CS4235 Project 3 Report
Christophorus William Wijaya/ cwijaya3/ 903551245

TASK 2

1. Why is this hashing scheme insecure?

   While using SHA256 as a hashing function helps in securing the users' data, it is still not considered as secured because the attacker can still obtain the guarded data by doing brute force attack, especially when they use an expensive and fast computer (it can compute a big number of hashes in a short time). Moreover, SHA256 requires users to create their own table of salt. Meaning that the users have to create their own salt function thus, it still easier for the attacker to break the hash since they might already knew what the possible salt is.

2. What steps could be taken to enhance security in this situation?

   We can improve the security of the password by changing the hashing function from SHA256 to PBKBDF2, BCRYPT, SCRYPT, or Argon2. Their hashing algorithm work by slowing down the hashing function or we called it as "key stretching". Thus, they are very useful in protecting password from attacker that use brute force attack (SHA256 vulnerable to brute force attack). In addition, those hashing function has an integrated salts so it is less troublesome for the users to add a salt in their password.

TASK 3

1. What steps did you follow to get private key?

   For the "get_factors" function, we know that n = p*q where both p and q are prime numbers; I assume that in the function I implement, the value of q is bigger than p. Since, number 2 is the only even prime number, we can conclude that if n is an even number (n%2=0), then the only possible value for p is 2 and the value of q is n divided by p. The other case is when n is an odd number. Firstly, I find the root of n and assign it to p because the value must be less than the square root of n. If p is and even number, I add 1 to p because we have solved the problem for even number n in the previous case. Then, I create an iteration to check the value of q by keep checking n%p and see whether it is equal to 0 or not. If it is not 0, I decrement the value p by 2 (-1 will create an even number) and if n%p is equal to 0, we assign the value of n divided by p to q. Then, we output the value of p and q.

   For the "get_private_key_from_p_q_e" function, where p, q, and e are given, we know from the formula of RSA that e*d = 1 mod phi and phi is equal to (p-1)*(q-1). From that, we can say that e*d must be equal to (x*phi)+1 where x is a positive integer since for all positive integer of x, the value of ((x*phi)+1) mod phi must be 1. Then, we can say that d=((x*phi)+1) divided by e. for that equation, I create an iteration where (1+phi) is the initial value and check whether the initial value % e is equal to 0 or not. We keep increment the initial value by phi until it can be fully divided by e. Then, we output the private key which is the value of the incremented initial value divided by e.

TASK 4

1. What makes the key generated in this situation vulnerable?

   Because in this particular case, the greatest common divisor between public keys n1 and n2 is not 1; meaning that they are not relatively prime and share a common factor other than 1. By finding the greatest common factor, the attacker can easily find the second prime factor of n1 and n2. By knowing both of the prime factors, the attacker will be able to compute and find the private keys.

2. What steps did you follow to get the private key?

   Firstly, for the “is_waldo” function, I check whether n1 and n2 share the same common factor other than 1. The return value of the function is a boolean value that return true when the greatest common divisor between n1 and n2 is not 1 (not relatively prime) and return false when they are relatively prime to each other. To get the private key, I implement the “is_waldo” function. However, instead of returning the boolean value, I assign the greatest common factor between n1 and n2 as the first prime factor and assign the value of n1 divided by the first prime factor as the second factor (I use n1 because we want to find my private key; not waldo’s private key). Then, I use both of the prime factor and the given e value to find the private key by using the function from task 3.

TASK 5

1. How does this attack work?

   It is known as the chinese remainder theorem. It happen as we try to encrypt the same message called as m with 3 different pubic keys (N1, N2, and N3) and same exponent value (e=3).
   C1 = m^3 mod N1
   C2 = m^3 mod N2
   C3 = m^3 mod N3
   From that we get C1, C2, and C3; each of them are the encrypted message that used the 3 different public key. By using the theorem, we can find the m^3 and can further find the cubic root of m^3; meaning that we will be able to read the original message without knowing the private key.

2. What steps did you follow to recover the message?

   Based on the chinese remainder theorem, we can get the value of m^3 by calculating the equation m^3 = (C1 * first + C2 * second + C3 * third) % (N1 * N2 * N3) where C1, C2, C3 is the given value for the encrypted message using the 3 given different public keys (N1, N2, & N3). The “first”, “second”, and “third” variable in the equation will be described below:

   To get those 3 value, we firstly find the inverse modulo of each N1, N2, and N3.
   n1_inverse = (n2*n3) mod n1
   n2_inverse = (n1*n3) mod n2
   n3_inverse = (n1*n2) mod n3

Then, I compute the “first”, “second”, and “third” value through these steps.
first = n1_inverse * N2 * N3
second = n2_inverse * N1 * N3
third = n3_inverse * N1 * N2

The next step I did is to compute the first equation m^3 = (C1 * first + C2 * second + C3 * third) % (N1 * N2 * N3) and use the m^3 to find the original message by using binary search in the “find_root” function. The “find_root” function will output the original message.

REFERENCES

https://nakedsecurity.sophos.com/2013/11/20/serious-security-how-to-store-your-users-passwords-safely/
https://crypto.stanford.edu/pbc/notes/numbertheory/crt.html