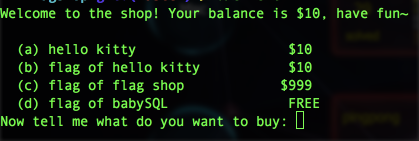
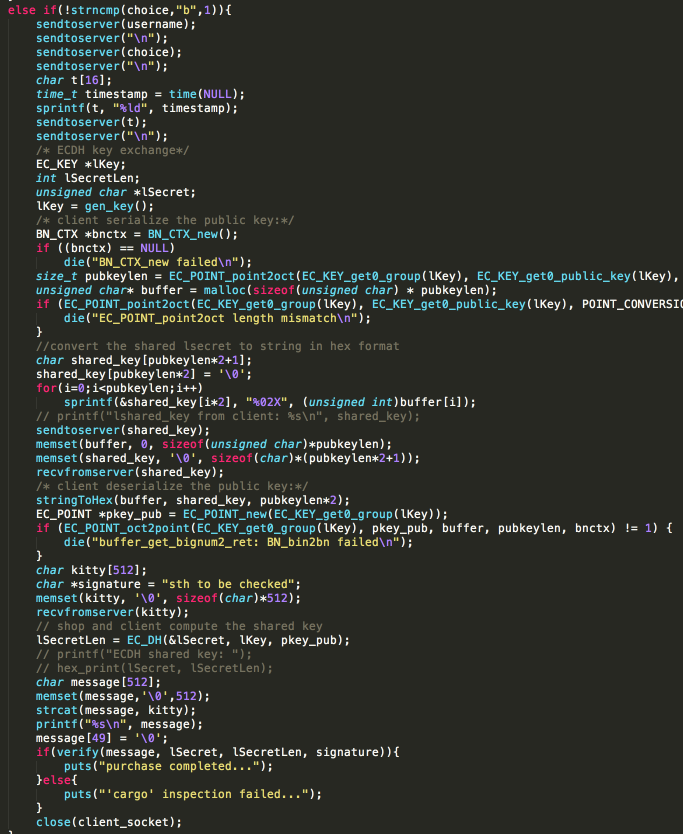
flag shop Writeup



The binary file for this challenge is coming from kittyshop so make sure you break kittyshop first.

4 options are displayed. Further analysis indicates that the client does not talk to the shop server except for option b. It's not hard to deduce that we need to follow the protocol in option b for the sake of the flag. The source code for option b is shown as follows:



The protocol of the communication between client and shop server can be illustrated as follows:

1. **client** connects to the shop server at 202.112.51.232:8080

2. **client** submits the order to shop server

3. **client** generates the lsecret key pairs and sends the public lsecret to shop server.

4. After receiving the order and the public lsecret, **shop server** (1). generates rsecret key pairs and sends the public rsecret to the client. (2) computes the ECDH shared key for signature from the private rsecret key and the public lsecret key. (3) forwards the order coming from client to wallet server (which is running on local host and not available to players) for validation (4) tells wallet the computed shared secret in a convert channel

5. **wallet** checks the balance and responses the result to shop server at 202.112.51.232:3344. Note the traffic of this step is captured and stored at *http://202.112.51.232:8888/pcap/shoppinghistory.pcap*. This URL is encoded in the client binary.

6. **shop** server compliances with wallet and sends the good to the client according to whether the result is 'accept' or 'reject'.

7. **client** receives the good from shop server.

8. **client** computes the ECDH shared key for signature from the private lsecret key and the public rsecret key.

9. **client** verify the message and print out what received.

To break this challenge, you need to find out the obvious vulnerability in this protocol that the ECDH key for session authentication and integrity can be computed at the client side. The exploit process is broken down to the following steps:

**malice** plays client role at step 1~4 that are exactly same as the origin protocol.

4. Next **malice** computes the ECDH shared secret. Now she has the power to sign the message.

5. By learning the protocol from the past traffic, **malice** pretends to be wallet, she tells the shop server that the order has been accepted at the head of the real wallet server.

6. Finally **malice** just plays client's role and waits for her good.



**convert channel**:

According to the hint from shop server, shared ECDH keys are xored with a secret. since shared ECDH keys are finally computed at client side, they have been xored with the secret and recorded in the pcap. This means we can use the transaction id to find out a session and compute the secret using the shared ECDH key in that session.