26 security-auth

保密性（Confidentiality）、完整性（Integrity）和可用性（Availability）是信息安全的三大基石。

1）保密性：保证信息不泄露给未经授权的用户。

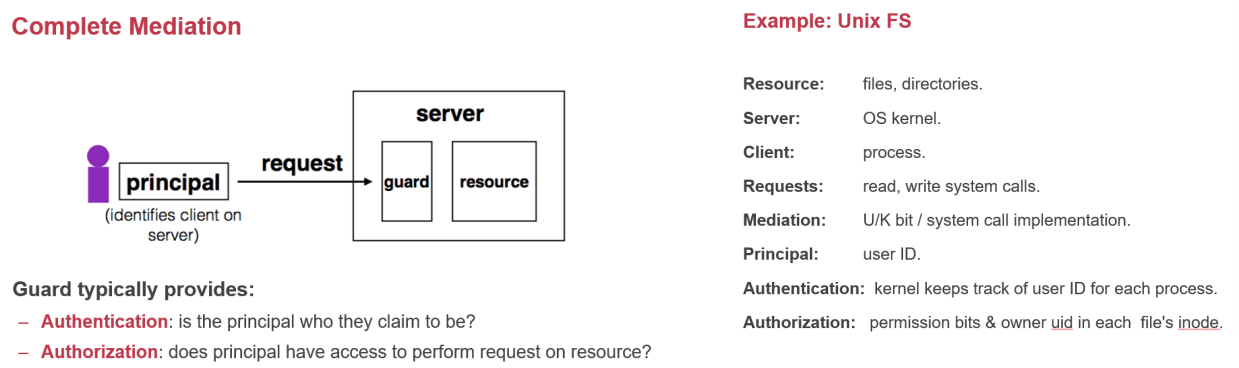
2）完整性：保证信息从真实的发信者传送到真实的收信者手中，传送过程中没有被非法用户添加、删除、替换等。

3）可用性：保证授权用户能对数据进行及时可靠的访问。

除CIA外，还有一些属性也是要求达到的，如可控性（Controllability）和不可否认性（Non-Repudiation）。

Guard Model

完全的中介——都要经过这个门，在这个门做安全检查



fd：防止没有权限的人访问

- 软件bug导致数据绕过防御
- user会犯错
- 值不值这样的安全等级

学术的评判标准：代码越少→更安全

E.G.

下载文件时直接改网址里的id也能获取信息

人的身份识别：

- 键盘输入特征
- 步伐特征
- 声音、虹膜

Example: Paymaxx.com (2005)

https://my.paymaxx.com/

- Requires username and password
- If you authenticate, provides menu of options
- One option is to get a PDF of the tax form

https://my.paymaxx.com/get-w2.cgi?id=1234

- Gets a PDF of W2 tax form for ID 1234

get-w2.cgi forgot to check authorization

- Attacker manually constructs URLs to fetch all data

密码匹配

可以通过pagefault的时间差等方法套出来 → 解决方法：hash

Store Hash of Password

username	hash(password)
arya	de5aba604c340e1965bb27d7a4c4ba03f4798ac7
jon	321196d4a6ff137202191489895e58c29475ccab
Sansa	6ea7c2b3e08a3d19fee5766cf9fc51680b267e9f
hodor	c6447b82fbb4b8e7dbcf2d28a4d7372f5dc32687

```
check_password(username, inputted_password):
    stored_hash = accounts_table[username]
    inputted_hash = hash(inputted_password)
    return stored_hash == inputted_hash
```

Store Hash of Password

What happens if an adversary breaks into a popular web site with 1M accounts?

- Adversary steals 1M hashes
- Problem: adversary can guess each password one-by-one
- Problem: worse yet, adversary can build table of hashes for common passwords
- Often called a "rainbow table"
- Users are not great at choosing passwords, many common choices

But 许多人就用几个常用的密码（如：1123456）→ 攻击者自己根据password套出密码

解决办法：salting

Salting

Salting: make the same password have different hash values

- Makes it harder to build a pre-defined lookup table

Choose random salt value when storing the password (& store the salt)

- Store hash of salt and password together
- Use the original salt to compute a matching hash when verifying password
- Every password has many possible hash values -> impractical to build rainbow table

Use a much more expensive hash function

- For reference: look up "bcrypt" by Provos and Mazieres

username	salt	hash(password | salt)
arya	5334900209	c5d2a9ffd6052a27e6183d60321c44c58c3c26cc
jon	1128628774	624f0ffa577011e5704bdf0760435c6ca69336db
Sansa	8188708254	5ee2b8effce270183ef0f4c7d458b1ed95c0cce5
Hodor	6209415273	f7e17e61376f16ca23560915b578d923d86e0319

```
check_password(username, inputted_password)
    stored_hash = accounts_table[username]
    inputted_hash = hash(inputted_password | salt)
    return stored_hash == inputted_hash
```

ASLR：随机化汇编代码位置→提高攻击难度

Session Cookies: Strawman

First check username and password, if ok, send:

`{username, expiration, H(server_key | username | expiration)}`

Use the tuple to authenticate user for a period of time

- Nice property: no need to store password in memory, or re-enter it often
- ***Server_key*** is there to ensure users can't fabricate hash themselves
- Arbitrary secret string on server, can be changed (invalidating cookies)
- Can verify that the username and expiration time is valid by checking hash

Session Cookies: Strawman

Problem: the same hash can be used for different username/expiration pairs!

- E.g., "**Ben**" and "**22-May-2012**" may also be "**Ben2**" and "**2-May-2012**"
- Concatenated string used to compute the hash is same in both cases!
- Can impersonate someone with a similar username

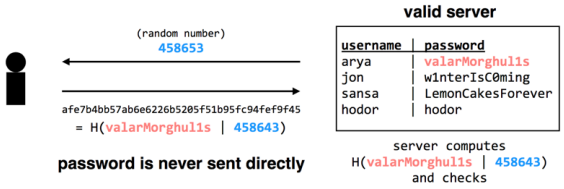
Principle: be explicit and unambiguous when it comes to security

- E.g., use an invertible delimiter scheme for hashing several parts together

不能直接拼接字符串，会带来歧义！

把密码发给服务器，但是服务器可能是冒充的！**不把密码直接发给服务器的方法：**

Technique 1: challenge-response scheme



Adversary only learns H(valarMorghul1s | 458643); can not recover the password from that

Server chooses a random value R, sends it to client

Client computes H(R + password) and sends to server

Tech 2: use passwords to authenticate the server

Make the server prove it knows your password

- Client chooses **Q**, sends to server, server computes **H(Q + password)**, replies
- Only the authentic server would know your password!

Unfortunately, not many systems use this in practice

- In part because app developers just care about app authenticating user...

机制1、2不能一起用！拿第二个机制里的服务器发过来的信息发给第一个机制里的服务器

Tech 3: turn offline into online attack

Turn phishing attacks from offline into online attacks

- If adversary doesn't have the right image, users will know the site is fake
- Adversary could talk to real site, fetch image for each user that logs in

Tech 4: Specific password

Make passwords specific to a site

- Instead of sending password, send H(servername + password)
- Just like a basic password scheme, from the server's point of view

Except impersonator on another server gets diff passwd

Recommendation

- E.g., LastPass

Why is it still useful, then?

- Requires more efforts on adversary's part to mount attack
- Even if adversary does this, bank can detect it
- Watch for many requests coming from a single computer
- That computer might be trying to impersonate site
- Turns an offline/passive attack into an online/active attack

Key insight

- Don't need perfect security, small improvements can help

Tech5: 只用一次的密码

服务器存算了100次hash后的密码，用户自己把算了99次hash的密码发给服务器，服务器再将存的内容改成算了99次hash的密码，下一次用户发算了98次hash的密码发给服务器……client要自己记着salt

Alternative design: include time in the hash (Google's 2-step verification)

Tech6: 把自己要做的事情和密码一起hash

Tech 5: one-time passwords

One-time Password

- If adversary intercepts password, can keep using it over and over
- Can implement one-time passwords: need to use a different password every time

Design: construct a long chain of hashes.

- Start with password and salt, as before
- Repeatedly apply hash, n times, to get n passwords
- Server stores x = H(H(H(H(...(H(salt+password)))))) = H^n(salt+password)

To authenticate, send token=H^{n-1}(salt+password)

- Server verifies that x = H(token), then sets x <- token
- User carries a printout of a few hashes, or uses smartphone to compute them.

Tech 6: bind authentication and request authorization

One way to look at problem: sending password authorizes any request

- Even requests by adversary that intercepts our password
- A different design: use password to authenticate any request
- **req = { username, "write XX to exam.txt", H(password + "write ..") }**

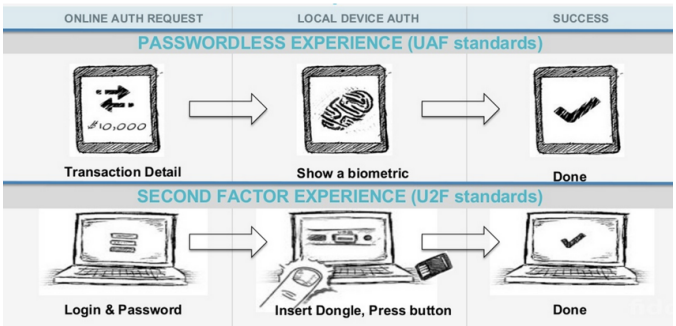
Server can check if this is a legitimate req from user using password

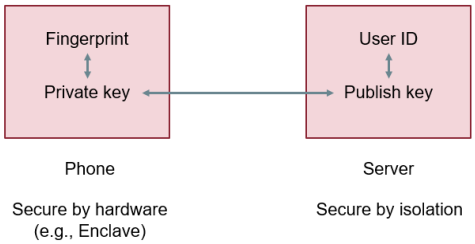
- Even if adversary intercepts request, cannot steal/misuse password
- In practice, don't want to use password, use some session token instead
- Could combine well with one-time passwords

Tech7: No password！插一个小设备（内含私钥，可以集成到电脑里）

被偷？小设备上加一个指纹验证

Tech 7: FIDO: Replace the Password





TA(Trust Zone): 手机将指纹存成primary key，但是手机里可能又恶意程序想从手机系统获取密码

密码修改：重置密码的URL如何生成？dev/random 生成随机数（用的是鼠标事件次数、键盘中断等等来生成）