Josia Andreas Gunawan
GTID: 903550798
Username: jgunawan6

CS-4235 Project 3 Reflection

Task 2 – Can I Have Some Salt with My Password?

-Why is this hashing scheme insecure?

SHA-256 is a relatively not very secure for storing password and SHA-256 is never meant to be a password-hashing function. The main problem is that the SHA-256 is not heavy to compute, thus creates some vulnerability to brute force attack. An attacker with decent gaming GPU could compute billions of possibilities in a second.

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

Some of popular algorithm to store passwords are scrypt, bcrpyt, and PBKDF2. Those algorithms use a technique called "key stretching" and is very hardware intensive. It includes work factor so it would be very slow to crack using brute force attack.

Task 3 – Attack A Small Key Space

-What steps did you follow to get the private key?

First thing I did is to get the factors. We knew that n=p*q and p&q are two large prime numbers. To get the factors, first I create a special condition where one of the primes is 2, since 2 is the only even prime number. Then, if the number wasn't 2, I rooted the n, since one of the factors should be smaller than the root of the n, and assigned it to p. If p is even, I converted it to the nearest odd number by adding +1. Next step, I looped through all the number from p to 0 and at each iteration I reduced p by two to keep the number odd (Since all prime number besides 2 are odd). At each loop, I tried to mod n with the current value of p, and I stopped if the result is 0, then that value of p is one of the prime factors. Finding the other factors is relatively simple, I just divided n with p to get q q=n/p.

Then to compute the private key, I used Extended Euclidean Algorithm. We know phi=(p-1) *(q-1) and p, q, e, are given. I used egcd(p,e) and extended Euclidean algorithm to find the modular inverse of e and return it as d.

Task 4 – Where's Waldo

-What makes the key generated in this situation vulnerable?

This key is vulnerable because the moduli shares a common prime factor with another key, so that we can easily find one of the prime factors by using GCD. Since we know one of the prime factors, it would not be hard to find the other one. We can divide n with one of the prime factors we got and then we got 2 prime factors that makes n. The 2 prime factors we got is enough to get the private key.

-What steps did you follow to get the private key?

Since the key's moduli shares a common prime factor with Waldo's key. We can easily find Waldo by using GCD, and if (GCD(n1,n2) !=1) we have found Waldo! We can also compute one of the prime factors using GCD (n1, n2) and assigned it as **p**. Since we can find one of the prime factors of the key using GCD, we can find the other factor by dividing n with **p** and assigned it as **q**. Using what I have done in task 3, I can compute the private key using p and q I have got.

Task 5 – Broadcast RSA Attack

-How does this attack work?

If e is 3 and n1, n2, n3 is three different public keys used to encrypt a massage "m", we would get three equations:

C1= m^(3)mod(n1)

C2= m^(3)mod(n2)

C3= m^(3)mod(n3)

We could solve m^(3) by using one of the number theory which is *Chinese Remainder Theorem.* After using *Chinese Remainder Theorem*, we would get m^(3) and we need to cubic root it. After we cubic root it, we will get m which is the message encrypted.

-What steps did you follow to recover the message?

Using the concept I mentioned before, I used *Chinese Remainder Theorem* to find m^(3). We can use Chinese Remainder Theorem to find a value where (C1)mod(n1) = (C2)mod(n2) = (C3)mod(n3) and I assigned it as m123. This value is equal to m^(3)mod(n1*n2*n3). Since m^(3)<(n1*n2*n3), m1230 is equal to m^(3) Then, I used binary search cube root to find the cube root of m^(3) to get m, so that I could know what the original message is.

References:

https://dusted.codes/sha-256-is-not-a-secure-password-hashing-algorithm

https://security.stackexchange.com/questions/90064/how-secure-are-sha256-salt-hashes-for-password-storage

https://crypto.stanford.edu/pbc/notes/numbertheory/crt.html