Tutorial 03 solutions
- OS security -

Security principles For each scenario identify the security principle that is the most relevant. There might be more than one possible answer. Discuss your answers in class.

1. Many people lock their most valuables objects in a safe in their house. They further lock the doors of their house.

   > Defence-in-Depth. For an attack to be successful the attacker will need to have managed to break both locks - these are two independent defence mechanisms.

2. Many people hide a duplicate of their house key under one particular in their garden just in case they forget or lose their own key

   > This mechanism contradicts the open design principle. It works as long as the adversary does not know the details.

3. Many cars come with a valet key which only opens the door and turns on the engine, but not the booth of the car, nor the glove compartment.

   > Principle of least privilege. A valet does not need to access to the booth or the glove-box to park a car.

4. Secret service walkie-talkies have robust encryption, but the default setting sends communication unencrypted.

   > This was not seen in class although discussed in the textbook. Consider the human factor. Agents will tend to forget to turn the encryption on. The communication will be as secure as the agent is security-conscious.

5. Shamir's secret sharing scheme allows you to split a "secret" between multiple people, so that all of them have to come together and collaborate in order to recover the secret.

   > This was not seen in class but this is yet a common design principle and useful in many settings, where centralisation with a single person accessing the secret might be unreasonable. Distribution of trust - everyone will need to be dishonest and colluding to get hold of the secret. A single honest party will be enough to protect the secret.

Permissions

1. Imagine `Myrto` is a member of group `staff` on a system that uses basic UNIX permissions. She creates a file `marks.txt`, sets its group as `staff` and sets the file permissions as `u=rw-` and `g=o=---`. Can `Myrto` read her file `marks.txt`? Can anyone else read that file?

   Solution

   Myrto is the owner of the file and her permissions are set to `u=rw-` so she has read and write access to that file. The administrator of the system (the root user) will also have read (and all other permissions) access to it. But no one else, not even from the group `staff`, will be able to read the content of that file because the group `staff` and all other users' permissions are set to `g=o=---`.

Passwords

1. What is the purpose of salting passwords?

   Solution

   The threat model here is that the attacker has got his hands on the password database. Salting passwords slows down the attacker when trying to find all users in this database that have picked a password included in his dictionary of common passwords.
   Without salt two user accounts with the same password will store the same hash value. So the attacker's complexity is just $\mathcal{O}(d)$ where d is the size of his dictionary. The attacker simply computes the hash of all the entries in his dictionary, and can amortise this computation for breaking any user account stored in that database that is associated with a password from his dictionary.
   On the other hand, if passwords are salted-and-then-hashed, even if two entries in that DB have the same password, the stored salted and hashed password differ. So for n users and d entries in the dictionary: the salt expands the search space of the attacker proportionally to the number of users $\mathcal{O}(n \times d)$.

2. If passwords are salted using 24-bits long random numbers, how big is the dictionary attack search space for a 200,000 words dictionary?

   Solution

   If the attacker does not have access to the salt associated with each user account (this is what this exercise assumes), he will then have to brute force the whole salt space. So for each of the 2^{24} possible values for the salt s, and each of the 200,000 passwords included in his dictionary p , the attacker needs to compute $H(s||p)$. The size of the search space is thus $2^{24} \times 200,000 < 2^{24+18} = 2^{42}$. This is something that an attacker can explore in a few days if the hash function is fast. Note that this might be further accelerated by implementing it on a GPU, and FPGA or an ASICS. However if a slow hash function is used this might already render the attack ineffective.

3. If the attacker further gets her hands on the file that associates userid with their 32-bit random salt value, as well as accessing the password file that contains the salted-and-hashed passwords for the 100 people in her class. If the attacker has a dictionary with 500,000 common passwords entries, and she is confident all her 100 classmates have chosen passwords from this dictionary, what is the size of the attacker's search space for performing a dictionary attack on all their passwords?

Solution

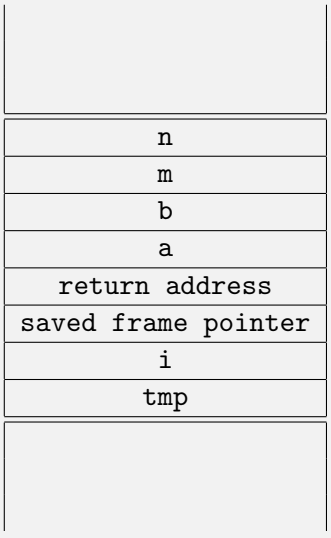
Having access to the specific user IDs of their victim, the attacker does not need to brute force the whole salt space. They merely need to iterate the dictionary attack for each of the passwords. The search space is reduced to the specific salt values for the victim accounts, that is $100 \times 500,000$.

Memory safety Consider the following C code.

```
void vuln(int a[], int b[], size_t m, size_t n) {
     for (size_t i = 0; i < m; i++) {
         int tmp = a[i];
         a[i] = b[n-i];
         b[n-i] = tmp;
     }
}
```

1. Depict the stack frame of a call to `vuln` before the execution of the `for` loop. You will assume that the code is executed on a 32-bit machine, with the size of the int data type being 4 bytes, and the size of the `size_t` data type being 4 bytes. `size_t` is defined by the C standard to be the **unsigned integer** return type of the `sizeof` operator.

Solution



Note that `a` and `b` may be pointing somewhere higher up in the stack, or somewhere in the heap.

2. This code is not memory-safe. Can you explain why and how could an attacker exploit this?

Solution

This code is vulnerable to a smashing the stack attack by overwriting a portion of the stack beyond the bounds of the `a` and `b` buffers. It can be exploited to mount a control hijacking attack, that executes arbitrary malicious code by taking the control flow of the application. In particular if `n` and `m` are greater than the size of `a` and `b` respectively.

3. Which of the following conditions on `a`, `b`, `m`, and `n` would ensure that `vuln()` be memory safe? If it is sufficient explain why. If not give an example of an input that would satisfy that condition but would be unsafe.

   (a) `a != NULL && b != NULL`

   Solution

   No memory-safe. Consider `a = [0]`, `b = [0]`, `m = 2`, `n = 1`.

   (b) `a != NULL && b != NULL && m == 0 && n == 0`

   Solution

   Memory-safe. The loop is never executed, so nothing is read from or written in memory.

   (c) `a != NULL && b != NULL && m == n`

   Solution

   Not memory-safe. Consider `a = [0]`, `b = [0]`, `m = 2`, `n = 2`.

   (d) `a != NULL && b != NULL && m < size(a) && n < size(b)`

   Solution

   Not memory safe. Consider `a = [0,1,2]`, `b = [0]`, `m = 2`, `n = 0`.

4. Suggest a better condition. Your solution should ensure that `vuln` is safe, and be as general as possible. Explain your solution.

Solution

`a != NULL && b != NULL && m <= size(a) && n < size(b) && m <= n+1`.