



Assignment of master's thesis

Title: Light-Weight Sandbox for Installers
Student: Bc. Artem Ustynov
Supervisor: Ing. Josef Kokeš
Study program: Informatics
Branch / specialization: Computer Security
Department: Department of Information Security
Validity: until the end of summer semester 2022/2023

Instructions

- 1) Research currently used methods for sandboxing applications.
- 2) Study the real-world scenarios of a malicious use of installers.
- 3) Propose a technical solution that would allow the user to detect and/or prevent such an attack.
- 4) Create a proof-of-concept of a lightweight application of your design.
- 5) Test your code against a real-world potentially unwanted installer.
- 6) Evaluate your results and discuss the options for improvement.



**FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE**

Master's thesis

Light-Weight Sandbox for Installers

Bc. Artem Ustynov

Department of Information Security

Supervisor: Josef Kokes

April 30, 2022

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the “Work”), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work for non-profit purposes only, in any way that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on April 30, 2022

.....

Czech Technical University in Prague

Faculty of Information Technology

© 2022 Artem Ustynov. All rights reserved.

This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Ustynov, Artem. *Light-Weight Sandbox for Installers*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Abstrakt

Při vyhledávání méně známých softwarových produktů nebo produktů vyvinutých nezávislími vývojáři je nutné vypořádat se s jejich instalátory. Tyto instalační programy často obsahují software třetích stran a někdy není možné nainstalovat požadovaný software, aniž by byl uživatel nucen instalovat nějaký další doprovodný program. Cílem této práce je analyzovat běžně používaný instalátor “InnoSetup” a vyvinout odlehčený sandbox, který zajistí, že instalátor nebude upravovat registry ani soubory na hostitelském počítači, což uživateli umožní učinit informovanější rozhodnutí týkající se bezpečnosti softwaru, který si přeje instalovat. Navrhované řešení je navrženo jako odlehčená alternativa k existujícím přístupům.

Klíčová slova lehký, sandbox, instalátor, softwarová emulace

Abstract

When searching for lesser-known software products or products developed by independent developers one has to deal with software installers. Such installers often bundle third party software and sometimes it is impossible to install the desired software without also installing some additional program. The goal of this work is to analyze the commonly used installer the “InnoSetup” and

develop a lightweight sandbox that will ensure that installer won't modify registers or files on the host computer, allowing the user to make a better-informed decision regarding the safety of the software they desire to install. The proposed solution is designed to be a lightweight alternative to existing approaches.

Keywords light-weight, sandbox, installer, software emulation

Contents

Introduction and motivation	1
1 Real-world scenarios of the malicious use of the installers	4
1.1 Official distribution	4
1.1.1 uTorrent	5
1.1.2 GOM player	6
1.2 Third party software distribution websites	7
1.2.1 Examining the distributed files	7
1.2.2 Motivation for PUP	10
1.3 Antivirus protection	10
2 Isolating installers	12
2.1 Docker	13
2.2 Virtual machines	13
2.3 Windows Sandbox	14
2.4 SHADE Sandbox	15
3 Design overview	17
3.1 Windows registry	17
3.2 Design	18
3.3 Sandbox design	22
4 Image file structure	25
4.1 Image file in memory	27
5 Implementation details	31
5.1 Function hooking	33
5.2 Registry key management	38
5.3 File management	44

6	Testing against real-world potentially unwanted installer	50
6.1	InnoSetup installation outside the sandbox	50
6.2	InnoSetup installation in the sandbox	52
7	Discussion	54
7.1	Options for improvement	55
7.2	Modifications and cooperation	56
8	Final conclusion	57
	Bibliography	60
A	Acronyms	63
B	Contents of enclosed CD	64

List of Figures

1.1 Cheat Engine PUPs	4
1.2 uTorrent additional software	6
1.3 GOM player PUPs	6
1.4 Downloadastro PUPs	8
1.5 No internet error message	8
1.6 End of the installation process window	9
2.1 Docker and VM distinctions [3]	13
2.2 Memory sharing for Windows Sandbox	14
2.3 SHADE sandbox. Placing program into the sandbox [8]	15
3.1 Schematic process of registry key creation	19
3.2 User level key creation imitation	19
3.3 Imitation of a registry key creation	21
3.4 Installer process start under sandbox mode	22
3.5 New process call interception diagram	23
3.6 Create file interception diagram	23
3.7 Modify file interception diagram	24
4.1 Portable executable file format. [14]	25
4.2 NT DOS HEADER location relative to DOS HEADER	26
4.3 Locate IMAGE_NT_HEADER structure	26
4.4 Export tables and symbols [17]	27
5.1 MoveFileW system calls	32
5.2 Locate PIMAGE_DOS_HEADER of loaded DLL libraries	33
5.3 Process of location of an ImageThunk from a DosHeader	34
5.4 Import by name process overview	35
5.5 Import by ordinal. Function metadata location.	35
5.6 Function names location, derived from AddressOfNameOrdinals	36
5.7 Import by name. Function swap schematic process.	37

5.8	Overwriting the function address for functions imported by ordinal	38
5.9	API call stack for predefined handle value equal to five	38
5.10	API call stack for predefined handle value equal to six	39
5.11	Containers used to store registry key data during sandbox run-time.	39
5.12	Registry key container class	40
5.13	Registry value container class	40
5.14	NtCreateKey emulation process.	41
5.15	NtOpenKey emulation process.	41
5.16	NtSetValueKey emulation process.	42
5.17	NtQueryValueKey emulation process.	42
5.18	NtQueryKey emulation process.	43
5.19	myNtClose process.	43
5.20	Files location with and without sandbox.	44
5.21	Name location in OBJECT_ATTRIBUTES structure	45
5.22	myNtOpenFile process.	46
5.23	myNtCreateFile process.	47
5.24	myNtQueryAttributesFile process.	47
5.25	MoveFile process.	48
5.26	LoadLibraryW process. Redirecting call.	48
5.27	CreateProcessW process. Redirecting call.	49
6.1	InnoSetup new process created.	50
6.2	InnoSetup new process at the final stage.	51
6.3	Files after InnoSetup installation	51
6.4	Origin of the child process during InnoSetup installation.	52
6.5	Content of the sandbox folder upon reaching the first screen.	53

Introduction and motivation

Nowadays, a lot of software is no longer supported by developers and can't be accessed through official websites, so users have to resort to downloading from third party resources. Often, such software comes in the form of an "installer" with bundled programs. Sometimes, even currently supported free-to-use programs come in bundles in order to monetize them. For example: Cheat Engine, uTorrent, and Daemon Tools. Some installers force the user to install at least one of the bundled programs in order to continue, and sometimes they don't give a choice of where the product is going to be installed, or won't even notify the user about the fact that there will be more than one program installed.

Furthermore, installed software needs to be uninstalled by the provided uninstaller and the result is often not what would be expected after complete removal. Programs often leave residue files in system folders. This can cause potential errors and crashes of other programs. And there is no easy way of knowing if these files weren't deleted by accident, or if they in conjunction with some other software are used to profile users. They can be used to track users' history of used programs and gain insight that will allow them to gain knowledge for malicious or commercial use.

A similar issue occurs with registry keys. They are often used as a less obvious way to store information on the machine. They can prevent a user from having a fresh install of the app and have similar potential malicious and commercial use as files. Examples of such behavior are often displayed by legitimate programs for example: Yandex Disk, Opera Browser, Gom Player, Viber and many other programs that leave residual files and registries after they were deleted.

Detecting such a file or a registry can be a rather difficult task since there are potentially millions of keys and files that can be present in the system and Windows doesn't provide a way to track what files or keys were modified or created by a given program.

While all the aforementioned changes are preventable via containers, vir-

tual machines, and Windows Sandbox, they are rather complicated to set up and require significant computation resources. They provide good levels of isolation from the host machine. Since they are all designed for professional use and not for everyday users process of setting up and running existing encapsulating solutions can be relatively difficult. HYPER-V technology that is used for Windows Sandbox and some Virtual Machines is not supported on some CPUs or might need an additional set-up in BIOS and not all users would be comfortable changing parameters there or even have access to them in the first place. Furthermore, installing the software might require installing additional dependencies and possibly even purchasing additional licenses. This makes the set-up and running of an installer in the emulated environment a task that requires more effort than an average user is willing to give. Software based sandboxes are less secure, but more convenient, so they are more likely to be utilized by the user.

Users can't fully rely on antiviruses to protect them from unwanted software, since antiviruses often depend on signatures and do not perform a full analysis of potentially installed software. The whole installer might be quarantined or deleted leaving the user no other choice except to disable the antivirus and install a program as-is especially if the desired software can't be obtained from other sources without the installer attached to it. See results of the experiment [\[1\]](#).

It is a common practice to have additional potentially unwanted software bundled with legitimate software to monetize free programs. Antiviruses sometimes see this practice as malicious and block the whole installer, but if the user is careful they can often avoid installing any additional programs so from the user's perspective the antivirus just stands in the way and makes obtaining otherwise legitimate software unnecessary difficult. However, this behavior from antivirus is expected since the vendor isn't obligated to openly state what software will be installed, especially if it is impossible to find alternatives.

A perfect solution would allow users to install software and verify its legitimacy without installing any additional programs or purchasing a professional license, that isn't going to be fully utilized. It is not necessary to develop an absolutely secure sandbox solution, but rather a solution that is good enough, to prevent opportunistic malicious agents from pushing potentially unwanted programs onto users. As it will be demonstrated in this thesis bundled programs are often free to use. They either used to serve ads to users or have some behaviour that promotes purchases of subscriptions or digital items. It isn't clear if original developers of bundled software are aware of methods that are being used to distribute their product. It is possible that they are contracting promoting agencies that incorporate the software in bundles, without informing the creator. If the distributor is being payed per-installment, they have strong instinctive to generate as many installations as possible and it is not important if the program will be used or not and software installers allow

to bundle multiple products maximizing the profits.

So end-user's PC isn't attacked by a sophisticated virus but rather the user has to deal with annoyances that come when unwanted programs are installed: unnecessary reboots, slowing down of the system, time that is spent waiting for the uninstallation process to finish plus time spend manually making sure that all key registers and files were deleted and do not persist past uninstall process. Software-based sandboxes can cover all of this issues.

Real-world scenarios of the malicious use of the installers

A malicious use of the installer would be defined as leveraging the software installation process to perform unsolicited or harmful changes or an attempt to install additional software that isn't crucial for the function of the application that the user originally intended to install.

1.1 Official distribution

Some officially distributed software is packed in form of the installers and can display some malicious behavior.

Cheat Engine [1] is software that can be used to quickly find the address of values that are changed during a program's run for reverse engineering purposes. The provided "InnoSetup" based installer is 3.4 MiB. And upon running, it will try to install two additional programs see figure 1.1. In the

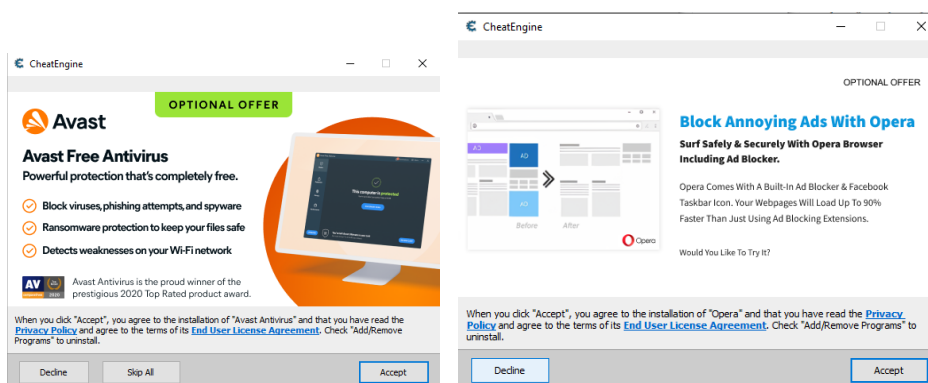


Figure 1.1: Cheat Engine PUPs

figure 1.1 we can see that the placement of the "Accept" button and the

fact that it is highlighted increases the chance of accidental clicks. In case the software was accepted, there are no further dialogues to specify target location and no way of going back other than closing the installer window. After the displaying of two potentially unwanted programs (PUPs) the installment will start immediately without clicking the dedicated “Install” button, a path for Cheat Engine is never asked by the installer.

Given the original size of the installer, it is obvious that the PUPs are being dynamically downloaded from the internet and their installation is completely silent.

If the user tries to run the installer without a connection to the internet, then no program is going to be installed but a success message is going to be displayed. In case the user already has PUP software installed then the installer won't be trying to download or install it again.

The observed behavior strongly indicates that the installer is used mainly to download real executables to reduce the perceived size of the program and as a source of magnetization for the application.

After uninstalling, there were “Cheat Engine” and “Cheat engine symbols” folders left in `user/Appdata/Local/Temp`. Even though this is a directory for storage of temporary files, the fact that some files were left there after the software was uninstalled violates the expected result of the application deletion process.

1.1.1 uTorrent

uTorrent is software designed to download files from peer-to-peer connections. This helps to reduce the load on an individual server and helps to improve the stability and speed of the download process. uTorrent uses the “installaware” installer. This installer executable is 5.4 MB. Upon an attempt to run this installer without an internet connection, it will fail, notifying the user that a connection is required.

If a connection to the internet is established, then the installer will try to install at least one additional software as shown in figure [1.2](#)

Similar to the previous example, this installer gives the user a chance to opt-out of additional programs. However, the UI is designed to trick users and make them continue the installation process by accident. Just like in the previous installer this one doesn't provide a dialog window to specify the desired path and additional software is installed silently in the background.

After uninstalling uTorrent, there are no residual files but `Computer\HKEY_CLASSES_ROOT\.torrent` registry key remains in the system.

1. REAL-WORLD SCENARIOS OF THE MALICIOUS USE OF THE INSTALLERS

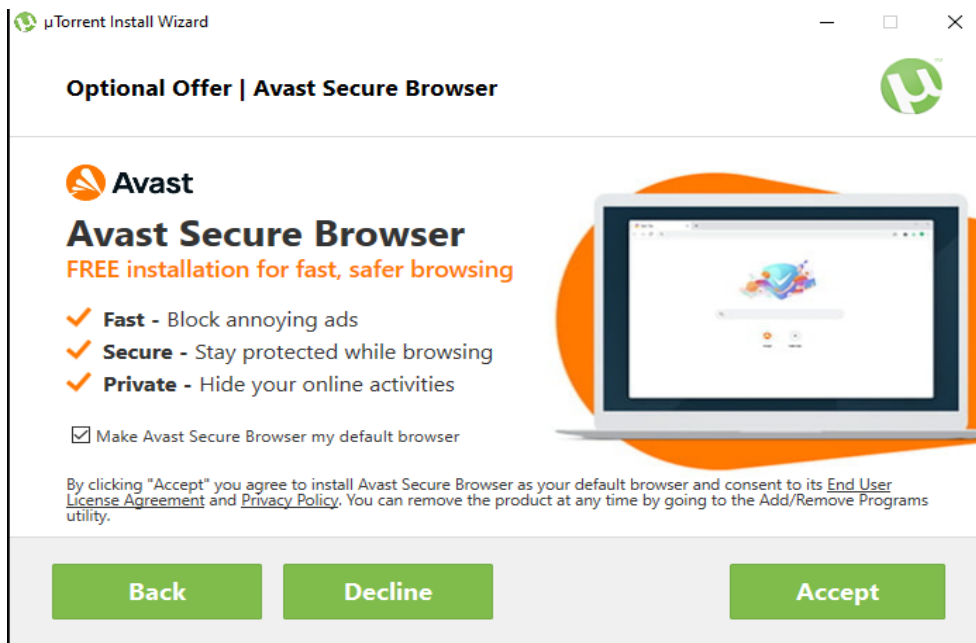


Figure 1.2: uTorrent additional software

1.1.2 GOM player

GOM player is software designed to playback audio and video files with a wide variety of encoding. The installer size is 40 MiB and it is based on the “NullSoft” installer.

Upon running the installer with an internet connection, two additional PUPs will attempt to be installed see figure [1.3](#).

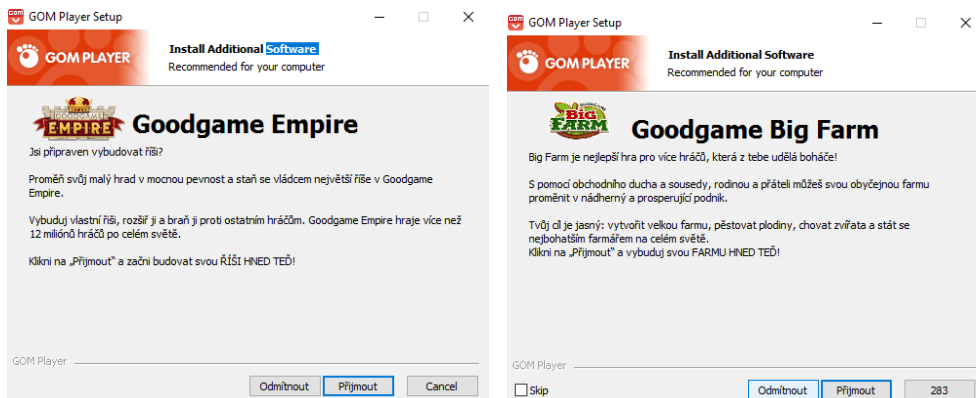


Figure 1.3: GOM player PUPs

The user isn't provided with a choice of where this software will be installed and UI elements placement encourages accidental clicks on the “Accept” button. Just like the previous examples, this installer doesn't have a dedicated

“Install” button and if a user clicks on the “Next” button the installation suddenly starts and can’t be interrupted.

After uninstalling the GOM player, these registry keys were left:

- `Computer\HKEY_CLASSES_ROOT\Gomplayer.Skinfile`
- `Computer\HKEY_CLASSES_ROOT\.gps\GOM Player Skin File`
- `Computer\HKEY_CLASSES_ROOT\DVD\shell\Play with GOM Player`

And these files and folders:

- `C:\Users\user\AppData\Local\Temp\Uninstall.exe`
- `C:\Program Files (x86)\GOM`

While none of these key registers or files are malicious the fact that they persist on the user’s PC after the uninstallation breaks the expectations of the software removal process and can potentially cause unexpected errors.

1.2 Third party software distribution websites

There exists a vast variety of websites that claim that one can download software there for free. Some websites work as a proxy between the user and the developer and serve a purpose similar to Microsoft Store or Steam, creating a convenient marketplace for developers to advertise their software and for users to download it legally. Often, this sort of website will distribute software in the same way as the official developer website, allowing users to download a free or paid version. They work as a platform that helps both developer and users and does not claim that software obtained from them is free, specifying the type of license.

The second category are websites that claim that all the software that they host is free and available for use without purchasing.

Websites of the second category are prone to malicious activity, thus will be examined more closely in the scope of this work. One of such websites is `downloadastro.com`. Here one can find that they are supposedly able to download programs that are protected by copyright.

Since downloading protected software is forbidden by law, in the scope of this work only free-to-use software will be examined.

1.2.1 Examining the distributed files

For direct comparison, “Cheat Engine”, “uTorrent” and “Gom Player” were selected. All of these programs, when downloaded from `downloadastro.com`, are 2.4 MB InnoSetup files with two PUPs as shown in figure [1.4](#).

[1.4](#)

1. REAL-WORLD SCENARIOS OF THE MALICIOUS USE OF THE INSTALLERS

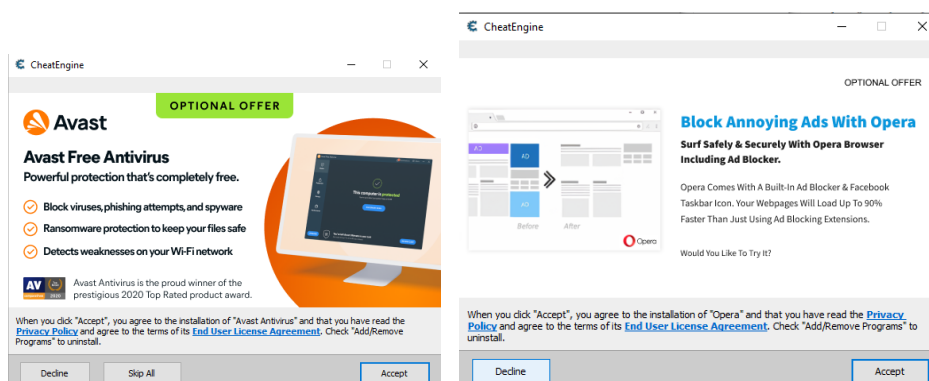


Figure 1.4: Downloadastro PUPs

Neither of the programs can be installed without an internet connection. Upon an attempt, the user will see an identical error message across all the installers as shown in figure 1.5.

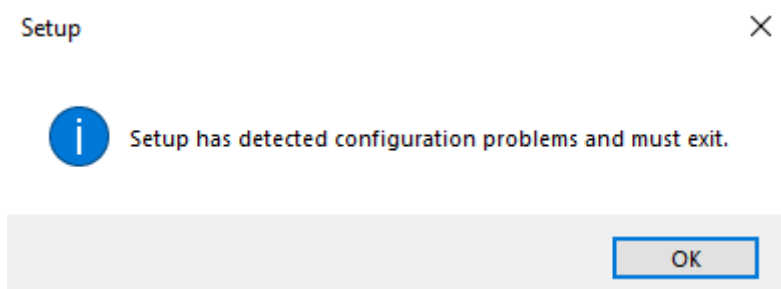


Figure 1.5: No internet error message

If the user accepts PUPs (on purpose or by accident) as shown in figure 1.4 then programs will be silently installed in the background without any indication of the fact that the process of installation has started. Regardless of whether the user accepts or declines the offered software, they will see window offering them to run the installer and open the developer's website as shown in figure 1.6.

Ignoring the user's choice regarding visiting the developer's website they will be redirected there to the download page.

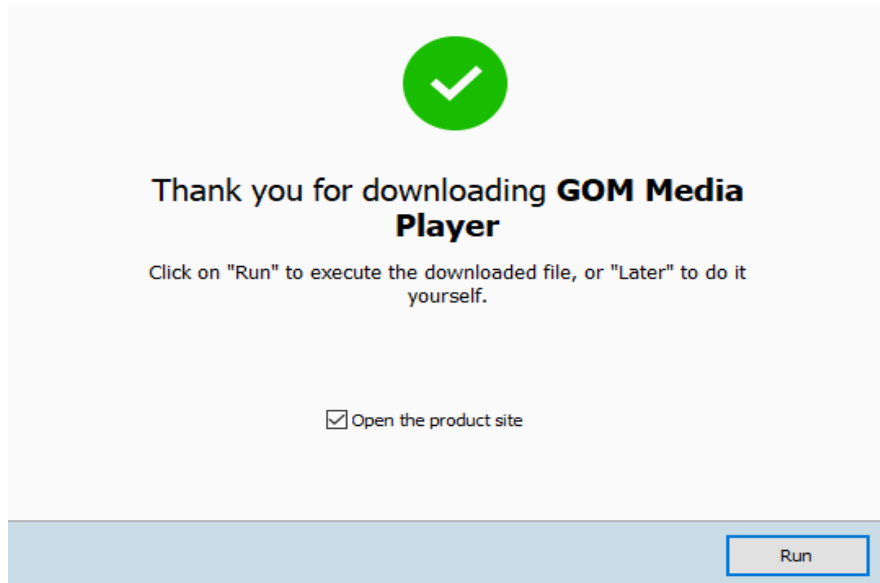


Figure 1.6: End of the installation process window

So downloadastro.com website only serves as a way to trick the users into thinking that they will be able to download the desired software, but in actuality, all they will get is a glorified URL to the developer's website. So the website is used exclusively to mislead the users into downloading unwanted software. Installers downloaded from developer's website will still have PUPs, so the potential amount of unwanted programs only increases.

1.2.2 Motivation for PUP

The main motivation for including potentially unwanted software into installers is the financial gain. There exists a framework that allows one to create installers with relative ease, for example: “Dynamic Downloader”. It describes itself as:

“Designed for downloading large files. The end user – instead of downloading a desired large file directly (through a browser) – downloads only a small Dynamic Downloader Client, which downloads the desired large file in small chunks in the background. This method allows you to offer the end-user additional values while his desired file is being downloaded. That value can be offered to download third-party software, or it can simply show ads, videos, websites, or any other online content.” [2] It is clear that developers of this software are aware of the fact that this product is used to push unwanted software onto users, and it serves to minimize user interaction and increase the chance of accidental downloads. Direct quote from the website: “Browser asks to save the file and starts the download process.

And here we go. This is the weak part of this way of downloading files. (Yeah, already the second step). After the download is confirmed (very often it is automatic) you see nothing but a progress bar, and it is even hidden somewhere in the corner!

For small software it is fine, but any larger software takes several minutes to download. It is pretty annoying for users to wait and also for you – because you have no profit from this time. The longer the download process takes the bigger chance you lose the customer is.

How to solve this? How to use this time effectively? What to do to avoid long waiting? Maybe showing the user your other valuable software? Or third party software he may like and you can earn some profit for recommending it to him?” [2]

This approach is designed to hide what software is going to be installed and the whole installation process. And there is no mention of whether “other valuable software” is going to be safe. Given that “Dynamic Downloader” can be purchased as a service by anyone, there are no guarantees regarding the safety of software received via the installer.

1.3 Antivirus protection

Antivirus software is often capable of detecting suspicious installers, but the detection is most-likely based on the signatures and the pattern recognition, so entire installer is quarantined without allowing the user to install the desired program.

To demonstrate this behavior, most common antiviruses were selected: Avast, ESET-NOD32, Microsoft Defender, and Norton 360. They were tested

against files downloaded from the official Cheat Engine website. In this test, all additional offerings were accepted. Results of the test:

- Avast Antivirus – the installer was isolated upon download. Had to be manually removed from quarantine, before running the program. Upon finishing the installation, the Opera Browser and the Cheat Engine were installed. Directories for both programs were scanned and no threats were detected.
- ESET-NOD32 – after an attempt to run the installer was isolated. It had to be manually removed from quarantine, before running the program. Upon finishing the installation, the Opera browser, Avast Antivirus, and the CheatEngine were installed. Directories for all programs were scanned, and no threats were detected.
- Microsoft Defender – no threats detected in the installer. Upon finishing the installation, the Opera browser, Avast Antivirus, and the Cheat-Engine were installed. Directories for all programs were scanned, and no threats were detected.
- Norton 360 – after an attempt to run the installer was isolated. It had to be manually removed from quarantine, before running the program. Upon finishing the installation, the Opera browser, Avast Antivirus, and the CheatEngine were installed. Directories for all programs were scanned, and no threats were detected.

It was shown that antiviruses behave similarly and only detect the installer as a threat. Since the official distribution of CheatEngine is bundled with additional software, users will ignore antivirus recommendations, to get the desired program. This becomes problematic if the users develop a habit of ignoring antivirus advice and warnings, rendering them useless.

Isolating installers

In the scope of this work, “unwanted change” would be considered as an installation of potentially unwanted software, or files and registry keys that were created and persisted after the software that created them was uninstalled, excluding files and registry keys that were deliberately created by a user with that software.

Currently, there exist several solutions that can be utilized to prevent unwanted changes. The most popular ones are virtual machines, Docker, Windows Sandbox and SHADE Sandbox (higher level software sandbox). Docker and VM are mainly geared towards developers and professional users. Their setup requires some knowledge of operating systems and networking as well as an understanding of hardware resources such as CPU and RAM to configure them for hosted OS or containerized applications. Windows Sandbox is designed to be easier to use, but it is not available on home versions of Windows, making it a rather expensive option. Higher level software sandboxes are usually less resource heavy and do not require specific CPU features like the HYPER-V technology. Price can vary depending on the sandbox.

Existing low-level isolating solutions are mostly geared toward professionals and advanced users. As such, it is not a problem for them to require some preparation time and hardware support and to have a relatively steep learning curve. A regular user often doesn't have time or willingness to learn new technologies, since that isn't their goal. So they rely on antivirus protection and trustworthy resources, and if such resources are not available, users resort to downloading software from third parties. In recent years, the situation has become better. Both virtual machines and containers are easier to install and configure. It is even possible to get this software for free. Windows Sandbox is available on professional versions of the OS only, but once configured it can be run and used as easily as any other program. Higher-level “SHADE sandbox” solution can be installed just like any other program.

2.1 Docker

From the point of view of a regular user containers are very similar in their functionality to VM. It is easier to run multiple instances of a container than VM, but setting up a container also requires a license for Windows and has all of the same drawbacks as VM when it comes to setting up of the environment also containers tend to have a steeper learning curve, so that might discourage an average user.

Just like VM, containers provide isolation from the host operating system and that is a strong argument in favor of their security.

So from the point of view of an average user the distinctions between virtual machine and container are not as significant since both of the solutions require significant computational resources, time and money investments to set up the environment.

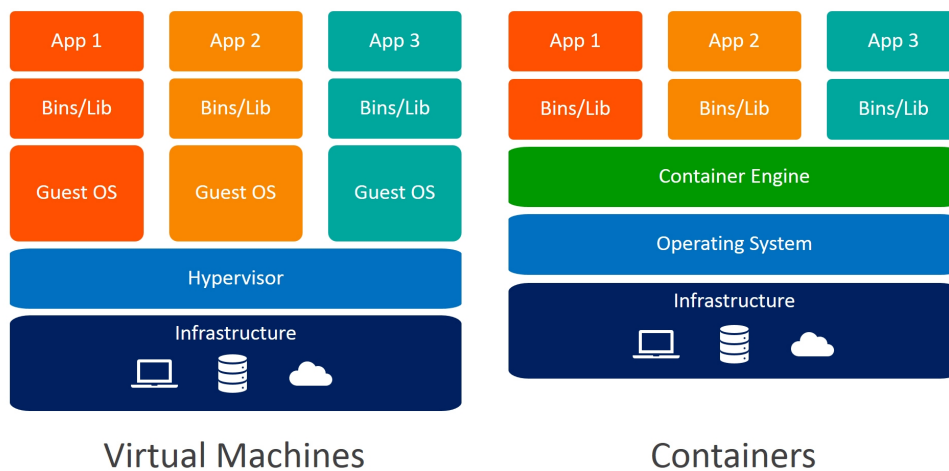


Figure 2.1: Docker and VM distinctions [3]

2.2 Virtual machines

System virtual machines (VM) (full virtualization virtual machines) provide virtualization of the entire OS. This type of emulation is rather resourced heavy. Minimum requirements for Windows 7 are 1 GB of RAM and 6 GB on hard drive [4], while newer versions require even more resources – 4 GB of RAM and 64 GB of storage [5]. For better performance, virtualization should also be supported by the CPU. This type of emulation requires HYPER-V to be disabled on the host OS, and modifying this parameter might not be permitted in certain types of environments, or simply above the effort level that the average user is willing to spend. An additional downside is that the purchase of an additional license is required in order to run Windows inside a

virtual machine, as well as licenses for all the additional software that might be required in order to install a new application. Also, when considering the average user, time of set-up should be taken into account since the majority of the applications is not critical and setting up VM might take several hours, so some users would rather risk and install software on their current system.

On the plus side, VM is isolated from the host OS and is system independent, so one can use VM to run Windows apps from a Linux host and the graphics interface makes navigating virtualized system identical to regular Windows.

2.3 Windows Sandbox

This is a utility that is available for Professional versions of Windows 10 and can be used to run untrustworthy applications in the Sandbox mode.

Because Windows Sandbox runs the same operating system image as the host, it has been enhanced to use the same physical memory pages as the host for operating system binaries via a technology referred to as “direct map.” For example, when NTDLL.dll is loaded into memory in the sandbox, it uses the same physical pages as those of the binary when loaded on the host. Memory sharing between the host and the sandbox results in a smaller memory footprint when compared to traditional VMs, without compromising valuable host secrets.^[6]

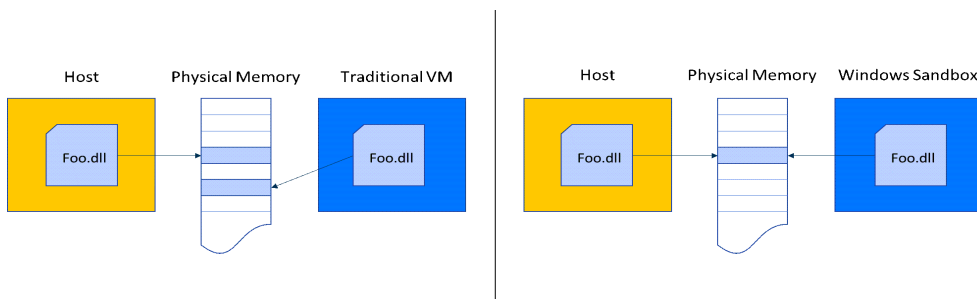


Figure 2.2: Memory sharing for Windows Sandbox

This and other optimizations allow Windows Sandbox to be relatively lightweight when compared to other solutions. However, systems requirements are far from being trivial ^[7]

- Windows 10 Pro, Enterprise or Education build 18305 or Windows 11 (Windows Sandbox is currently not supported on Windows Home edition)
- AMD64 architecture
- Virtualization capabilities enabled in BIOS

- At least 4 GiB of RAM (8 GB recommended)
- At least 1 GB of free disk space (SSD recommended)
- At least two CPU cores (four cores with hyperthreading recommended)

In addition to paying for a professional license, which might not be needed for the average user, the setup process also requires access to BIOS and the use of PowerShell [7]. Those two factors can be enough to prevent people from even trying to use an official solution.

In contrast to containers and virtual machines Windows Sandbox can't preserve its state, so every new launch only includes the default Windows applications. While this can be desired behavior for some use cases, it can also mean additional work from the users' side.

The strong sides of Windows Sandbox would include official support from Microsoft and ease of interaction. Users can just drag and drop files they want to test, and this eliminates unnecessary barriers for beginners.

2.4 SHADE Sandbox

SHADE Sandbox is a simulated virtual environment in your system. Instead of running software and browsing the internet directly from your system, you can perform these activities through a safe virtual environment. This advanced mechanism enables a highly controlled setting for running new or suspicious applications without risking the integrity of your operating system [8].

SHADE Sandbox offers an easy way to run a program under software sandbox. All that is required from a user is to drag and drop an application of their choice into the program's window, and it will run in the sandbox mode as shown in figure 2.3.

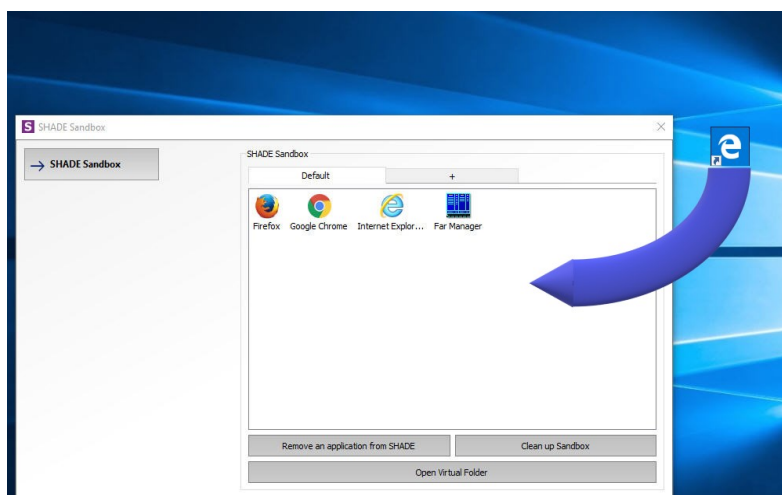


Figure 2.3: SHADE sandbox. Placing program into the sandbox [8]

2. ISOLATING INSTALLERS

SHADE sandbox is a closed source project. User can choose between the free trial version, a yearly subscription and a perpetual licence. Hardware requirements are minimal:

- Operating System: Windows 10, Windows 11
- CPU: 1GHz
- RAM: 2GB

There is no in-depth explanation of how exactly it functions. But since it is required to run in background to intercept applications that were just dragged and dropped into the sandbox and the fact that it utilizes the actual file system on the user's PC, sandboxing functionality is probably achieved by hooking API calls redirecting them into a predetermined "sandbox" location.

This solution offers software isolation that would probably be good-enough for a regular user. It will most likely protect them from unwanted software bundles by placing them into simulated folders, making it easy to remove if deemed necessary, or the user can keep using an untrustworthy application from within the sandbox. The Previously discussed isolating solutions are geared towards professional usage and are deployed to production environments in various corporations offer better isolation and as the result more security, but for an average user the trade-off in convenience isn't worth it since they are not likely to be subjected to highly sophisticated attacks in the first place.

Small drawbacks are:

- Closed sourced nature of the product. If it doesn't find a commercial success it might become abandoned.
- The trial period is limited to 30 days, after that purchasing of the license is required.
- Sandbox affects the performance of the application by making it slower to start and less responsive.

On the plus side, it is a much more user-friendly solution compared to VM, containers, and Windows Sandbox. Installing it is not more difficult than any other software and running programs in the sandbox or purging the sandbox storage is an easy and intuitive process, there are no special hardware features that are required in order for it to function and the minimum requirements match those of the OS it is designed for.

Design overview

The goal of this work is to develop a solution that will contain all the changes that were performed by installer in a predetermined scope while not requiring a complicated set-up and preparation process. By deleting the containment scope, all files and registry keys that were created should be removed from the system without leaving any trace and no files or registry keys should be edited.

3.1 Windows registry

The Microsoft Computer Dictionary, Fifth Edition, defines the registry as:

“A central hierarchical database used in Windows 98, Windows CE, Windows NT, and Windows 2000 used to store information that is necessary to configure the system for one or more users, applications, and hardware devices.

The Registry contains information that Windows continually references during operation, such as profiles for each user, the applications installed on the computer and the types of documents that each can create, property sheet settings for folders and application icons, what hardware exists on the system, and the ports that are being used.

The Registry replaces most of the text-based .ini files that are used in Windows 3.x and MS-DOS configuration files, such as the Autoexec.bat and Config.sys. Although the Registry is common to several Windows operating systems, there are some differences among them. A registry hive is a group of keys, subkeys, and values in the registry that has a set of supporting files that contain backups of its data. The supporting files for all hives except HKEY_CURRENT_USER are stored in the %SystemRoot%\System32\Config folder on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, and Windows Vista. The supporting files for HKEY_CURRENT_USER are in the %SystemRoot%\Profiles\Username folder.” [9].

This definition is crucial, since it helps in understanding of how and why programs are using registry instead of regular files. Knowing this it becomes clear why the proposed sandbox solution should also isolate registry storage. Data stored in the registry isn't limited to small numerical or string values. It is possible that the software that appears to be legitimate can store the malicious code inside the registry key, hiding it from the antivirus programs. And even if the software that originally placed the malicious code in the registry was deleted, but registry key remained, it is possible that some other program will take an advantage of it in the future.

3.2 Design

For a running process to interface with an OS, it has to utilize system calls. A system call is a way for a program to interface with the OS kernel. Assuming the synchronous call model process performs a system call to the OS kernel through a predefined interface and passes control from user space to kernel space, once control is returned to the process, the program continues as usual. System calls are utilized to interact with the file system, create new processes or threads, and perform other kernel-level functions. Windows OS has close to 2000 system calls [10]. Not all the system calls are documented. Often there exist only wrapper interfaces e.g. `CreateProcessW`, a part of `Kernel32.dll`, is a documented function in MSDN [11] that is utilizing an undocumented function `NtCreateProcess` from `NTDLL.DLL` [12]. Calling undocumented functions isn't a good practice since the interface or error reporting might change and that would cause problems or crashes of the application that are hard to debug.

Import Address Table (IAT) is a structure that holds addresses of the symbols that are being imported. These addresses are technically called "virtual addresses" even though they are the actual memory addresses of the symbols.

Knowing how image files are structured and how they are loaded into memory allows developers to intercept calls to the dynamically linked libraries by replacing the import function addresses (a process of locating function addresses and replacing them will be explained in more detail in the chapter: "Implementation details" section: "Function hooking" [5.1])

All operations that might cause an unwanted change can be reduced to:

- Creating files.
- Editing files.
- Creating registry keys.
- Editing registry keys.

In Windows, these functions are handled by the OS—in case of working with registers by AdvApi.dll on the user level and by NTDLL.dll on the system level as shown in figure 3.1.

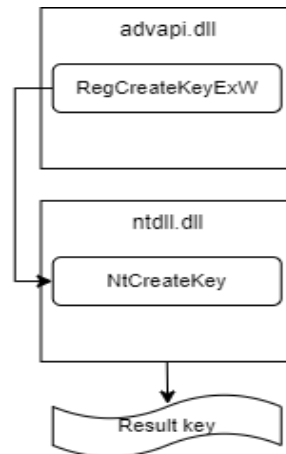


Figure 3.1: Schematic process of registry key creation

In case of files calls to the DLL functions can be intercepted and redirected to prevent changes in the user's system. This can be achieved with a combination of remote thread execution and DLL injection.

It is possible to intercept system calls on the user level (advapi.dll etc.) and the system level (NTDLL.dll). There are drawbacks and benefits to both approaches.

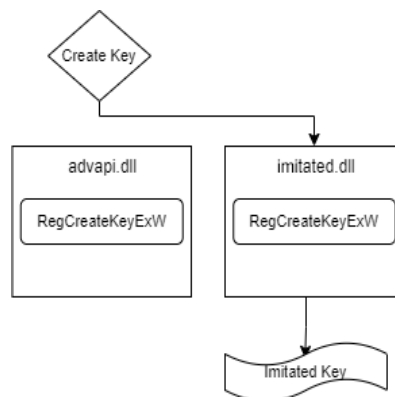


Figure 3.2: Schematic process of imitation of a registry key creation on user level

Intercepting of function calls on user-level as shown in figure 3.2 has benefits such as:

3. DESIGN OVERVIEW

- Functions are well documented and described on the official Microsoft website
- They are called less frequently compared to NTDLL functions
- Interfaces are often simpler (functions have fewer input parameters and fewer data structures are involved)
- Their declaration is much more stable, rarely changing over time.

Drawbacks should also be considered:

- There are many variations of functions that achieve a similar goal (OpenKey, OpenKeyExW, OpenKeyExA etc.).
- Third party libraries that will serve a a convenient wrapper around the NTDLL functions are relatively easy to develop and use instead of the f.e. AdvApi.dll and other official user-level libraries.
- It is possible to call system-level libraries directly.
- They are not guaranteed to be loaded into the process.

While this is a valid approach it is also can be defeated relatively easily and there are no guarantees that user-level functions are going to be utilized in the installer. Designers of installers might want to use their libraries to optimize their code in some way. Due to these reasons, this approach wasn't implemented, and instead, the interception was performed on the system level as shown in figure [3.3](#).

Since NTDLL is guaranteed to be present in the installer process and creating a file or key without invoking NTDLL functions isn't officially supported in Windows, even if someone should discover a way to somehow circumvent the invocation of the NTDLL library, it would most likely be unstable and eventually blocked by Microsoft. A significant effort that isn't guaranteed to pay off and the inherent instability (due to reasons described above) of any solution that will somehow avoid using NTDLL for file or registry key manipulation makes it unsuitable. Using it in installers would be counterproductive since one of the main reasons to use the installer in the first place is its ability to create an environment required for program execution across different versions of Windows and independently of the security patch installed.

Due to the aforementioned reasons interception of system-level functions is a good candidate for proof of concept (POC) of a sandbox. By replacing the NTDLL functions it won't be possible for an installer to create a change in the user's system [3.3](#). Problem of avoiding NTDLL library doesn't apply to the proposed solution, because the goal is to mimic system behaviour and not to circumvent NTDLL. Interactions with the system will be mocked, so from point of view of the original application everything will be the same,

the only difference is that the operations on registers will be simulated in the memory and operations on files will be redirected into the predetermined location. In case the interface for the intercepted functions will be changed it will be necessary to update the sandbox application, but underlying logic isn't likely to be significantly different from the originally proposed.

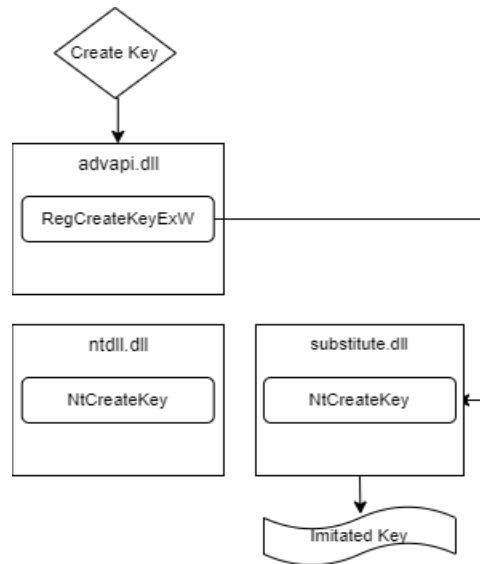


Figure 3.3: Schematic process of imitation of a registry key creation on a system level.

3.3 Sandbox design

To execute an application in sandbox mode, several actions must be executed. First, the application should be started, and then Sandbox.dll that will hook API calls should be injected. To achieve this, three main components of the sandbox application were identified and designed.

Initializer – a component responsible for the initial start of an installer application. Since the application should not perform any actions while being outside of the sandbox mode, this component starts the application in a suspended state. After the installer process is started, the Initializer creates the Injector process. Once Injecting is complete, the installer process is resumed.

Injector – a component responsible for injecting DLL into the running installer process. This component allocates memory inside the target process, writes a DLL into the allocated space and starts the remote thread. Once finished, this process exits.

Sandbox.dll – the main logical component that is responsible for interception of the API calls and emulating registry key calls and redirecting of file system calls to a predetermined location.

It is expected that the installer might utilize multiple threads during the installation process, so Sandbox.dll has critical regions that will ensure proper order of execution.

To intercept all potential changes, the sandbox has to inject the DLL right as the installer program starts. This can be achieved by starting the installer process in suspended mode, performing interception, and then resuming the installer process as shown in figure [3.4](#)

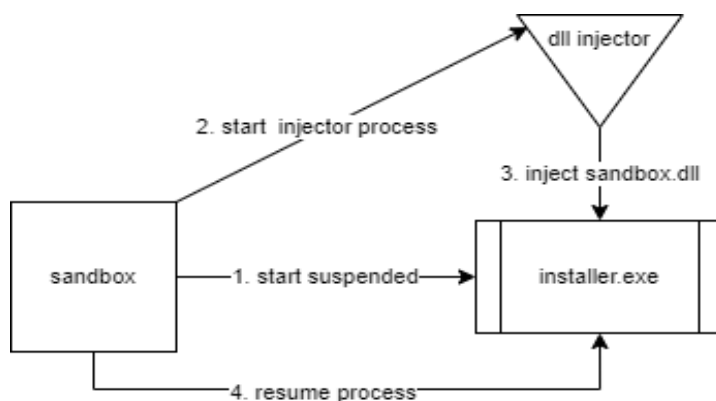


Figure 3.4: Installer process start under sandbox mode

Since there are two main architectures of Windows applications, 64 and 32-bit versions, the proposed solution should support both of them. To achieve that, two versions of Sanbox.dll and Injector components are required: 64 and 32-bit accordingly. After the Initializer starts the installer.exe process, it is

possible to get the process's architecture and based on the received information decide what version of Injector and Sandbox.dll should be dispatched.

During the execution of the installer.exe process, a new process may be created. To ensure that the created process won't escape the sandbox, Sandbox.dll should intercept API calls to create a new process and inject Sandbox.dll into that process as well as shown in figure 3.5.

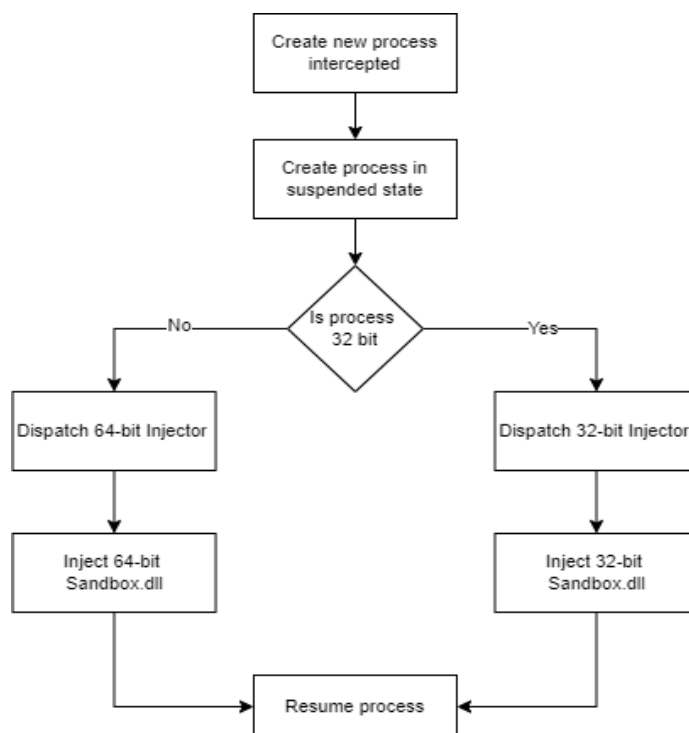


Figure 3.5: New process call interception diagram

It is possible that the installer might have some libraries loaded via `LoadLibrary`, these calls should also be intercepted and functions that might cause a change on the user's PC should be hooked by `Sandbox.dll`

To prevent the creation of files in a location outside the predetermined one, `Sandbox.dll` should redirect the destination location of the file about to be created as shown in figure 3.6.



Figure 3.6: Create file interception diagram

Since the installer can also open and modify existing files on the user's PC simple redirection won't work, since the file won't be present in the predeter-

mined location and the application might fail. To counter this, Sandbox.dll should copy files that are accessed with modify permission enabled into a pre-determined location and allow all the manipulations required by the installer on the copy of the original file. In this way, the installer will have access to the same files as if it wasn't running in the sandbox, but no changes will be performed on the user's files, and after the deletion of the sandbox all modified files will be removed but originals will be untouched as shown in figure 3.7. In case the file in question was created by the installer, then all modifications will be performed on the file in the sandbox, since creation of the file was redirected as well.

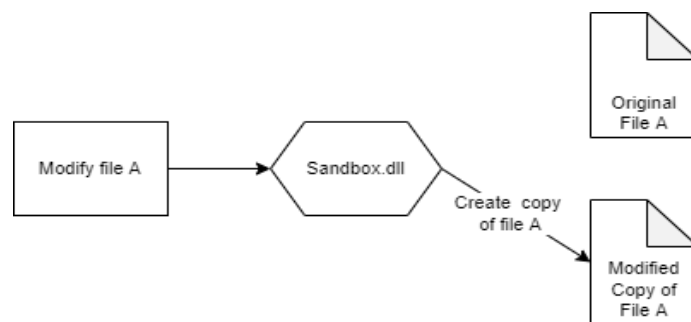


Figure 3.7: Modify file interception diagram

Registries are rather similar to files. Thus, registry key creation and modification processes are similar from the point of view of intercepting and sandboxing API calls. The main difference is that all registry keys will be saved into the SQLite database which can be reviewed or deleted after the installation is finished.

Image file structure

The software-base sandbox solution is possible because of the structure of the executable file, its layout in the memory, and the way it interacts with dynamically linked libraries. Some elements of the executable file format that are required for the functioning of the sandbox will be described in this section. Executable file either .EXE or .DLL is called an image file. An image file can be thought of as a memory image. The term image file is usually used instead of executable file, because the latter sometimes is taken to mean only a .EXE file [13].

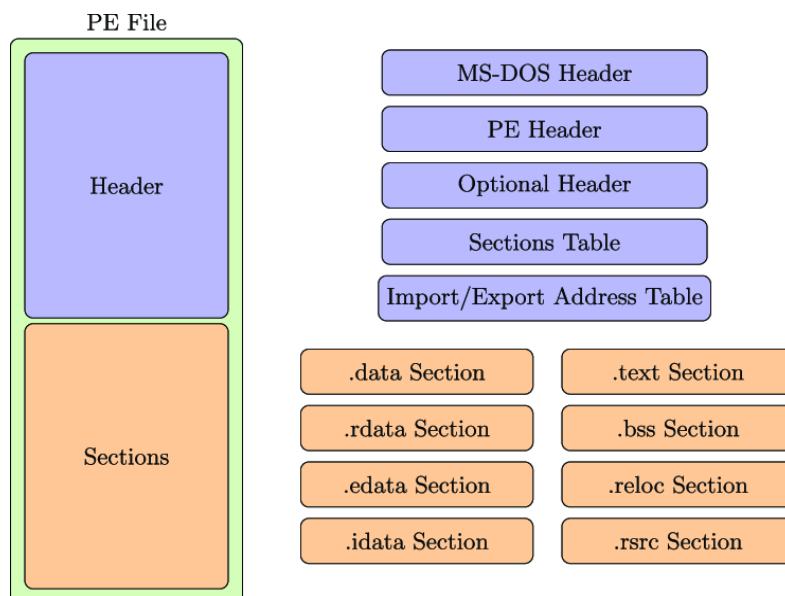


Figure 4.1: Portable executable file format. [14]

The MS-DOS Header – Every PE file begins with a small MS-DOS executable. The need for this stub executable arose in the early days of Windows,

4. IMAGE FILE STRUCTURE

before a significant number of consumers were running it. When executed on a machine without Windows, the program could at least print out a message saying that Windows was required to run the executable.

The first bytes of a PE file begin with the traditional MS-DOS header, called an `IMAGE_DOS_HEADER`. The only two values of any importance today are `e_magic` and `e_lfanew`. The `e_lfanew` field contains the file offset of the PE header. The `e_magic` field (a `WORD`) needs to be set to the value `0x5A4D`. There's a `#define` for this value, named `IMAGE_DOS_SIGNATURE`. In ASCII representation, `0x5A4D` is `MZ`, the initials of Mark Zbikowski, one of the original architects of MS-DOS. [15]

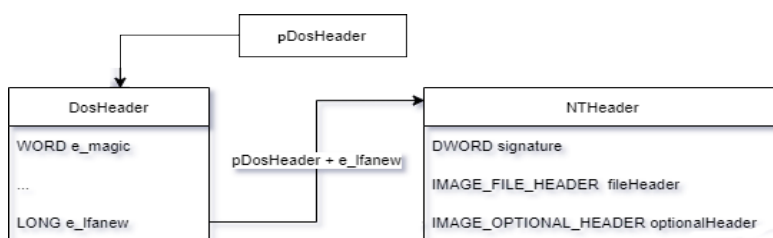


Figure 4.2: `NT_DOS_HEADER` location relative to `DOS_HEADER`

The `IMAGE_NT_HEADERS` structure (see figure 4.3) is the primary location where the specifics of the PE file are stored. Its offset is given by the `e_lfanew` field in the `IMAGE_DOS_HEADER` at the beginning of the file. There are actually two versions of the `IMAGE_NT_HEADER` structure, one for 32-bit executables and the other for 64-bit versions. The only correct, Microsoft-approved way of differentiating between the two formats is via the value of the Magic field in the `IMAGE_OPTIONAL_HEADER` [15]

```
1 typedef struct _IMAGE_NT_HEADERS {
2     DWORD Signature;
3     IMAGE_FILE_HEADER FileHeader;
4     IMAGE_OPTIONAL_HEADER32 OptionalHeader;
5 } IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;
```

Figure 4.3: Locate `IMAGE_NT_HEADER` structure

In an image file, the address of an item after it was loaded into memory, with the base address of the image file subtracted from it is called Relative virtual address (RVA). The RVA of an item almost always differs from its position within the file on disk (file pointer). In an object file, an RVA is less meaningful because memory locations are not assigned. In this case, an RVA would be an address within a section (described later in this table), to which a relocation is later applied during linking. For simplicity, a compiler should just set the first RVA in each section to zero. [16]

Knowing how RVA works allows us to locate some of the crucial parts of the image file, that can be used to locate dynamically loaded libraries and their functions. One of such parts is the Optional Header. Every image file has an optional header that provides information to the loader. This header is optional in the sense that the object file format doesn't require it. For image files, this header is required. An object file can have an optional header, but generally this header has no function in an object file except to increase its size [16].

From optional header it is then possible to locate the Export Directory Table as shown in figure 4.4. The export symbol information begins with the export directory table, which describes the remainder of the export symbol information. The export directory table contains address information that is used to resolve imports to the entry points within this image. [16].

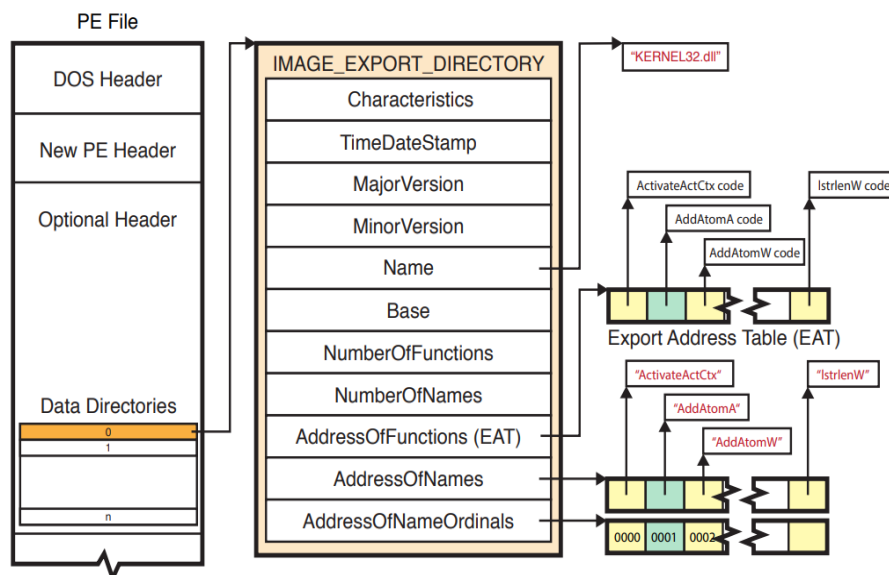


Figure 4.4: Export tables and symbols [17]

4.1 Image file in memory

Image file gets loaded into the memory with Windows loader utility. The image loader lives in the user-mode system DLL NTDLL.dll and not in the kernel library. Therefore, it behaves just like a standard code that is a part of a DLL and is subject to the same restrictions in terms of memory access and security rights. What makes this code special is the guarantee that it will always be present in the running process (NTDLL.dll is always loaded) and that it is the first piece of code to run in user mode as part of a new application. (When the system builds the initial context, the program counter, or

instruction pointer, is set to an initialization function inside NTDLL.dll. [18])

Because the loader runs before the actual application code, it is usually invisible to users and developers. Additionally, although the loader's initialization tasks are hidden, a program typically does interact with its interfaces during the run time of a program—for example, whenever loading or unloading a DLL or querying the base address of one. Some of the main tasks the loader is responsible for include [18]:

- Initializing the user-mode state for the application, such as creating the initial heap and setting up the thread-local storage (TLS) and fiber-local storage (FLS) slots
- Parsing the import table (IAT) of the application to look for all DLLs that it requires (and then recursively parsing the IAT of each DLL), followed by parsing the export table of the DLLs to make sure the function is actually present (Special forwarder entries can also redirect an export to yet another DLL.)
- Loading and unloading DLLs at run time, as well as on demand, and maintaining a list of all loaded modules (the module database)
- Allowing for run-time patching (called hotpatching) support, explained later in the chapter
- Handling manifest files
- Reading the application compatibility database for any shims, and loading the shim engine DLL if required
- Enabling support for API sets and API redirection, a core part of the MinWin refactoring effort
- Enabling dynamic runtime compatibility mitigations through the SwitchBranch mechanism

Without this functionality applications wouldn't be able to run. Since the Windows loader is part of NTDLL all processes are subjected to the same loader behaviour. Loader is also responsible for import-parsing (processing IAT and making imported functions available for calling within a program)

During this step, the loader will do the following:

- Locate each DLL referenced in the import table of the process' executable image.
- Check whether the DLL has already been loaded by checking the module database. If it doesn't find it in the list, the loader opens the DLL and maps it into memory.

- During the mapping operation, the loader first looks at the various paths where it should attempt to find this DLL, as well as whether this DLL is a “known DLL”, they are just like any other DLL except that the operating system always looks for them in the same directory in order to load them. Certain deviations from the standard lookup algorithm can also occur, either through the use of a .local file (which forces the loader to use DLLs in the local path) or through a manifest file, which can specify a redirected DLL to use to guarantee a specific version.
- Relocation happens after the DLL has been found on disk and mapped. The loader checks whether the kernel has loaded it somewhere else. If the loader detects relocation, it parses the relocation information in the DLL and performs the operations required. If no relocation information is present, DLL loading fails.
- The loader then creates a loader data table entry for this DLL and inserts it into the database.
- After a DLL has been mapped, the process is repeated for this DLL to parse its import table and all its dependencies.
- After each DLL is loaded, the loader parses the IAT to look for specific functions that are being imported. Usually this is done by name, but it can also be done by ordinal (an index number). For each name, the loader parses the export table of the imported DLL and tries to locate a match. If no match is found, the whole loading process is aborted.
- The import table of an image can also be bound. This means that at link time, the developers already assigned static addresses pointing to imported functions in external DLLs. This removes the need to do the lookup for each name, but it assumes that the DLLs the application will use will always be located at the same address. Because Windows uses address space randomization, this is usually not the case for system applications and libraries.
- The export table of an imported DLL can use a forwarder entry, meaning that the actual function is implemented in another DLL. This must essentially be treated like an import or dependency, so after parsing the export table, each DLL referenced by a forwarder is also loaded and the loader goes back to the first step.
- The DllMain function is called when the system starts or terminates a process or thread, it calls the entry-point function for each loaded DLL using the first thread of the process. The system also calls the entry-point function for a DLL when it is loaded or unloaded using the LoadLibrary and FreeLibrary functions [19].

4. IMAGE FILE STRUCTURE

After all imported DLLs (and their own dependencies, or imports) have been loaded, all the required imported functions have been looked up and found, and all forwarders also have been loaded and processed, the step is complete: all dependencies that were defined at compile time by the application and its various DLLs have now been fulfilled. During execution, delayed dependencies (called delay load), as well as run-time operations (such as calling LoadLibrary) can call into the loader and essentially repeat the same tasks 18

Implementation details

To create a working proof of concept, it is crucial to study the installer application and its use of Windows API. In previous chapters, it was shown that there exist websites that host “InnoSetup” based installers and that there are frameworks designed to work with it to push unwanted software to users. This installer will be used as a reference. It is a legitimate software that is popular among the developers and isn’t necessarily malicious it is being abused by the packaging parties, because it is free to use and exists for over twenty years, so malicious frameworks and strategies were designed around it. InnoSetup version 6.2.0 uses itself for install so it was used to study the normal behavior of the installer and develop a proof of concept. To identify API calls that are typical for this installer, “APImonitor” by Rohitab software was used ¹. Since InnoSetup uses a pascal scripting language to configure its installer and doesn’t provide control over what system API functions will be used it is safe to assume that in the standard case all InnoSetup based installers will have a similar set of utilized API calls ²⁰. Usage of untypical APIs can be considered as a highly suspicious behaviour.

This Windows API calls were identified as those that required hooking for registry keys:

- `NtCreateKey` – function creates a new registry key or opens an existing one.
- `NtOpenKey` – function opens an existing registry key.
- `NtSetValueKey` – function creates or replaces a registry key’s value entry.
- `NtQueryValueKey` – function returns a value entry for a registry key.
- `NtQueryKey` – function provides information about the class of a registry key, and the number and sizes of its subkeys.

¹Link to the APImonitor by Rohitab website – <http://www.rohitab.com/apimonitor>

5. IMPLEMENTATION DETAILS

- `NtClose` – function closes an object handle.

File API calls:

- `NtOpenFile` – opens an existing file, device, directory, or volume, and returns a handle for the file object.
- `NtCreateFile` – creates a new file or directory, or opens an existing file, device, directory, or volume.
- `NtQueryAttributesFile` – retrieves basic attributes for the specified file object.
- `MoveFileW` – moves an existing file or a directory, including its children.
- `MoveFileExW` – moves an existing file or directory, including its children, with various move options.

`MoveFile` and `MoveFileExW` functions are not part of the NTDLL library, since internally they are utilizing `memcpy` mechanics that are not specific to file move calls as shown in figure 5.1.

innosetup-6.2.0.tmp	MoveFileW ("C:\Program Files (x86)\Inno Setup 6\is-7NDD0.tmp", "C:\Progra...	BOOL	TRUE
KERNELBASE.dll	RtlDosDeviceName_U ("C:\Program Files (x86)\Inno Setup 6\unins000.e...	ULONG	0
KERNELBASE.dll	RtlDosPathNameToNtPathName_U ("C:\Program Files (x86)\Inno Setup 6\...	BOOLEAN	TRUE
KERNELBASE.dll	NtOpenFile (0x0019e8e8, DELETE FILE_READ_ATTRIBUTES SYNCHRONIZE,	NTSTATUS	STATUS_SUCCESS
KERNELBASE.dll	NtQueryInformationFile (0x00000484, 0x0019e8b4, 0x0019e888, 8, FileAtt...	NTSTATUS	STATUS_SUCCESS
KERNELBASE.dll	RtlDosPathNameToNtPathName_U ("C:\Program Files (x86)\Inno Setup 6\...	BOOLEAN	TRUE
KERNELBASE.dll	RtlAllocateHeap (0x00960000, 0, 120)	PVOID	0x00a1aa00
KERNELBASE.dll	memcpy (0x00a1aa0c, 0x009803c0, 104)	void*	0x00a1aa0c
KERNELBASE.dll	NtSetInformationFile (0x00000484, 0x0019e8b4, 0x00a1aa00, 120, FileRe...	NTSTATUS	STATUS_SUCCESS
KERNELBASE.dll	RtlFreeHeap (0x00960000, 0, 0x00a1aa00)	BOOLEAN	TRUE
KERNELBASE.dll	NtClose (0x00000484)	NTSTATUS	STATUS_SUCCESS
KERNELBASE.dll	RtlFreeHeap (0x00960000, 0, 0x00980528)	BOOLEAN	TRUE
KERNELBASE.dll	RtlFreeHeap (0x00960000, 0, 0x009803c0)	BOOLEAN	TRUE
innosetup-6.2.0.tmp	GetLastError ()	DWORD	ERROR_SUCCESS

Figure 5.1: `MoveFileW` system calls

During the execution process InnoSetup creates temporary files in the `C:\Users\user\AppData\Local\Temp\is-*.tmp` folder, where `*` represents random four to five character alphanumeric string. One of those files is an executable that performs the installation process. The original file that user interacts with is just a launcher for it and finishes its execution after the secondary process finishes. All interactions with the UI are handled by the secondary process. After studying the API calls it was determined that InnoSetup uses `CreateProcessW` function. Also, during research it was discovered that some libraries are loaded with `LoadLibraryW` function, so two additional functions were hooked:

- `CreateProcessW` – creates a new process and its primary thread. The new process runs in the security context of the calling process.
- `LoadLibraryW` – loads the specified module into the address space of the calling process. The specified module may cause other modules to be loaded.

System calls description was sourced from official MSDN documentation website [\[21\]](#).

5.1 Function hooking

In order to intercept function calls after DLL had been attached to the installer process, `DLLMain` will be called with `fdwReason` equal to `DLL_PROCESS_ATTACH`. Since this code will be executed in the address space of the target process it is possible to locate `TEB` – thread environment block and from there `PIMAGE_DOS_HEADER` of all data table entries can be located as shown in figure [5.2](#).

```

1     PTEB teb = (PTEB) NtCurrentTeb();
2     PPEB peb = teb->ProcessEnvironmentBlock;
3     PPEB_LDR_DATA peb_ldr_data = peb->Ldr;
4     PLIST_ENTRY list_head = &peb_ldr_data->
5     InMemoryOrderModuleList;
6     PLIST_ENTRY list_entry = list_head;
7
8     while ((list_entry = list_entry->Flink) != list_head) {
9         PLDR_DATA_TABLE_ENTRY data_table_entry =
10            CONTAINING_RECORD(list_entry,
11            LDR_DATA_TABLE_ENTRY, InMemoryOrderLinks);
12
13         locateAndSwapAll((PIMAGE_DOS_HEADER)
14            data_table_entry->DllBase, STEAL);
15     }

```

Figure 5.2: Locate `PIMAGE_DOS_HEADER` of loaded DLL libraries

After the location of `IMAGE_DOS_HEADER` pointers, it is possible to locate a pointer to `IMAGE_THUNK_DATA` as shown in figure [5.3](#).

5. IMPLEMENTATION DETAILS

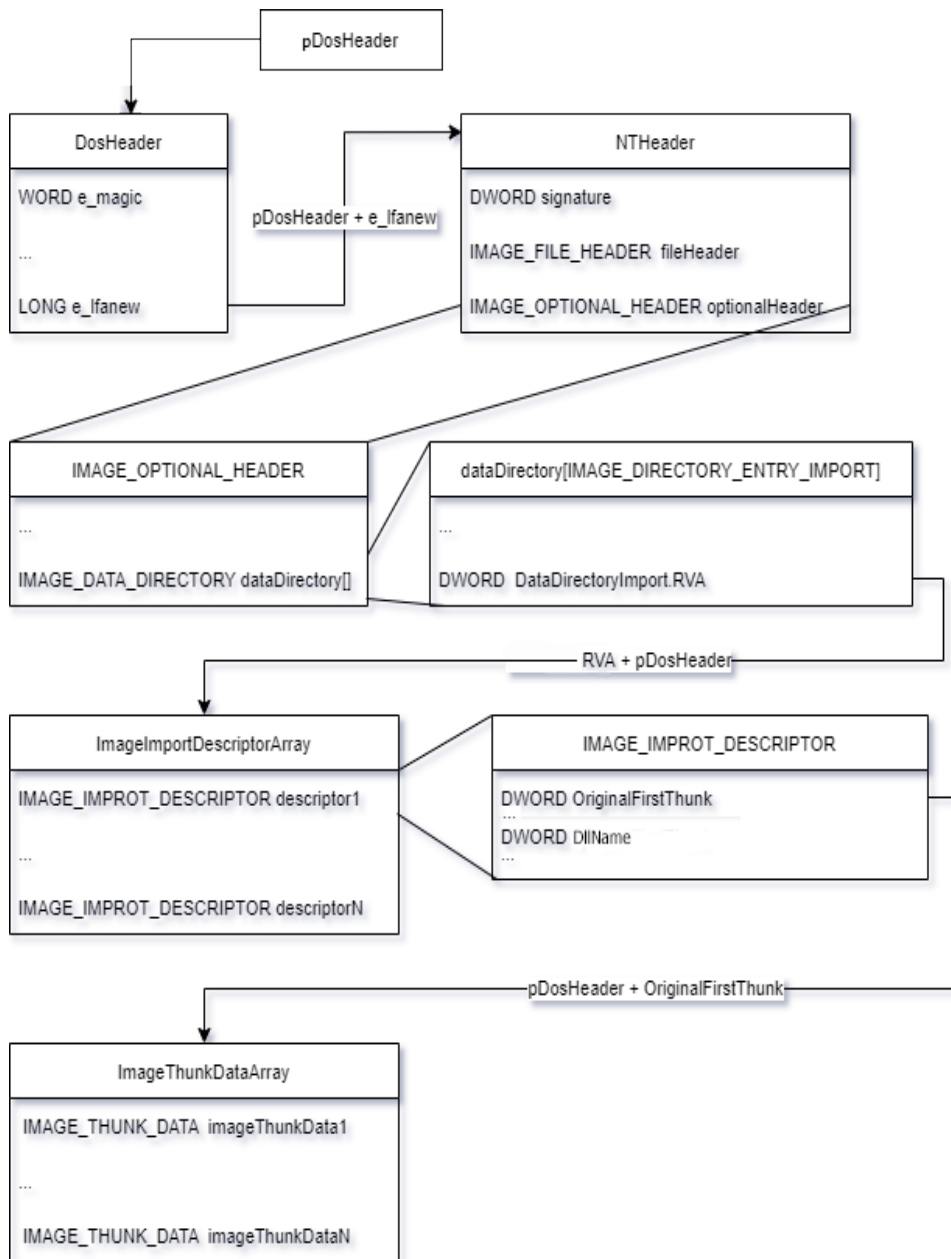


Figure 5.3: Process of location of an ImageThunk from a DosHeader

To identify a function it is required to identify the DLL name, that can be computed as

`pImageImpDescArray[imageDescIndex].Name + ((BYTE *) pDosHeader`
and function name. There are two main ways how Windows handles function imports – import by name shown in figure 5.4 and import by ordinal [22].

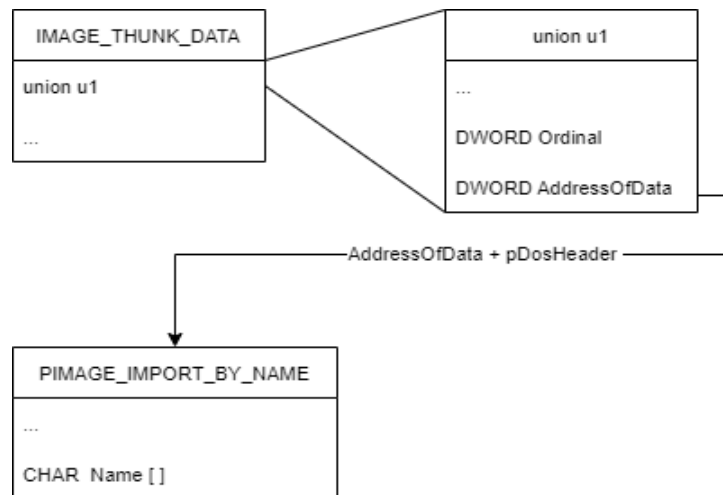


Figure 5.4: Import by name process overview

To determine whether the function in question is imported by name or by ordinal, it is necessary to check the most significant bit of the ordinal. If this bit is set, then the function is imported by ordinal as shown in figure 5.5.

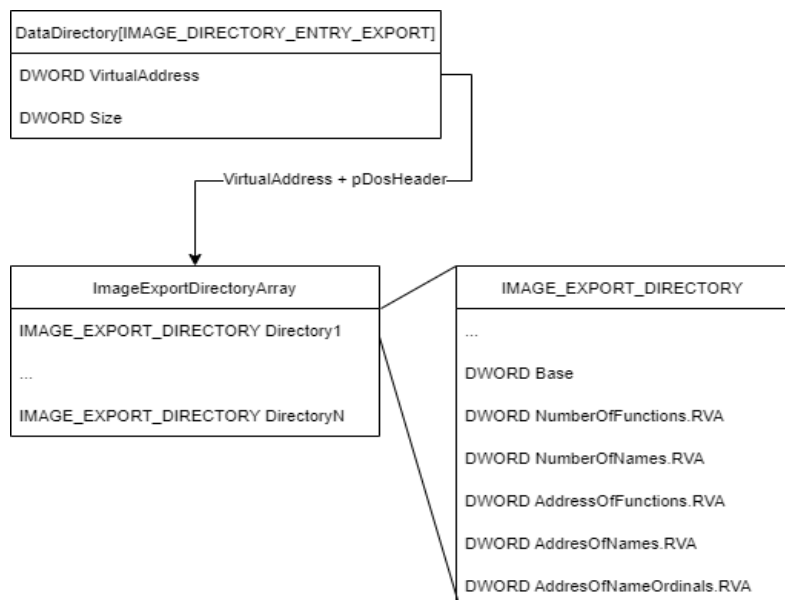


Figure 5.5: Import by ordinal. Function metadata location.

An export's ordinal is the index into the `AddressOfFunctions` array (the 0-based position in this array) plus the "Base" mentioned above. In most cases, the "Base" is 1, which means the first export has an ordinal value of 1, the second has an ordinal value of 2 and so on.

After the "AddressOfFunctions" RVA we find a RVA to the array of 32-bit-RVAs to symbol names "AddressOfNames", and a RVA to the array of 16-bit-ordinals "AddressOfNameOrdinals". Both arrays have "NumberOfNames" elements. The symbol names may be missing entirely, in which case the "AddressOfNames" is 0. Otherwise, the pointed-to arrays are running parallel, which means their elements at each index belong together. The "AddressOfNames"-array consists of RVAs to 0-terminated export names; the names are held in a sorted list [22] (i.e. the first array member is the RVA to the alphabetically smallest name; this allows efficient searching when looking up an exported symbol by name). According to the PE specification, the "AddressOfNameOrdinals"-array has the ordinal corresponding to each name [22]. Based on this information, it is possible to locate the name of a function.

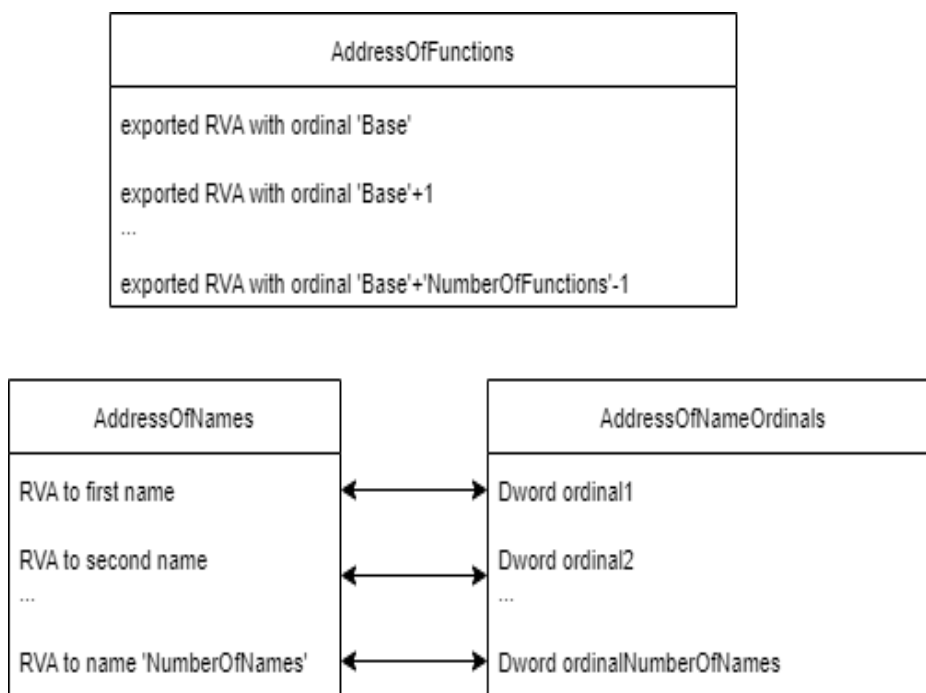


Figure 5.6: Function names location, derived from `AddressOfNameOrdinals`

Once the DLL name and function name have been identified, it is possible to hook the function by swapping the function pointer from injected `Sandbox.dll` with the original function pointer. Since there are two different ways to import a function, the process of hooking the functions will be different as well. Before process functions can be swapped corresponding memory page should

made be writable, this can be achieved with a call to the `VirtualProtect` function. And once the process of swapping is complete, it is necessary to restore the original page attributes.

For functions imported by name, the process of hooking is relatively straightforward. `FirstThunk` of the `IMAGE_IMPORT_DESCRIPTOR` that was used to locate function name should be used. `FirstThunk` is an RVA pointing to the `ImageThunkDataArray` its elements correspond to those in in the `OriginalFirstThunkArray` as shown in the figure 5.7. After the original function was located its address is saved in the `Sandbox.dll` and the original address is overwritten with the injected function as shown in figure 5.7.

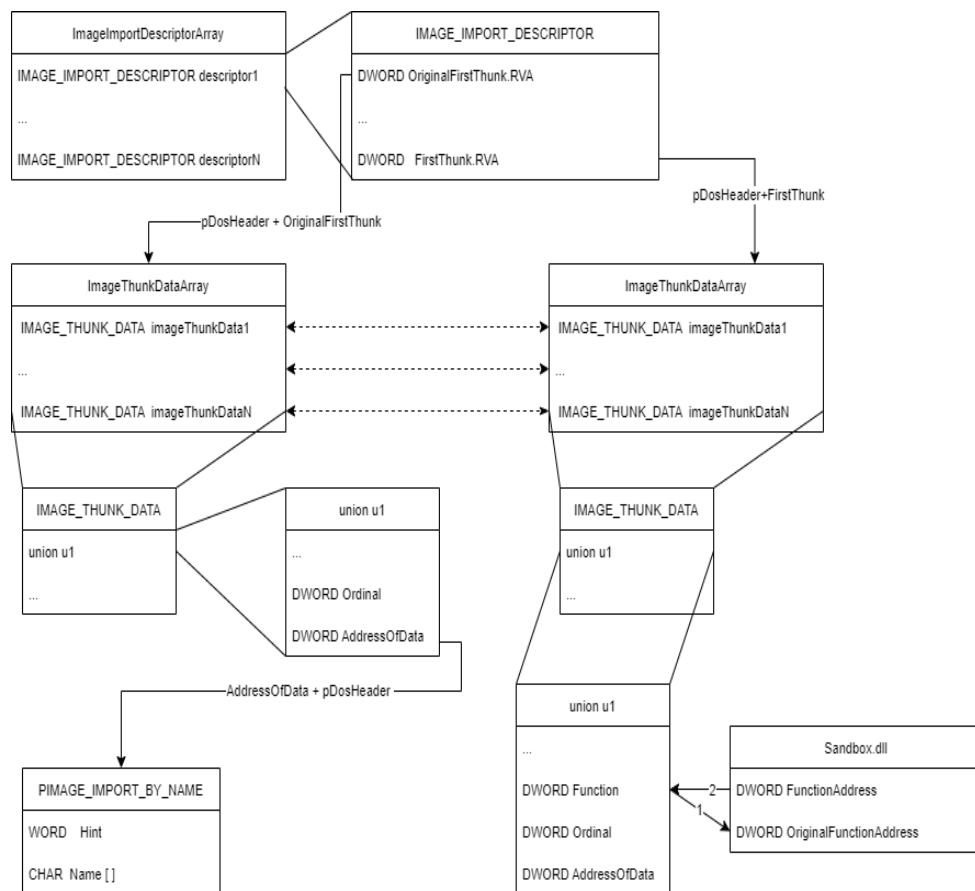


Figure 5.7: Import by name. Function swap schematic process.

For functions imported by ordinal it is required to first locate function address in the `AddressOfFunctionsArray` that can be accessed from `IMAGE_EXPORT_DIRECTORY` and function address has to be computed 5.8

Since one function can be imported multiple times, it is important to make sure that no cycles will be created during the substitution process, so a check

5. IMPLEMENTATION DETAILS

```
AddressOfFunctionsArray[ordinal - pImageExpDescArray[0].Base]
= functioFromInjectedDll - pDosHeader
```

Figure 5.8: Overwriting the function address for functions imported by ordinal

should be placed to ensure that the address of the function from DLL is not equal to the function from Sandbox.dll

Once all functions were identified and swapped all subsequent calls from the installer application will be redirected to Sandbox.dll and handled there in a manner that prevents modifications on the user's PC.

5.2 Registry key management

To prevent changes in the registry original operations to change registry value such as `NtSetValueKey` and `NtCreateKey` are never called within the sandbox, so these operations are 100% emulated. Accessed registries and values are stored in an SQLite database that allows users to review what values were created, changed, or accessed during the execution. Preventing changes in a registry hive also helps against programs establishing persistence on the user's PC.

It was attempted to allow users to define registry keys in the SQLite database before sandbox and to define returned handles manually. However, some values caused unpredictable behavior from the OS on subsequent calls to this handle. This happened when predefined handles had small values as shown in figure 5.9.

tib.exe	RegQueryValueExW (0x00000005, "param", NULL, 0x006ffc0, 0x006ffc0, 0x006fca4)	LONG
KERNELBASE.dll	RtlInitUnicodeStringEx (0x006ff96c, "param")	NTSTATUS
RPCRT4.dll	RtlEnterCriticalSection (0x76cdb9b4)	NTSTATUS
KERNELBASE.dll	RtlInitUnicodeStringEx (0x006ff8c8, "rpcrt4.dll")	NTSTATUS
KERNELBASE.dll	LdrLoadDll (1, 0x006ff8b8, 0x006ff8c8, 0x006ff8bc)	NTSTATUS
RPCRT4.dll	NtQuerySystemInformation (SystemBasicInformation, 0x006ff8b0, 44, NULL)	NTSTATUS
RPCRT4.dll	RtlGetNtProductType (0x006ff8ac)	BOOLEAN
RPCRT4.dll	_wcsicmp ("tib.exe", "lsass.exe")	int
KERNELBASE.dll	RtlGetCurrentTransaction ()	HANDLE
KERNELBASE.dll	RtlSetCurrentTransaction (NULL)	LOGICAL
KERNELBASE.dll	RtlAcquireSRWLockExclusive (0x754988a8)	VOID
KERNELBASE.dll	RtlReleaseSRWLockExclusive (0x754988a8)	VOID
KERNELBASE.dll	RtlSetCurrentTransaction (NULL)	LOGICAL
KERNELBASE.dll	RtlCreateUnicodeStringFromAsciiz (0x006ff844, "Software\Microsoft\Rpc")	BOOLEAN
KERNELBASE.dll	NtOpenKeyEx (0x006ff8d4, KEY_READ, 0x006ff794, 0)	NTSTATUS
teeCgrind32.dll	EnterCriticalSection (0x648de904)	void
teeCgrind32.dll	EnterCriticalSection (0x648de94c)	void
MSVCP140D.dll	EnterCriticalSection (0x64daa3d8)	void
MSVCP140D.dll	LeaveCriticalSection (0x64daa3d8)	void
MSVCP140D.dll	EnterCriticalSection (0x64daa3d8)	void
MSVCP140D.dll	LeaveCriticalSection (0x64daa3d8)	void

API	EnterCriticalSection
Module	Kernel32.dll
Category	Critical Section

Figure 5.9: API call stack for predefined handle value equal to five

As is shown on figure 5.9, after calling for `RegQueryValueExW` with the han-

dle value equal to five `LdrLoadD11` is invoked and library “`rpct4.dll`” is loaded. According to the official documentation: “Microsoft Remote Procedure Call (RPC) defines a powerful technology for creating distributed client/server programs. The RPC run-time stubs and libraries manage most of the processes relating to network protocols and communication. This enables you to focus on the details of the application rather than the details of the network.” [23].

The same calls with different handle values (six) resulted in different call stacks as shown in figure 5.10.

Function Name	Return Type	Out
RegQueryValueExW (0x00000006, "param", NULL, 0x005af7e0, 0x005af7ec, 0x005af7d4)	LONG	ERF
RtlInitUnicodeStringEx (0x005af49c, "param")	NTSTATUS	STA
memset (0x005af1f8, 0, 394)	void*	0x0
NtQueryKey (0x00000006, KeyNameInformation, 0x005af1f8, 392, 0x005af1f0)	NTSTATUS	STA
EnterCriticalSection (0x64daa3d8)	void	
LeaveCriticalSection (0x64daa3d8)	void	
EnterCriticalSection (0x64daa3d8)	void	
LeaveCriticalSection (0x64daa3d8)	void	
EnterCriticalSection (0x64daa3d8)	void	
LeaveCriticalSection (0x64daa3d8)	void	
API LeaveCriticalSection	void	

Figure 5.10: API call stack for predefined handle value equal to six

After some testing, it was observed that this behavior only occurs when handle values are small numbers. Since specifying custom handles isn’t a core functionality, the reason for this behavior wasn’t determined and this feature wasn’t developed. It is not guaranteed that this behavior won’t be displayed for some other values, but it wasn’t observed outside described scenario. To mitigate this issue it is possible to intercept also `RegQueryValueExW` but since this call is not a part of `NTDLL` and wasn’t required to be intercepted it is outside the scope of the current thesis.

To improve the performance of the sandbox during run-time, registries are stored in `map` containers as shown in figure 5.11.

```
map<wstring, MyRegKeyOpen> openedRegisters;
map<MAX_WORD, wstring> openedRegisterLocator;
```

Figure 5.11: Containers used to store registry key data during sandbox run-time.

`openedRegisters` is a `map` that keeps the information about opened registries. Key – registry key path. Value – the class that holds all required information about a registry key shown in figure 5.12. Since one registry key can have multiple values it has `map` container that holds information about key values. The key to this `map` is the name of the Value – data stored in

```
class MyRegKeyOpen {
    ACCESS_MASK accessMask;
    ULONG OpenOptions;
public:
    HANDLE resHandle = NULL;
    size_t reference = 0;
    bool emulated = false;
    map<wstring, MyRegValue> created_values;
    OBJECT_ATTRIBUTES objectAttributes;
    ...
}
```

Figure 5.12: Container class used to store registry key data during sandbox run-time.

the registry key's value as shown in figure [5.13](#). `openedRegisterLocator` is

```
class MyRegValue {
    HANDLE KeyHandle = NULL;
public:
    PUNICODE_STRING ValueName = NULL;
    ULONG TitleIndex = NULL;
    ULONG Type = NULL;
    PVOID Data = NULL;
    ULONG DataSize = NULL;
    NTSTATUS status;
    ...
}
```

Figure 5.13: Container class used to store registry key's values during sandbox run-time.

used to locate registry names based on the handles of the registry keys. Every time a registry key is opened or created, a pair handle – registry key name is stored in this map. To emulate registry key creation, it is necessary to return some value that will serve as a handle for possible future operations. Because `NtCreateKey` creates a new registry, opens it and returns a handle or opens one if already exists [\[24\]](#), it is reasonable to place all created key registers in `openedRegisters` container. As a returned value for created registers the address of the `created_values` map was used.

Having all of the above information, it is now possible to locate all registries based on their name or based on the handle. That is sufficient to intercept and emulate all the calls that were defined above in chapter [5](#). The original

`NtCreateKey` is never called during execution and the call is fully emulated as shown in figure 5.14

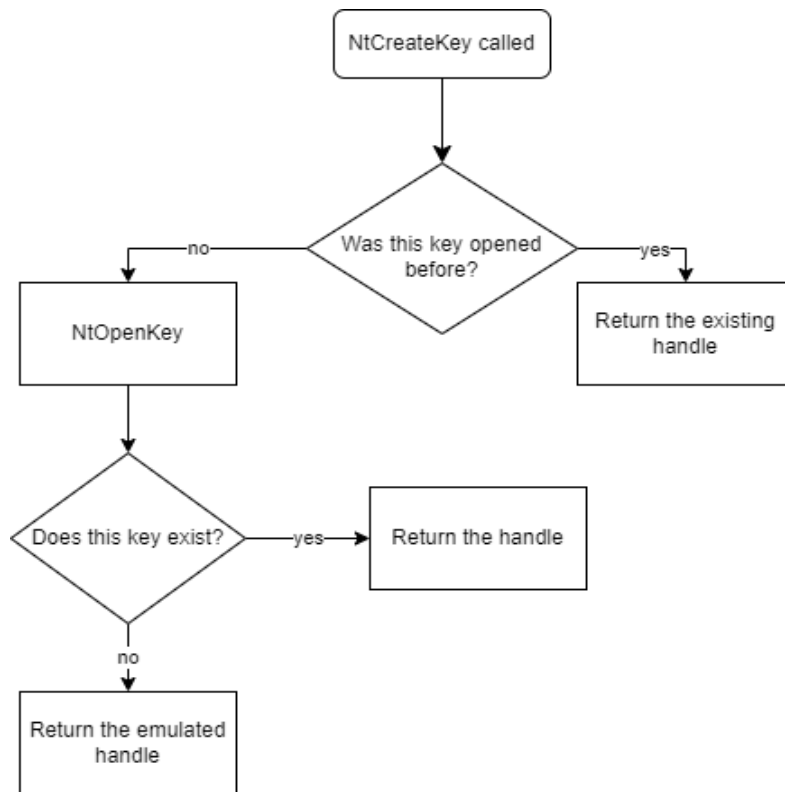


Figure 5.14: `NtCreateKey` emulation process.

Since `NtOpenKey` doesn't cause any changes in the system registry hive emulation is a straightforward process as shown in figure 5.15.

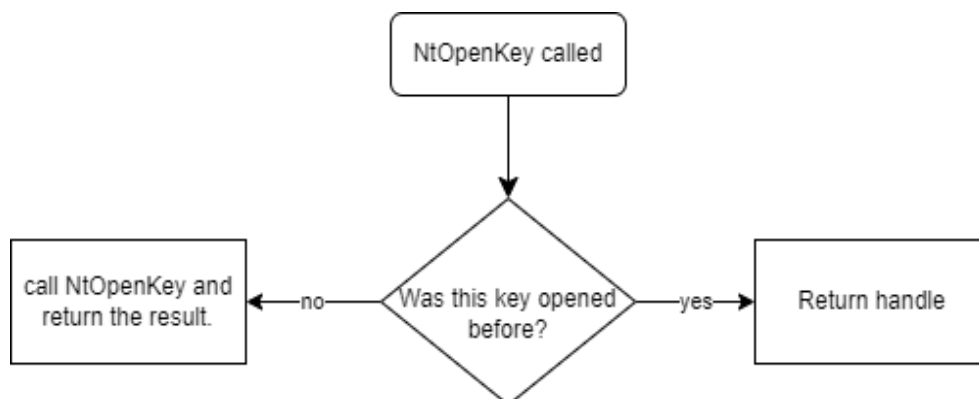


Figure 5.15: `NtOpenKey` emulation process.

`NtSetValueKey` takes a handle to an opened register where the value will be

set. The call to this function has to be fully emulated and the original function should never be called. During the development of the sandbox application, it was discovered that contrary to the MSDN documentation: “handle is created by a successful call to `ZwCreateKey` or `ZwOpenKey`” [golden’set’value] sometimes handles are used without a call to `NtOpenKey` or `NtCreateKey`. This behavior is probably an optimization from Windows OS to reuse recently opened key handles by accessing cached values. In case such a behavior is detected, `Sandbox.dll` retrieves a registry key name from the handle and adds this registry to `openedRegisters` and `openedRegisterLocator` maps as shown in figure 5.16.

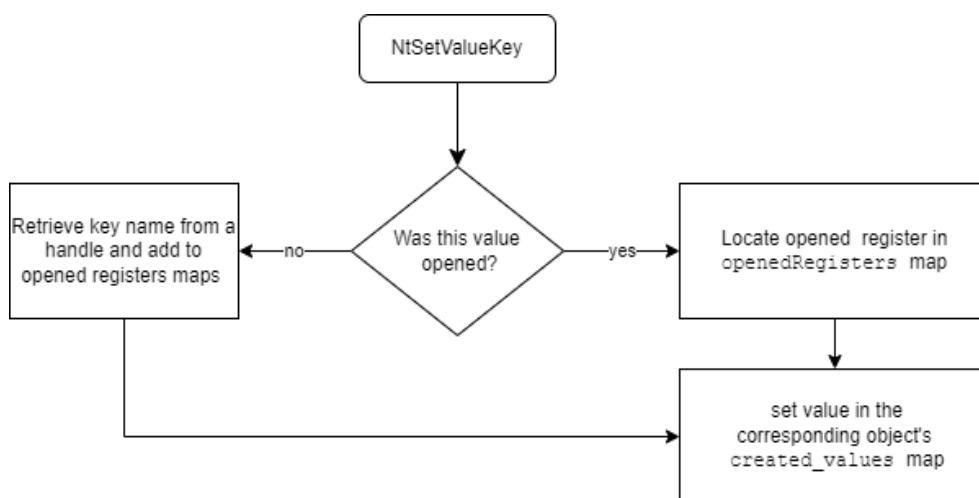


Figure 5.16: `NtSetValueKey` emulation process.

`NtQueryValueKey` function is used to retrieve the data stored in the registry key value. Since this call doesn't change registry key data its original function can be called. If the key was created by an installer program it will be emulated and data will be retrieved from `openedRegisters` map, if this value wasn't created by an installer, then it will be retrieved from the registry hive as usual.

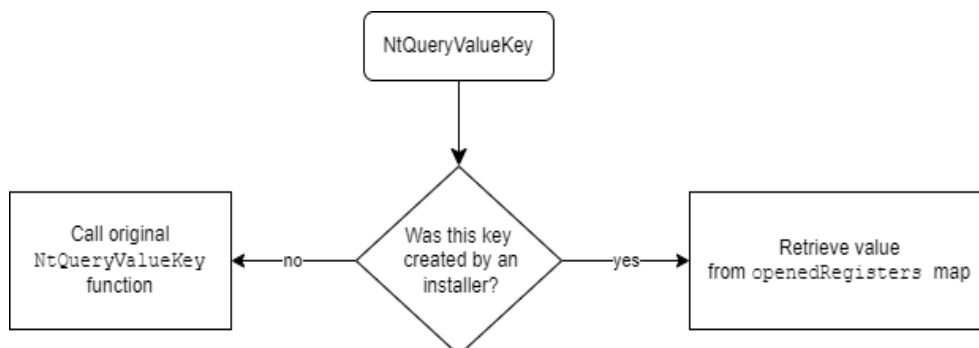


Figure 5.17: `NtQueryValueKey` emulation process.

Cached handles may be used in this function call, but since information is queried and not written, it is safe to call the original function as shown in figure 5.17

`NtQueryKey` function is sometimes called inside the `NtSetValueKey` (if this key was created under the sandbox library), so it has to be emulated. Since the target registry key could be one created by the installer itself and thus exists only in the RAM of the `Sandbox.dll` it is not always possible to directly call the original function even if it doesn't cause any changes. 5.18.

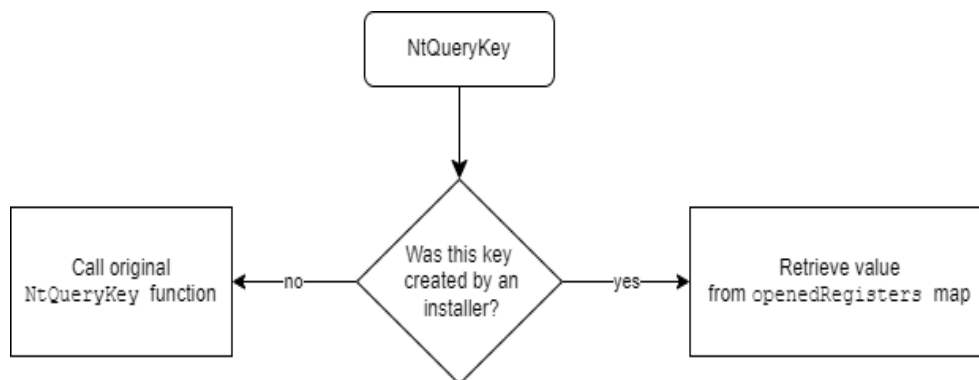


Figure 5.18: `NtQueryKey` emulation process.

`NtClose` this function closes handles, not necessarily ones that were created by `NtOpenKey` or `NtCreateKey` functions. So, if a handle in question was completely emulated by `Sandbox.dll` the function will always return success, otherwise the original function will be called as shown in figure 5.19.

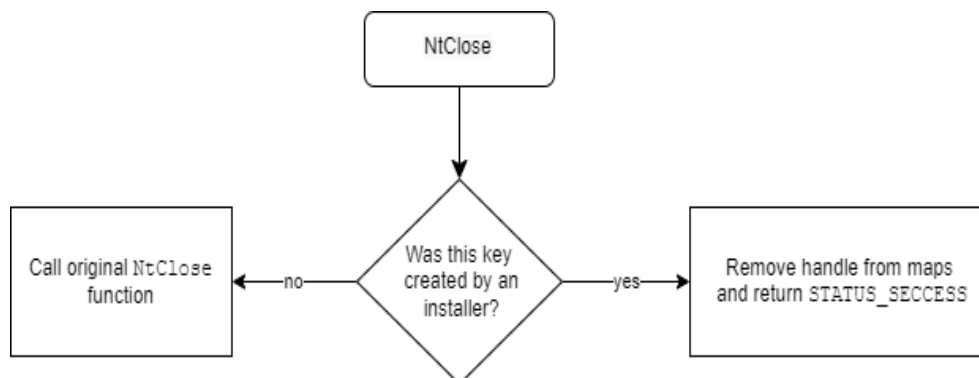


Figure 5.19: `myNtClose` process.

The process of writing data to the SQLite database involves functions that are being hooked by the sandbox. To avoid interception of the internal SQLite-related functions, a mechanism was introduced to call the original functions instead of the hooked ones. When an SQLite function is invoked, a critical

section is entered, then a boolean variable is set to true, while it is true all emulation will be paused and original functions will be called, once the process is finished the variable is set to false, the critical region is exited and emulation resumes.

5.3 File management

The main reason to intercept function calls related to file system is that installers will often create files across many directories in the file system, and if the user had to disable antivirus during the installation process and wants to scan files once it is finished they are left with a choice of scanning only directories that were specified as target ones while there is no guarantee that it is the only place where files are added, or do a whole system scan that might take multiple hours so it isn't likely to be desired. By redirecting all created files to a predefined location, it becomes possible to scan a single folder for viruses after a program was installed as shown in figure [5.20](#).

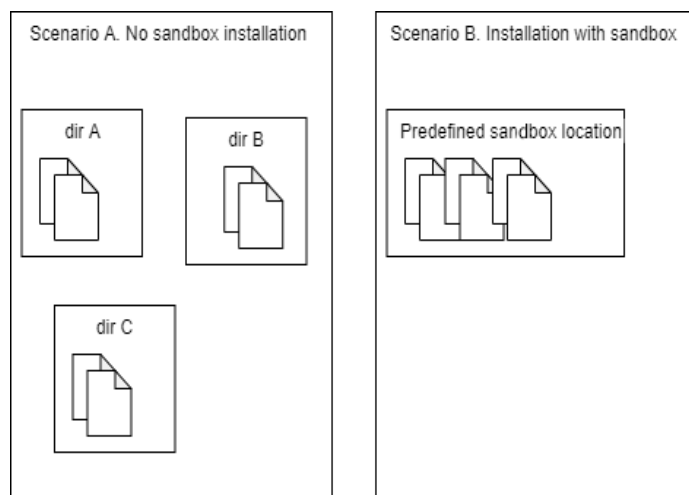


Figure 5.20: Files location with and without sandbox.

In addition, the installer might want to create files in a location inaccessible by the user and utilizing the sandbox mechanics, it becomes possible to install and review software in otherwise restrictive environments or run installers that were not designed to be run under a limited user without writing access to certain directories. To keep track of the original file name and redirected file `map<wstring, wstring> translation_map` container structure was utilized, where the key — original file path, value — new file path. After redirecting the file, the new path can be significantly longer and according to the official documentation: “In the Windows API (with some exceptions discussed in the following paragraphs), the maximum length for a path is MAX_PATH,

which is defined as 260 characters. A local path is structured in the following order: drive letter, colon, backslash, name components separated by backslashes, and a terminating null character. For example, the maximum path on drive D is "D:\some 256-character path string<NULL>" where "NULL" represents the invisible terminating null character for the current system codepage. (The characters < > are used here for visual clarity and cannot be part of a valid path string.) [25] Since the new path can be longer than the original one all the addresses are in form of "\\?\..." since this form allows having a longer path length of approximate 32,767 characters. Approximate because the "\\?\\" prefix may be expanded to a longer string by the system at run time, and this expansion applies to the total length [25]. In order to redirect the file path for NT functions it is necessary to modify `OBJECT_ATTRIBUTES` shown in figure 5.21 structure value – `ObjectName`. This structure holds the target's file name. It was discovered that if the original `UNICODE_STRING` `ObjectName` is modified – `Buffer` attribute replaced and `Length` `MaximumLength` values shown in figure 5.21 were recalculated, then program will fail in unpredictable manner, but if `ObjectName` is fully replaced by a structure that was allocated on the heap, then file redirection will be successful, so all functions that utilize `OBJECT_ATTRIBUTES` had their `ObjectName` replaced to one that would point them inside of the sandbox.

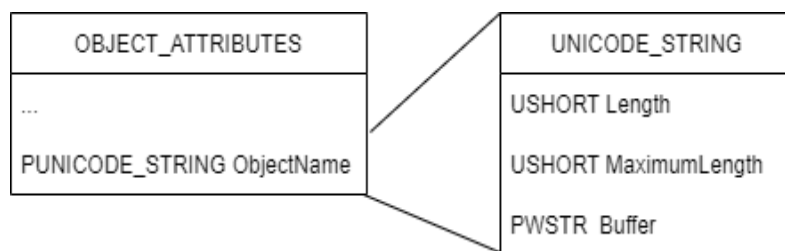


Figure 5.21: Name location in `OBJECT_ATTRIBUTES` structure

`NtOpenFile` is used to get open handle to a file. `ACCESS_MASK` `DesiredAccess` expresses the types of file access desired by the caller. If the desired access doesn't specify the access level of write or delete then this call is passed to an original function. Files that are opened with write access are copied to the predetermined location first, following the "copy first strategy". Since calls to `CopyFileW` are internally utilizing `NtOpenFile` functions the sandbox sets a flag, and when it is set original NTDLL functions are being called. After `CopyFileW` is called function `SetLastError(0)` is invoked to reset any error that could have occurred during this process, so if installer program will utilize `GetLastError` the original error will be reported as shown in figure 5.22.

5. IMPLEMENTATION DETAILS

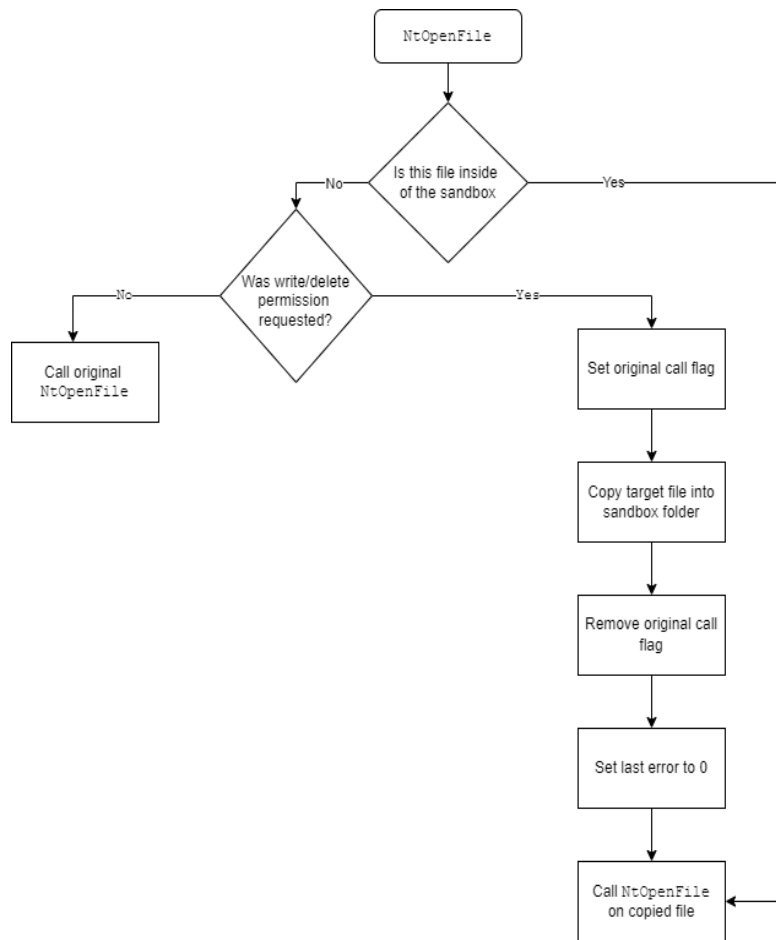


Figure 5.22: `myNtOpenFile` process.

`NtCreateFile` opens or creates a file. This function has a similar implementation logic to the `NtOpenFile` with the exception that if the file doesn't exist it will be created and that regardless of desired access file it will be first copied to the `Sandbox.dll` directory.

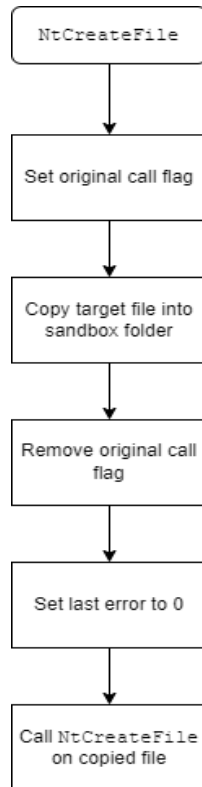


Figure 5.23: `myNtCreateFile` process.

`NtQueryAttributesFile` – retrieve file attributes. If the file in question exists in `translation_map` then the file attribute of the file in the sandbox folder is queried, otherwise the original file is queried as shown in figure 5.24.

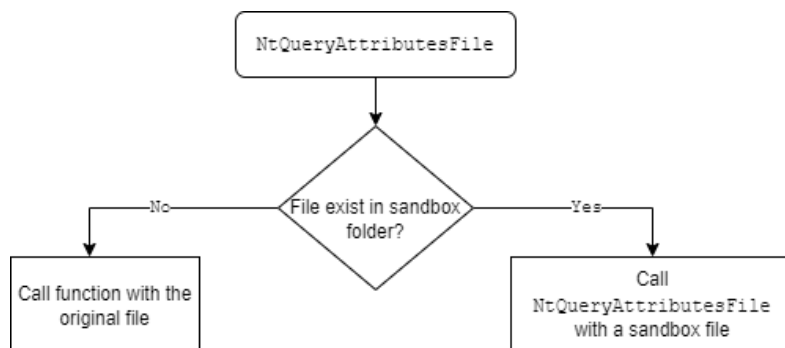


Figure 5.24: `myNtQueryAttributesFile` process.

`MoveFileW` and `MoveFileExW` – moves files to the specified location. Since this function doesn't utilize NTDLL hooking is done on user level in the `Kernel32.dll`. A call is intercepted, and the destination address is swapped for the address within a sandbox folder as shown in figure 5.25.

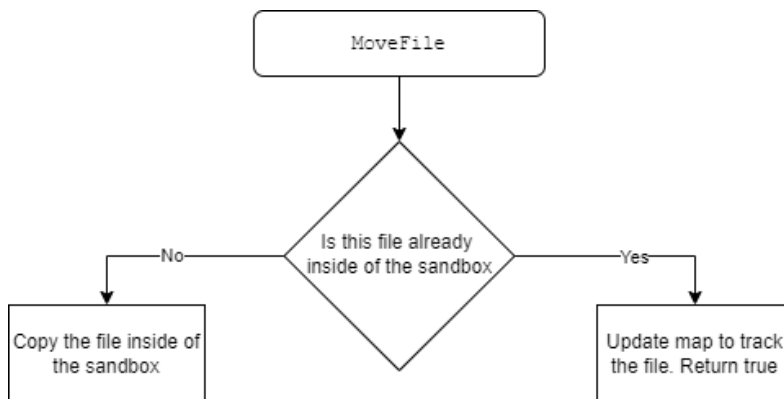


Figure 5.25: `MoveFile` process.

An installer can emit libraries that later will be loaded with a call to `LoadLibraryW`. Since newly created files are redirected into the sandbox library it is necessary to swap the original library path with one located in `translation_map`.

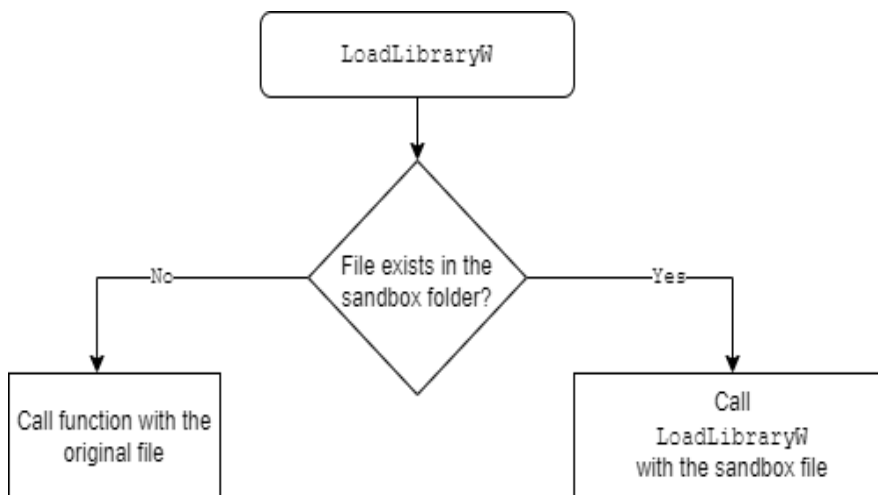


Figure 5.26: `LoadLibraryW` process. Redirecting call.

After the library is loaded into the memory, it is checked for referencing functions that should be managed by `Sandbox.dll`, and if this library imports them, function calls are swapped with ones that are managed by the sandbox. This process is identical to the process described in the above "Function hooking. Implementation details" section 5.1, with main difference being that

PIMAGE_DOS_HEADER can be obtained by a simple static cast of the LoadLibraryW return value into pointer to a DOS header. After obtaining the header it is possible to do the substitution as shown in section 5.1

CreateProcessW function can also be affected by the fact that the files are relocated into the sandbox folder. The strategy for executable files is rather similar to dynamically linked libraries. If the files are tracked by translation_map then the executable should be loaded from the sandbox folder, in other cases the original file should be called as shown in figure 5.27

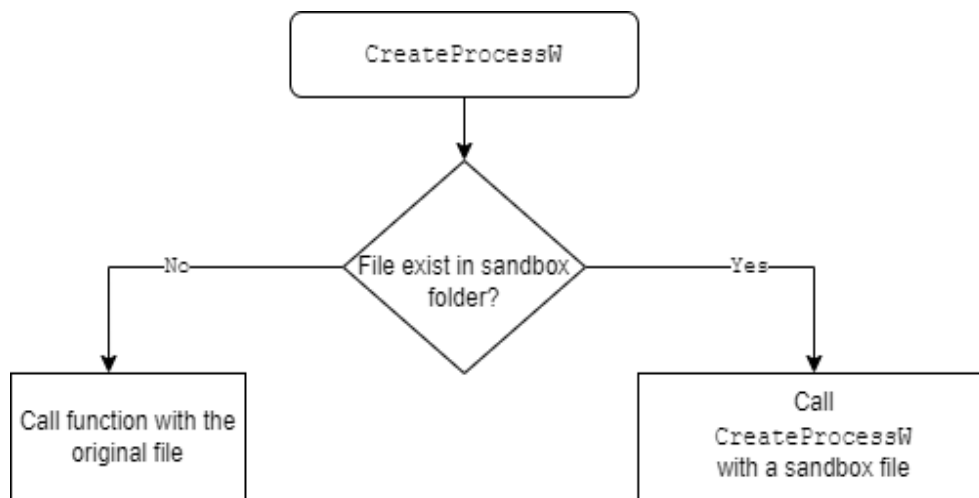


Figure 5.27: `CreateProcessW` process. Redirecting call.

After the executable file was substituted, the process is started in the suspended mode and PID is obtained by calling `lpProcessInformation->dwProcessId`. When PID of the process was obtained and architecture was determined by calling `IsWow64Process` with handle obtained from calling `OpenProcess`. After process architecture is established, the Injector can be invoked and the rest of the process is identical to initial startup described above as shown in section 5.1. This process doesn't account for the case when new process or DLL can be called from current directory or relative address, since InnoSetup doesn't exhibit such behaviour it is not handled, however it can be supported by expanding relative address first and then redirecting the call.

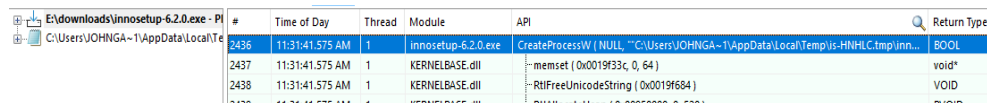
Testing against real-world potentially unwanted installer

As it was discussed in the chapter “Real-world scenarios of a malicious use of installers” [1] there exists a framework around InnoSetup that allows adding PUP to a program to enable monetization. Since InnoSetup is a rather popular tool to use when it comes to pushing software, it was chosen as a real-world software that will be used to test the proposed sand-boxing technique.

InnoSetup uses itself for the installation process, so it will be a good representation of other programs using it for potentially malicious use. To test the installation process, all the customizable parameters were left at the default values.

6.1 InnoSetup installation outside the sandbox

The installation process begins with InnoSetup creating two folders in `C:\Users\user\AppData\Local\Temp\is-*.tmp`, where `*` is a random five letter alphanumeric string. Both folders contain only one file each – `innosetup-6.2.0.tmp`. One of the files is used to create a new process as shown in figure [6.1].



#	Time of Day	Thread	Module	API	Return Type
2436	11:31:41.575 AM	1	innosetup-6.2.0.exe	CreateProcessW (NULL, ""C:\Users\OHNGA-1\AppData\Local\Temp\is-HNHLC.tmp\inn...	BOOL
2437	11:31:41.575 AM	1	KERNELBASE.dll	memset (0x0019f33c, 0, 64)	void*
2438	11:31:41.575 AM	1	KERNELBASE.dll	RtlFreeUnicodeString (0x0019f684)	VOID

Figure 6.1: InnoSetup new process created.

The installation continues from the new process. In the final stages of the installation another process is created

6.1. InnoSetup installation outside the sandbox

PID	Parent PID	Process Name	Working Set	Private Bytes	Working Set - Private Bytes
298928	1	innosetup-6.2.0.tmp	1000	1000	0
298929	298928	kernelbase.dll	1000	1000	0
298930	298928	kernel32.dll	1000	1000	0

Figure 6.2: InnoSetup new process at the final stage.

After the installation is finished folders in the `\Temp` directory are removed and folder `C:\Program Files (x86)\Inno Setup 6` and registry key in location `Computer\HKEY_CLASSES_ROOT\InnoSetupScriptFile` were created.

If the user will uninstall InnoSetup from Windows:

`Control Panel\Programs\Programs and Features`. Then all files mentioned above are deleted, including the registry keys. So during the normal installation/deinstallation process, InnoSetup doesn't leave files on the user's PC.

Now suppose that due to some external reason the installer fails during the installation process. This can be imitated by killing the process with Windows' task manager utility. If done at the right time, the registry keys and InnoSetup files may be already created but the program still isn't visible from the Windows uninstall utility. And temporary files are also not removed as shown in figure [6.3](#).

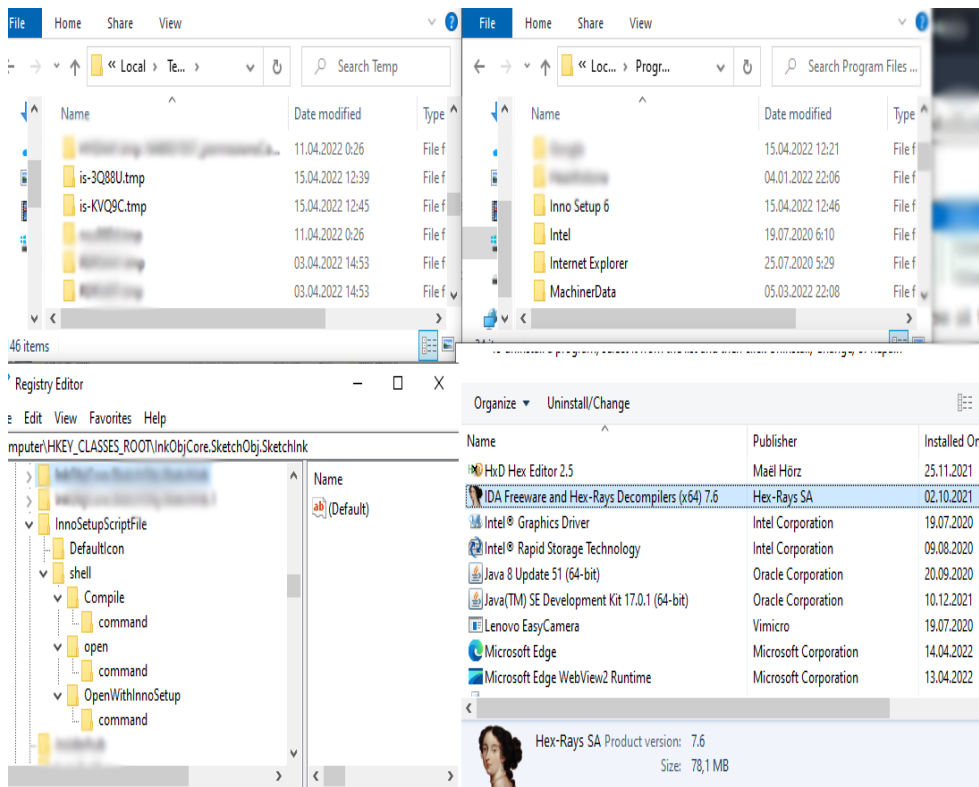


Figure 6.3: Files and registry keys on the user's PC after InnoSetup installation process was interrupted.

As it is demonstrated even installers created by developers of the software can't recover after some unexpected external errors. Such as a power outage or a random hardware fault, or a background application failing and causing the whole system to shut down. Now even if a user was paying attention and selecting carefully where the program is going to be installed, they would have no way of knowing that some files were also created in the `C:\Users\user\AppData\Local\Temp` folder and that these files should also be removed from the computer. It is easy to overlook registry keys that were created by the installer. So subsequent attempts to install the software might be unsuccessful — even if the “InnoSetup” original installer can recover from it, other installers might not be able to. The problem of folders with random names being created on the system isn't very likely to cause issues, however, some values might potentially be overwritten and data might be lost.

In the case of malicious or potentially unwanted software, it may try to create copies of itself in different places on the drive. Even some legitimate programs can try to establish persistence so that they are run on every boot of the system, and if this behavior is unsolicited, then it becomes annoying and the user might want to remove the application due to broken trust.

So as it was demonstrated installing an application with the assumption that it can always be fully removed later isn't always true.

6.2 InnoSetup installation in the sandbox

To test the sandbox application InnoSetup was run and examined. It was important to determine whether the sandbox application can contain all created files within a directory and to simulate registry key creation and modification. After running InnoSetup in the sandbox mode and upon reaching the first dialog window sandbox folder contained several files. Upon inspecting them there were DLL libraries, font files, and files used by the “InnoSetup” to launch a secondary process that handles the UI and installation as shown in figure 6.5. No folders were created in the `C:\Users\user\AppData\Local\Temp` path and from ApiMonitor the path to process executable it can be observed to be originating in the sandbox folder as shown in figure 6.4. After installation is finished all files are located in the sandbox folder and no registries were created by the installer. If, after inspection of created files and registries, the user chooses to delete the program, all they have to do is to remove the sandbox folder and the SQLite database file.

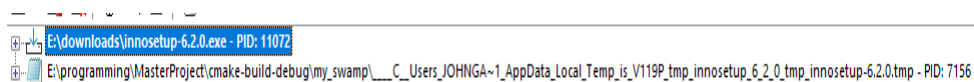
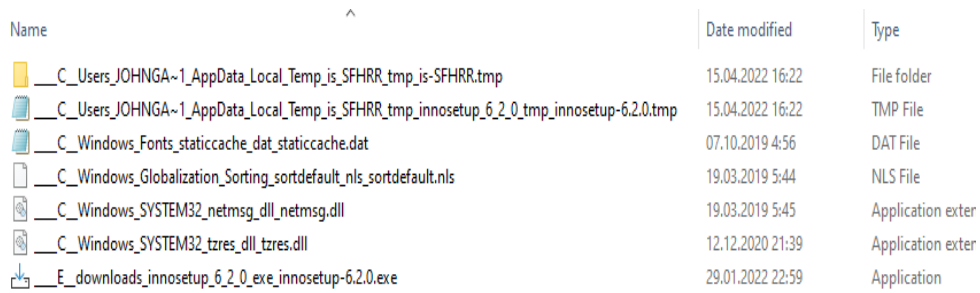


Figure 6.4: Origin of the child process during InnoSetup installation.

6.2. InnoSetup installation in the sandbox



Name	Date modified	Type
__C_Users_JOHNGA~1_AppData_Local_Temp_is_SFHRR_tmp_is-SFHRR.tmp	15.04.2022 16:22	File folder
__C_Users_JOHNGA~1_AppData_Local_Temp_is_SFHRR_tmp_innosetup_6_2_0_tmp_innosetup-6.2.0.tmp	15.04.2022 16:22	TMP File
__C_Windows_Fonts_staticcache_dat_staticcache.dat	07.10.2019 4:56	DAT File
__C_Windows_Globalization_Sorting_sortdefault_nls_sortdefault.nls	19.03.2019 5:44	NLS File
__C_Windows_SYSTEM32_netmsg_dll_netmsg.dll	19.03.2019 5:45	Application exter
__C_Windows_SYSTEM32_tzres_dll_tzres.dll	12.12.2020 21:39	Application exter
__E_downloads_innosetup_6_2_0_exe_innosetup-6.2.0.exe	29.01.2022 22:59	Application

Figure 6.5: Content of the sandbox folder upon reaching the first screen.

Now suppose the installer failed due to external reasons and a user wants to remove the generated files. Unlike the case where the program was installed outside of the sandbox, process of deletion of created files doesn't rely on the software provider creating an uninstallation utility, and once the sandbox folder is deleted all the files associated with the installer will be removed.

Removing process of the program installed within the sandbox is more intuitive and secure because it doesn't depend on the third party that is directly interested in tracking users' data and having their software on as many PCs as possible.

Discussion

Because there is a market for potentially unwanted programs installers themselves might be modified to detect and avoid proposed sandboxing solutions.

Since sandbox is injected as a DLL it is possible to list all libraries that are being used by the current process and if a library matches a predefined name then the program should change its behavior. This technique can be categorized as blacklisting. For example, if the installer detects that it is being run in sandbox mode it might not perform actions that can be categorized as suspicious and only install software that was originally advertised. Then after inspection users will think that the program is legitimate and install it without sandbox, and might get a PUP installed on their PC. Implementing this logic would require a low amount of effort from the attacker's side.

To mitigate this the injected sandbox DLL can have a random name, this will help to avoid being detected, especially if the name will be overwritten dynamically to some well-known library that is often used, but not loaded in the current process, and if such a library becomes loaded later, then sandbox would have to change its name. This is a moderate to low effort amount of work that would be required from the sandbox side.

Another approach that can be implemented by malicious installers is whitelisting. By only allowing a certain set of libraries they would be able to detect any unregistered DLLs and change the program's behavior to act as legitimate software. This approach is more reliable, but also harder to implement since installers are designed to be flexible and defining a certain set of possible libraries might be too restrictive, so it is less likely to be implemented.

The current version of the sandbox is using function names as a way to identify the function that needs to be swapped. The name doesn't necessarily have to be there. Functions can be imported exclusively by ordinals. Since ordinals and function addresses would be still possible to locate, logic could be implemented that matches assembly instructions to functions and decides if swapping is required. This approach from the attacker's side would require a moderate amount of effort, but from the side of the sandbox, a rather

significant amount of work would be required.

As an ultimate defense against sandbox the attackers might implement all the functions targeted by sandbox in assembler and use their functions instead. However, this approach would require a significant amount of effort and maintenance, since system functions can change between versions of Windows and be completely incompatible within themselves. And since the installer has to work on as many versions of Windows as possible, otherwise having one defeats the purpose a lot of the versions would have to be supported and constantly updated, so this approach is rather impractical.

7.1 Options for improvement

The proof of concept that was designed in the scope of this work is geared towards working with installers and the typical installation process. Some design decisions were chosen based on the normal case for the installer and would not support in the current stage more general use-cases. And as a result, this sandbox might not work with a general program.

Registry management is currently accomplished by maintaining map containers in the process memory. While this approach works with a single process and multiple threads, if the installer or some other program creates a new process that depends on the registry data from the original process, it will fail, since it is currently not shared. This can be resolved by completely removing the dependency on maps and using SQLite database not only for logging but as main the storage and communication channel between processes. In this case, processes would have access to the same data. But the problem described in the “Registry key management” section would [5.9](#) have to be resolved. It is possible that Windows reserves some handle values and if they are invoked, libraries are being loaded as a part of some optimization process. In this case, such handle values would have to be identified and avoided. It is also possible that during the `NtOpenKey` process Windows has to perform some internal preparations and so the handle needs to be registered, in this case, the issue can be resolved by calling the original `NtOpenKey` and then return some random handle value, this is a more likely scenario, based on the observed behavior.

Another improvement that could be implemented that potentially could help to resolve some anti-sandbox techniques described in the previous section is the ability to write through the installed files. Currently, after inspection of the files in the sandbox, a user would have to install the program again to use it as it was designed, but it is possible to record all the intercepted actions and play them back in reverse, effectively installing the application without the installer. This would help to reduce the risk of the installer detecting that it is being sandboxed and installing only advertised software, since even if a sandbox was detected, only verified files and registry keys would be deployed,

so the installer won't have an opportunity to run outside the sandbox. As a part of this feature, users might be able to choose if they want to write through all the changes, or not. For example, they might not want a program to run on system start-up, which is a common practice for even legitimate programs to do by default, forcing users to search and disable it in the setting post factum.

To improve user-friendliness, some sort of UI can be implemented, so the user can drag and drop programs that they want to run in the sandboxed mode. Sandbox can also be extended to generate a report that will help users to spot and understand different suspicious behaviors and their severity.

7.2 Modifications and cooperation

The core idea developed in this work can be easily adapted to function as a logger for files created and changed by some program. This would allow programs to work normally but users would no longer depend on the developer to remove the software because they would be able to delete all files that were ever created by the program. So the problem of residual files and registries will be solved. If the strategy of "copy on access" would be preserved then it would be possible not only to delete all the files and registers but also to restore files that were modified by the program to their original state. This can be especially useful in cases when there are no alternatives to the software, but it is rather obnoxious and forces some themes or extensions or widgets on the user that has to be removed separately after the main program is removed. In the real world, this behavior is typical for some antivirus software, that will try to add various utilities and browser extensions upon installation. Also, this would allow avoiding forceful data collection practices that companies implement when it is impossible to remove their software without answering some questionnaire first.

Software sandboxing might also become a part of an antivirus. It would allow users to install suspicious software in a safer environment. As was described above, software sandbox doesn't guarantee absolute security and it can be defeated by a sophisticated attack, but most commonly the easiest route is chosen and installers are more commonly pushing potentially unwanted legitimate programs that might be willingly installed by some users, and are only problematic when installed without proper disclosure or if installer designed to encourage miss inputs and accidental agreements. This also allows for software distributors that are utilizing installers to remain in the legal zone, even if it isn't a morally right thing to do.

If software emulation was officially supported by Windows it would be the best approach, since they have access to all proprietary functions and can implement it on a lower level that would have the potential to be impossible to defeat by a regular program without access to low-level functionality.

Final conclusion

In this work, it was shown that some frameworks and companies exist with the purpose to push unwanted programs on the user. And even taking into an account that more and more work can be done in the browser companies have equally increased incentive to develop a proprietary application to collect as much data as possible and by leaving some files on the user's PC they can profile them better and gather the information that user wouldn't be willing to share otherwise. Another thing to consider is that with every year amount of abandoned software increases and there is no other choice for a user other than to use distributors that are doing their best to be as profitable as possible even if that means tricking users into installing programs that they don't want if distributors operate on "pay per install" model.

Sandbox application was successfully implemented and tested on the real-world program. This work demonstrates a way how to reduce the risk associated with installing programs from sources that can't be fully trusted even if they are primary distributors of the software. By allowing users to install a program in the sandboxed mode it becomes possible for them to examine the created files. In addition all the files will be located in a single folder that is easy to scan and registries will be accessible from an SQLite database file, so if the software in question doesn't meet the user's expectations then the remove process is as simple as deleting a file and a folder, and there is no need to rely upon provided uninstaller. This can be especially useful if one has to rely on software distribution websites to access some f.e. abandonware or freeware. In cases of abandoned software and free one, there is a strong incentive from the distributor to push some potentially unwanted programs to recuperate the costs of running the website and to make a profit. An additional benefit of using the sandbox for installation is that in case of some unexpected exception or improperly handled error due to system incompatibility the installer won't just quit and leave unaccounted files and registries that can be a cause of errors later.

The main difficulties in implementation are related to the fact that the

underlying code is proprietary, and function description doesn't always correspond to the actual behavior. Also since the path of emulating NTDLL functions was chosen this means that the solution has to account for the ways that the NTDLL library is used within other libraries. As a direct consequence of frequent involvement by various subroutines debugging process becomes a lot harder and zeroing in on a problem might require a significant effort.

There are also significant benefits to choosing lower-level functions. NTDLL is always loaded and is always available in the process memory space, so a lot of libraries rely on it to function and this creates an additional challenge for anyone who attempts to create an installer that will be resilient to the proposed solution. If functions were intercepted on a user level it would be relatively easy to develop a library that implements all the required functionality from AdvApi.dll and Kernel32.dll and generate a random name for those functions, so intercepting them would require an advanced code recognition techniques. Or just adapt the code to use the NTDLL library directly, without invoking any user-level code.

Benefits compared to using existing isolating solutions:

- Lightweight – system requirements are low.
- Shared environment – no need to purchase additional licenses to run separate versions of Windows or auxiliary programs that might be required for an installer in question to run.
- Software-based sandbox – doesn't require support from hardware to work.
- No additional setup – running the installer inside the sandbox is no harder than executing the installer itself.
- File conglomeration – after installation all files are located in a single directory making the scanning process easier. Even if users install and test software in the VM they would still have to find and scan all created or changed files and this process can be time-consuming.
- Registry key storage – since information about registry keys is located inside the SQLite database it is possible to inspect this file and use SQL language to execute a more advanced search to help in the detection of suspicious behavior.

Drawbacks compared to other solutions:

- Less isolation – since the installer process can still directly interact with a real file system it carries the risk of a program escaping the sandbox higher.

-
- Tied to the underlying OS – unlike VM or Container, the proposed solution only runs in the current operating system, so different versions of the OS can't be used.
 - Privilege level – other solutions allow the user to have full admin access inside the emulated environment. The proposed solution can't give a user more access than they had originally.
 - Higher risk – since the installation process is run on the user's system there is more potentially valuable data that can be damaged by a malicious agent. If a malicious program infects a VM, Container, or Windows Sandbox, the image can be disposed of and no valuable data will be affected.

So software sandboxing is and will remain a good additional level of protection. While not as sophisticated as VM or Containers it is easier to run and doesn't require hardware support and is a good enough solution to prevent opportunistic agents from dictating their terms.

Bibliography

1. CHEATENGINE. *Downloads*. 2021. Available also from: <https://www.cheatengine.org/downloads.php>.
2. DYNAMICDOWNLOADER. *Dynamic downloader*. 2018. Available also from: <http://www.dynamic-downloader.com/#about>.
3. WEAVEWORKS. *A practical guide to choosing between Docker Containers and VMS*. 2020. Available also from: <https://www.weave.works/blog/a-practical-guide-to-choosing-between-docker-containers-and-vms>.
4. ORACLE. In: 2013. Available also from: https://docs.oracle.com/cd/E36500_01/E36503/html/desktop-images.html.
5. MICROSOFT. *Windows 11 Specs and System Requirements: Microsoft*. 2022. Available also from: <https://www.microsoft.com/en-us/windows/windows-11-specifications?r=1>.
6. SIMPSON, Daniel. *Windows Sandbox Architecture - Windows Security*. 2020. Available also from: <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-sandbox/windows-sandbox-architecture>.
7. SIMPSON, Daniel. *Windows Sandbox - Windows Security*. 2022. Available also from: <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-sandbox/windows-sandbox-overview>.
8. WEBMASTER. *Sandbox web browser: Security for windows PC*. 2022. Available also from: <https://www.shadesandbox.com/what-is-a-sandbox>.
9. HAN, Deland. *Windows Registry for Advanced Users - Windows Server*. 2022. Available also from: <https://docs.microsoft.com/en-us/troubleshoot/windows-server/performance/windows-registry-advanced-users>.

10. MATEUSZ, Jurczyk. *Windows WIN32K.SYS System Call Table*. 2020. Available also from: <https://j00ru.vexillum.org/syscalls/win32k/32/>.
11. BRIDGE, Karl. *CREATEPROCESSW function (processthreadsapi.h) - win32 apps*. 2022. Available also from: <https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessw>.
12. UNDOCUMENTED. *NtCreateProcess NtCreateProcess*. [N.d.]. Available also from: <http://undocumented.ntinternals.net/index.html?page=UserMode%5C%2FUndocumented+Functions%5C%2FNT+Objects%5C%2FProcess%5C%2FNTCreateProcess.html>.
13. MICROSOFT. *Microsoft Portable Executable and Common Object File Format Specification. Revision 6.0*. 1999.
14. GIBERT, Daniel; MATEU, Carles; PLANES, Jordi. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications*. 2020. Available from DOI: [10.1016/j.jnca.2019.102526](https://doi.org/10.1016/j.jnca.2019.102526).
15. PIETREK, Matt. *Inside windows: Win32 portable executable file format in detail*. 2019. Available also from: <https://docs.microsoft.com/en-us/archive/msdn-magazine/2002/february/inside-windows-win32-portable-executable-file-format-in-detail>.
16. BRIDGE, Karl. *PE format - win32 apps*. 2021. Available also from: <https://docs.microsoft.com/en-us/windows/win32/debug/pe-format>.
17. KOKEŠ, Josef; ZAHRADNICKÝ, Tomáš. *Code generation*. 2022.
18. RUSSINOVICH, Mark; SOLOMON, David A.; IONESCU, Alex. Image Loader. In: *Windows internals: Part 1*. Microsoft Press, 2012, pp. 232–244.
19. WHIMS, Steve. *DllMain entry point (process.h) - win32 apps*. 2021. Available also from: <https://docs.microsoft.com/en-us/windows/win32/dlls/dllmain>.
20. JRSOFTWARE. *Pascal Scripting: Support Functions Reference*. 1999. Available also from: <https://jrsoftware.org/ishelp/index.php?topic=scriptintro>.
21. BRIDGE, Karl. *System services - win32 apps*. 2021. Available also from: https://docs.microsoft.com/en-us/windows/win32/api/_base/.
22. LUEVELSMEYER, Bernd. *The PE file format*. 1999. Available also from: <http://www.pelib.com/resources/luevel.txt>.
23. STEVEWHIMS. *Remote procedure call (RPC) - win32 apps*. 2022. Available also from: <https://docs.microsoft.com/en-us/windows/win32/rpc/rpc-start-page>.

BIBLIOGRAPHY

24. GOLDEN, Barry. *Zwcreatekey function (WDM.H) - windows drivers*. 2022. Available also from: <https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/nf-wdm-zwcreatekey>.
25. ASHCRAFT, Alvin. *Maximum path length limitation - win32 apps*. 2021. Available also from: <https://docs.microsoft.com/en-us/windows/win32/fileio/maximum-file-path-limitation?tabs=cmd>.

Acronyms

API Application Programming Interface

CPU Central Processing Unit

DLL Dynamically linked library

IAT Import Address Table

MB Mega Bytes

MSDN Microsoft Developer Network

OS Operating system

PC Personal Computer

PEB Process environment block

PE Portable executable

PID Process ID

POC Proof of concept

PUP Potentially unwanted program

RAM Random access memory

TEB Thread environment block

UI User interface

URL Uniform Resource Locator

VM Virtual machine

Contents of enclosed CD

	readme.txt	the file with CD contents description
	executables	the directory with executables
	src	the directory of source codes
	code	implementation sources
	thesis	the directory of L ^A T _E X source codes of the thesis
	text	the thesis text directory
	thesis.pdf	the thesis text in PDF format