Application Security through Sandbox Virtualization

Liberios Vokorokos, Anton Baláž, Branislav Madoš

Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Letná 9, 042 00 Košice, Slovakia
liberios.vokorokos@tuke.sk, anton.balaz@tuke.sk, branislav.mados@tuke.sk

Abstract: This article is aimed at the creation of a secure, virtualized system sandbox environment at the level of the respective applications. The proposed sandbox model allows us to generate a secure environment for various untrusted applications and resolve potential security incidents, such as zero day vulnerabilities. The resulting work is a functional sandbox within the MS Windows operating system, which protects the system against potentially hazardous applications. The sandbox has a minimal impact on the semantics and the time of the executed program and provides an efficient sandbox configuration interface.

Keywords: system sandbox; operating system security; virtualization; security policy

1 Introduction

Security threats are closely related to the overall level of software security in computer systems. When writing applications, various errors occur; these are then eliminated by testing and real-life usage. Insufficiently tested applications cause a number of problems, especially operating system infections, caused by various kinds of malware. These errors are often found only post-mortem. The attacker managed to abuse the fault. Subsequently, the system vendor creates a patch and updates the application. However, the attack has already happened and the system may have remained infected. Another problem is that the end user must react immediately to the various security measures, since (s)he has a direct interest in the operating system security policy or the application itself. There is a significant amount of malware on the Internet – claiming to be useful and necessary, but containing adware, viruses or other malicious code. Antivirus solutions are not capable of detecting these programs on time and reliably at all times.

The method of using system sandboxes is based on the idea of preferring prevention instead of detection in case of security. In general, there are multiple categories of sandboxes and system containers – each of these has its own specific

use. The most widely used are virtual systems isolating the individual applications, including the operating system itself. This type of – full – virtualization degrades the system in terms of performance and it has a significant administration overhead too. In this article, we describe a lightweight application container, which would resolve the above security threats and have minimal impact on the system.

2 Sandboxes and Virtualization

The system sandbox comprises a number of programs. The specific definition depends on the solution type. Hoopes defined the sandbox universally as software providing a monitored and controlled environment, in which no unknown program may cause any damage to the system it is running in [1].

A more specific definition states that a sandbox allows applications to be executed so that these applications are not allowed to read or write the data beyond the specified path, i.e. beyond the sandbox. In a broader sense, one has to add the control and allocation of operating system resources to this definition, such as network services, hardware management, low-level access, etc.

The concept of controlled execution was first time introduced by R. Wahbe et al. in the context of software bug protection [2]. The term coined for this technique – sandboxing – stands for a method of isolating untrusted modules executed in the same address space as the trusted modules, with a minimal impact on execution time.

Security in general, system security and the field of system sandboxes are all closely related to virtualization. Virtualization is a set of procedures and techniques, which allow the separation of the available system resources into multiple environments. The virtualized environment may be tailored to the needs of the users, it is easier to use or to hide the details (such as the physical location of the hardware resources) from the users. Virtualization is performed by a virtual machine (VM). A virtual machine represents an independent environment and/or a software implementation of a machine executing programs, similarly to a physical computer [3].

Virtualization is possible at various levels: from the whole computer down to its individual hardware components, or even a certain specific software environment. The various virtualization types may differ in the level of isolation, the resource requirements, the execution costs, scalability and flexibility. In general, the “closer” the virtualization level is to hardware, the more the virtual machines are isolated and separated from the host computer; however, those require also more resources and are less flexible. The "farther" the virtualization level is from the hardware, the more powerful and scalable these virtual machines are.

When used correctly, virtualization has a number of advantages – Hoopes summarized these as follows: *consolidation*, *reliability* and *security* [1]. Consolidation includes a more efficient use of computers, simplifies migration to newer versions of various systems, significantly shortens time needed for development and testing and allows a single physical platform to host various operating systems. Reliability has become a priority of many IT companies, more than ever. A system error in the virtual machine does not affect the other parts of the system on the same hardware platform – this ensures the reliability and the consistency of the system as a whole. Technology providing protection against application faults provides isolation from security faults. If the security of a specific part of the virtual machine is compromised, it may be terminated at any time.

In accordance with the characteristics of the respective virtualization methods, the system sandbox is a specific virtual machine. Similarly to virtualization excluding consolidation, reliability and security are the basic motives of the creation of system sandboxes. A sandbox and/or its virtualization layer may be created at various levels. Isolation of multiple virtual machines or the virtual machine and the host environment makes virtual machines an efficient platform to execute potentially error-prone and untrusted applications. While using virtual machine technology, it is a common requirement to execute potentially dangerous operations in the created VM, which is an instance of the operating system (OS) host environment. This can be achieved by using a virtual machine at hardware level. However – as it was stated above – this is not too efficient, because virtual machines are fully isolated from each other and each of these has a separate OS running in it. The initialization of such a machine has a high overhead in terms of resources and has a start-up delay too. Considering the distance of the virtualization layer from hardware, in most cases software designated as a sandbox is implemented as a layer between the operating system and the individual processes. In this solution, we focus on the isolated environments implemented at operating system level, depending on the OS virtualization model, in accordance with the scheme depicted in Figure 1.

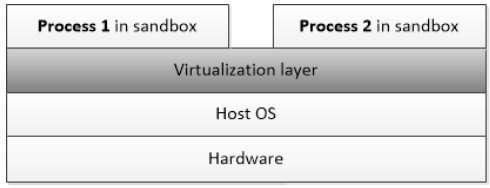

Figure 1
The position of the sandbox virtualization layer within the OS

3 Implementing the Security Policy using a System Sandbox

Access to the individual system resources is implemented by means of a unified interface provided by the operating system. Therefore, the system sandbox must provide access control of the individual OS components within this security implementation. The most important OS components are:

- files
- the registry
- network interfaces
- the CPU
- I/O
- locks
- processes
- other

Seen from the aspect of the proposed solution of secure sandboxes in MS Windows, determined by the architecture of the OS itself, the key system resources are the following:

- Files: files contain all user and operating system data. Changes in the file system without the knowledge of the user could be fatal. Removed or modified system files would directly compromise OS security. Therefore, controlling file access is a must in case of any application container.
- The registry: the registry contains the configuration data of the operating system and the respective applications. Malware should never access or alter the important data stored in the registry. In spite of the registry being stored in files, access to it occurs by means of a separate interface. Therefore the registry, as a specific component of the OS must be adequately secured.
- Network interface: computer worms often contact remote servers, which coordinate their activities within the system. They try to steal confidential data and infect further computers. Controlling network access is therefore an equally significant factor in these cases.
- System resources: The ability to strictly limit the number of active processes, the usage of operating memory or the processor load may fully eliminate malware execution, or, eventually, minimize the damages and prevent the OS from being inaccessible. Controlled access to these system resources shall thus increase the isolation of processes in the sandbox.

3.1 File Access

The architecture used as a starting point of the proposed solution is the sandbox core created as a layer between the MS Windows operating system and the respective applications. Our concept of supporting virtualization at OS level is based on file virtualization and/or redirecting and constraining file access requests from the virtual machine to the local, root partition of the host OS. E.g. if a process in the V1 virtual machine tries to access a file with the path C:\foo\bar, the virtualization level may redirect the request to the file C:\V1\foo\bar. If a process in another environment (e.g. V2) wants to access C:\foo\bar, the path may be diverted to the file C:\V2\foo\bar, different from file \foo\bar in V1. This mapping is done transparently in the virtualization layer. The process accessing the file C:\foo\bar cannot tell it is in fact accessing a different file.

The targets of the file system access requests may be mapped to the directory containing the file system root, which will contain an equivalent tree structure, or a file container. This container will appear to the host system as a single file. A sandbox knowing the specification of the container may access all elements within the container and control the file access requests for the virtualized process transparently. As to the implementation, the file container may be a simple archive (e.g. a ZIP file), or it may contain a full and separate file system and make use of the existing file formats as VHD, VMDK, etc. specifications [17].

A relevant aspect of file virtualization is whether the application in the virtual machine "sees" the other files and disk volumes in the host environment. There are two different access types in this aspect:

- the process does not know the content of the host file system;
- the process may read arbitrary files in the host system, as part of the specified security policy.

In the first case the root directory changes fully, similarly to the *chroot* tool of Unix systems. This approach has advantages: absolute file isolation and a decreased performance overhead. On the other hand, all libraries and files required for the problem-free execution of the applications in the sandbox must be present in the virtual root directory. This may be a problem in the MS Windows systems, because it is not easy to find out and resolve all DLL dependencies. A further problem is loading duplicates of DLL libraries into the operating memory and wasting disk space. The other solution allows the virtualized process to read arbitrary processes in the host system. The redirected requests result in the following: each file accessed by the process is copied into the redirected directory. However, resource duplication has a cost in terms of performance, it requires space in the primary memory and also increases the overhead related to the initialization and removal of the sandbox. Therefore, the virtual machine shares the majority of its resources with the host and creates private copies of files in the virtualized directory only if it is necessary – upon file write or modification

operations. This approach is known as the copy-on-write mechanism [18]. When using the copy-on-write approach, we have to pay attention to an important fact. Files will be located at two places: in the host environment and in the sandbox environment. When a process running in the VM requests to read the content of a specific directory, the resulting data should contain the unification of the two locations, omitting the duplicate entries in the host directory.

Removing or renaming files initiated by the process running in the sandbox should not affect the host environment. Thus, if the sandboxed application tries to remove/rename a file located beyond the sandbox, the virtualization layer shall not remove the file. The process running in the sandbox shall not know that the file was not really removed. Upon a request of the same process of the specific sandbox to access the removed file, the virtualization layer shall report to the process that the file doesn't exist.

In the proposed solution we copy the file upon opening it with the write flag set. Unlike the copy-on-write approach, the process accessing the existing file is not diverted to a private directory until the first request to open a file with the write flag set appears. This approach has advantages: the virtualization layer overhead is low, which means also a simpler implementation. The disadvantage is wasting storage space and time required to perform a copy when opening the file. The request target does not use a file container but rather a separate directory, the content of which shall mirror the tree structure of the file system.

3.2 Registry Access

Malware, except for modifying files, usually accesses the Windows system registry too [19]. The registry is a hierarchic database containing system information and program settings. In addition to other things, the registry contains the configuration data of the operating system, including the keys required for automatic program execution [20]. Most programs write to the registry at least during installation. Due to all of these reasons, the registry is a critical part of the system, accesses to it must be controlled. Therefore, the sandbox catches the system calls to the registry and modifies the requested keys and values. The principle is the same as with file access, slightly altered, with a lower overhead, when the whole key and value tree is moved to the sandbox [9].

In the proposed solution, registry filtering is based on copying keys on write operations and/or when altering the values. It is analogous to the copy-on-write method. The key or value is created in the sandbox only upon creating the keys and their values or upon their change. The scheme of accessing files is depicted in Figure 2.

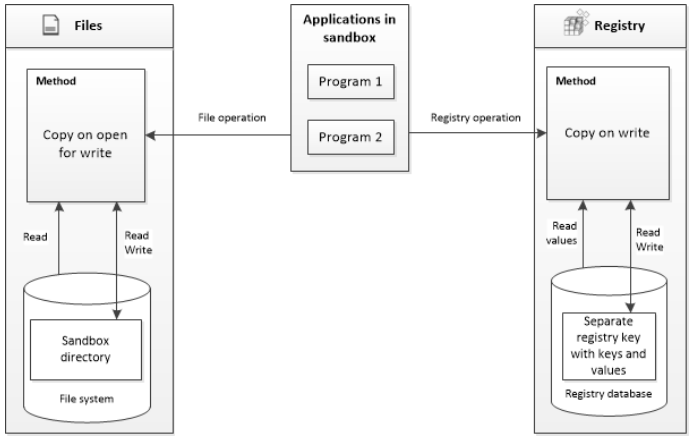

Figure 2
File and registry access

3.3 Network Isolation

Network isolation – in a broader sense – is the creation of an independent environment for an arbitrary process communicating on the local network and the Internet [21]. Any such process gets a unique virtual IP address within the system. The support for this kind of isolation is not widely used in system sandboxes. Most often it appears in solutions providing full virtualization implementations at OS-level, where it is not desirable to have the virtual machines appear under the same IP address and interfere with the network operation of each other. Another approach, emulating a separate network subsystem in the operating system for each process is a very good way to analyze malware behavior. The implementation of such a mechanism is a complex and tedious job, therefore it appears primarily in hardware virtualization, when a separate operating system is running in the virtual machine.

Network isolation – in the more specific sense – means controlling and limiting access during communication on the local network and the Internet. The sandbox should block Internet access and prevent sending out confidential information, especially when the program has access to the host files. The proposed solution blocks network access by means of the firewall integrated into the Windows operating system.

3.4 Controlling System Resources

The control of system resources provides a means of protection from overloading and losing access to the system resources. The sandbox may control the use of the file system, the registry, the operating memory and the CPU load. Depending on the settings it then applies the limitations.

Moreover, in addition to resource management and as part of the privilege system, the sandbox may also contain a policy to modify the system settings (icons, fonts, screen settings, power management, etc.), logging out, system restart and shutdown. The sandbox contains an implementation of controlling resources and system settings by means of Windows Job objects. This technology allows the individual processes to be assigned to the respective containers and apply the requested properties. Job objects allow the processes to be handled as uniform units. The set of constraints may be specified within a single object, which forces the use of the constraint on each process in the process file. An overview of the methods used in network isolation and system resource and setting control is depicted in Figure 3.

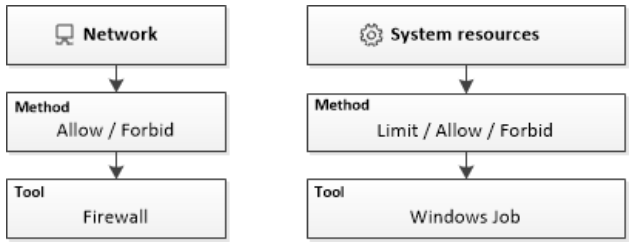

Figure 3
Network isolation, system resource and setting control

4 Sandbox Architecture

The proposed sandbox consists of three independent parts. The core of the sandbox is executed at the OS kernel level. The main module consists of a software driver. It filters file, registry and process operations. A Windows service loads the driver into the system and implements network isolation. The communication with the user occurs using a third, separate program. The user controls the sandbox by means of a graphical interface, which communicates with the driver and the service. The basic architecture of the sandbox is depicted in Figure 4.

The article contains a detailed description of the sandbox driver architecture. This component is the core of the sandbox.

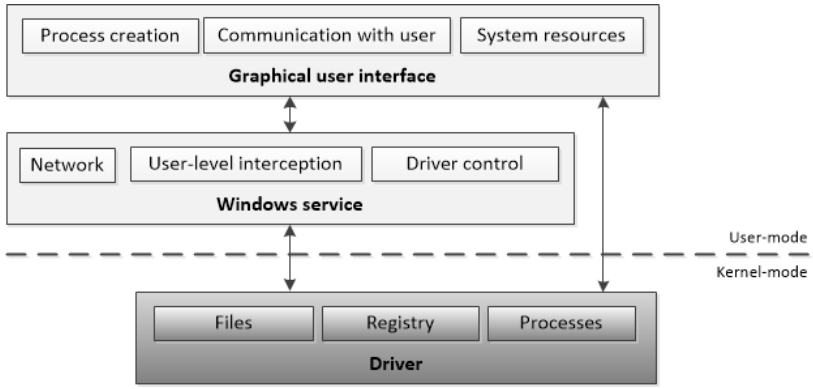

Figure 4
Basic sandbox architecture

4.1 Minifilters

With the advent of Windows XP SP2, Microsoft created a model to ease the creation of file system filters as an alternative to the previously used legacy filter model. In this the drivers – minifilters – are managed by the filter manager (FM). Most system calls issued by the applications, sent to the drivers are in I/O request packet (IRP) format. In the minifilters, the IRP requests are first processed by the FM. The minifilter registers the callbacks of the requested operations, which are then called by the FM. The FM encapsulates the IRP into a structure called CallbackData and sends the structure gradually to the individual minifilter instances, just as it was in the case of the legacy filters. A minifilter instance is an abstraction of the minifilter at the specific disk partition.

An important part of the filter architecture is context support. The context is a structure defined by the minifilter to store arbitrary data associated with the filter manager objects. The context may be associated to minifilter instances, files, file objects, open file streams, disk partitions and transactions.

4.2 Files

The elementary mechanism providing the file name modification is based on setting the reparse status flag (STATUS_REPARSE) of the corresponding IRP operation in the pre-operation. This indicates that the I/O manager discards the current IRP and initiates a new IRP with the file object containing the file location in the sandbox domain. In addition to altering the path and the state of the IRP operation the kernel must finish the current operation to make sure that the IRP structure does not access the file system. This is achieved by returning the appropriate state from the pre-operation to the filter manager. An important problem arises with this redirection operation: How does the filter know, which

request has been redirected and which has not. With the advent of Windows Vista, each creation operation may contain a reference to an ECP (Extra Create Parameter) context. This context is retained even after finishing the IRP, when the I/O manager adds a reference to the context to the new IRP structure. The kernel uses the ECP context to mark the redirected operation in the pre-operation phase and to mark the creation operation initiated by the sandbox in the redirection between disk partitions in the whole driver. The ECP context is only available in the creation pre-operation and post-operation phases. For a simple identification of the request and further required data, the ECP context is transformed to a context associated with an open file stream (SHC) in the creation post-operation phase, where the file object already exists (unlike the pre-operation phase), therefore this context may be associated with it. The kernel associates the SHC only with the file object representing a file in the sandbox and it is available only to our minifilter, unlike the ECP context available for all drivers taking part in the creation operation.

4.2.1 Opening Files

When opening/creating a file, each application tells the OS how to behave if the file exists/does not exist. Windows knows six different dispositions, as stated in Table 1.

Table 1
File opening dispositions

Disposition	**The file exists**	**The file doesn't exist**
CREATE	failure	created
OPEN_IF	open	created
OPEN	open	failure
OVERWRITE	open and overwritten	failure
SUPERSEDE	replaced	created
OVERWRITE_IF	open and overwritten	created

The program also specifies the requested file access: read, write, erasable, etc.. The disposition and access specifications are the main elements in selecting when the kernel should redirect the request, copy the file to the sandbox domain or ignore the request. The situations, which may arise in conjunction with the specified disposition and the selected flag are depicted in Figure 5.

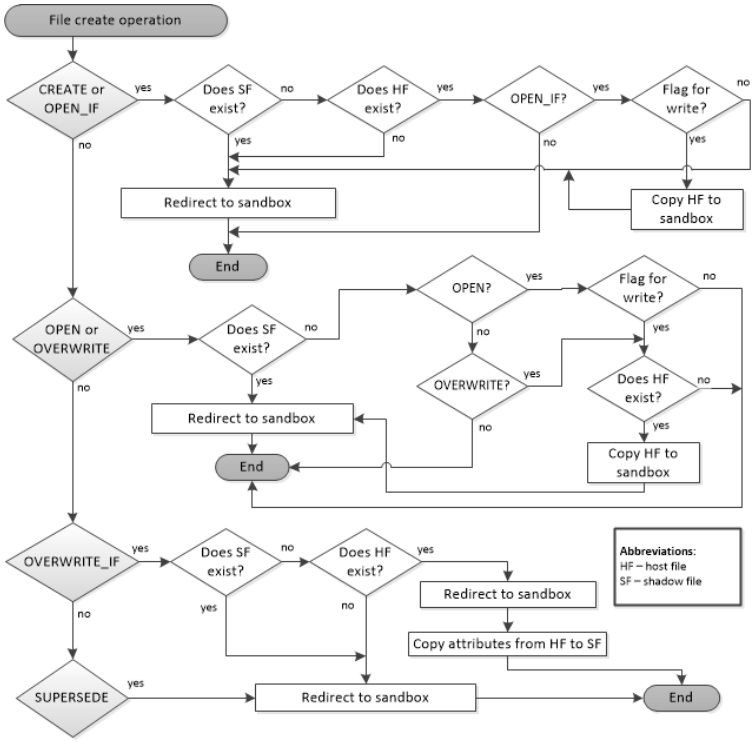

Figure 5
Request flow upon opening a file

4.2.2 Removed Files

The sandbox core registers the information setting operation (IRP_MJ_SET_INFORMATION) and selects the disposition structure. Then, the whole path to the file is stored in the table of removed files. If the file comes from the sandbox, further execution is left over to the I/O manager. If the file comes from the host environment, the core initiates its own information setting operation with the file removal disposition flag unset. Depending on the result of the initiated operation, the kernel will apply the final state and finish the original information setting operation.

The kernel manages a separate table with the list of removed files for each logical sandbox. To keep the state of the sandbox even after operating system shutdown and restart, the table contents are stored in a file and read into the memory upon the creation of the logical sandbox.

The table implementation influenced the decision process in the redirection specified above. If the file is being opened, the first step is to check the existence of the file path in the table. If the file is in the table, the request is denied, the operation is finished. Otherwise, the above process is executed.

4.2.3 Renamed Files and Hard Links

The sandbox controls file renaming and hard link creation. The kernel is primarily interested in the first three phases of these operations. Opening a file to rename; opening the target file with the new name; and catching the IRP called IRP_MJ_SET_INFORMATION associated with the file object of the first operation. The first phase is executed in the sandbox kernel in accordance with the file opening algorithm. The second phase is a little different from the standard opening algorithm, because the target file does not exist. Actually, the request contains the flag indicating that the calling process tries to open the target of the renaming operation or to create a hard link. The sandbox kernel receives the flag and forwards the target to the sandbox domain without further analyzing the request. The third phase is different, depending on whether the operation is aimed at renaming a file or creating a hard link.

When renaming a file, the filter retrieves source file information in the third phase. If the source file comes from the sandbox domain, the kernel is sure that the host cannot be modified and the rest of the operation is left over the I/O manager. If the source file is with the host, the kernel retrieves information of the target file, copies the source file from the host to the sandbox domain under a new name, adds the source file to the table of removed files, stops further execution and indicates success. Renaming within disk partitions is not possible due to the principle of file system independence; such operations are performed internally as copy and deletion operations.

4.2.4 Directory Enumeration

The result of the enumeration is a set of files and directories from the queried directory. Since the proposed sandbox uses the copy-on-write technique, the files may be at two locations. If the isolated process requests information on the items residing in a specific directory, the result must contain a unification of both locations, omitting the duplicate entries and the entries listed in the table of deleted files. Duplicate entries represent files and directories located both in the sandbox domain and the host domain. The result shall not contain the entries from the host domain in these cases.

4.3 Registry

The registry filter is a driver filtering the registry calls. The configuration manager (CM) implementing the registry allows filtering any process call related to the registers. Similarly to files, the register filter receives pre-notifications and post-notifications. The driver catches all notifications. If the given notification is irrelevant to the manager, it must at least return the execution to the configuration manager. If the operation is filtered during the pre-notification phase, the requested processing may be done and one of the three status types is returned:

- *success (STATUS_SUCCESS) – the CM processes the registry operation and then initiates a post-notification containing the required data and the resulting status of the operation;*
- *failure – the operation is rejected and the process error status is returned immediately, without calling the post notification and the assistance of the CM;*
- *bypass (STATUS_CALLBACK_BYPASS) – all processing must be done by the driver and the registry operation terminates with success and without initiating post-notification and CM assistance.*

As with minifilters, registry filters have contexts too. The context may be associated with registry filters, registry notifications and registry objects representing open keys. Figure 6 shows the principle of filtering pre-notifications in the registry filter, including specific notification types.

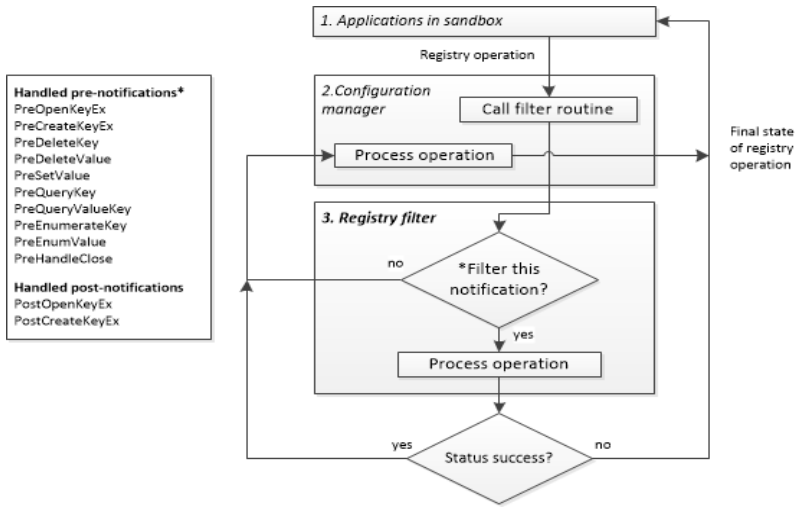

Figure 6
The principle of filtering pre-notifications of registers in the sandbox

4.4 Processes

The driver, the minifilter, the registry filter or other components catch all system calls by default. The aim of the solution is to filter specific programs. The sandbox kernel must determine the context of the currently executed thread for all callbacks and associate them with the processes. Each process in the operating system has a unique numerical identifier, the PID. Windows allows each driver to register a callback initiated upon process creation and termination. The client part of process execution sends the PID to the sandbox, which stores it in the global process table. The callback contains not only the PID of the process being created,

but also the PID of the parent process, which created the process. If the isolated process creates another process, the sandbox core identifies the new process using the parent PID and adds it to the process table.

5 Evaluation of the Solution

To evaluate the solution, we have performed a functionality test of the proposed secure sandbox. For creating snapshots of the file system and the registry database we used System Explorer and OS Forensics software. The sandbox was tested in a virtual machine with an updated 32-bit Windows 7 OS without antivirus software. The tests could be performed on the MS Windows XP and MS Windows Vista too. We have purposefully downloaded and executed 11 malicious applications in the proposed sandbox using Internet Explorer. We have selected malware according to its current popularity. According to the statistics of the company G Data Software AG, the most widespread are various forms of adware. Trojans, spyware and worms are the largest threats, therefore we have included them in the test, in spite of the fact that these are less widespread globally.

We have tested the following threats (the names come from the virus database of the company ESET):

1. ZeroAccess (called Trojan-Dropper.Win32.Smiscer.hf in the Kaspersky database) is one of the most widespread and most dangerous rootkits. This malware is part of various key generators and cracks. When executed in a 32-bit OS, ZeroAccess overwrites the existing driver and loads itself into kernel space. The rootkit usually hides in a hidden partition. In a 64-bit OS, ZeroAccess does not contain code executed in kernel space. In this case it hides in the Global Assembly Cache (GAC) of the .NET framework.

   Result: The rootkit could not be executed in the proposed sandbox. The system was not infected.

2. Win32/Injector.AAKO – carrier of various kinds of malware, trojans catching passwords and other confidential information from the infected computer. This specific variant named AAKO does also copy the payload to the C:\Windows\InstallDir directory and creates a value in the registry database, in the HKLM\Software\Microsoft \Windows\CurrentVersion\Run\HKLM key.

   Result: The program was executed in the sandbox. The registry and file sandbox caught all executed changes. After terminating the sandbox, all malicious processes were terminated. The system was not infected.

3. Win32/TrojanDropper.VB.OJG – changes the Internet connection settings in the registry and attempts to switch off the firewall. Further, it also copies an executable with a random name to the TEMP directory, executes it and writes malicious data in the GDIPFONTCACHEV1.DAT file, normally used as a font cache. In addition to this, it copies an svchost.exe executable file into the C:\ProgramData directory, pretending to be a service of the operating system.

Result: The sandbox successfully caught all executed changes and processes, without infecting the OS.

4. MSIL/Injector.CUXtrojan – behaves similarly to the previous program, but pretends to be a Java update in the registry.

Result: The sandbox successfully caught the executed changes and processes.

5. Adware.Relevant.CC – currently the most widespread adware currently, part of many freeware programs. It analyzes user activity on the computer (including Internet surfing), it adds an exception to the firewall and from time to time it opens a window to fill in the user data.

Result: unsuccessful installation in the sandbox.

6. Win32/Spy.Zbot.YW – a trojan stealing passwords and other confidential information. It also serves as a backdoor, it may be controlled remotely.

Result: This trojan was not execute in the sandbox. Upon start, the program terminated with an error, so the system was not infected.

7. Win32/Kryptik.BFCO – Ransomware, encrypting the file system and offering unencryption for money. In addition to this, it switches off Windows Firewall, Windows updates, Windows Defender and other antivirus software. It is automatically started by means of registry values and it was copies itself to various places.

Result: This malware was successfully executed and started encrypting the files. However, the original files remained untouched; all changes remained limited in the sandbox. The sandbox contained a number of encrypted files. The host system (files and the registry) remained uninfected and the OS settings remained unchanged. During the test it was not possible to execute the task manager and terminate the processes manually. After terminating the sandbox, the file encryption processes were terminated and the task manager could be switched on again.

8. Win32/Spy.Hesperbot – the known banking trojan spreading by email as an attachment named zasilka.pdf.exe, most active in Turkey and the Czech Republic. Win32/Spy.Hesperbot records keystrokes, creates screenshots, records video using the web camera of the computer and creates a remote proxy.

Result: This malware started successfully, it stored system information in an encrypted file and copied its code into the newly created attrib.exe and explorer.exe files, then terminated. The active network isolation prevented it from communicating with the server. The host system was not modified. After terminating the sandbox, both infected processes were terminated.

9. Win32/Dorkbot.B – a very widespread computer worm spreading through Skype, Facebook Chat, Twitter messages, etc. This worm steals passwords and contains a backdoor.

Result: None of the tested variants of this worm could be started in the sandbox.

10. Win32/LockScreen.ALY – another example of ransomware.

Result: Without administrator privileges it was not possible to execute the malware in the sandbox. With administrator privileges and active limitations the sandbox caught the changes.

The chart in Figure 7 shows the results. The chart shows a total of 11 experiments, since the behavior of malware no. 10 was significantly dependent on the method of testing.

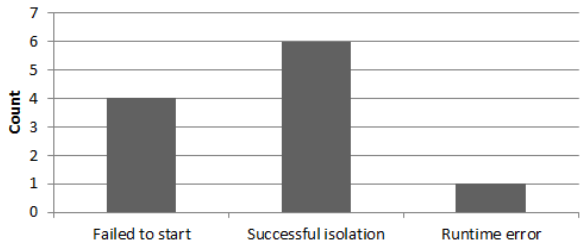

Figure 7
Security test results

Malicious code did not manage to infect the host system in any of the cases. Some code did not even start in the sandbox. If the malware started, in some cases it was capable of performing its malicious activities; however, after terminating the sandbox, it was terminated successfully. Network access constraints and mainly the restrictions in the number of active processes and system settings play a significant role in sandbox efficiency.

To evaluate the additional computing costs of the solution, we have measured both the initial and repeated program start-ups. We have tested the start-up of the Firefox 28 Internet browser, WinRAR 5 compression software, Foxit Reader 6 PDF reader and Emule 0.5 file transfer software. When measuring the initial start-up, we have restarted the OS after each attempt. The measurement results are in Figure 8. The average initial start-up overhead was 8%, subsequent start-up overhead amounted to 17%. The smaller percentage of the initial startup is related to the increased time of loading the libraries from the hard drive.

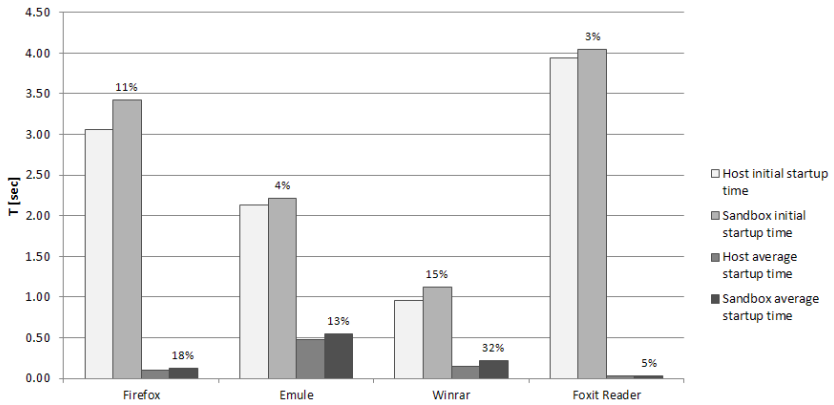

Figure 8

Initial and subsequent startup times of the programs in the sandbox

Standard usage of programs degrades the execution time of programs minimally. Increased time costs are presented in case of programs which are heavy on I/O operations. All other system calls are executed natively, which is an advantage in comparison with hardware-level virtual machines. The whole performance penalty depends on the application in use; it may range from 5% to 20%.

Conclusion

The goal of this work was to design a system sandbox in the MS Windows system, providing security in case of the executed applications. The proposed architecture was implemented using standard methods offered by the Windows operating system. The behavior of the sandbox was primarily based on the copy-on-write model, which creates a redundant isolated OS environment. This article presents the design of the sandbox and the main parts of the implementation in the MS Windows OS. The execution of the respective applications is not functionally limited in the sandbox. The applied copy-on-write approach at the level of OS virtualization has a smaller performance penalty than full virtualization; therefore the applications are executed without greater latency in the sandbox. The performed security test showed the functionality of the proposed sandbox model in solving current security problems, where the user is forced to react to various security issues. A drawback of the solution is the possible waste of storage space. In future, the virtualization of inter-process communication should be improved and the network isolation with deeper integration.

Acknowledgements

This work was supported by the Slovak Research and Development Agency under the contract No. APVV-0008-10 and project KEGA 008TUKE-4/2013: Microlearning environment for education of information security specialists.

References

[1] Hoopes John: Virtualization for Security including Sandboxing, Disaster Recovery, High Availability, Forensic Analysis, and Honeypotting, Burlington 2009

[2] Wahbe Robert et al: Efficient Software-based Fault Isolation, ACM SIGOPS Operating Systems Review, Vol. 27, No. 5, New York 1993, pp. 203-216

[3] Smith James Edward, Nair Ravi: Virtual Machines: Versatile Platforms for Systems and Processes (The Morgan Kaufmann Series in Computer Architecture and Design), San Francisco 2005

[4] Barrett Diane, Kipper Gregory: Virtualization and Forensics: A Digital Forensic Investigator's Guide to Virtual Environments, New York 2010

[5] Moya del Barrio Victor: Study of the Techniques for Emulation Programming [online], Barcelona School of Informatics, Barcelona 2001, Available on: http://personals.ac.upc.edu/vmoya/docs/emuprog.pdf [quoted on April 12, 2013]

[6] Calzolari Federico: High Availability using Virtualization: Doctor's thesis, University of Pisa, Pisa 2009, p. 83

[7] Nakajima Jun, Mallick Asit: Hybrid-Virtualization – Enhanced Virtualization for Linux, Proceedings of the Ottawa Linux Symposium, Vol. 2, Ottawa 2007, pp. 87-96

[8] Chaudhary Vipin et al: A Comparison of Virtualization Technologies for HPC, Proceedings of 22nd International Conference on Advanced Information Networking and Applications, IEEE Computer Society, Buffalo 2008, pp. 861-868

[9] Yu Yang: Os-Level Virtualization and its Applications: Doctor's thesis, State University of New York at Stony Brook, Stony Brook 2007, p. 120

[10] Shan Zhiyong et al: Facilitating Inter-Application Interactions for OS-Level Virtualization, ACM SIGPLAN Notices – VEE, Vol. 47, No. 7, New York 2012, pp. 75-86

[11] Laadan Oren, Nieh Jason: Operating System Virtualization: Practice and Experience, ACM Proceedings of the 3rd Annual Haifa Experimental Systems Conference, New York 2010, pp. 1-12

[12] Mihályi Daniel, Novitzká Valerie: Towards the Knowledge in Coalgebraic Model of IDS, Computing and Informatics, Vol. 33, No. 1, Bratislava 2014, pp. 61-78

[13] Kampert Paulus: Taxonomy of Virtualization Technologies: Thesis, Delft University of Technology, Faculty of Systems Engineering Policy Analysis & Management, Delft 2010, p. 93

[14] Jakubčo Peter, Šimoňák Slavomír, Ádám Norbert: Communication Model of emuStudio Emulation Platform, Acta Universitatis Sapientiae: Informatica, Vol. 2, No. 2, Cluj-Napoca 2010, pp. 117-134

[15] Kamp Poul-Henning, Watson Robert: Jails: Confining the Omnipotent Root, Proceedings of the 2nd International SANE Conference, Maastricht 2000, pp. 116-127

[16] Ma Kun, Yang Bo, Abraham Ajith: A Template-based Model Transformation Approach for deriving multi-tenant SaaS applications, Acta Polytechnica Hungarica, Vol. 9, No. 2, Budapest 2012, pp. 25-41

[17] Singh Jasmet. et al: An Application Sandbox Model based on Partial Virtualization of Hard-Disk and a Possible Windows Implementation, International Journal of Computer Applications, Vol. 7, No. 57, New York 2012, pp. 16-21

[18] Kasampalis Sakis: Copy On Write Based File Systems Performance Analysis And Implementation: Thesis, Technical University of Denmark, Department of Informatics, Lyngby 2010, p. 83

[19] Apap Frank et al: Detecting Malicious Software by Monitoring Anomalous Windows Registry Accesses, Recent Advances in Intrusion Detection: Lecture Notes in Computer Science, Berlin 2002, pp. 36-53

[20] Honeycutt Jerry: Microsoft Windows Registry Guide, Redmond 2005, p. 1327

[21] Vokorokos Liberios, Pekár Adrián, Ádám Norbert, Darányi, Peter: Yet Another Attempt in User Authentication, Acta Polytechnica Hungarica, Vol. 10, No. 3, Budapest 2013, pp. 37-50

[22] Vokorokos Liberios, Baláž Anton, Trelová Jana: Distributed Intrusion Detection System using Self Organizing Map, Proceedings of 16th IEEE International Conference on Intelligent Engineering Systems, Lisbon 2012, pp. 131-134