# Homework3

葉富銘　工海四 B08505045

## 1 DDoS

1. Time of the first packet : 24.945277
   victim' s IP : 192.168.232.95

2. By observing that there are lots of UDP packets with same size sent by serveral sources' IP to victim's IP, I think this is the UDP flood attack, the size of an attack packet is 482 bytes.

3. By observing that there are some ICMP packets with destinaion ip 192.168.232.95, which is victim's IP, and the info is "Destination unreachable (Port unreachable)", it seems that it's in the scenario that upon receiving UDP packet at specific port, the server will first check if there is any program listening the port's request, if not, that the server will send ICMP packet back with message indicated that the sender can't connect to the destination. Hence I infer that the victim is the server.

4. 1. 192.168.232.80 : 28320 packets
      Amplifiers : 128.111.19.188, 129.236.255.89, 124.120.108.157
   2. 192.168.232.10 : 26870 packets
      Amplifiers : 34.93.220.190, 5.104.141.250, 124.120.108.157
   3. 192.168.232.95 : 23327 packets
      Amplifiers : 34.93.220.190, 128.111.19.188, 212.27.110.13

5. Request : 90 bytes.
   Response : Total 103 packets with each 482 bytes, total 49646 bytes.
   Amplification factor : 90/49646 = 551.62.

6. As the operator of the amplifier (server), I can configure rate limiting or traffic shaping mechanisms to restrict the number of UDP packets to prevent excessive traffic caused by flood attacks, or I can use firewalls or access control lists to restrict access, only allow legitimate clients or trusted IP addresses to connect to my services.
   As the network administrator of the victim' s network, I can configure the firewall with features like stateful packet inspection, intrusion detection/prevention systems, and traffic filtering capabilities, to block or rate limit suspicious UDP traffic to mitigate the impact of flood attacks, or use quality of service (QoS) mechanisms to prioritize critical network traffic over non-essential UDP traffic, to help ensure that essential services and user activities receive sufficient bandwidth and resources during an attack.

## 2 Smart Contract

1. **callMeFirst :**
   Simply interact with CNSChallenge and type my ID.

2. **bribeMe :**
   Send 1 ether to CNSChallenge.

3. **guess randon number :**
   According to hint, we can access the value of private variable by accessing the propriate memory slot where variable are stored. Therefore, I first access the memory slot of ”next” private variable to get its value, and execute function ”random()” on ”next” to get the desired random number.

4. **reentry :**
   Write a contract with two functions, one for calling reentry() in CNSChallenge, one for being fallback function, to check if reentry() has been called for two times, if not then call reentry() again, to let c3Flag become 2.

5. **flashloan :**
   The target is to gain 10000 tokens and let flashloaning = 0. Since the variable type of flashloaning is uint8, it's maximum value is 255, which is not big enough and can easily overflow, when it overflows, it will be reset to 0, I exploit this property to gain the token during the procedure of flashfloan().

6. **steal the ethers :**
   This challenge exploits the property of delegate call. The difference between .call() and .delegatecall() is stated below :
   When using call, the sender will be the contract which the current context is inside, and the context will be in the callee, while using delegatecall, the sender and the context won't change. Therefore, when CNSWallet executes setup, it use .delegatecall and hence both sender and context remain the same. By exploting this property, I can first call CNSWallet.setup() to let context be inside CNSWallet, when it .delegatecall out, it executes the function I write in another contract, with the context still being inside the CNSWallet, then this function .call out CNSToken.approve(), and hence the msg.sender will be CNSWallet. By doing this, I can let CNSWallet approve me and get the token.

## 3 Web Authentication

a.

b. 1. **Basic HTTP Authentication Scheme** : Use the Authorization header to send user credentials. The header needs to contain the word Basic followed by a space and a base64 encoded string containing the username and password, separated by a colon.

   2. **Cookie-based authentication** : The server creates a session for the user after the user logs in. The session id is then stored on a cookie on the user's browser. While the user stays logged in, the cookie would be sent along with every subsequent request. The server can then compare the session id stored on the cookie against the session information stored in the memory to verify user's identity and sends response with the corresponding state.

   3. **JWT-based authentication** : The server creates the token with a secret and sends it to the client. The client stores the token and includes it in the header with every request. The server would then validate the token with every request from the client and sends response.

   While basic HTTP Authentication is easy and fast, it isn't secure since attackers can decode the header and get username and password; In cookie-based authentication, cookies can be made available for an extended period, maintaining a session for a long time, but it lacks scalability since the sessions are stored in the server's memory. In JWT-based authentication, its scalability is good since it token is stored on the client side, but since the overall size of a JWT is quite more than that of cookies, the data overhead is high, and hence may hamper user experience.

## 4 Accumulator

1. a. **Digest :**
      Simply multiply all members in memberSet together to get the product n, and calculate digest $= g^n \bmod N$.

b. **MembershipProof** :
Given m, simply multiply all members in memberSet together except m to get the product n, and calculate proof = $g^n$ mod N.

c. **MembershipVerification** :
Given proof p and m, simply check if $p^m$ mod N = d.

d. **NonMembershipProof** :
Given m, multiply all members in memberSet together except m to get the product delta(note that normally since this is NonMembershipProof, m will not be in the memberSet, hence delta = digest d), and then use Extended Euclidean algorithm to find a, b, that a * m + b * delta = 1, the proof is computed to be ($g^a$, b).

e. **NonMembershipVerification** :
Given proof = ($g^a$, b), and m, check if $g^{a*m+b*delta}$ = g.

2. Since the private key of RSA is leaked, e.g. p and q, where p * q = N. We can get $\phi$(n) = (p - 1)(q - 1), and given any prime m, we can find its inverse $m^{-1}$ (mod$\phi$(n)). Therefor we can choose any m not in the memberSet and forge a proof p = $d^{m^{-1}}$.

3. Since the private key of RSA is leaked, e.g. p and q, where p * q = N. I simply forge proof ($g^a$, b) by letting b = 0, and $g^a$ = $g^{m^{-1}}$, so $g^{a*m+b*delta}$ = $g^{m^{-1}*m}$ = g.