Tutorial 03
- 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.
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
3. Cars often come with a valet key which only opens the door and turns on the engine. Valet keys however do not open the booth of the car, nor the glove compartment.
4. Secret service walkie-talkies have robust encryption, but the default setting sends communication unencrypted.
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.

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?

Passwords

1. What is the purpose of salting passwords?
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?
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?

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.

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

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`

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

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

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

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