Security Design Principles: Fail-safe Defaults Least Privilege

CS-3113: PRINCIPLES OF CYBER SECURITY

BENJAMIN R. ANDERSON

Review: The Eleven Design Principles

General/Fundamental Design Principles

1. Simplicity (related to Economy of Mechanism but not exactly the same)
2. Open Design
3. Design for Iteration
4. Least Astonishment

Security Design Principles

5. Minimize Secrets (not specifically identified by Saltzer and Schroeder)
6. Complete Mediation
7. ***Fail-safe Defaults***
8. ***Least Privilege***
9. Economy of Mechanism
10. Minimize Common Mechanism (related to Least Common Mechanism)
11. Isolation, Separation and Encapsulation

Review: Saltzer and Schroeder's Security Design Principles

Many security issues are a result of poor coding techniques which lead to flaws in the program which can result in a security hole that can be exploited

Saltzer and Schroeder came up with a list of 8 security design principles that, if followed, would help programmers reduce the number of errors and design more secure software

1. Economy of mechanism – A simple design is easier to test and validate
2. ***Fail-safe defaults –In computing systems, the safe default is generally "no access" so that the system must specifically grant access to resources***
3. Complete mediation – Access rights are completely validated every time an access occurs
4. Open design –secure systems, including cryptographic systems, should have unclassified designs
5. Separation of privilege – A protection mechanism is more flexible if it requires two separate keys to unlock it, allowing for two-person control and similar techniques to prevent unilateral action by a subverted individual
6. ***Least privilege – Every program and user should operate while invoking as few privileges as possible***
7. Least common mechanism – Users should not share system mechanisms except when absolutely necessary
8. Psychological acceptability – users won't specify protections correctly if the specification style doesn't make sense to them

Fail-Safe Defaults

There are two obvious parts to this principle:

- ***Fail-safe***
- ***Defaults***

We will define defaults first which, according to the Meriam-Webster dictionary is:

- *A preselected option adopted by a computer program or other mechanism when no alternative is specified by the user or programmer*

For fail-safe, Meriam-Webster defines it as:

- *incorporating some feature for automatically counteracting the effect of an anticipated possible source of failure*

The idea here is that, if not specifically told to do something by the user, programmer or operator, the system will use the most secure settings possible

- This includes when there is a failure of some kind

Fail-safe Examples

Vehicles:

- For a vehicle with air brakes (busses, heavy trucks, trains, etc.) if the air pressure drops too low a mechanical brake automatically engages
- Lane keeping software will keep the vehicle from crossing the lane markings unless the driver puts enough force on the steering wheel to override the system

Traffic Signals:

- If a traffic light controller goes out, the lights default to blinking red in all directions

Elevators:

- If a cable breaks in an elevator, mechanical brakes automatically engage preventing the elevator car from falling

Electronic locks:

- By default, if there is a fire, electronic locks will open (called “fail-safe” mode)
- It requires someone to specifically configure them to stay locked (called “fail-secure”) and is rarely done
- This is required by the National Fire Protection Agency (NFPA) in the NFPA 101: Life Safety Code

Computing:

- A computer that detects it is getting too hot will go into thermal shutdown before catching on fire
- Before this feature, failed data center cooling systems could result in so much overheating that the glass in server racks melted – and destroyed all the hardware in the building
- An error state during authentication will result in access being denied

Related Concepts

Definitions from the NIST Glossary:

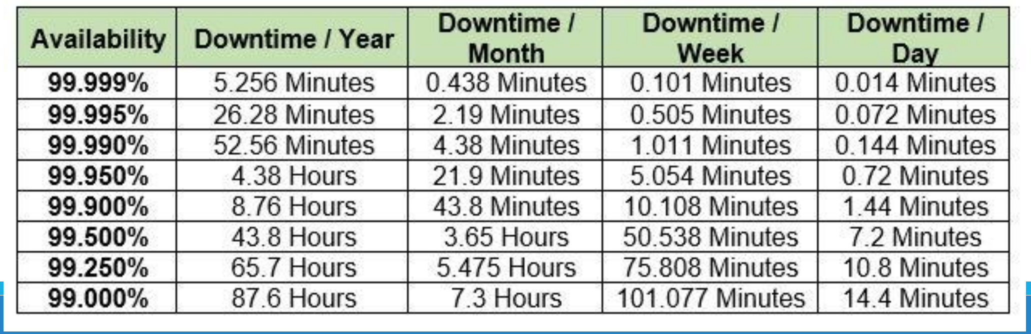
- ***Fail Secure:*** A mode of termination of system functions that prevents loss of secure state when a failure occurs or is detected in the system (but the failure still might cause damage to some system resource or system entity).
- ***Fail soft***: Selective termination of affected, non-essential system functions when a failure occurs or is detected in the system

A definition from Simplicable:

- ***Safety by Design***: A mechanism that is automatically triggered by failure that reduces or eliminates harm
- https://simplicable.com/new/fail-safe

Five-nines Availability

- This is the percentage of uptime that is desired – this is a type of availability management
- From the Amazon Public Sector Blog, Ryan Reynolds states:
  - *The accepted availability standard for emergency response systems is 99.999% or "five nines" – or about five minutes and 15 seconds of downtime per year (see table below). To achieve five nines, all components of the system must work seamlessly together. Software applications, compute resources, networking functionality, and physical data center services must be highly available to achieve five nines*
  - https://aws.amazon.com/blogs/publicsector/achieving-five-nines-cloud-justice-public-safety/



Availability	Downtime / Year	Downtime / Month	Downtime / Week	Downtime / Day
99.999%	5.256 Minutes	0.438 Minutes	0.101 Minutes	0.014 Minutes
99.995%	26.28 Minutes	2.19 Minutes	0.505 Minutes	0.072 Minutes
99.990%	52.56 Minutes	4.38 Minutes	1.011 Minutes	0.144 Minutes
99.950%	4.38 Hours	21.9 Minutes	5.054 Minutes	0.72 Minutes
99.900%	8.76 Hours	43.8 Minutes	10.108 Minutes	1.44 Minutes
99.500%	43.8 Hours	3.65 Hours	50.538 Minutes	7.2 Minutes
99.250%	65.7 Hours	5.475 Hours	75.808 Minutes	10.8 Minutes
99.000%	87.6 Hours	7.3 Hours	101.077 Minutes	14.4 Minutes

Safety by Design

This has been a standard approach in engineering for decades

- For example, apartment buildings have to use fire walls and fire doors to protect different areas of the building from fires
- Ships (and submarines) use bulkheads (walls) to separate the ship into separate fire- and water-proof zones
  - In the event of a hull breach or fire, these can be closed to keep the ship or submarine from sinking and personnel safe

The goal of safety by design is to design out the health and safety risks of a system

Image from the Daily Mail: https://www.dailymail.co.uk/home/moslive/article-1318268/HMS-Talent-Five-days-aboard-Britains-silent-warriors.html

Safety by Design

In software this idea can be seen in how the system reacts to “broken” applications

- If a program or window stops responding, it can usually be killed
- This is also used in servers where the web server process (HTTPD or HTTP Daemon) is separate from the SSH process (SSHD)
- This way, in the event of a website failure, an administrator can still access the system and restart the application or perform other remediations

Reminder: Accessing things like health care information or emergency services make uptime a safety-critical function

This is also seen in hardware through the use of redundant servers, network equipment, power and other required services

Data centers will generally have (at least) 2 power feeds, and 2 network feeds

- This allows for normal operations to continue if one of the services is disrupted
- Under heavy network load, it can also use both network feeds to ensure enough bandwidth is available

Cloud computing also uses approaches like Availability Zones (AZ’s) that allow for automatic failover in the event of a failure

- You do have to pay for this feature

Five-nines Uptime

Five-nines is the “standard” definition for high availability

There are three principles of system design to help achieve this.

From Wikipedia:

- *Elimination of single points of failure. This means adding or building redundancy into the system so that failure of a component does not mean failure of the entire system.*
- *Reliable crossover. In redundant systems, the crossover point itself tends to become a single point of failure. Reliable systems must provide for reliable crossover.*
- *Detection of failures as they occur. If the two principles above are observed, then a user may never see a failure – but the maintenance activity must.*
- *https://en.wikipedia.org/wiki/High_availability*

Five-nines Uptime

A properly implemented cloud architecture can help achieve high availability

Read: *Achieving "five nines" in the cloud for justice and public safety* by Ryan Reynolds

- https://aws.amazon.com/blogs/publicsector/achieving-five-nines-cloud-justice-public-safety/
- Pay attention to their architecture that supports the principles from the previous slide

Watch: Netflix: Multi-Regional Resiliency and Amazon Route 53

- This shows how Netflix maintains uptime using Amazon services
- https://www.youtube.com/watch?v=WDDkLOT8SCk

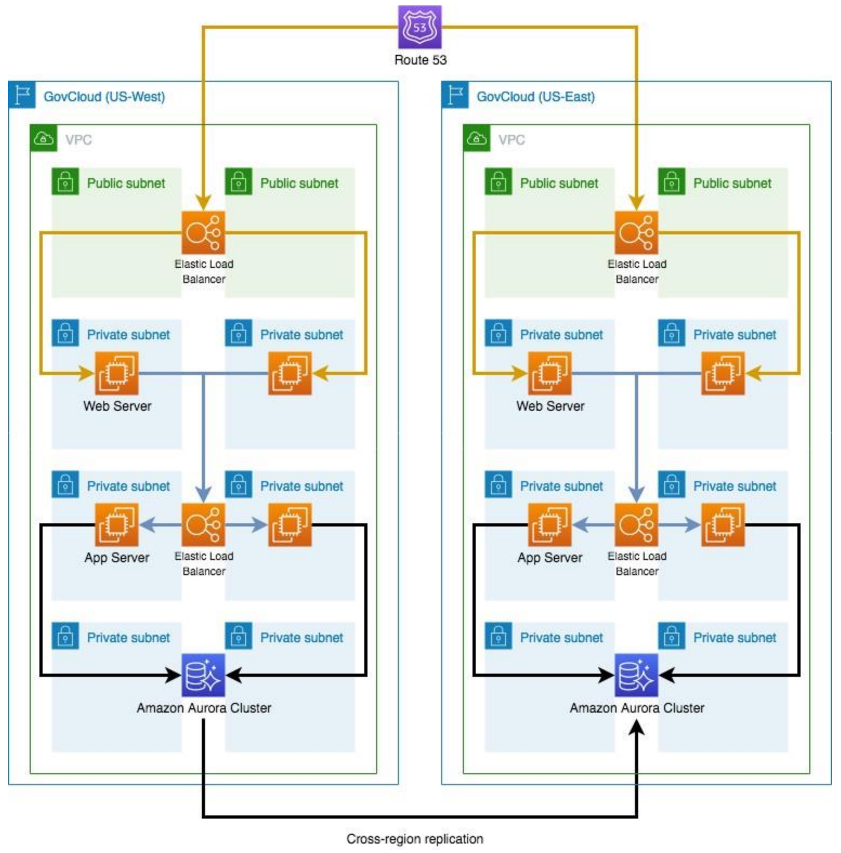


Image from: https://aws.amazon.com/blogs/publicsector/achieving-five-nines-cloud-justice-public-safety/

Active and Passive Safety

Active Safety

- A design method that seeks to avoid problems or failures from occurring (risk avoidance)
- For example, many vehicles, such as the Tesla Model 3, have an emergency breaking system that detects obstacles and applies the brakes
- In nuclear power systems, this can include engaging the control rods if temperatures get too high

Passive Safety

- A design method that minimizes losses or impact when a risk is realized
- Engages once a problem is detected – often designed to not require any human intervention
- In nuclear power systems, these measures could include:
  - Engineering the reactor so it can’t go critical with a mechanical failure (like a CANDU reactor)
  - Convection currents in the coolant being enough to cool the core in the event of power loss
- For vehicles this could include:
  - Air bags
  - Crumple zones
  - Seatbelts

Fail-safe Default in Computing

For access control, the fail-safe default is deny

- In the event of some kind of system failure, the safe thing to do is deny access when an error occurs
- “Deny by default” ensure confidentiality is maintained

Access decisions should be based on a user having permissions – not a user being denied permission

Failing Securely

From US-CERT:

When a system fails, it should do so securely. This typically involves several things: secure defaults (default is to deny access); on failure undo changes and restore to a secure state; always check return values for failure; and in conditional code/filters make sure that there is a default case that does the right thing. The confidentiality and integrity of a system should remain even though availability has been lost. Attackers must not be permitted to gain access rights to privileged objects during a failure that are normally inaccessible. Upon failing, a system that reveals sensitive information about the failure to potential attackers could supply additional knowledge for creating an attack. Determine what may occur when a system fails and be sure it does not threaten the system.

- https://www.cisa.gov/uscert/bsi/articles/knowledge/principles/failing-securely

Failure Modes

According to Viega and McGraw in their book *Building Secure Software: How to Avoid Security Problems the Right Way*:

- *Any sufficiently complex system will have failure modes. Failure is unavoidable and should be planned for. What is avoidable are security problems related to failure. The problem is that when many systems fail in any way, they exhibit insecure behavior. In such systems, attackers only need to cause the right kind of failure or wait for the right kind of failure to happen. Then they can go to town.*
- For example, what happens if a credit card reader fails?
  - If the chip fails, it can default to swiping
  - If the swiping fails, it can default to manual entry
- Credit cards used to require other information
  - Sometimes you needed to show an ID to use a card
  - Had to call your credit card company if you were making a large purchase
  - What is the failure mode?
    - The credit card companies simply accept the minor amount of fraud that happens
    - In return for this risk, they get higher customer convenience and goodwill

Failure Modes

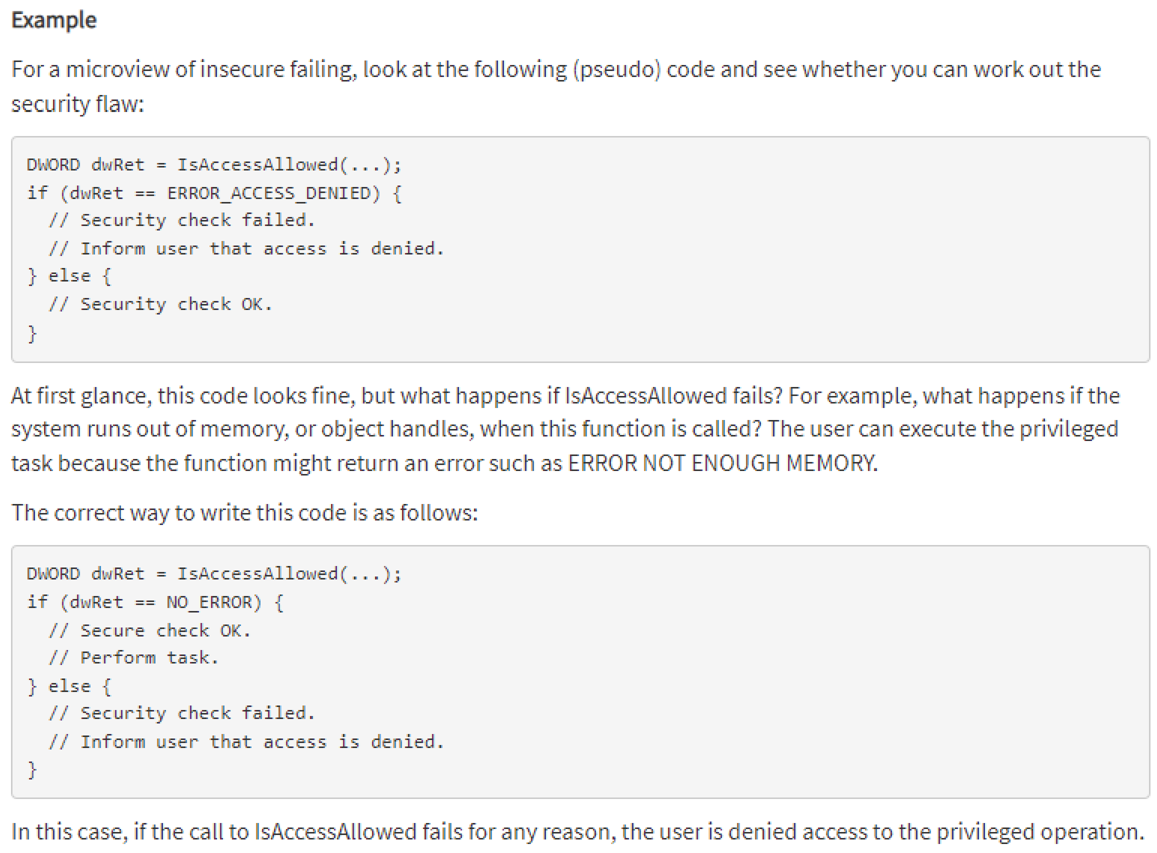
From Howard and LeBlanc in *Writing Secure Code, Second Edition* regarding “Plan on Failure”

As I've mentioned, stuff fails and stuff breaks. In the case of mechanical equipment, the cause might be wear and tear, and in the case of software and hardware, it might be bugs in the system. Bugs happen—plan on them occurring. Make security contingency plans. What happens if the firewall is breached? What happens if the Web site is defaced? What happens if the application is compromised? The wrong answer is, "It'll never happen!" It's like having an escape plan in case of fire—you hope to never have to put the strategy into practice, but if you do you have a better chance of getting out alive.

This is related to the Zero Trust Security Model

- This is where you trust nothing – no user or system – inside or outside of your network
- Everything needs to authenticate
- Everything needs to be constantly verified
- ***Read***: https://www.akamai.com/our-thinking/zero-trust/zero-trust-security-model

Fail-safe Example



Example

For a microview of insecure failing, look at the following (pseudo) code and see whether you can work out the security flaw:

```
DWORD dwRet = IsAccessAllowed(...);
if (dwRet == ERROR_ACCESS_DENIED) {
  // Security check failed.
  // Inform user that access is denied.
} else {
  // Security check OK.
}
```

At first glance, this code looks fine, but what happens if IsAccessAllowed fails? For example, what happens if the system runs out of memory, or object handles, when this function is called? The user can execute the privileged task because the function might return an error such as ERROR NOT ENOUGH MEMORY.

The correct way to write this code is as follows:

```
DWORD dwRet = IsAccessAllowed(...);
if (dwRet == NO_ERROR) {
  // Secure check OK.
  // Perform task.
} else {
  // Security check failed.
  // Inform user that access is denied.
}
```

In this case, if the call to IsAccessAllowed fails for any reason, the user is denied access to the privileged operation.

Screenshot from US-CERT: https://www.cisa.gov/uscert/bsi/articles/knowledge/principles/failing-securely

Insecure Defaults

When it comes to consumer devices, a manufacturer runs into two problems

- Users want the latest features – and they want it now
- Devices need to be simple to set up and configure – or at least troubleshoot

Since companies exist to make a profit, that means that anything slowing down development or sales – like security – is a secondary priority (at best)

- Developing a secure system is one reason BlackBerry lost out to iPhone
  - They focused on security and use in corporate environments
  - However, everyone wanted something cool instead

The IoT is a prime example of an area that uses insecure defaults

- Hardcoded passwords, unencrypted communication, open ports, etc. means they are often exploitable
- ***Read***: https://www.venafi.com/blog/top-10-vulnerabilities-make-iot-devices-insecure

Default Password Reset Questions

You've probably seen password reset questions like:

- What is your mother's maiden name?
- What was your first car?
- What street did you live on in high school?

How many of those questions could be answered by close friends or family?

These are insecure default questions

One way to combat this is to use a fictional character as your answer

- If you picked Dorothy from the Wizard of Oz, the street would be Yellow Brick Road

This approach allowed a college student to hack Sarah Palin's Yahoo! Email account

- https://www.wired.com/2008/09/palin-e-mail-ha/

Least Privilege

As defined by OWASP:

The principle of least privilege recommends that accounts have the least amount of privilege required to perform their business processes. This encompasses user rights, resource permissions such as CPU limits, memory, network, and file system permissions.

For example, if a middleware server only requires access to the network, read access to a database table, and the ability to write to a log, this describes all the permissions that should be granted. Under no circumstances should the middleware be granted administrative privileges.

- https://wiki.owasp.org/index.php/Least_privilege

From US-CERT:

Only the minimum necessary rights should be assigned to a subject that requests access to a resource and should be in effect for the shortest duration necessary (remember to relinquish privileges).

- https://www.cisa.gov/uscert/bsi/articles/knowledge/principles/least-privilege

This applies at all levels of an organization:

- **Program**: Code/Module
- **Component**: Supply Chain/Interactions
- **System**: Interactions between components/programs
- **Enterprise**: Everything – including polices and procedures

Least Privilege

From Microsoft:

If low-privileged processes are compromised, they will do a lot less damage to a system than high-privileged processes are capable of doing. Consequently, using a non-administrator account instead of an administrator account while completing daily tasks offers the user added protection against infection from a host of malware, external or internal security attacks, accidental or intentional modifications to system setup and configurations, and accidental or intentional access to confidential programs or documents

- https://docs.microsoft.com/en-us/previous-versions/tn-archive/cc700846(v=technet.10)

Unfortunately, it usually requires having elevated privileges to make changes to a system, which can cause regular users from having these privileges

- For example, you may need to run an installer as Administrator when using Windows
- Linux also requires root privileges for some installations and/or executing programs
  - For example, root is required to bind a program to a reserved port (ports below 1024) such as HTTPD

Least Privilege

From LeBlanc, David (2009-09-03). *24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them:*

While the word "privilege" has some very distinct meanings on different operating systems, the core of what they're trying to say is that the user account used to perform a given task should have as little access as possible.

If there are ways to further reduce the capabilities of the user account but still get the job done, then you should reduce the capabilities of the process.

A closely related problem is designing the application so that privileges are used for the shortest possible time.

The military security rule of “need-to-know” is also part of least privilege

- Even if someone has a clearance to view something, if they don’t also have a need-to-know, then they are not given access

Privilege Creep

As people move to different roles in an organization, they often keep their previous privileges

- Over time they end up with far more privileges than necessary, creating a security risk

Usually there is a well-defined process when someone leaves

- Accounts are disabled
- Security tokens are returned
- Badges are handed in

However, when someone changes roles, none of these occur

This is further complicated since people often assist their previous group for a few weeks while the workload gets distributed and sorted out

- They need to keep their privileges while they do this
- Not usually a set time for the privileges to expire

This makes auditing privileges a necessary security function

Least Privilege

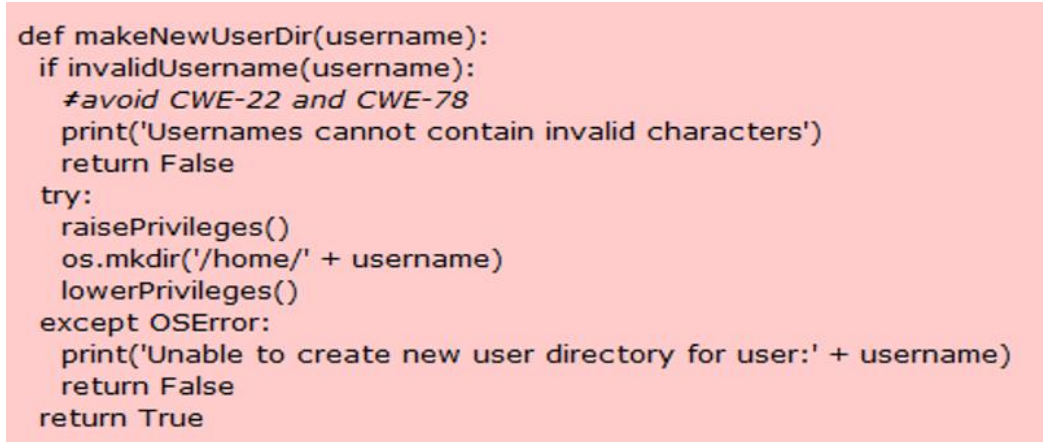
Having unnecessary privileges is common enough that it has its own CWE (the list of common software weaknesses)

- ***CWE-250: Execution with Unnecessary Privileges***
- **Description*:*** The software performs an operation at a privilege level that is higher than the minimum level required, which creates new weaknesses or amplifies the consequences of other weaknesses.
- **Extended Description**: New weaknesses can be exposed because running with extra privileges, such as root or Administrator, can disable the normal security checks being performed by the operating system or surrounding environment. Other pre-existing weaknesses can turn into security vulnerabilities if they occur while operating at raised privileges.
  Privilege management functions can behave in some less-than-obvious ways, and they have different quirks on different platforms. These inconsistencies are particularly pronounced if you are transitioning from one non-root user to another. Signal handlers and spawned processes run at the privilege of the owning process, so if a process is running as root when a signal fires or a sub-process is executed, the signal handler or sub-process will operate with root privileges.
- https://cwe.mitre.org/data/definitions/250.html

Least Privilege

In an application, higher privileges should only be used when necessary

- For example, creating a directory for a new user on Linux the program should:
  - Raise the privileges of the process
  - Create the directory
  - Lower the privileges back to the default
- The code should also be designed so a failure in the privileged operation should still result in the privileges being lowered
  - The code to the right will not lower the privileges if the mkdir operation fails
  - Python skips the rest of the try block when there is an exception raised



```
def makeNewUserDir(username):
  if invalidUsername(username):
    #avoid CWE-22 and CWE-78
    print('Usernames cannot contain invalid characters')
    return False
  try:
    raisePrivileges()
    os.mkdir('/home/' + username)
    lowerPrivileges()
  except OSError:
    print('Unable to create new user directory for user:' + username)
    return False
  return True
```

Least Privilege

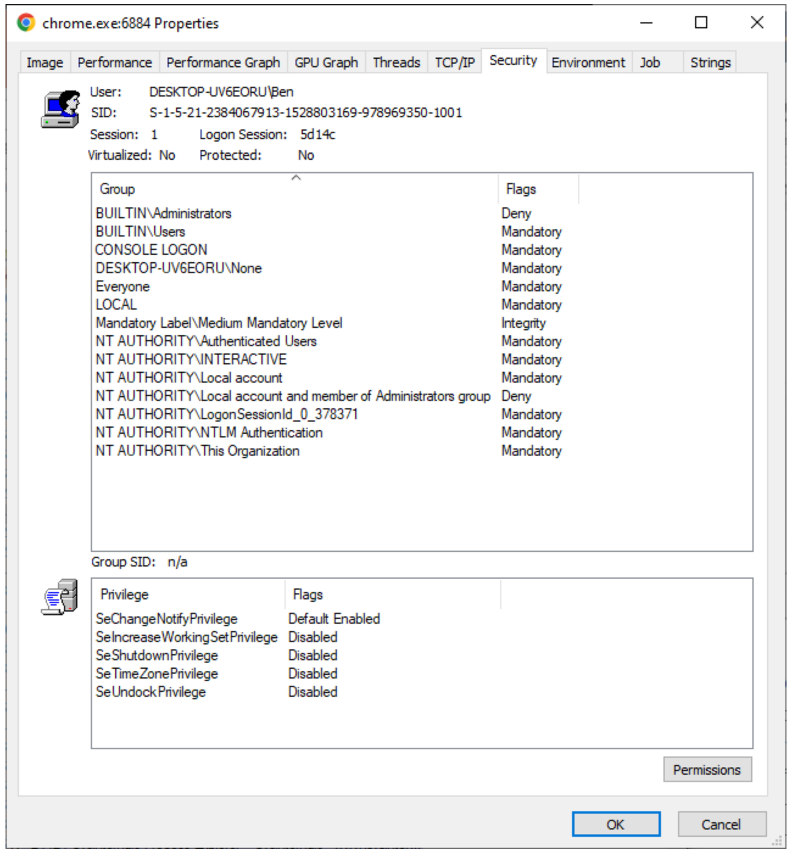
You can check if a program is using too many permissions when it is running

On Linux, BSD, or MacOS you can use the ‘ps’ command

- https://man7.org/linux/man-pages/man1/ps.1.html

On Windows, you can run the SysInternals Process Explorer application

- procexp64.exe
- You can go to a process, look at the properties, and go to the security tab to see the privileges that the process has



User: DESKTOP-UV6EORU\Ben
SID: S-1-5-21-2384067913-1528803169-978969350-1001
Session: 1 Logon Session: 5d14c
Virtualized: No Protected: No

Group	Flags
BUILTIN\Administrators	Deny
BUILTIN\Users	Mandatory
CONSOLE LOGON	Mandatory
DESKTOP-UV6EORU\None	Mandatory
Everyone	Mandatory
LOCAL	Mandatory
Mandatory Label\Medium Mandatory Level	Integrity
NT AUTHORITY\Authenticated Users	Mandatory
NT AUTHORITY\INTERACTIVE	Mandatory
NT AUTHORITY\Local account	Mandatory
NT AUTHORITY\Local account and member of Administrators group	Deny
NT AUTHORITY\LogonSessionId_0_378371	Mandatory
NT AUTHORITY\NTLM Authentication	Mandatory
NT AUTHORITY\This Organization	Mandatory

Group SID: n/a

Privilege	Flags
SeChangeNotifyPrivilege	Default Enabled
SeIncreaseWorkingSetPrivilege	Disabled
SeShutdownPrivilege	Disabled
SeTimeZonePrivilege	Disabled
SeUndockPrivilege	Disabled

Least Privilege

Mitigations can be applied in all phases of development

- Architecture and Design:
  - Design the code to run with the lowest privileges required
  - For example, the Apache web server often has the user “apache” instead of using root
- Implementation:
  - Ensure all user inputs are properly validated
  - Ensure users are unable to execute privileged code
  - As mentioned previously, ensure that any escalated privileges are dropped – even if there is an error
- Operation:
  - Run the code as an unprivileged user to verify it still works
  - Ensure the software runs properly on a properly configured and locked-down system

System Configuration

- There have been many approaches to properly secure a system
- The US Government has used:
  - The Federal Desktop Core Configuration (FDCC)
  - The US Government Configuration Baseline (USGCB)
  - Security Technical Implementation Guides (STIGs)
    - https://public.cyber.mil/stigs/
- There is also NIST SP800-70 Rev. 4 *National Checklist Program for IT Products – Guidelines for Checklist Users and Developers*
  - This described the National Checklist program
  - The checklist repository can be found here: https://ncp.nist.gov/repository
  - This repository also includes STIGs

Least Privilege

Unfortunately, it can be hard to set the “granularity” of permissions

For example, it can be difficult to determine what specific medical records a specific hospital employee needs to access

- Emergencies also have to be considered – the doctor or nurse present may need to be the one to access the records
- HIPAA sets standards for patient privacy and confidentiality

When Jussie Smollett was checked into a hospital, over 50 employees were fired for looking at his profile and records

- Is there a technical fix for this situation? Or does it rely on professional behavior?
- https://www.nbcchicago.com/news/national-international/northwestern-memorial-hospital-employees-fired-jussie-smsollett-records/2383/

Consider this issue in the context of a banking system

- How would the database be designed to minimize the privileges of individuals looking at account numbers, personal information, balances, and transactions?
- Would there need to be different bank account types?
- What kind of database users or roles would you create?
- How would you handle bank managers that might have multiple roles?

These questions might have different answers based on the type and size of a bank

- A local credit union might have a single type of role
- A global bank like J.P. Morgan might have entire divisions focused on personal accounts vs. business accounts
- They might even have specific people for accounts that are more than $1 billion

Summary

Lack of access should be the default

Access decisions should be based on permission rather than exclusion

A design or implementation mistake in a mechanism that gives explicit permission tends to fail by refusing permission, a safe situation that can be quickly detected

Every program and every user of the system should operate using the least set of privileges necessary to complete the job

This principle limits the damage that can result from an accident or an error

People don't generally question authority with respect to least privilege (immediacy and authority).

- Do executives really need access to everything?