

SECURITY BY VIRTUALIZATION

Ha Nguyen. ha.nguyen@unil.ch. tel: 41-21-6922207
University of Lausanne. January 16, 2006. 10:03

NEW APPROACH NEEDED

Time has proven that traditional solutions (native Windows security, Anti-virus...) are not very efficient. Some new approach is necessary to control system resources access beyond native Windows security. There exist two schools of thought: the behavior detection and the virtualization approach.

By studying actions of known malwares, it is possible to recognize and list commonly used behaviors. The difficulty of this behavior detection stems from the fact that Windows offers a large set of powerful but dangerous functionalities. They are "legal" to use. For example, many malwares install a keylogger to spy the user. But, some IM clients also do it in order to be aware of user activity status.

In the virtualization approach, only trusted programs run in the real Windows and the user is provided with the opportunity to execute untrusted or unknown applications in a virtual environment. Initially the virtual environment is an exact copy of the real one and supports efficiently (almost) all kinds of applications. As expected, with time this environment will deviate from the original one and probably gets corrupted or infected. Then, it can be restored easily as the real environment is still there. Although the idea looks very attractive, it should be challenging to implement it and third party products should take some time to develop and mature as they require lots of intervention inside the Windows intimate core.

We are now concentrating on the virtualisation approach as it appears a better solution. This memo has been written mostly from the point of view of an user, with few Windows internals knowledge. Some ideas may be impossible or difficult to implement. Nevertheless, this brainstorming effort will help us in choosing the right product that would fit our needs.

THE VIRTUALISATION APPROACH

File trust

From a security point of view, there are 3 types of files:

- ☞ Executable files: they contain codes that can be executed. When launched by some running process, an executable file yields a new process. Such a file may be launched several times in a row, each time creating a separate process.
- ☞ Active content files: they contain codes that can be executed. When launched, an active content file is executed inside the launching process. No separate process is created.
- ☞ Passive content files: they don't contain any executable code.

A Trusted file is the one that somehow we know for sure that it is not malicious by itself. For example, we can safely consider digitally signed files as Trusted. An Untrusted file is the one that comes from some unsure or unknown source and can be harmful, but we still insist on making use of it. It could be a malicious (or buggy) application file or an infected active content file (e.g. Word document containing a macro virus). Actually, the trust property concerns only executable and active content

files, but not passive content files.

Trust of executable file process

The following table describes the trust relationship between an executable file, its corresponding launched process and the launching process:

<i>Exec file</i>	<i>Launching process</i>	<i>Launched process</i>	<i>Remarks</i>
Trusted	Trusted	Trusted	
Trusted	Untrusted	Untrusted	
Untrusted	Trusted	Trusted	Launching process Trust_child= on
		Untrusted	Launching process Trust_child= off (default)
Untrusted	Untrusted	Untrusted	

When the launching process is Trusted and the executable file is Untrusted, it is safe to consider by default the launched process as Untrusted. But, it is useful to introduce a Trust_child property in order to ease installation of applications that involves many executable files:

- ∞ Mark the main installer as Trusted with Trust_child enabled.
- ∞ When the installer launches an executable file, the resulting child process will inherit trust properties from the installer: it becomes also Trusted and its Trust_child property enabled. If this process launches in turn an other executable file, again the same trust inheritance will be effective.

Active Content File and Process Contamination

An active content file contains instructions. When opened by an appropriate process, these instructions get executed inside the opening process. For example, when the WORD process (which is originally considered as Trusted) opens an Untrusted document that contains macros, there is a contamination risk as WORD is going to execute malicious actions as directed by the macros. The WORD process should then be changed to Untrusted in order to protect the real Windows.

Script files are considered active content files only when they are opened by an appropriate application. As there are many application-specific script languages, an application process is only contaminable by some specific format files. For example, an Untrusted BAT file can contaminate CMD, but is harmless when opened by WORD. So, the term “active content” is only meaningful to the concerned application. By extension, one can also include DLL's in the active content category, as they are executed inside the context of the calling process. Nevertheless, there exist many uncontaminable applications (e.g. NOTEPAD) as they don't interpret any scripting language nor call any DLL.

The following table describes the trust relationship between an active content file and the launching process:

<i>Active content file</i>	<i>Launching process</i>	<i>Remarks</i>
Trusted	Trusted->Trusted	
Trusted	Untrusted->Untrusted	
Untrusted	Trusted->Untrusted	Launching process Contaminable= on
	Trusted->Trusted	Launching process Contaminable= off
Untrusted	Untrusted->Untrusted	

Actually, the contamination risk is only possible if we allow Trusted processes to see Untrusted files (see also discussion below).

Resources

In Windows, the 2 most important resource groups are files and the Registry. Only Trusted processes can modify the “real” resources (the real environment). Untrusted processes are allowed to read “real” resources, but actually creates a “virtual” instance (the virtual environment) when trying to modify a real resource. We will use the terms Trusted and Untrusted to replace respectively the terms real and virtual when dealing with resources/environments, as this naming convention is more user-centric.

When a process reads a resource, the instance actually accessed is:

<i>Resource</i>	<i>Trusted process</i>	<i>Untrusted process</i>
Trusted instance only	Trusted instance	Trusted instance <i>note 1</i>
Untrusted instance only	See <i>note 2</i>	Untrusted instance
Trusted & Untrusted instances	Trusted instance	Untrusted instance

Notes:

1. Untrusted process sees the Trusted resource except when the resource Secret property is on.
2. One can choose to allow Trusted processes to see Untrusted resources (transparency principle), in which case it is required to implement the contamination mechanism. Alternatively, it is also possible to make Untrusted resources invisible to Trusted processes (opaqueness principle). Such a solution is simpler to implement (no contamination) but less flexible.

How can one differentiate Trusted files from Untrusted ones ? Windows XP SP2 creates an NTFS file alternate data stream (ADS) called Zone.Identifier to carry the security zone information when a file is downloaded with IE. In this way, when one attempts to execute the file, Windows will display a warning and prompt for confirmation. Using the ADS mechanism to flag file trust can be a solution. Nevertheless, it is not easy to deal with the case where there exists at the “same” location a Trusted and an Untrusted instances of the same filename.

This difficulty can be overcome by using an other approach where a dedicated folder (let's call it %UNTRUSTED%) holds all Untrusted files. In addition, the approach is independent of the filesystem type. Some preliminary definitions are necessary here:

- œ Expected location: during a file save operation, the process indicates the location where it expects to save the file. This location is also where the process expects to subsequently find the file.
- œ Visible location: the location where the process can see a file.
- œ Implant location: the location in the Windows filesystem where the file actually resides. The implant location of Trusted files is the same as in a “pure” Windows, whereas Untrusted files are actually located in %UNTRUSTED%.

The same approach can be used to implement the Trusted and Untrusted parts of the Registry.

Interoperability with other products

One can predict the following difficulties in regard to existing products:

- œ Difficulty to cope with Windows frequent patches.
- œ Conflict with traditional security applications (firewall, anti-virus, antispymware...).
Difficulty to make these applications see simultaneously Trusted and Untrusted resource instances.
- œ Difficulty to choose trust properties for certain applications e.g. the EXPLORER process, especially in the same desktop approach (see below).

SEPARATE DESKTOPS

The user sees 2 separate environments: a Trusted environment (the real Windows) and an Untrusted environment (the virtual Windows).

- œ The user explicitly switches from one environment to the other.
- œ Each environment has its own desktop and keyboard. The 2 environments can be simultaneously visible on the same physical screen, one environment being viewed as an application window by the other. Or at any time, all the screen is allocated to a single environment.
- œ Processes in each environment can't communicate with those in the other environment.

Trusted environment

- œ All processes running in this environment are Trusted (and remain Trusted). Their created resources are Trusted.
- œ For Trusted resources: expected location= visible location = implant location.
- œ To keep a Trusted file/folder confidential, one can explicitly declare it as invisible to the Untrusted environment (see the Secret property).

- œ Opaqueness principle: in order to avoid being contaminated by an Untrusted content, a Trusted process does not see Untrusted files at their expected location. So, an Untrusted normal.dot template file cannot fool a Trusted WORD process. Likewise, an application that has been installed in the Untrusted environment cannot be inadvertently launched. In other words, for a Trusted process, the expected location of Untrusted files is empty, but it can be useful to make the visible location not empty (= implant location).
- œ An utility is useful to make a Trusted copy out of an Untrusted file or to change file trust.

Untrusted environment

- œ All processes running in this environment are Untrusted. Their created resources are Untrusted.
- œ For Untrusted-instance-only (and dual instance, Untrusted instance being taken into account) resources: expected location= visible location. But the implant location %UNTRUSTED% remains invisible to the Untrusted environment. So, when an Untrusted process saves the file MyFile to C:\MyFolder, the file can be subsequently opened as C:\MyFolder\MyFile, even if its actual pathname (as far as Windows is concerned) is %UNTRUSTED%\MyFolder\MyFile.
- œ For Trusted-instance-only resources: expected location= visible location.
- œ The Untrusted environment has read-only access privilege to all Trusted resources, except the ones with the Secret property. If one attempts to read-modify-save a Trusted resource, actually an Untrusted instance is created and the Trusted instance remains unchanged.
- œ Cleanup operation: all Untrusted resources are deleted.
- œ Sync operation: Untrusted instances that have a corresponding Trusted instance are deleted. Then, the Untrusted environment will see Trusted instances.

Trust Visual Marking

In the Trusted environment, by way of the opaqueness principle, the situation is simple as the Trusted EXPLORER displays only Trusted files. On the other hand, it is highly desirable to be able to visually distinguish Trusted from Untrusted files while working in the Untrusted environment (see “Fig: Trust visual marking (variant 1)” as an example):

- œ File has only one instance: if Trusted, there is no special marking (e.g. T.txt). If Untrusted, surround with a red rectangle (e.g. Firefox Setup 1.0.6.exe).
- œ File has a Trusted and Untrusted instances: mark with a red triangle (e.g. Word.doc).



An other variant (variant 2) is also possible: when a file has 2 instances, they appear separately, the Trusted one without any special marking and the Untrusted one surrounded by a red rectangle.

Advantages

- œ Easy to use.
- œ 2 separate environments. Once in an environment, there is no risk to modify the other.
- œ The Trusted environment keyboard and screen are protected from “sniffing” by Untrusted processes.
- œ While in the Trusted environment, there is no risk to start an Untrusted application. Likewise, a Trusted process can't be unwillingly contaminated by an Untrusted file.
- œ An application installed in Trusted environment can be shared with the Untrusted environment, each environment having its own application settings and dynamically created variables.
- œ By design, there is no trust ambiguity problem (see below).

Disadvantages

- œ The user is responsible for choosing which environment to use. He can inadvertently access unsafe sites while in the Trusted environment. Partial solutions:
 - œ At login, go directly to the Untrusted environment.
 - œ Restrict access from the Trusted environment to only the internal network.

SAME DESKTOP

- œ The user sees a single desktop.
- œ Trusted and Untrusted processes use automatically and respectively the Trusted and Untrusted environments. No user action is required to switch from one environment to the other.
- œ Concerning resource visibility, the transparency principle applies to Untrusted processes, whereas transparency or opaqueness can be chosen for Trusted processes.

Trust Visual Marking

Even if the EXPLORER process is of Trusted type, it is highly desirable that it lets the user see all file instances and visually distinguish Trusted from Untrusted

instances. For example, one can visually differentiate the following cases (see “Fig: Trust visual marking (1)” as an example):

- ⌘ File has only one instance: if Trusted, there is no special marking (e.g. T.txt). If Untrusted, surround with a red rectangle (e.g. Firefox Setup 1.0.6.exe).
- ⌘ File has a Trusted and Untrusted instances: mark with a red triangle (e.g. Word.doc).

An other variant (variant 2) is also possible: when a file has 2 instances, they appear separately, the Trusted one without any special marking and the Untrusted one surrounded by a red rectangle.

The Trust Ambiguity Problem

In the separate desktop solution, all processes (including EXPLORER) and resources created in an environment have the same trust as the hosting environment. On the other hand, management of processes (e.g. EXPLORER) and resources in the single desktop approach is expected to be more complex.

What happen when one double-clicks on a file ? This action is equivalent to a Trusted process (the EXPLORER process) opening a file. In the variant 1 of trust visual marking, the following sequence would happen:

- ⌘ If the file is an application (executable file):
 - ⌘ Single instance: the single instance is opened and a process is created with the same trust as the single instance. For user convenience, it is also possible to override this default trust behavior by right-clicking to launch the instance as a process with the opposite trust.
 - ⌘ 2 instances: the Trusted EXPLORER process chooses to open by default the Trusted instance and a Trusted process is created. For user convenience, it is also possible to override this default trust behavior by right-clicking and launching the Untrusted instance (as an Untrusted process).
- ⌘ If the file is not an application, EXPLORER must first select the file instance then the associated application instance:
 - ⌘ File instance selection: select single instance or Trusted instance if there are 2 instances (EXPLORER selects by default the instance with same trust as EXPLORER process). Right-clicking to select the Untrusted instance is also possible.
 - ⌘ Associated application selection:
 - ⌘ Associated application has only 1 instance: a process is created with the same trust as the application, except in contamination case (application is Trusted and file is Untrusted and of active content type) where the process becomes Untrusted. This contamination occurs only with the transparency principle option where the Trusted application can see the Untrusted file. Otherwise (opaqueness principle), a “No associated application” warning message should pop up.
 - ⌘ Associated application has 2 instances: application instance with same trust as selected file is used to create a process also with same trust.

One can see that variant 1 requires lots of modifications to EXPLORER operations (open, copy & paste, drag & drop...) to cope with the dual instance ambiguity. In contrast, variant 2 is simpler to implement and easier to use: there is no ambiguity as the user chooses directly the desired instance. The following events happen with the file opening example:

- ☞ If the file is an application (executable file): a process is created with the same trust as the application file. For user convenience, it is also possible to override this default trust behavior by right-clicking to launch a process with the opposite trust.
- ☞ If the file is not an application, EXPLORER must select an appropriate associated application instance:
 - ☞ Associated application has only 1 instance: a process is created with the same trust as the application, except in contamination case (application is Trusted and file is Untrusted and of active content type) where the process becomes Untrusted. This contamination occurs only with the transparency principle option where the Trusted application can see the Untrusted file. Otherwise (opaqueness principle), a “No associated application” warning message should pop up.
 - ☞ Associated application has 2 instances: application instance with same trust as selected file is used to create a process also with same trust.

Concerning the copy & paste and drag & drop operations, the resulting file must have the same trust properties as the original one.

If we look at applications that take some input files then generate output files (e.g. compression utility, compiler/linker), output files should be Untrusted unless all input files are Trusted. To approximate this behavior:

- ☞ These applications are declared Trusted with the Contaminable property enabled and are contaminable by any Untrusted file (regardless of the file format).
- ☞ The transparency principle is adopted so these applications (that are started as Trusted) can see Untrusted input files.

Advantages

- ☞ Usage is more natural (no manual switching of environments).
- ☞ More tolerant to human errors.

Disadvantages

- ☞ The Trusted and Untrusted environments share I/O devices (screen, keyboard...). Even if a spyware runs as Untrusted, it can see the window content of a Trusted process.
- ☞ Implementation is more difficult (see the trust ambiguity problem) especially when the transparency principle is adopted.