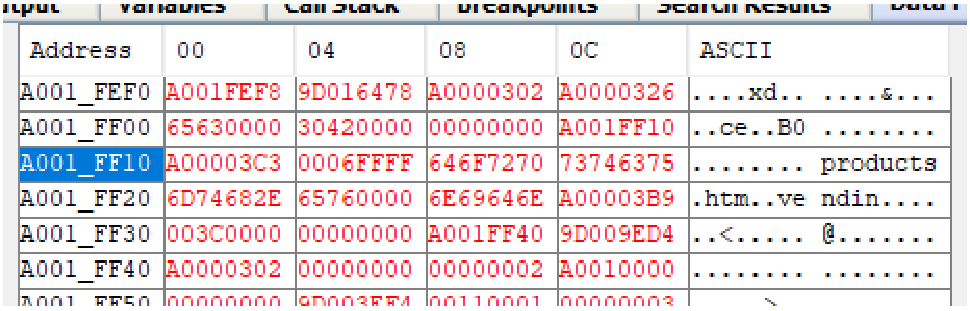


```
        {// Check if this is the last one
            if(TCPIsGetReady(sktHTTP) == curHTTP.byteCount)
                len = curHTTP.byteCount - 1;
            else // Wait for more data
            return HTTP_IO_NEED_DATA;
        }
// Make sure we don't overflow
        if(len > HTTP_MAX_DATA_LEN-2)
        {
            curHTTP.byteCount -= TCPGetArray(sktHTTP, NULL, len+1);
            continue;
        }
```

| Name | Type | Address | Value |
|---|---|---|---|
| ⊟ ☑ name | unsigned char[20] | 0xA001FF18 | "\\x 70\\x 72\\x 6f\\x 64\\x 75\\x 63\\x 74\\x 7... |
| name[0] | unsigned char | 0xA001FF18 | 'p'; 0x70 |

Figure 5: MLab Variable Window showing the contents of the **name** character array that corresponds to which website made the request.

And finally, if I look in memory at that array, the full name of the requesting webpage is found to be **products.htm**. Which is exactly what we expected.



| Address | 00 | 04 | 08 | 0C | ASCII |
|---|---|---|---|---|---|
| A001_FEF0 | A001FEF8 | 9D016478 | A0000302 | A0000326 | ....xd.. ....&... |
| A001_FF00 | 65630000 | 30420000 | 00000000 | A001FF10 | ..ce..B0 ........ |
| A001_FF10 | A00003C3 | 0006FFFF | 646F7270 | 73746375 | ........ products |
| A001_FF20 | 6D74682E | 65760000 | 6E69646E | A00003B9 | .htm..ve ndin.... |
| A001_FF30 | 003C0000 | 00000000 | A001FF40 | 9D009ED4 | ..<..... @....... |
| A001_FF40 | A0000302 | 00000000 | 00000002 | A0010000 | ........ ........ |

Figure 6: MPLab Memory View of the website name array.

## 2 Findings

Overall, I found the Wireshark program to be really helpful. Being able to capture all the traffic going to a specific IP address made looking at the messages really simple. I think it would have been more useful for me to look close at the structure of the TCP packet – especially with regard to the POST method. However, for a simple homework like this, it was useful to just easily look at all the information being sent. If there were multiple webpages being served by the PIC, or perhaps many users at a given time, I could see Wireshark being invaluable.

On the other hand, I did not find the memory view particularly helpful, as it seemed to just add a layer of obfuscation on top of the variable window, which was already really helpful. Related to that, it was nice to see the functionality behind the implementation of those GET and POST servicing functions.