# Skynet assignment1 report

Qgua2322, mxia9416, nzha9510

•**What was your choice of Diffie-Hellman key exchange parameters and what made you select them specifically? Make reference to RFC 3526.**

The prime of Diffie-Hellman key exchange is selected from RFC 3526 4096-bit MODP Group. The original 1536 bits prime is replaced due to relatively low level of safety for being too short. The power ‘a’ is generated by a cryptograph library called ‘secret’ which is newly implemented in Python 3.6. As the library provides CSPRNG, the number it produces is resistant to predict.

•**What was your choice of cipher? What mode of operation does it use? Why did you make these choices?**

Our choice of cipher is AES encryption. We choose to perform the AES encryption with CBC mode, e.g. block cipher.

We choose AES as encryption algorithm because AES’s CBC mode is chosen to be TLS (https encryption standard) and IPSec encryption standard due to its level of security. AES has salt value built in the algorithm, therefore even plaintexts and IVs are the same, ciphertexts are not the same. It prevents attackers from guessing plaintexts’ contents because of the randomness of AES’s ciphertexts. Therefore increases the level of security.

•**How do you prevent attackers from tampering with messages in transit?**

In our design, the messages’ authentications and integrities are protected by employing HMAC in transit.

After the encryption of plaintext, the outcome (ciphertext) will be hashed by HMAC using keys that are generated from shared secret. The resultant MAC value will append to ciphertext. The whole packet (ciphertext+MAC) will be send to receiver. When receivers receive the packet, they first hash the received ciphertext using HMAC and then compare this value to the HMAC value they received. Due to the property of hash functions, if the message were modified during transit, the new hash value will not match the received one. Therefore if hash values match each other, the message can be determined as unmodified i.e. Integrities are ensured. Also only sender and receiver have the same HMAC’s keys, so authentication is ensured as well.

The reason of choosing HMAC rather than other hash functions in pyCrypto is because HMAC is relatively secure against length extension attack.

The SHA384 hash algorithm is chosen for HMAC because it is much stronger than the default MD5 in terms of collision-resistance and preventing chosen text attack. In fact, both SHA384 and SHA512 are good enough for almost any imaginable collision-resistance applications. Since they are both practical, SHA384 is chosen to reduce the need of computation and let the program respond faster. In addition, the SHA384 hash function is integrated in pyCrypto library which makes it convenient to use.

**How do you prevent replay attacks?**

We prevent replay attack by using nonces as our HMAC keys. We use nonces as keys to hash the encrypted date, such nonce come from the random generator whose seed is shared secret. Because the seeds are the same for both sender and receiver, we can ensure that both sides have the same salt at the same time (iteration).

The mechanism works like following:

Sender: ciphertext → ciphertext → hash(ciphertext,key) → send(ciphertext+MAC)

Receiver: received data → ciphertext+receive\_MAC → hash(ciphertext,key) → compare new\_MAC vs receive\_MAC

Therefore if Alice sends an message to Bob and an attacker Eve captures this packet which contains ciphertext+MAC. After a while Eve decides to send this whole packet to Bob pretending this is from Alice. Because the nonce which receiver is using to produce MAC value is different from before, the newly created MAC value will not match the old MAC value even if the messages are the same. Therefore receiver will not proceed this message and discards it. Nonce here acts like an tag/token that only be used once to prevent replay attack.

Sender: ciphertext → hash(ciphertext,key1) → send(ciphertext+MAC1)

After a while……………………

Eve: send(ciphertext+MAC1)

Receiver: received data → ciphertext+ MAC1 → hash(ciphertext,key2) → compare MAC2 vs MAC1 → different MACs → discard

•**Why might we want to allow for peer-to-peer file transfers between bots? What are the advantages and disadvantages to using a central web server (pastebot.net in our case, similar to pastebin.com) to dis-tribute files when controlling a botnet?**

We want to allow for peer-to-peer file transfer between bots because P2P is difficult to detect and remediate. By employing P2P we increase the level of security in terms of preventing others from controlling our botnets.

When using central web server to distribute files:

Advantage: it is easy to build and maintain, the botmaster can easily control the botnet.

Disadvantages: The botmaster can be easily traced. The whole network is dependent on the server; if the server is hacked the attacker can then manipulate every bot on the botnet. The server therefore is a juicy target for who wants to control the botnet.

•**Explain how this botnet, if used in the real world, could be trivially controlled by other hackers or government agencies. How might one attempt to stop it?**

Due to the property of P2P network, each bot acts as a client as well as a server, therefore it is vulnerable to exploits. If a hacker gets control of one bot in the botnet, he/she can insert malware in a file and spread it over the botnet. The whole botnet can be infected. One attempt to stop it is to employ servers (mixed P2P), unlike the centralized servers, servers in P2P botnet will not have data (files) in them. They contain information about bots in the botnet and monitor the activities in the botnet.