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)` – out of bounds index

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

ECURITY PRINCIPLES

1. Deference-in-depth

   → For an attack to be successful, the attacker will need to have managed to break both locks, which are 2 indep. defense mechanisms

2. This mechanism contradicts the open design principle

   → It works as long as the adversary does not know the details.

3. Principle of least privilege

   → A valet does not need to access the booth/glove-box to park a car

4. 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. Distribution of trust

   → Everyone will need to be dishonest & colluding to get hold of the secret

   → A single honest party will be enough to protect the secret

---

PERMISSIONS

no execute
↓ permission!

→ Myrto is owner of file and her permissions are set to u=rw–
so she has read & write access to tht. file.

→ Root user will also have read (and all other permissions) access to it

→ No one else, not even frm the group staff, will be able to read the content of tht file bc. the group staff and all other users' permissions are set to g = o = – – –

PASSWORDS

1. → The threat model here is tht the attacker has got his hands on the pswd dbase
   → Salting pswd slows down the attacker when trying to find all users in this dbase tht. have picked a pswd included in his dictionary of common pswds
   → W/o salt, 2 user accs w/ same pswd will store same hash value. So the attacker's complexity is just O(d) where d is size of dictionary
   → w/ salt, even if 2 entries have same pswd, the stored salted and hashed pswd differs. So for n users: $O(n \times d)$

2. → If attacker does not have access to the salt associated w/ each user acc., he will have to brute force the whole salt space

→ So for each 2^{24} possible values for the salt s, and each of the 20 000 passwords in his dictionary p, he needs to compute $H(s \| p)$

→ The size of search space is $2^{24} \times 200\,000 < 2^{24} \times 2^{18} = \boxed{2^{42}}$

→ If hash fn is fast, he can explore in few days.

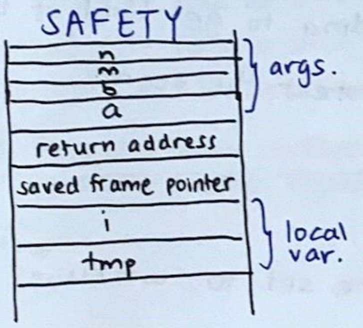
If slow hash fn used, attack might be ineffective.

3. → Now the attacker does not need to brute force the whole salt space

→ She needs to iterate the dictionary attack for each of the pswds:

$\boxed{100 \times 500\,000}$

MEMORY SAFETY

1. 

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

2. → This code is vulnerable to stack smashing by overwriting a portion of the stack beyond the bounds of a and b

→ It can be exploited to mount a control hijacking attack tht. executes malicious code by taking the control flow of the application

→ In particular if n & m are greater than size of a & b respectively

3. (a). NO! a = [0], b = [0], m = 2, n = 1

(b). YES! Loop is never executed.

(c). NO! a = [0], b = [0], m = n = 2

(d). NO! a = [0, 1, 2], b = [0], m = 2, n = 0

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