

# Loosen That Ratchet

## Network Security // 800 points

### Description

A great deal of data was exfiltrated via HTTP traffic ("loosen\_that\_ratchet\_http.pcapng") from ShoppingBaba's website to its fake domain name [www.sh0ppingbaba.com](http://www.sh0ppingbaba.com) during its cyber attack.

The packet headers are in clear, however, the packet body is encrypted. But there seems to be an orderID tagged to each GET packet which suggest that there might be some ordering used in the encryption protocol...

Come join the incident response team and inspect the packets carefully to find a clue and get the critical flag hidden in the encrypted packets.

### Solution

After opening the pcap file, we are greeted with some orders in sequential order with order number starting from 000001 and going up. We apply the filter to only get the HTTP requests using `http.request.method == GET`.

The image shows the Wireshark network protocol analyzer interface. The top toolbar includes icons for opening files, saving, and various view options. Below the toolbar, a filter bar displays the active filter: `http.request.method == GET`. The main packet list pane shows a series of 14 HTTP GET requests, all originating from 192.168.152.1 and destined for 192.168.152.129. Each request is for the path `/myorders.php?orderID=` followed by a sequential order number from 000001 to 000014. The selected packet (No. 2) is expanded in the packet details pane, showing the Ethernet II, Internet Protocol Version 4, Transmission Control Protocol, and Hypertext Transfer Protocol layers. The Hypertext Transfer Protocol layer indicates a GET request for `/myorders.php?orderID=000001`. The packet bytes pane at the bottom displays the raw data in hexadecimal and ASCII, showing the start of the HTTP request line: `GET /myorders.php?orderID=000001 HTTP/1.1`.

No.	Time	Source	Destination	Protocol	Length	Info
2	0.000001	192.168.152.1	192.168.152.129	HTTP	198	GET /myorders.php?orderID=000001 HTTP/1.1
5	0.005158	192.168.152.1	192.168.152.129	HTTP	214	GET /myorders.php?orderID=000002 HTTP/1.1
8	0.010264	192.168.152.1	192.168.152.129	HTTP	214	GET /myorders.php?orderID=000003 HTTP/1.1
11	0.015598	192.168.152.1	192.168.152.129	HTTP	214	GET /myorders.php?orderID=000004 HTTP/1.1
14	0.021297	192.168.152.1	192.168.152.129	HTTP	198	GET /myorders.php?orderID=000005 HTTP/1.1
17	0.026381	192.168.152.1	192.168.152.129	HTTP	214	GET /myorders.php?orderID=000006 HTTP/1.1
20	0.031322	192.168.152.1	192.168.152.129	HTTP	214	GET /myorders.php?orderID=000007 HTTP/1.1
23	0.036443	192.168.152.1	192.168.152.129	HTTP	198	GET /myorders.php?orderID=000008 HTTP/1.1
26	0.041343	192.168.152.1	192.168.152.129	HTTP	214	GET /myorders.php?orderID=000009 HTTP/1.1
29	0.046373	192.168.152.1	192.168.152.129	HTTP	214	GET /myorders.php?orderID=000010 HTTP/1.1
32	0.051574	192.168.152.1	192.168.152.129	HTTP	198	GET /myorders.php?orderID=000011 HTTP/1.1
35	0.056581	192.168.152.1	192.168.152.129	HTTP	214	GET /myorders.php?orderID=000012 HTTP/1.1
38	0.061674	192.168.152.1	192.168.152.129	HTTP	214	GET /myorders.php?orderID=000013 HTTP/1.1
41	0.071412	192.168.152.1	192.168.152.129	HTTP	198	GET /myorders.php?orderID=000014 HTTP/1.1

Frame 2: 198 bytes on wire (1584 bits), 198 bytes captured (1584 bits) on interface 0  
Ethernet II, Src: Vmware\_c0:00:08 (00:50:56:c0:00:08), Dst: Vmware\_74:dd:da (00:0c:29:74:dd:da)  
Internet Protocol Version 4, Src: 192.168.152.1, Dst: 192.168.152.129  
Transmission Control Protocol, Src Port: 62784, Dst Port: 80, Seq: 319, Ack: 1, Len: 144  
[2 Reassembled TCP Segments (462 bytes): #1(318), #2(144)]  
Hypertext Transfer Protocol  
Data (144 bytes)

0000 00 0c 29 74 dd da 00 50 56 c0 00 08 08 00 45 00 ..}t...P V.....E.  
0010 00 b8 4b 7c 40 00 00 06 fc ef c0 a8 98 01 c0 a8 ..K|@... ..  
0020 98 81 f5 40 00 50 ee f8 ad 9b bb ff 16 27 50 18 ..@.P... ..'P.  
0030 10 0a 61 9a 00 00 2c e6 81 47 44 03 ce 58 1b c1 ..a... ..GD.X.  
0040 d6 9e 49 de af 17 35 64 ff b6 d0 85 34 d7 3e 06 ..I...5d ....4.>  
0050 ab 7a 0d 23 71 89 e7 ea 94 ef e6 cb 5f df 11 24 ..z.#q... ..\_\$  
0060 5e be 2f 08 f7 7d 98 fd b8 16 90 f5 08 44 6a 7d ^./... ..Dj}  
0070 66 fc 33 6d f9 98 8d f2 0a 02 5b 23 2e f1 31 83 f.3m... ..[#..1.

Frame (198 bytes) Reassembled TCP (462 bytes)  
Frame (frame), 198 bytes  
Packets: 6060 · Displayed: 2020 (33.3%) Profile: Default

However, even after doing so there isn't any valuable information that we can get out from here as everything seems to be encrypted. Or so I thought...

Since this challenge involves encryption, we will first think of the length of the encrypted ciphertext. Sorting the packets by length reveals something special as shown, the HTTP data is actually in plaintext!

The image shows a Wireshark packet capture window titled "loosen\_that\_ratchet\_http.pcapng". The filter bar shows "http.request.method == GET". The packet list shows several HTTP GET requests to "/myorders.php?orderId=...". The packet details pane for packet 3122 shows the following information:

- Frame 3122: 196 bytes on wire (1568 bits), 196 bytes captured (1568 bits) on interface 0
- Ethernet II, Src: Vmware\_c0:00:08 (00:50:56:c0:00:08), Dst: Vmware\_74:dd:da (00:0c:29:74:dd:da)
- Internet Protocol Version 4, Src: 192.168.152.1, Dst: 192.168.152.129
- Transmission Control Protocol, Src Port: 63836, Dst Port: 80, Seq: 319, Ack: 1, Len: 142
- [2 Reassembled TCP Segments (460 bytes): #3121(318), #3122(142)]
- Hypertext Transfer Protocol
- Data (142 bytes)
  - Data: 4d6573736167653a53656e64696e6720696e697469616c20...
  - [Length: 142]

The packet bytes pane shows the raw data of the packet, which is a JSON object containing an AES key and IV. The data is: {"message": "Cyberthon is an exciting and awesome event!", "next-id": 000252, "key": "f9a5f256e3bcfb8804d6e1448203d2eb", "iv": "6e0c228c5f7bd3d4a96a05b2ddd7600d"}. The key is 16 bytes long, and the IV is 16 bytes long.

This shows that the current packet (order 1041) is the initial packet from the client, and sends back the AES key and IV for use in the `Next-ID` order.

The key is 16 bytes, and thus indicate that it is an AES-128-CBC cipher (you can guess the mode of operation by guess and check using the next order, 303, and it turns out to be CBC)

Decoding the next message (order 303) we get: `Message: Cyberthon is an exciting and awesome event!; Next-ID: 000252; Key: f9a5f256e3bcfb8804d6e1448203d2eb; IV: 6e0c228c5f7bd3d4a96a05b2ddd7600d;`

Therefore, all we need to do now is to follow the flow and obtain the final packet which should contain the flag.

We export all the packets shown to a JSON file, then using nodeJS to parse and finally obtain the message that contains the flag. The initial position, key and IV have been given to the script.

```
const fs = require('fs');
const crypto = require('crypto');
```

```

const packets = JSON.parse(fs.readFileSync('./packets.json'));

const START = 303;
const LEN = packets.length;

let curr = START;
let key = "734159a92fff69578ba0954659a3e0a4";
let iv = "37ba7e8d277a79a3ae04fd9a4af669dd";

const pmap = new Map();

// Create a map of the orderID to the hex data of the order
packets.forEach(packet => {
  const info = packet._source.layers.http;
  const data = packet._source.layers.data;

  const idraw = info[Object.keys(info)[0]]
  ['http.request.uri'].slice('/myorders.php?orderID='.length);
  const id = parseInt(idraw, 10); // Strip leading 0s

  pmap.set(id, data['data.data'].replace(/:/g, ''));
});

// Process and follow every order
for (let i=1; i<LEN; i++) {
  const data = pmap.get(curr);
  const decipher = crypto.createDecipheriv('aes-128-cbc', Buffer.from(key,
'hex'), Buffer.from(iv, 'hex'));
  let dec = decipher.update(data, 'hex', 'ascii');
  dec += decipher.final('ascii');
  dec = dec.toString();

  const tokens = dec.split(';');
  tokens.forEach(t => {
    const p = t.split(':');
    if (p[0] == 'Message') console.log(t);
    else if (p[0] == "Next-ID") {
      curr = parseInt(p[1], 10); // Set next ID and strip leading 0s
    }
    else if (p[0] == " Key") {
      key = p[1];
    }
    else if (p[0] == " IV") {
      iv = p[1];
    }
  });
}

```

(Do note that my script prints out all the messages, so it can get quite spammy in the console, but this can be fixed easily with a few line changes)

## Footnotes

- This challenge actually is a crypto challenge, and combining it with forensics makes this challenge good and interesting.
- However, some coding is actually required as there is no sane person that will manually decrypt 2020 orders (would you even be able to find the flag before the competition ends?)
- It seems very daunting and hard at first, but as a forensics challenge there should be some digging around trying to look for clues to get started.