Cybersecurity

Subject: Process security

The aim of this challenge is to learn and apply in practice the mechanisms used in Windows to protect the operating system against abuse by untrusted application processes. We will focus on protecting the application processes themselves against unwanted interactions with other (potentially dangerous) processes. The main goal of these mechanisms is to reduce the risk of possible malware infections spread in the Windows OS, hence their in-depth knowledge and the ability to use them correctly are very important for a safe use of that operating system.

1. Application processes security

1.1 Security and access tokens

The security parameters related to a process or thread form the *security context* of the processing. In MS Windows, the context is represented by the security token. It includes, among others, account, group and session identifiers, access rights, level of obligation (see section 1.2) and UAC virtualization status (see section 2). When a user logs in, the winlogon process creates an initial token and assigns it to the first process started for that user (i.e. userinit, by default). Newly created processes inherit the token from their parent process. When a process tries to access a protected object, its security token permissions are additionally confronted with the restrictions imposed on the object, stored in its *security descriptor* (containing, inter alia, ACL lists).

1.1.1 Impersonation

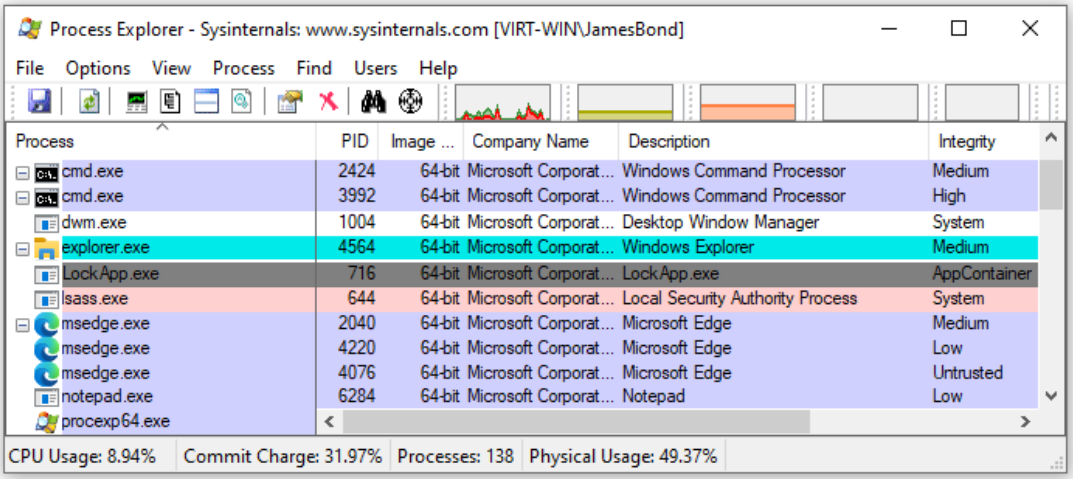
The impersonation mechanism allows the process thread to use a different token than its current one, thus allowing (temporarily) to perform activities that require different permissions. This is a functionality used in client-server applications (e.g. RPC) to temporarily "take over" the client's identity by the server in order to gain access to resources on behalf of the client (see the rights delegation mechanism). Any administrative user account processes can use impersonation to use the full administrative token.

The simplest case of using the impersonation mechanism is starting the application with the "Run as different user" option, or—in the command line—with the runas command.

1.2 Integrity levels

In Windows, there is an additional process isolation mechanism, even for processes operating under the same account. This mechanism, known as *Mandatory Integrity Control* (MIC), assigns a process an additional *integrity level*, which specifies the rights to modify sensitive elements of the operating system (e.g. system registry keys). The Security Reference Monitor (SRM) system module takes these levels into consideration when controlling access to objects (integrity level value is specified in their security descriptor, within a structure called *mandatory label*).

The Windows operating system distinguishes the following levels: untrusted (level 0, identified by SID S-1-16-0), low (1, S-1-16-0x1000), medium (2, S-1-16-0x2000), high (3, S-1-16-0x3000), system (4, S-1-16-0x4000) and protected (5, S-1-16-0x5000). For instance, processes running with elevated privileges (in administrative mode) get the high integrity level (i.e. level 3). Level 0 is assigned to processes that run under the Guests (*Anonymous*) group account.



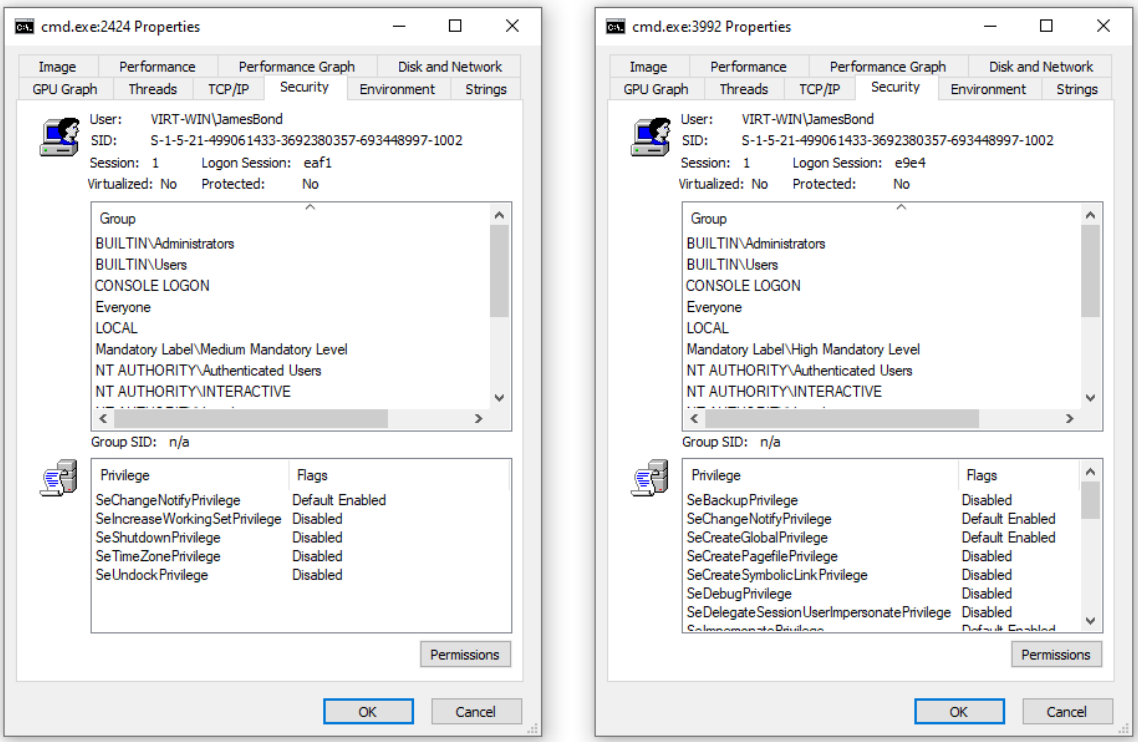
2. User Account Control (UAC)

UAC was introduced in Windows to limit the possibility of uncontrolled administrative operations. When this mechanism is active and a user with administrative rights logs into the system, 2 tokens are created for him: full—containing full administrative rights, and a *restricted token*—deprived of most security-critical rights (the so-called Filtered Admin Token).

2.1 UAC Elevation

If a user works under an administrative account and uses a limited Filtered Admin Token, he can "switch" to his full token using the UAC elevation mechanism. This mechanism is activated on demand, e.g. when starting the application with the "Run as administrator" option, in a system-controlled manner or by the application itself.

In the figure below, you can compare the permissions of two text command processor instances, on the left running normally and on the right running in "Run as administrator" mode:



2.2 File system and registry virtualization

Carelessly prepared applications, often modifying (usually unnecessarily) critical system components, have always been a common problem in MS Windows operating system. In order to protect against the catastrophic consequences (also from the security point of view) of such operations, and at the same time to enable the correct installation and operation of such applications, UAC introduces the ability to automatically virtualize the most critical parts of the file system and the system registry for applications running without administrative rights. Processes called in *impersonation* mode, all system services and applications containing an appropriate declaration in their UAC manifest (*requestedExecutionLevel*) are not being virtualized.

Areas that can be virtualized are `%ProgramFiles%`, `%ProgramData%` and `%SystemRoot%` excluding some specific subdirectories. Any executable files, including `.exe`, `.bat`, `.scr`, or `.vbs` are also excluded. Virtualized access is redirected to `%LocalAppData%\VirtualStore\`. In the registry, almost the entire `HKLM\Software` branch is virtualized, except for the most important ones, such as `HKLM\Software\Microsoft\Windows` or `HKLM\Software\Classes`.

💻 Resources:

Sysinternals Utilities (http://technet.microsoft.com/en-us/sysinternals/)

🕮 Problems to discuss:

- What mandatory label is given to objects created by a process operating with a high level mandatory (e.g., by an administrator)?
- What does “AppContainer” mean in the integrity level field?
- Please find information how MS Windows can bypass the process restrictions (mandatory levels and UAC), e.g. using the *Event Viewer* tool.
- What is `CreateRemoteThread()` system function used for?