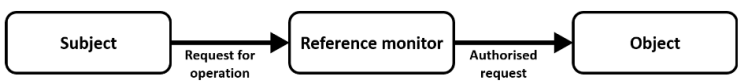
CS Revision Lecture 19, 20, 21

- ## Lecture 19 - Privilege separation: UNIX security model
  - ### Privilege separation
    - Modern computers are
      - multi-users
      - multi-tasking
    - **Goal:**
      - Prevent potentially misbehaving users and/or applications from harming the rest of the system
    - **Permissions system:**
      - Mechanisms for achieving separation between components
  - ### Central question
    - "**Who** is allowed to access **what** and **how**?"
      - **The subject (who)** - e.g. user, application, process
      - The object (what) - protected resource, e.g. hardware device, network socket, memory, files, directories, etc.
      - The access operation (how) - e.g. read, write, execute
  - ### Key assumptions for separation
    - 1. The system know who the user is - user has authenticated, e.g. using username / password
    - 2. **Complete mediation** - all requests are mediated - all requests go to the reference monitor that enforces specified access control policies



    - The **reference monitor** grants permission to users to apply certain operations to given resource

- Users
  - Two types of accounts each with a unique identifier, the user ID(uid):
    - 1. User accounts - associated with humans
    - 2. Service accounts - associated with background processes

```
marapini@myrto-thinkpad:~$ more /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
```

  - One entry in the /etc/passwd per account with the fields:
    - username:password:uid:gid:uid info:home:shell
  - **uid 0 - user root uid**
- Groups
  - Groups are sets of users that share resources
  - Every group has a name and a unique identifier, the group ID (gid)
  - Allow for easier users management and monitoring

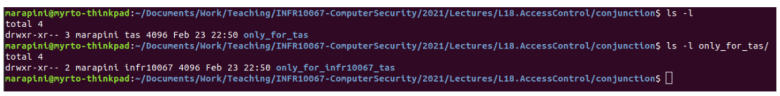
```
marapini@myrto-thinkpad:~$ more /etc/group
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:syslog,marapini
tty:x:5:
disk:x:6:
lp:x:7:
mail:x:8:
```

  - One entry in the /etc/group per group with the fields:
    - group name:password:gid:group list
- File permissions

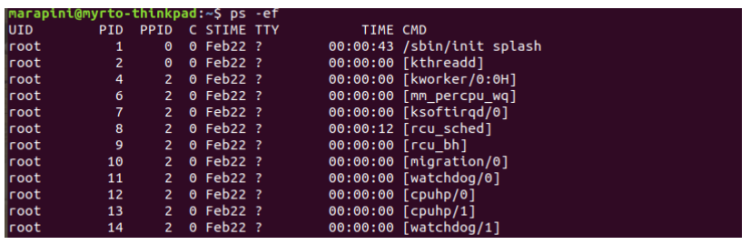
```
marapini@myrto-thinkpad:~/Documents/Work/Teaching/INFR10067-ComputerSecurity/2021/Lectures/L18.AccessControl$ ls -l
total 352
drwxrwxr-x 2 marapini marapini   4096 Feb 23 17:27 Images
-rw-r--r-- 1 marapini marapini   1839 Feb 23 17:31 L18.AccessControl.aux
-rw-r--r-- 1 marapini marapini  47121 Feb 23 17:31 L18.AccessControl.log
-rw-r--r-- 1 marapini marapini    835 Feb 23 17:31 L18.AccessControl.nav
-rw-r--r-- 1 marapini marapini      0 Feb 23 17:31 L18.AccessControl.out
-rw-r--r-- 1 marapini marapini 258111 Feb 23 17:31 L18.AccessControl.pdf
-rw-r--r-- 1 marapini marapini      0 Feb 23 17:31 L18.AccessControl.snm
-rw-rw-r-- 1 marapini marapini   9769 Feb 23 18:05 L18.AccessControl.tex
-rw-rw-r-- 1 marapini marapini  23638 Feb 20 01:28 L18.AccessControl.tex~
-rw-r--r-- 1 marapini marapini      0 Feb 23 17:31 L18.AccessControl.toc
```

- All resources (sockets, directories, files) are managed as files
- 3 defined permissions: **read (r), write (w), execute (x)**
- Permissions are defined for the owner, the owner's group, and other users
- **Root and owner** can change file **permissions**
- Only **root** can change file **ownership**

- Directory permissions
  - Execute permission on a directory allows traversing it
  - Read permission on a directory allows lookup
  - Quiz: Imagine you have the following groups:
    - infr10067 - for any user involved with the ComputerSecurity course
    - tas - for all Informatics TAs
    - How can you have a folder only for Computer Security TAs



- Processes
  - Each process has a unique identifier, the process ID (pid)
  - Each process is associated with the user that spanned it



  - When a user runs a process, it runs with that user's privileges, i.e. they can access any resource that user has permissions for
  - By default, a child process inherits its parent's privileges

- Processes are isolated in memory

- Process user IDs
  - Every process has:
    - **Real user ID(uid)** - the user ID that started that process
    - **Effective user ID (euid)** - the user ID that determines theprocess' privileges
    - **Saved user ID (suid)** - the effective user ID before the lastmodification
  - Users can change a process' IDs:

```
setuid(x)          seteuid(x)
  uid ← x            uid ← uid
  euid ← x           euid ← x
  suid ← x           suid ← suid
```

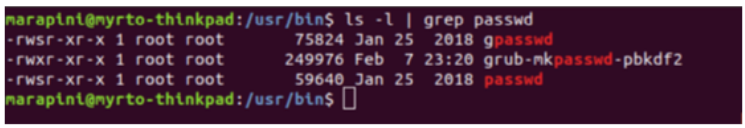
  - Root can change euid/uid to arbitrary value x
  - Unprivileged users can only change euid to uid or suid
- Dropping privileges with *setuid*
  - Imagine a program that runs as root and wants to fork a process with lower privileges using the following code:

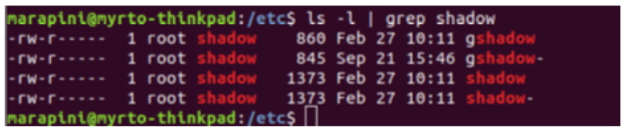
```
if (auth(uid, pwd) == SUCCESS) {
  if (fork() == 0) {
    seteuid(uid);
    exec("/bin/bash");          ← the user can call
  }                                seteuid(0)
}                                  and become root!!
```

```
if (auth(uid, pwd) == SUCCESS) {
  if (fork() == 0) {
    setuid(uid);
    exec("/bin/bash");          ← the user cannot
  }                                change uid
}
```

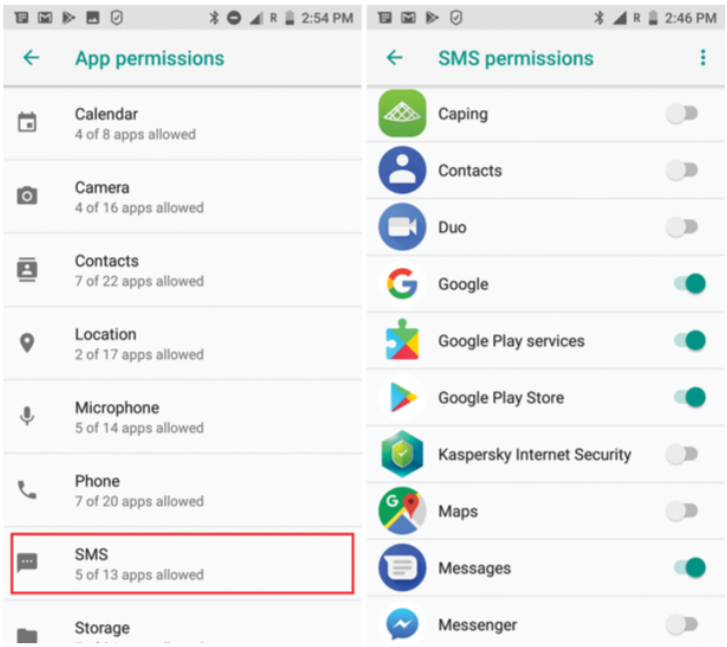
- Elevating privileges - *setuid* programs

- An executable file can have the set-user-ID property(**setuid**) enabled
- If A executes a setuid file owned by B, then the euid of the process is B and not A
- Writing secure setuid programs is tricky because vulnerabilities may be exploited by malicious user actions
- Some programs that access system resources are owned by root and have the setuid bit set(setuid programs)





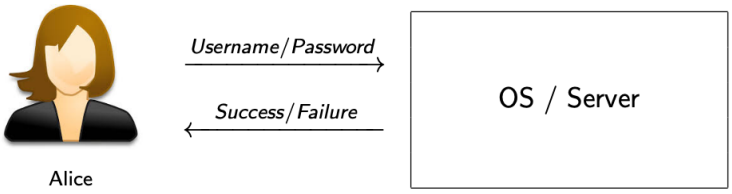
- UNIX permissions are too coarse-grained
  - All application installed by a single user account have the same privileges
  - What if the software is malware?
    - Better delegate capabilities associated with specific root powers
- Android permissions
  - Each app runs with a different User ID
  - Apps do not interact
  - Permissions are set per app



- Take aways
  - The UNIX security model provides a simple and flexible model, but permissions are too **coarse-grained**
    - same permissions for all applications ran under a single user account
    - many utilities have the setuid bit enabled
    - $\longrightarrow$ many opportunities for privilege escalation attacks
    - $\longrightarrow$ better use capabilities when delegating privileges

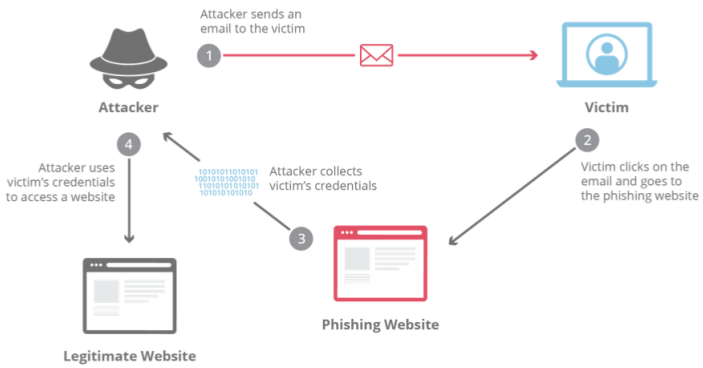
Lecture 20 - Password authentication

- Password authentication
  - The question: "who is allowed to access the resources in a computer system?"
  - How does the operating system securely identify its users?
  - **Authentication: determination of the identity of a user**
  - Standard authentication mechanism: **username** and **password**

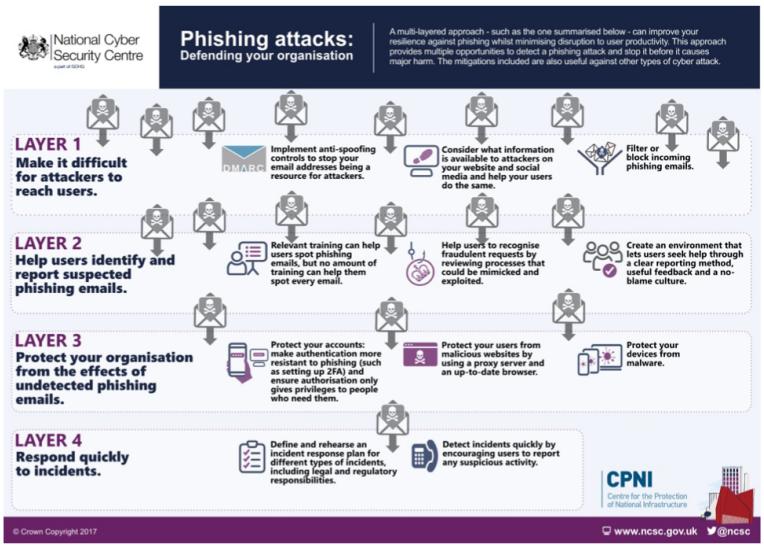


- Usability vs. Security
  - Passwords need to be **hard to guess** yet **easy to remember**
- Network attacks

  - Defending against eaversdropers
    - **Encrypt communication** using e.g. TLS
- Social engineering & Phishing attacks

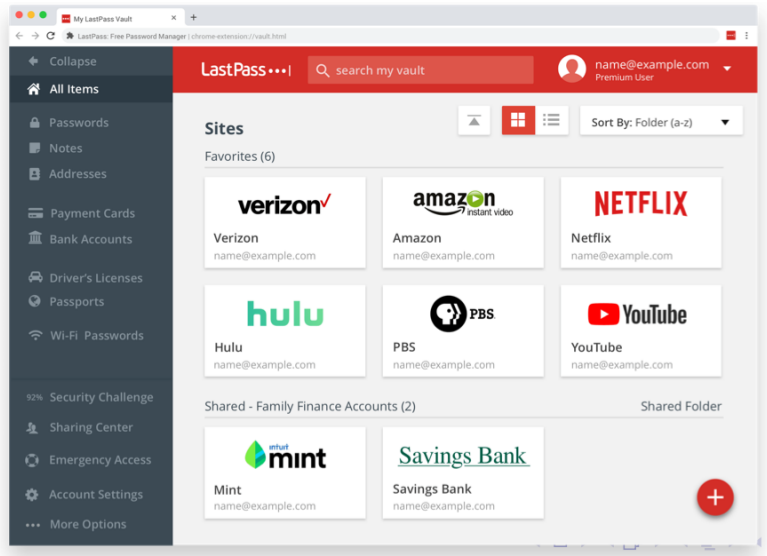


- Defending against phishing - NCSC guidance



- Layer 1
  - Make it difficult for attackers to reach users
- Layer 2
  - Help users identify and report suspected phishing emails
- Layer 3
  - Protect your organisation from the effects of undetected phishing emails
- Layer 4
  - Respond quickly to incidents

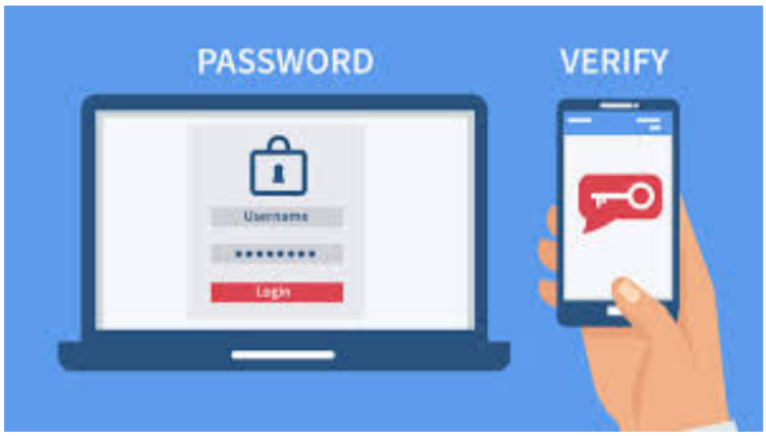
- Defending against phishing - password managers



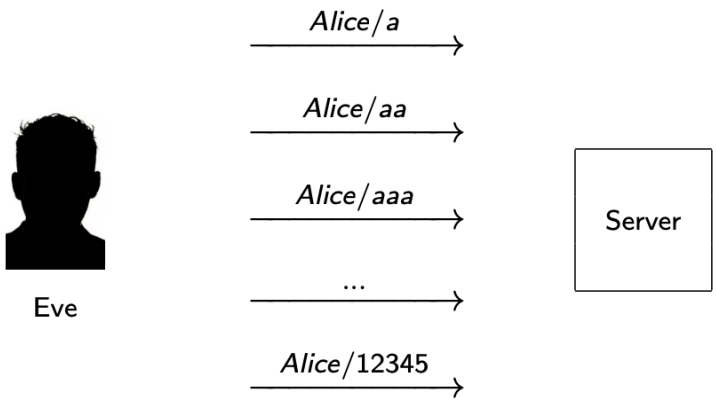
  - Password managers often fill username & password for user based on URL
  - The password manager will not enter credentials for amazon.co.uk or barclays.co.uk on any other attacker controlled website

- Malware attacks
  - **Malware attack** - users will often have malware installed on their machine - this malware might contain a **key-logger** that records keyboard stroke and intercept passwords when typed

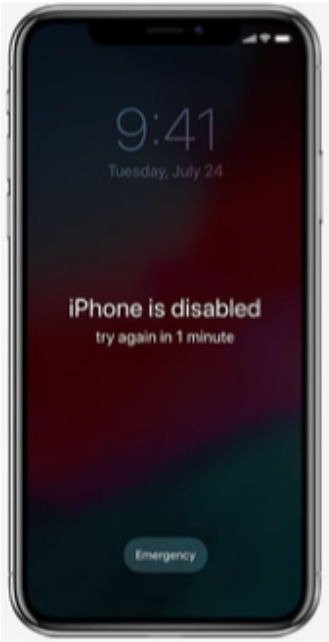
- Key-logger mitigation - use **two factor authentication** (2FA)



- Online guessing attacks



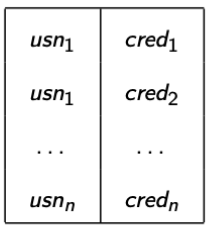
- Defending against online guessing attacks
  - **Choose a good password** - length, capital letters, symbol characters, not a word etc.
  - **Rate limit** - impose a limit on the number of failed password attempts before locking the system for a set amount of time



- Include **captchas** - include a captcha puzzle to be solved along the submission of the username and password in order to prevent automated password guessing

- Offline guessing attacks
  - Most common password-related attacks target the server

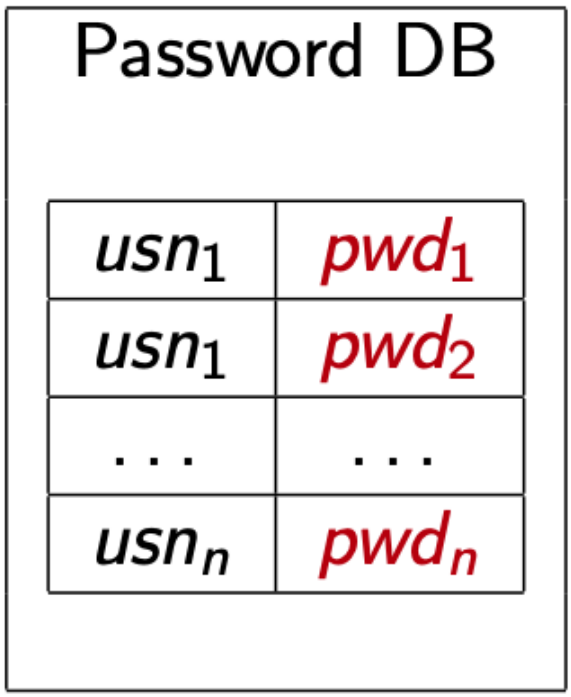
Eve



DBpwd

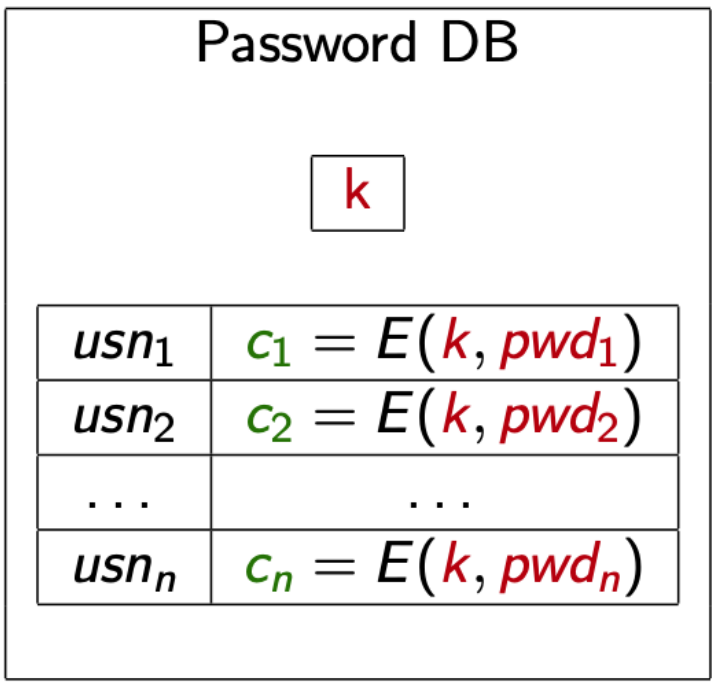
  - Our goal
    - Defend from attacks that leak the password database

- Attempt #1: store passwords unencrypted



  - Whoever accesses the password DB can login as any user
  - Might leak user login information to other services/accounts

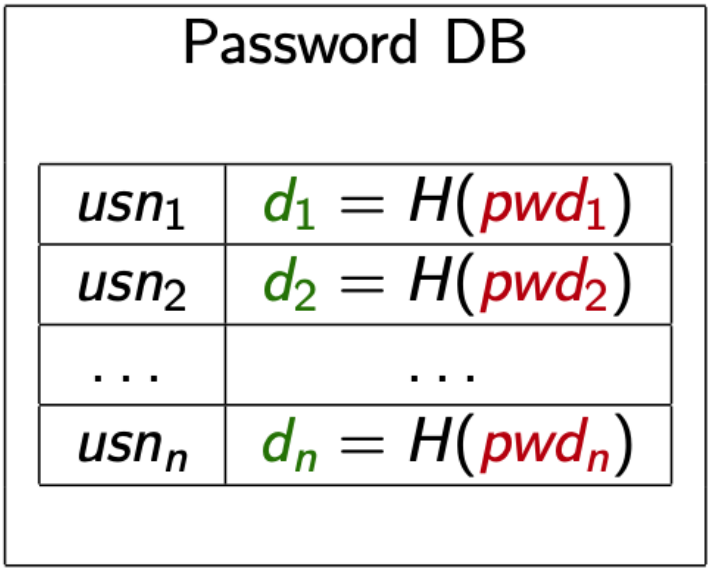
- Attempt #2: encrypt passwords



  - \+ Stolen encrypted passwords cannot be decrypted

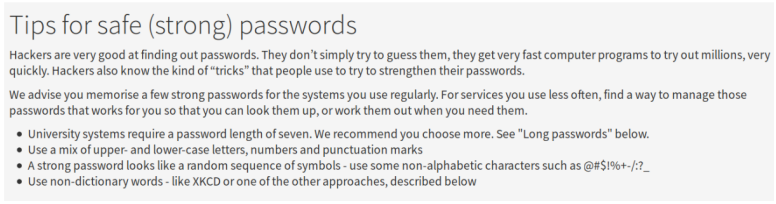
- + Only admins have the key. If a user forgets their password, admins can just look it up for him
- - If attacker managed to steal passwords, why assume the key cannot be stolen?
- - Anyone with the key(admins) can view passwords

- Attempt #3: hash passwords



- Stolen hashed passwords cannot easily be cracked
- - Once a hash is cracked, the password is know for all accounts using the same password
- - Humans tend to pick weak/guessable passwords
  - Frequency analysis
  - Dictionary attack

- Brute force attack
- Try all passwords in a given space
  - K : number of possible characters
  - l : password length
  - $\rightarrow K^l$: possible passwords



Tips for safe (strong) passwords

Hackers are very good at finding out passwords. They don't simply try to guess them, they get very fast computer programs to try out millions, very quickly. Hackers also know the kind of "tricks" that people use to try to strengthen their passwords.

We advise you memorise a few strong passwords for the systems you use regularly. For services you use less often, find a way to manage those passwords that works for you so that you can look them up, or work them out when you need them.

- University systems require a password length of seven. We recommend you choose more. See "Long passwords" below.
- Use a mix of upper- and lower-case letters, numbers and punctuation marks
- A strong password looks like a random sequence of symbols - use some non-alphabetic characters such as @#$!%+-/:?_
- Use non-dictionary words - like XKCD or one of the other approaches, described below

UoE password guidelines

⇝ Assuming a standard 94 characters keyboard, there are $94^7 = 6.4847759e^{+13}$ possible passwords.

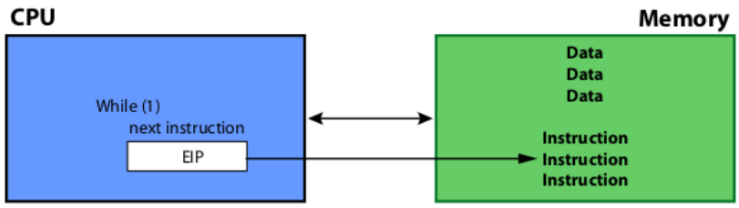
- Do we need to try all K^l passwords?

Rank	2011[4]	2012[5]	2013[6]	2014[7]	2015[8]	2016[3]	2017[9]	2018[10]
1	password	password	123456	123456	123456	123456	123456	123456
2	123456	123456	password	password	password	password	password	password
3	12345678	12345678	12345678	12345	12345678	12345	12345678	123456789
4	qwerty	abc123	qwerty	12345678	qwerty	12345678	qwerty	12345678
5	abc123	qwerty	abc123	qwerty	12345	football	12345	12345
6	monkey	monkey	123456789	123456789	123456789	qwerty	123456789	111111
7	1234567	letmein	111111	1234	football	1234567890	letmein	1234567
8	letmein	dragon	1234567	baseball	1234	1234567	1234567	sunshine
9	trustno1	111111	iloveyou	dragon	1234567	princess	football	qwerty
10	dragon	baseball	adobe123[a]	football	baseball	1234	iloveyou	iloveyou
11	baseball	iloveyou	123123	1234567	welcome	login	admin	princess
12	111111	trustno1	admin	monkey	1234567890	welcome	welcome	admin
13	iloveyou	1234567	1234567890	letmein	abc123	solo	monkey	welcome
14	master	sunshine	letmein	abc123	111111	abc123	login	666666
15	sunshine	master	photoshop[a]	111111	1qaz2wsx	admin	abc123	abc123
16	ashley	123123	1234	mustang	dragon	121212	starwars	football
17	bailey	welcome	monkey	access	master	flower	123123	123123
18	passw0rd	shadow	shadow	shadow	monkey	passw0rd	dragon	monkey
19	shadow	ashley	sunshine	master	letmein	dragon	passw0rd	654321
20	123123	football	12345	michael	login	sunshine	master	!@#$%^&*
21	654321	jesus	password1	superman	princess	master	hello	charlie
22	superman	michael	princess	696969	qwertyuiop	hottie	freedom	aa123456
23	qazwsx	ninja	azerty	123123	solo	loveme	whatever	donald
24	michael	mustang	trustno1	batman	passw0rd	zaq1zaq1	qazwsx	password1
25	Football	password1	000000	trustno1	starwars	password1	trustno1	qwerty123

- (2016) the 25most common passwords made up more than 10% of surveyed passwords.
- Most common password of 2016, ”123456”,makes up 4% of surveyed passwords.
- 30% of password surveyed in top 10000

- Dictionary attack
  - Try the top N most common passwords,
  - Try words in English dictionary,
  - Try names, places, notable dates,
  - Try Combinations of the above,
  - Try the above replacing some characters with digits and symbols

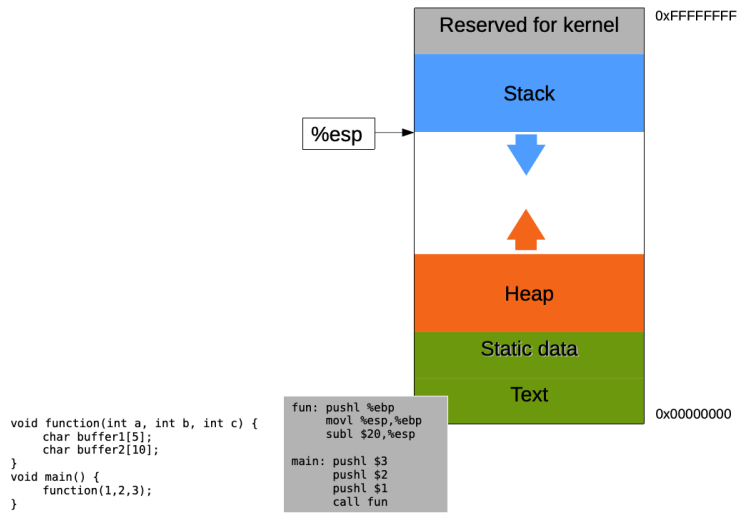
        - e.g. : iloveyou, il0vey0u, i10v3y0u, ...
    - Attempt #4: salt and hash passwords
        - + Since every user has different salt, identical passwords will not have identical hashes
        - + No frequency analysis
        - + No precomputation: when salting one cannot use preexisting tables to crack passwords easily
        - store **salted hashes** of passwords
        - use a **slow hash function** - eg. $H(pwd) = h^{1000}(pwd)$
    - Take aways
        - **1. Password authentication**
            - principles
            - network attacks
            - phishing attacks
            - keylogger attacks
            - offline attacks
            - online attacks
        - **2. Password cracking**
            - Brute force attack
            - Dictionary attack
        - **3. How to store passwords:**
            - Store salted hashed of passwords
            - use a slow hash function
        - **4. Enable 2FA**
        - **5. Use a password manager**
- Lecture 21 - Memory management
    - From source code to execution



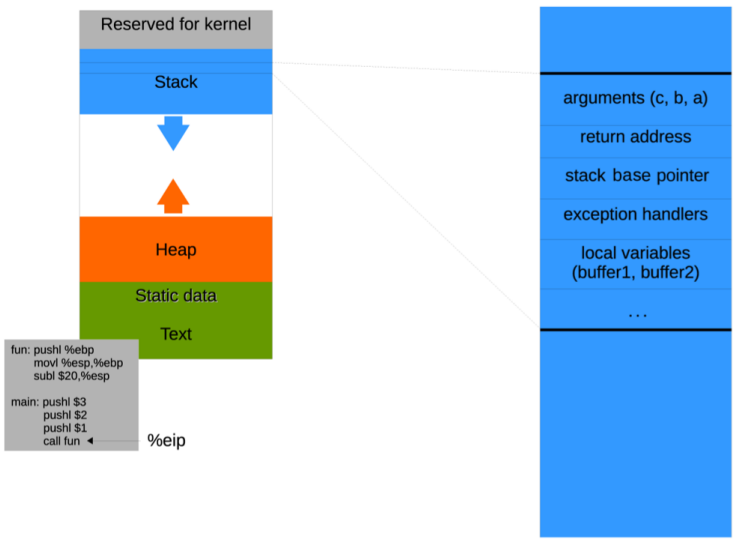
- 1. The **compiler** converts C code to assembly code
- 2. The **assembler** converts assembly code to machine code
- 3. The **linker** deals with dependencies and libraries
- 4. The **loader** sets up address space in memory and load machine code in memory, and jumps to the first instruction of the program
  - The address space contains both the code for running the program its input data, and it working memory
- 5. The CPU interprets instructions
  - %eip points to next instruction
  - %eip incremented after each instruction
  - %eip modified by **call, ret, jmp**, and conditional **jmp**

- x86-32 registers
  - Temporary registers: %eax, %ebx, %ecx, %edx, %edi, %esi
    - These registers are like variables built in the processor
    - Most of the instructions perform on these registers
  - Extended stack pointer: %esp
    - Points at the top of the stack
  - Extended base pointer: %ebp
    - Points to the base of the stack frame of the current function call

- x86 process memory layout (simplified)



- Stack frame



- Stack and functions: Summary
  - **Calling function**
    - 1. Push arguments onto the stack (**in reverse**)
    - 2. Push the return address, i.e., the address of the instruction to run after control returns
    - 3. Jump to the function's address
  - **Called function**
    - 4. Push the old frame pointer onto the stack (%ebp)
    - 5. Set frame pointer (%ebp) to where the end of the stack is right now(%esp)
    - 6. Push local variables onto the stack

- **Returning function**
  - 7. Reset the previous stack frame: %esp = %ebp, %ebp = (%ebp)
  - 8. Jump back to return address: %eip = 4(%ebp)

以上内容整理于 幕布文档