Assignment #1

B05902120 / Yu-Ting, TSENG

Apr 5, 2019

CIA

Please explain three major security requirements: confidentiality, integrity and availability. For each security requirement, please give an example in the real world.

The other easily-understanding word of confidentiality would roughly be privacy. To be more precisely, the information should not be exposed under unauthorized people. In common case, company would ensure confidentiality by designed some mechanism to prevent sensitive information from reaching the wrong people, while making sure that the right people can in fact get it. "Eavesdropping" and "password file stealing" are two examples violate confidentiality. Data encryption is a method to efficiently ensure confidentiality; two-factor authentication is becoming the norm. Another instance is that major facetime lets you hear the audio of the person before they pick up.

Integrity means that any operation toward data must be legal and authorized. It involves maintaining the consistency, accuracy, and trustworthiness of data. To put it simply, data must not be changed and altered by unauthorized people. These measures include "file permissions" and "user access controls", we can ensure the data is access only by authorized people. Furthermore, version control maybe used to prevent erroneous changes or accidental deletion by authorized users becoming a problem. For example, if DNS table is modified, user would be redirect the way to hacker.

Availability ensure all hardware being maintained and repaired, authorized people can get the information when needed immediately. It provides smooth feel when making use of these services. Yo achieve availability, providing adequate communication bandwidth and preventing the occurrence of bottlenecks are equally important. Moreover, safeguards against data loss or interruptions in connections must include unpredictable events such as natural disasters and fire. A great variety of malicious actions such as denial-of-service (DoS) attacks violate availability.

Hash Function

Please explain three properties of a cryptographic hash function: one-wayness, weak collision resistance and strong collision resistance. For each property, please give an example applied in the real world.

One-Wayness means that we can hardly guess the original plaintext from the output ciphertext through a hash function, included md5. Supposed the output result is n-bit, and the possible results scatter uniformly, which indicates that Prob($H(\texttt{plaintext}) = \texttt{ciphertext} = 2^{-n}$. In this situation, if we want to guess the original value, it will cost 2^{n-1} trial on average.

Weak Collision Resistance indicates that we can hardly find a collision with one certain input via a hash function, such as sha-1. The same assumption is adopted. We could discover that in such case, it takes us 2^{n-1} trial to find the other plaintext to have same hash value.

Strong Collision Resistance indicates that we can hardly find a collision of two different input via a hash function, such as sha-256. The same assumption is adopted. Different from the above mention, there is no assigned input plaintext. Under this circumstance, it costs us $2^{n/2}$ times for trial.

Threshold Signature

You should revise the setting of Shamir's Secret Sharing and BLS signature scheme to accomplish BLS threshold signature, such that a valid threshold signature can be signed only if t out of n users collaborate. Everyone should be able to verify the user signature and the threshold signature by performing the verification procedure.

In SetUp Algorithm, most of the thing are same as mentioned. Supposed there are n users. Let $U(x)$ be a random polynomial of degree n with $U(0) = sk$. Each user is assigned a public key pk and a secret key sk. Note that for any set of t participants, every $sk = U(0)$.

In Signing Algorithm, the situation is similar to SetUp Algorithm. Each participant computes its own partial signature σ and broadcast the pair (pk, σ).

In Verifying Algorithm, things has become much more different. Because of the designed algorithm and the combination of Shamir's Secret Sharing, we can indicated that the signature σ on m is only valid if and only if $e(\sigma, g) = e(h, pk)$.

Babe Crypto

Platform: Unix (Macbook)

Language: Python 2.7

Flag: BALSN{CRYPT0_1S_3ASY_XDD}

There are 4 rounds in total. The first is Caesar Cipher, what we should do is to list all possible solution and pick one as input of the server. Getting into the second round, we find that the location of " " and punctuation are the same, what is changed is only the alphabets, so we assume that it is a kind of substitution cipher. We then observe the regular pattern, finding that the difference of between alphabets in plaintext and those in ciphertext is repeatedly appear. The third round is the most difficult one. " " and punctuation are located in different place; furthermore, the same alphabet correspond to different results. During DIP course, a bold idea come to my mind. It might be encrypted by some graphic. After a few tries, it is zigzag encryption! The last round is the most easy part. Based on "=" at last, we know that it must be base64 encryption. After solving those problem, we could be able to get the flag.

One Time Padding

Platform: Linux (Workstation)

Language: Python 3.6

Flag1: BALSN{7ime_Se3d_Cr4ck!n9}

Flag2: BALSN{Trial_4nd_3rr0r_AnD_Get_Fla9<3!}

The main concept would be that the random numbers are not truely random. Those with the same random seed would have the same results. Therefore, if we get the correct random seed we might be capable of capturing the flag. Look deeply into the code, we find that the random seed is `int(time.time())` which is just the roughly time. In this case, we can get the random seed by trying several time according to the time we connect to the server. Eventually, we get the correct flag.

The next problem is similar to the first one. We can take use of Gaussian elimination method after obtaining enough data. However, we can not keep trying for the random seed this time, otherwise, it would take too much time. Discussing to B05902013 Zong-Han, WU, we deleted the data when `0.2 < time.time() - int(time.time()) < 0.8` which might be wrong. Running our program for several times, we might get the flag.

MD5 Collision

Platform: Unix (Macbook)

Language: Python 2.7

Reference: https://github.com/thereal1024/python-md5-collision

Flag: BALSN{MD5_Ch3cK5Um_!5_Br0k3N}

We need to create two python code that will output "MD5 is secure!" and "Just kidding!" respectively. The most important is that those code have the sam MD5 hash value. In python code, we can add " " without having influence on the output result; hence, we create two python codes padding numerous " ", leading to the same hash value. Then encrypted the code by base64 and get the eventual flag.

Flag Market

Platform: Unix (Macbook)

Language: Python 2.7

Flag: BALSN{L3ngTh_3xeT3n5i0N_4tTacK_i5_34sY_w1tH_H4shPump}

A useful tool `hashpump`. It's a method to execute hash length extension attack. We can pad plenty of character after the original string and get the hash value. In this problem, we add `&BALSN_Coin=1001` at last. However, we don't know how long is the original string, but there is a range provided. We test via brute force, and output the return string which is the flag.

RSA

Platform: Unix (Macbook)

Language: Python 2.7

Reference: https://github.com/kur0mi/CTF-RSA/blob/master/

Flag: BALSN{Therefore_We_5hould_Not_Choose_4_5mall_Public_Key...}

Because of small integer `e`, we can record the receive message `e` times manually. Based on Chinese Remainder Theorem, we can get $\texttt{plain}^e mod N$ ($N = n_1 * n_2 * ... * n_k$), and we know that if $k = e$ $\texttt{plain}^e < N$; as a result, we can get `m` turned into string by ascii, which is the flag.

The Backdoor of Diffie-Hellman

Platform: Unix (Macbook)

Language: Python 2.7

Flag: BALSN{black magic number}

The hint is really useful. In the sake of that 691829 is a small integer, we can actually get `a` equal to 352497 by brute force. Next, we compute the value of `s`. According to the code, we can assume that $(\texttt{cipher} * \texttt{s}^{-1})\%\texttt{p} = \texttt{int(flag.encode('hex'), 16)}$. Fermat's little theorem enable us to know $\texttt{s}^{-1} = \texttt{s}^{\texttt{p}-2}$. In the last step we translate the numbers to character by ascii table and capture the resultant flag.