



WIKIPEDIA  
The Free Encyclopedia

[Main page](#)

[Contents](#)

[Featured content](#)

[Current events](#)

[Random article](#)

[Donate to Wikipedia](#)

[Wikipedia store](#)

[Interaction](#)

[Help](#)

[About Wikipedia](#)

[Community portal](#)

[Recent changes](#)

[Contact page](#)

[Tools](#)

[What links here](#)

[Related changes](#)

[Upload file](#)

[Special pages](#)

[Permanent link](#)

[Page information](#)

[Wikidata item](#)

[Cite this page](#)

[Print/export](#)

[Create a book](#)

[Download as PDF](#)

[Printable version](#)

[Languages](#)



[Čeština](#)

[Español](#)

[فارسی](#)

[Français](#)

[Italiano](#)

[日本語](#)

[Polski](#)

[Русский](#)

[中文](#)

★A 5 more

[Edit links](#)

Not logged in [Talk](#) [Contributions](#) [Create account](#) [Log in](#)

Article

[Talk](#)

Read

[Edit](#)

[View history](#)

# Principle of least privilege

From Wikipedia, the free encyclopedia

*Not to be confused with [Rule of least power](#).*



This article **needs additional citations for [verification](#)**. Please help [improve this article](#) by [adding citations to reliable sources](#). Unsourced material may be challenged and removed.

*Find sources:* ["Principle of least privilege"](#) – [news](#) · [newspapers](#) · [books](#) · [scholar](#) · [JSTOR](#) (*April 2019*) ([Learn how and when to remove this template message](#))

In [information security](#), [computer science](#), and other fields, the **principle of least privilege (PoLP)**, also known as the **principle of minimal privilege** or the **principle of least authority**, requires that in a particular [abstraction layer](#) of a computing environment, every module (such as a [process](#), a [user](#), or a [program](#), depending on the subject) must be able to access only the information and [resources](#) that are necessary for its legitimate purpose.<sup>[1][2]</sup>

## Contents [hide]

- [Details](#)
- [History](#)
- [Implementation](#)
- [Similar principles](#)
- [See also](#)
- [References](#)
- [Bibliography](#)
- [External links](#)

## Details  [[edit](#)]

The principle means giving a [user account](#) or process only those privileges which are essential to perform its intended function. For example, a user account for the sole purpose of creating backups does not need to install software: hence, it has rights only to run backup and backup-related applications. Any other privileges, such as installing new software, are blocked. The principle applies also to a personal computer user who usually does work in a normal user account, and opens a privileged, password protected account (that is, a [superuser](#)) only when the situation absolutely demands it.

When applied to [users](#), the terms *least user access* or *least-privileged user account* (LUA) are also used, referring to the concept that all [user accounts](#) at all times

should run with as few [privileges](#) as possible, and also launch [applications](#) with as few privileges as possible.

The principle of least privilege is widely recognized as an important design consideration in enhancing the protection of data and functionality from faults ([fault tolerance](#)) and malicious behavior ([computer security](#)).

Benefits of the principle include:

- Better system stability. When code is limited in the scope of changes it can make to a system, it is easier to test its possible actions and interactions with other applications. In practice for example, applications running with restricted rights will not have access to perform operations that could crash a machine, or adversely affect other applications running on the same system.
- Better system security. When code is limited in the system-wide actions it may perform, vulnerabilities in one application cannot be used to exploit the rest of the machine. For example, Microsoft states “Running in standard user mode gives customers increased protection against inadvertent system-level damage caused by “[shatter attacks](#)” and [malware](#), such as [root kits](#), spyware, and undetectable [viruses](#)”.<sup>[3]</sup>
- Ease of deployment. In general, the fewer privileges an application requires the easier it is to deploy within a larger environment. This usually results from the first two benefits, applications that install device drivers or require elevated security privileges typically have additional steps involved in their deployment. For example, on Windows a solution with no [device drivers](#) can be run directly with no installation, while device drivers must be installed separately using the Windows installer service in order to grant the driver elevated privileges.<sup>[4]</sup>

In practice, there exist multiple competing definitions of true least privilege. As [program complexity](#) increases at an exponential rate,<sup>[citation needed]</sup> so do the number of potential issues, rendering a predictive approach impractical. Examples include the values of variables it may process, addresses it will need, or the precise time such things will be required. Object capability systems allow, for instance, deferring granting a single-use privilege until the time when it will be used. Currently, the closest practical approach is to eliminate privileges that can be manually evaluated as unnecessary. The resulting set of privileges typically exceeds the true minimum required privileges for the process.

Another limitation is the granularity of control that the operating environment has over privileges for an individual process.<sup>[5]</sup> In practice, it is rarely possible to control a process's access to memory, processing time, I/O device addresses or modes with the precision needed to facilitate only the precise set of privileges a process will require.

## History [\[ edit \]](#)

---

The original formulation is from [Jerome Saltzer](#).<sup>[6]</sup>

Every program and every privileged user of the system should operate using the least amount of privilege necessary to complete the job.

Peter J. Denning, in his paper "Fault Tolerant Operating Systems", set it in a broader perspective among four fundamental principles of fault tolerance.

Dynamic assignments of privileges was earlier discussed by Roger Needham in 1972.<sup>[7][8]</sup>

Historically, the oldest instance of least privilege is probably the source code of *login.c*, which begins execution with [super-user](#) permissions and—the instant they are no longer necessary—dismisses them via *setuid()* with a non-zero argument as demonstrated in the [Version 6 Unix source code](#).<sup>[9]</sup>

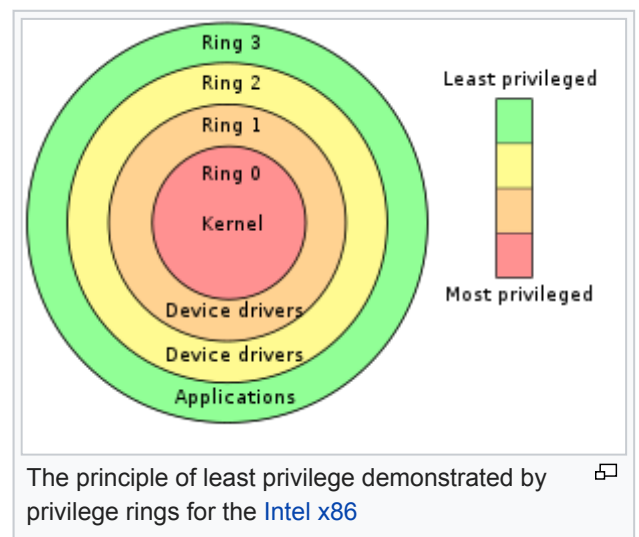
## Implementation <sup>[edit]</sup>

The [kernel](#) always runs with maximum privileges since it is the [operating system](#) core and has hardware access. One of the principal responsibilities of an operating system, particularly a multi-user operating system, is management of the hardware's availability and requests to access it from running [processes](#). When the kernel crashes, the mechanisms by which it maintains [state](#) also fail. Therefore, even if there is a way for the [CPU](#) to recover without a [hard reset](#), security continues to be enforced, but the operating system cannot properly respond to the failure because it was not possible to detect the failure. This is because kernel execution either halted or the [program counter](#) resumed execution from somewhere in an endless, and—usually—non-functional [loop](#).<sup>[citation needed]</sup> This would be akin to either experiencing [amnesia](#) (kernel execution failure) or being trapped in a closed maze that always returns to the starting point (closed loops).

If execution picks up after the crash by loading and running [trojan code](#), the author of the trojan code can usurp control of all processes. The principle of least privilege forces code to run with the lowest privilege/permission level possible. This means that the code that resumes the code execution—whether trojan or simply code execution picking up from an unexpected location—would not have the

ability to perform malicious or undesirable processes. One method used to accomplish this can be implemented in the [microprocessor](#) hardware. For example, in the [Intel x86](#) architecture the manufacturer designed four (ring 0 through ring 3) running "modes" with graduated degrees of access—much like [security clearance](#) systems in defence and intelligence agencies.<sup>[citation needed]</sup>

As implemented in some operating systems, processes execute with a *potential privilege set* and an *active privilege set*.<sup>[citation needed]</sup> Such privilege sets are



inherited from the parent as determined by the semantics of [fork\(\)](#). An [executable file](#) that performs a privileged function—thereby technically constituting a component of the [TCB](#), and concomitantly termed a trusted program or trusted process—may also be marked with a set of privileges. This is a logical extension of the notions of [set user ID](#) and [set group ID](#).<sup>[[citation needed](#)]</sup> The inheritance of [file privileges](#) by a process are determined by the semantics of the [exec\(\)](#) family of [system calls](#). The precise manner in which potential process privileges, actual process privileges, and file privileges interact can become complex. In practice, least privilege is practiced by forcing a process to run with only those privileges required by the task. Adherence to this model is quite complex as well as error-prone.

## Similar principles <sup>[[edit](#)]</sup>

---

The [Trusted Computer System Evaluation Criteria](#) (TCSEC) concept of [trusted computing base](#) (TCB) minimization is a far more stringent requirement that is only applicable to the functionally strongest assurance classes, viz., B3 and A1 (which are *evidentially* different but *functionally* identical).

Least privilege is often associated with [privilege bracketing](#): that is, assuming necessary privileges at the last possible moment and dismissing them as soon as no longer strictly necessary, therefore ostensibly reducing fallout from erroneous code that unintentionally exploits more privilege than is merited. Least privilege has also been interpreted in the context of distribution of [discretionary access control](#) (DAC) permissions, for example asserting that giving user U read/write access to file F violates least privilege if U can complete his authorized tasks with only read permission.

## See also <sup>[[edit](#)]</sup>

---







- [User Account Control](#)
- [Capability-based security](#)
- [Compartmentalization \(intelligence\)](#)
- [Confused deputy problem](#)
- [Encapsulation \(object-oriented programming\)](#)
- [Need to know](#)
- [Privilege bracketing](#)
- [Privilege escalation](#)
- [Privilege revocation \(computing\)](#)
- [Privilege separation](#)
- [Protection ring](#)
- [setuid](#)
- [sudo](#)

## References <sup>[[edit](#)]</sup>







---

- <sup>^</sup> Saltzer & Schroeder 75
- <sup>^</sup> Denning 76
- <sup>^</sup> Jonathan, Clark; DABCC Inc. "Virtualization Guru Writes "User-mode is a Good Thing - Deployment to Locked-down Accounts without Security Elevation"<sup>[[↗](#)]</sup>.











Retrieved 15 Mar 2013.

4. <sup>^</sup> Aaron Margosis (August 2006). "Problems of Privilege: Find and Fix LUA Bugs" . Microsoft.
5. <sup>^</sup> Matt Bishop, *Computer Security: Art and Science*, Boston, MA: Addison-Wesley, 2003. pp. 343-344 cited Barnum & Gegick 2005 
6. <sup>^</sup> Saltzer, Jerome H. (1974). "Protection and the control of information sharing in multics". *Communications of the ACM*. **17** (7): 388–402. [CiteSeerX 10.1.1.226.3939](#) . doi:10.1145/361011.361067 . ISSN 0001-0782 .
7. <sup>^</sup> Roger Needham, *Protection systems and protection implementations*, Proc. 1972 Fall Joint Computer Conference, AFIPS Conf. Proc., vol. 41, pt. 1, pp. 571-578
8. <sup>^</sup> Fred B. Schneider. "Least Privilege and More"  (PDF).

## Bibliography [[edit](#)]

- Ben Mankin, *The Formalisation of Protection Systems*, Ph. D thesis, University of Bath, 2004
- P. J. Denning (December 1976). "Fault tolerant operating systems". *ACM Computing Surveys*. **8** (4): 359–389. doi:10.1145/356678.356680 .
- Jerry H. Saltzer, Mike D. Schroeder (September 1975). "The protection of information in computer systems" . *Proceedings of the IEEE*. **63** (9): 1278–1308. [CiteSeerX 10.1.1.126.9257](#) . doi:10.1109/PROC.1975.9939 .
- Deitel, Harvey M. (1990). *An introduction to operating systems*  (revisited first ed.). Addison-Wesley. p. 673. ISBN 978-0-201-14502-1. page 31.
- Sean Martin (April 2012). "Are security basics getting lost under the cover of cloud and mobile?" . *SC Magazine*.
- SANS Institute (May 2013). "20 Critical Security Controls"  (PDF). *SANS Institute*. Archived from [the original](#)  (PDF) on 2013-11-01.

## External links [[edit](#)]

- [Managing least privileges from the cloud by Monique Sendze](#) 
- [The Saltzer and Schroeder paper cited in the references.](#) 
- [NSA \(the one that implemented SELinux\) talks about the principle of least privilege](#) 
- [A discussion of the implementation of the principle of least privilege in Solaris](#) 
- [Tom's IT Pro: Most Organizations Unaware of Employees With Admin Rights](#) 
- ["Proof that LUA makes you safer" by Dana Epp](#) 
- [Applying the Principle of Least Privilege to User Accounts on Windows XP, by Microsoft](#) 
- [Privilege Bracketing in the Solaris 10 Operating System, Sun Microsystems](#) 
- ["Commercial enterprises are putting our critical infrastructure at risk" CSO](#) 
- [How to successfully implement the principle of least privilege](#) 

V · T · E	Object-capability security	[ <a href="#">hide</a> ]
<b>Concepts</b>	<b>Principle of least authority (POLA)</b> · Confused deputy problem · Ambient authority · File descriptor · C-list · Object-capability model · Capability-based security · Capability-based addressing · Zooko's triangle · Petnames	
<b>OS kernels</b>	Hydra · NLTSS · KeyKOS · EROS · CapROS · iMAX 432 · Midori · seL4 · Genode · Fuchsia	

<b>Programming languages</b>	<a href="#">Joule</a> · <a href="#">E</a> · <a href="#">Joe-E</a> · <a href="#">Cajita</a>
<b>Filesystems</b>	<a href="#">Tahoe-LAFS</a>
<b>Specialised hardware</b>	<a href="#">Plessey System 250</a> · <a href="#">Cambridge CAP</a> · <a href="#">IBM System/38</a> · <a href="#">Intel iAPX 432</a>

Categories: [Information theory](#) | [Computer security](#)

This page was last edited on 24 September 2019, at 16:55 (UTC).

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#). Wikipedia® is a registered trademark of the [Wikimedia Foundation, Inc.](#), a non-profit organization.

[Privacy policy](#) [About Wikipedia](#) [Disclaimers](#) [Contact Wikipedia](#) [Developers](#) [Cookie statement](#)

[Mobile view](#)

