Loosen That Ratchet

Network Security // 800 points

Description

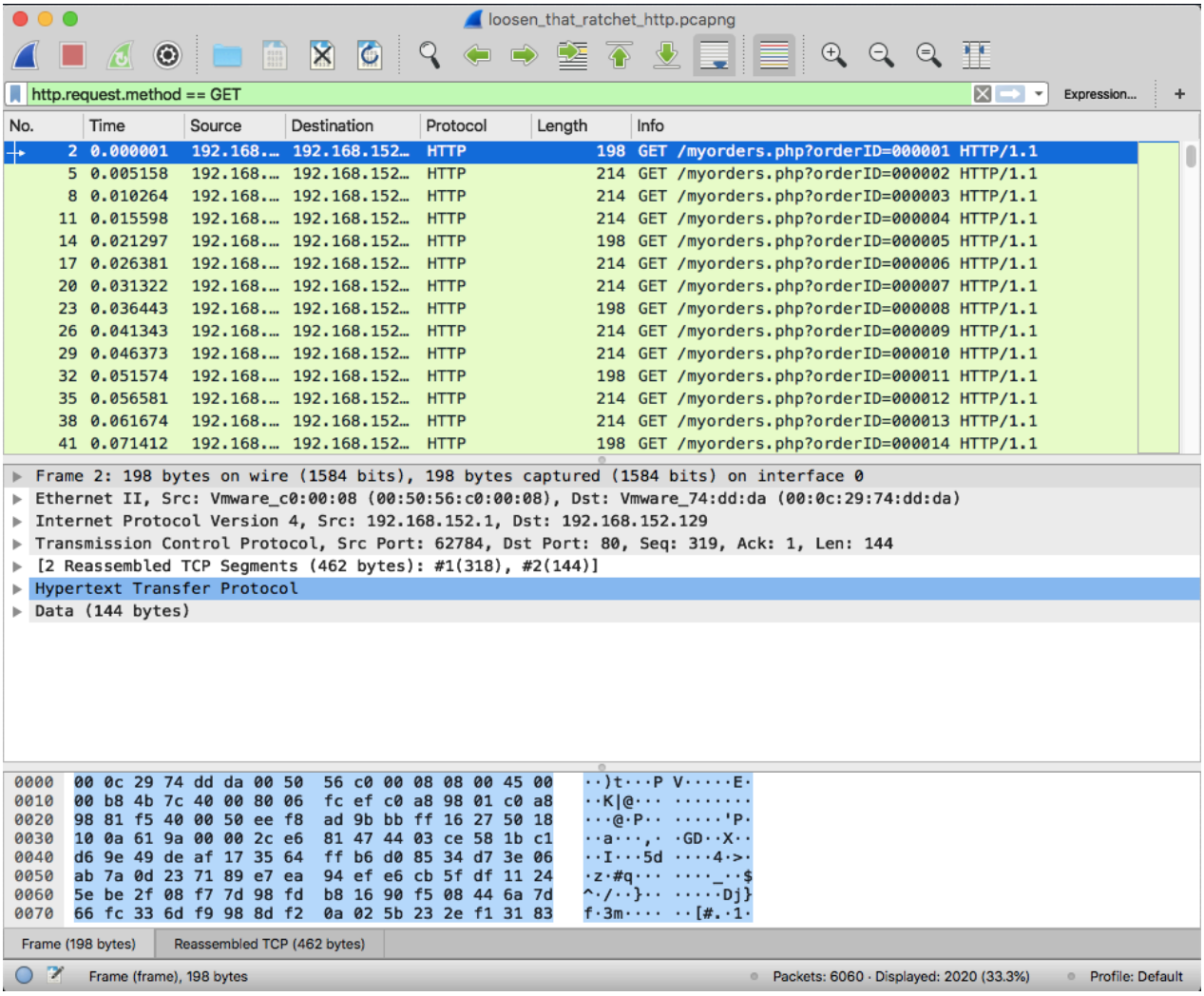
A great deal of data was exfiltrated via HTTP traffic ("loosen_that_ratchet_http.pcapng") from ShoppingBaba's website to its fake domain name www.sh0ppingbaba.com during its cyber attack.

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.

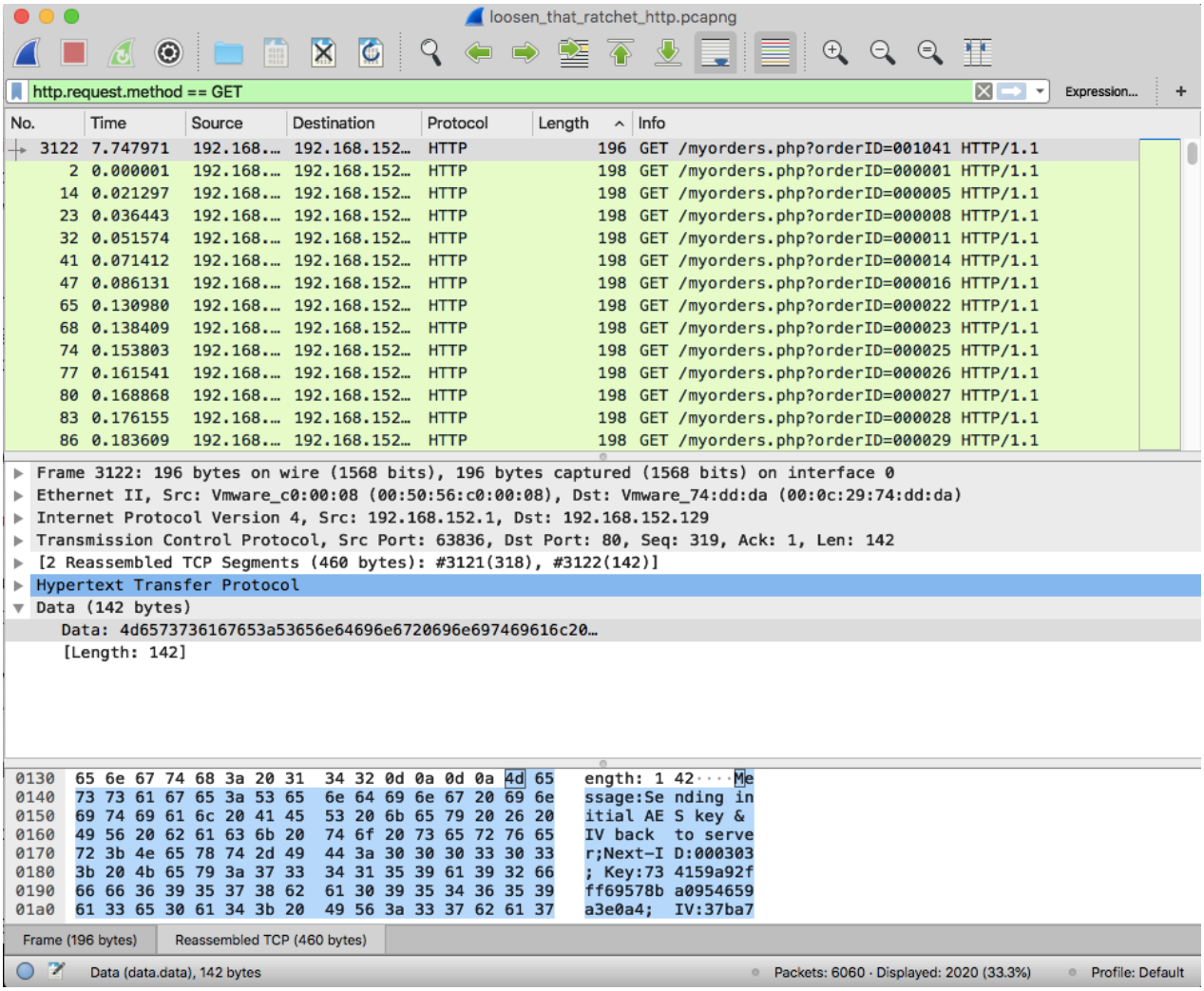
Solution

After opening the pcap file, we are greeting 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`.



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!



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.