

Implementation of the Least Privilege Principle on Windows XP, Windows Vista and Linux

L.Scalbert and P.S.Dowland

Centre for Information Security and Network Research,
University of Plymouth, Plymouth, United Kingdom
e-mail: info@cscan.org

Abstract

The least privilege principle can solve some of the problems in Operating System security. It consists of running programs with only the needed rights for the operations expected from them. Thus, the unexpected actions on a program such as an attack are blocked. This paper compares its current implementation on Windows XP (service pack 3), Windows Vista (service pack 1) and Linux (kernel 2.6).

Keywords

Least privilege principle, security by isolation, separation of privileges, Operating System Security, Linux Security, Windows Security

1 Introduction

1.1 Approaches in computer security

Modern Operating Systems are designed for being more and more secured. Besides adding new security mechanisms, four main approaches have been adopted for improving the security of Operating Systems (Rutkowska 2008). The first one consists of thinking the security from their design stage. This is called security by correctness. Methodologies such as the Microsoft's Secure Development Life-cycle are intended to reduce the number of bugs in code that might result in security holes. Development tools include now some heuristic algorithms for detecting security bugs. Think for example of the argument *-wall* on the *gcc* compiler.

Since bugs are (and will still be) unavoidable, a second approach called security by obscurity completes the first one. Even if a flaw is found on the code, it should not be exploitable (or at least difficultly exploitable). Some attacks target some components that are constants or located at the same place. Randomizing some elements in the code and their location in memory could make such an attack defeat. Microsoft Windows Vista includes for example Address Space Layout Randomization that places the elements of the kernel randomly on memory.

The security by reduction of the surface of attack consists of disabling the unnecessary programs and services on the computer. The purpose of this approach is to avoid a potential attacker from using them for attacking the system. Some

potentially dangerous programs are not installed by default out-the-box on the newest versions of the Operating Systems. The tool *tftp* is now more and more uninstalled out-of-the-box as it can be used by an attacker that got an access to the command-line interpreter and want then to download tools to go further (Johansson 2006).

The fourth approach, which the paper focuses more on, is the security by isolation. It consists of breaking down the elements of the system into smaller and independent parts. The set of actions that could be carried out on a computer system is commonly called the privileges. The principle of least privilege is a kind of security by isolation. Indeed, it breaks down the privileges into small individual parts. Then it makes sure that programs run only with the needed privileges for the action expected from them. The purpose of the least privilege principle is to prevent attacker (may be either a hacker or malware) from breaking the security of the system. The difficulty resides in defining properly the privileges so that programs have sufficient privileges for their actions but attackers not.

1.2 Privileges and users

Privileges are based on the Access Control mechanism that denies or grants the access of the users (the subjects) to the objects of system. Privileges are not actually always determined by Access Control, for example, under Windows, the right of shutting down the computer is for example determined by a policy instead of an Access Control List. Access Control Lists (ACL), which are attached to each object, determine which users can access an object or, in other words what the privileges of the users are on a given object.

Windows and Linux define basically two classes of user accounts. The super-users (called *Administrator* or *System* on Windows, *root* on Linux) have the full privileges on the system whereas the normal users have only a limited set of privileges. A user is actually represented by programs that run under its identity and inherit normally its privileges. In fact, it should inherit only the needed privileges of the users if the least privilege principle is applied. This is discussed later.

It is possible of performing a lot of malicious thing as a normal user such as deleting user files and installing a Trojan malware but as super-user it is possible to go a step further. Despite the Super-user has full access rights on the objects of system, it can also loads some programs (called *modules* on Linux, *drivers* on Windows) on the kernel-mode. Such a program, called *rootkit*, has the ability of taking over the mechanisms that are located on kernel-mode. Therefore, a rootkit can modify the Reference Monitor that is responsible for the Access Control. It can also hide processes and files related to a malware and even disable an antivirus.

Since everything is possible from a super-user account, the system is often targeted by attacks, called escalation of privilege attacks, which consists of getting an access to the privileges of the super-user from a normal user session. These attacks target generally the super-user programs that run from the computer start-up. Number of techniques such as the buffer overflow and shatter attacks involve redirecting the super-user program to a malicious piece of code, called *shellcode*. Since the

shellcode inherits the privileges of the super-user, it can open an access to the super-user account. However, bear in mind that some malware does not involve any complex attack. A malicious program may simply prompt the person behind the screen for elevating the privileges, which succeeds in most of the cases.

What would be desirable is to be preventing any attack and malware from compromising the system. Without changing the current model, it is possible to limit the privileges of the normal user and super-user to only to these that are necessary. The permissions of a program would therefore not be determined by the rights of a user but the privileges that are allowed to it. For example, if the privileges of a program are got by any manner, the attacker will not get the privileges of a user but only the one of the program.

Dispatching the privileges throughout only two users has also another disadvantage. Indeed, the term user is ambiguous as it can be either a person, the system itself or a program. The system does not make actually the difference between a real person and a program. *root* is used on Linux for running daemons but also by the person whom the duty is to administrate the system. A user may trust a real person but not a program as it might be a malware.

2 Windows XP

The privileges on Windows XP are not adroitly dispatched throughout the two kinds of user. Indeed, the privileges of the normal user do not allow using properly the system. A normal user cannot for example connect to a wireless network, view the calendar when it clicks to the clocks of taskbar... Furthermore, some programs have not been designed to work with a normal user account. The command *run as* can normally be used for getting the super-user privileges for executing administrating tasks. However, although the super-user privileges are needed for the elements of the control panel, this command is not available there. Since the normal user account is not quite usable, the super-user account *administrator* is by default used instead out-of-the-box. Thus, all actions that are performed on the computer are carried out as a super-user, which equals not handling the least privileges at all. Note that the privileges of *administrator* are actually a bit lower than the one of the super-user *system* but sufficiently high to install a driver into kernel-mode (and so a rootkit). As a consequence, *administrator* has full privileges on the system directly or indirectly.

On Windows XP (and also Vista), the privileges of the processes and threads are determined by their *Access Token*. Access Tokens contain the identifiers (called *Security Identifier* or *SID*) of users and groups that are used as subjects in the Access Control mechanism. Groups are basically a set of users that shares the same rights on objects. Thus, associating (respectively dissociating) a process with a group permits adding (respectively removing) it a set of object permissions. A method, called *impersonation*, can make some modification on the Access Token. Some SIDs can be added or removed for adjusting the privileges of a process or thread. However, the privileges cannot be set finely. The impersonation is actually used in most of cases for switching from the normal user privileges to the super-user privileges.

Some privileges are not based on Access Control. For example, the privilege of shutting down the computer is not but it still associated to some users. These privileges are set in the security policy. They are associated to the user and editable through the management console *secpol.msc* in *local policies* and then *local rights assignment*. If the administrator account needs to be used for maintaining the compatibility with applications, it should recommend then removing the privileges *load and unload device drivers* to the current administrator account and using another administrator account when installing device driver is necessary. This setting can prevent efficiently rootkit infections. Note this setting does not fortunately avoid a driver delivered with the Windows Operating System from being installed with the Plug and Play feature. Therefore, it is still to use for example USB storage key if this setting is applied.

3 Windows Vista

On Windows Vista, the privileges of the normal users have been reviewed. Normal user accounts are fully operational. The privileges can always be elevated to super-user privileges in a specific tasks need them.

Windows Vista implements the least privilege principle with the User Account Control feature. By default, programs run with privileges of the normal user. The system decides whether a program needs super-user privileges prior to its execution. If the program does not contain a header with the required level of privileges, a heuristic algorithm determines it instead. The case elevating the privileges is needed generally arises when a new software or device is going to be installed. A dialog box prompts then the computer user for elevating the privileges.

User Account Control avoids using the super-user privileges when it is not necessary. However to our opinion, it is not sufficient for fully implementing the least privileges principle. Indeed, the privileges are set to either one of two levels of privileges instead of the privileges the process really needs. It would have been better at least to design an intermediate level of privileges. Installing software should not require the same level of privilege as installing device drivers. Indeed, the driver might be in fact a rootkit. An installer program should instead run at an intermediate level of privileges that would prevent it from installing drivers. If a driver needed to be installed, another dialog box could be displayed for asking the user if it really wants to do so. The dialog box would need to be designed in a manner the user would understand that it going to install a driver for a new hardware (or may be for security software).

Our recommendation for preventing a driver from being installed without the user's consent is to disable the privileges *load and unload device drivers to the administrator* as recommended for Windows XP as well.

The user interface of User Account Control could have been designed in a better manner. The dialog boxes for elevating the privileges propose only two choices: "continue" or "cancel". The user does not even have the choice of running the application with restricted privileges in case it does not trust the application. If it

really needs to run the application, it will click “continue”, which will execute the application at its own risk.

A common complain about User Account Control is that too many dialog boxes request actions from users (Johansson 2007). Firstly, it was experienced that some installation programs triggered two or more dialog boxes although only one would have been necessary. Secondly, when navigating through the elements of the control panel, too many dialog boxes are displayed. If the authorization were kept temporarily, at least for the Microsoft’s elements of the Control Panel. The high number of UAC dialog boxes has certainly a perverse effect on the computer user. Indeed, the user can become so accustomed to them that it might reply automatically “continue” anytime it see one of them without thinking to the consequences (Johansson 2007). The dialog boxes can also become so annoying for the user that it might decide to disable completely UAC. That would be bad for the security as that would result in coming back to the privilege model of Windows XP.

Another bad point with UAC is that there is an algorithm decides that a program requires administrator privileges. If the system decides the elevation of privilege is needed, there is no way by default to run the application at a low level of privileges. Indeed, the User Account Control dialog box suggests only two choices *continue* or *cancel*. It is recommend that disabling the policy *Detect application installations and prompt for elevation* (in *secpol.msc*, *local policies* and then *security options*) in order to disable the algorithm. Thus, all the applications run at a low level of privileges unless specifically requesting them to be loaded at the high level of privileges.

Mandatory Integrity Control is the feature of Windows Vista whom the duty is to put boundaries between the different levels privileges. It is used as a part of the User Account Control mechanism but provides also two more security features. The first one consists of preventing the shatter attacks, which target the graphical windows of super-user process for escalating privileges, by introducing a kind of Access Control on graphical windows. The second one is more related to our discussion as it provides a way of executing program at low level of privileges. This feature is similar to a sandboxing mechanism. The processes that run at the low level of privileges can only access files in read-only mode (excepting the setting files that are related to the specific program) in order to keep their integrity. Internet Explorer is as far the only one application that takes benefit of the feature. Thus, a malware that come from Internet cannot alter system files and personal documents. Note that the application needs to be designed for running in this sandboxing mode.

4 Linux

By default on Linux, all actions are performed as normal user. Like on Windows, the privileges of the super-user can be obtained on demand. Linux shared also the same problem as Windows. As soon the privileges are elevated, everything could be done on the system, including installing a rootkit.

Some programming methods exist for limiting the privileges of the super-user program. If they are used, they can considerably limit the consequence of an

escalation of privilege attack. Indeed, in case the attack succeeds, the attacker just gets the privileges that were allowed to the targeted program.

The capabilities were introduced with the kernel 2.2. They “divide the privileges traditionally associated with super-user into distinct units that can be independently enabled and disabled” in a program. For example, the capability `CAP_SYS_MODULE` and `CAP_SYS_TIME` control respectively the permissions of loading kernel modules and changing the system clock. The best practice regarding the least privilege principle consists of allowing only the necessary capabilities to a super-user program. Note this is only efficient if the capability `CAP_SETPCAP` that controls the ability of changing privileges is disabled.

Note also the utility `lcap`, provided with Linux, is able to disable the capability for whole system. It can be used on a start-up script in `/etc/init.d` (or an equivalent directory depending of the Linux distribution) for removing some capabilities to root from the machine start-up. Thus, a simple script can disable `CAP_SYS_MODULE`, which results of preventing rootkit from being loaded. However, the script (or the `lcap` program itself) can be easily deleted in order to remove the protection at the next reboot. It can be recommend instead building a static Linux kernel, which does not support modules. This can be done by disabling the *Enable loadable module support* option during the configuration stage of the kernel compilation and then selecting all the need the needed kernel components. Note that selecting the right components that will be built with the kernel is not an easy task but the method is very efficient as the protection cannot be disabled (NB a rootkit can still be installed by patching the kernel).

Root Set UID programs have the particularity of being executed by a normal user but with the privileges of the super-user root. They could be therefore targeted by an escalation of privilege attacks. It is strongly recommended to use the capabilities for limiting the privileges of root Set UID programs. Another technique should be used as well. It consists of not using the identity of root when it is not necessary during the execution of the Set UID program. The identity can be dropped temporarily to the one of the normal user if it needs to be restored afterwards or permanently otherwise. However, the developer should not assume that the identity always drops as requested. Indeed, the capability `CAP_SETUID` controls the ability of changing the ability. If it is disabled, the identity does not change as expected.

Another way of implementing the least privilege principle is to implement Mandatory Access Control. This consists of denying by default any access from a subject to an object unless it has been explicitly authorized on a policy. SE Linux is one of its popular implementations on Linux. It does not replace the traditional Discretionary Access Control mechanism but completes it. Indeed, if the access is granted by the Discretionary Access Control, then the Mandatory Access Control checks whether the access is permitted by the policy. Since it would be impossible to include on a policy all the authorized combinations of accesses from subjects to objects, SE Linux implements *type enforcement*. A *type* is tagged to every subject and object. The policy defines actually which types of subject can access to which types of objects. If no rule concerns the access from a type of subject to a type of object, the access is denied. SE Linux makes also the difference between a real

person and a program and so adjusts the privileges to the situation. Thus, some actions like changing a password can be only performed by a real person and so not by a malware. However, the difficulty with Mandatory Access Control is the policy is difficult to set properly. Indeed our experience with SE Linux showed that, if the rules contained on the policy are too permissive, they are not efficient enough. And on the other side, if they are too strict, some accesses can unwillingly be denied. Note that SE Linux is only efficient if it is set in a way it cannot be disabled. However, this means once the SE Linux is applied it cannot be changed. This is only conceivable in a corporate environment where system images are deployed on every computer. Thus changing the policy can be done by changing a previous system image that does not contain SE Linux applied.

Linux has also a powerful command *chroot* that provides a way of executing program in a safe context. *chroot* does not actually protect the system by dropping the level of privileges the application run at. It acts as a sandbox mechanism, which means the application is executed in a virtual context separated from the real one. The program works on a virtual directory that is totally separated from the rest of the file system. Thus, the confidentiality and the integrity of the data present on the file system are assured. This sandbox mechanism can be used for running critical root applications on a separate context. Since the programs on the sandbox do not use the real file structure, it can be difficult to make some programs run on the sandbox. This is the limit the possibility of applications of *chroot*.

5 Conclusion

The privilege model of security of Windows XP is clearly obsolete in comparison to the Windows Vista and Linux security model. It is difficult to determine which architecture is better secured between Linux and Windows Vista. The fact is there is a lot of Linux distribution that does not all offer the same level of security.

We suggested some security settings for reducing the privileges of the user. Some of them can avoid the systems from being infected by a rootkit. All these settings can significantly improve the security of these systems. To our opinion, the Linux system has a security advantage. Indeed, Linux has the ability of controlling everything. Thus it is possible applying more finely settings that can significantly improve the privilege model of the system. The Windows system is more closed and some components look actually like black boxes.

We pointed also out two main weaknesses that are shared on the three systems. First of all, the privileges on the system are determined by only two levels of users. Some enhancements to this privileges model are available on the three systems (we suggested some of them). It would be necessary redefining entirely the privileges model. A solution would consist of breaking down the privileges into small entities and another of differentiating the super-user account that is used by the administrator from the one that is used by the system.

If some boundaries exist to isolate processes on the user-mode, no boundary exists on the kernel-mode. Therefore, the security model can be bypassed by rootkit by

inserting drivers into the kernel mode. This paper has suggested some tips for preventing rootkit infection. The best thing would be that Operating System designers place the device drivers on a separate layer on top of the kernel and make the kernel static. Midori, which is the Microsoft research project for developing the Windows's successor, uses such architecture (Worthington 2008). Since the Operating Systems that are designed in this way are totally different from the traditional Operating Systems, the old application cannot be re-used normally. The problem can nevertheless be sorted out by using virtualization technologies.

6 Reference

Johansson 2006. *Anatomy of a network hack: How to get your network hacked in 10 easy steps - webcast*. Microsoft (Ed) Journées Microsoft de la sécurité. Paris. <http://www.microsoft.com/france/vision/WebcastTechnet.aspx?EID=941a1463-43aa-49b0-b52c-789cae0b2569>

Johansson. 2007. Security watch the long-term impact of user account control. <http://technet.microsoft.com/en-us/magazine/cc137811.aspx>. [Accessed 25-08-2008]

Rutkowska. 2008. The three approaches to computer security. <http://theinvisiblethings.blogspot.com/2008/09/three-approaches-to-computer-security.html>. [Accessed 4-09-2008]

Worthington 2008. Microsoft's plans for post-windows os revealed. http://www.sdtimes.com/microsoft_s_plans_for_post_windows_os_revealed/about_cloudcomputing_and_mobiledevelopment_and_net_and_soasaas_and_softwaredevelopment_and_windows_and_microsoft/32627. [Accessed 18-09-2008]