UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA

FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ

SPECIALIZAREA ÎN INFORMATICĂ

LUCRARE DE LICENŢĂ

Titlu

Conducător ştiinţific
Dr. CZIBULA Istvan Gergely, Profesor Universitar

Absolvent
BUZAS Laura Andrada

2019

BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION IN COMPUTER SCIENCE

DIPLOMA THESIS

Title

Supervisor
PhD. CZIBULA Istvan Gergely, Professor

Author
BUZAS Laura Andrada

2019

Contents

1. Introduction

Privacy is one of the things humanity has been keen on in the last few decades. To achieve safety software-wise, as a first thing, you would like to protect personal data.

What is important to keep in mind regarding data security is a matter of three concepts, namely availability, confidentiality and integrity, each one them having a high importance. Data availability means that it is available at any time, in case the user tries to access it.

Confidentiality is all about data that cannot be accessed without adequate authorization. A solution to take into account when it comes to confidentiality would be encryption. This is a proper solution, regarding the fact that a user's data will be available to read only for him. The process allows the user to "hide" their information, by using an algorithm in order to turn the real data into data that would seem random to other users, making it unable to be read.

Integrity means that user's data cannot be modified without authorization. In case integrity is violated, the data would be in an inconsistent state, not what the user is expecting. This would also render the data unavailable because the user does not have access to the old data anymore.

To achieve data safety you can consider having your data encrypted, but even with it, the integrity problem would still be open.

The main objective of this paper is to describe the process of design and implementation of a solution that tackles the integrity problem mentioned above. The designed solution is composed of: a minifilter driver that contains the monitoring and protection logic, a DLL that is an abstract interface over the minifilter functionality, and a Windows Desktop application built on top of WPF (Windows Presentation Foundation).

The second chapter of this paper contains four sections. First section is an high level overview of a few existing products, containing their description and functionality. Second section is divided in three main subjects: Windows Filter Manager where I'll be discussing the minifilter framework and how it can be used to achieve our proposed objective, C and .NET where I will motivate why I've chosen to use these languages. Third section is a detailed view of solution's design and implementation, especially the minifilter logic and communication. Last section focuses on how the application was tested.

Third chapter describes the further work that can be done to add more functionality to the current application.

Last chapter is a conclusion of the paper, summing up the main ideas.

1.1 Personal Contribution

Designing an SDK (Software Development Kit) that can be integrated by third-parties. My work consists of providing a documented interface for integrators in order to easily manage file protection rules that are based on Windows Kernel-level mechanisms.

The purpose of this is to provide an abstraction. //TODO

2. File Protection

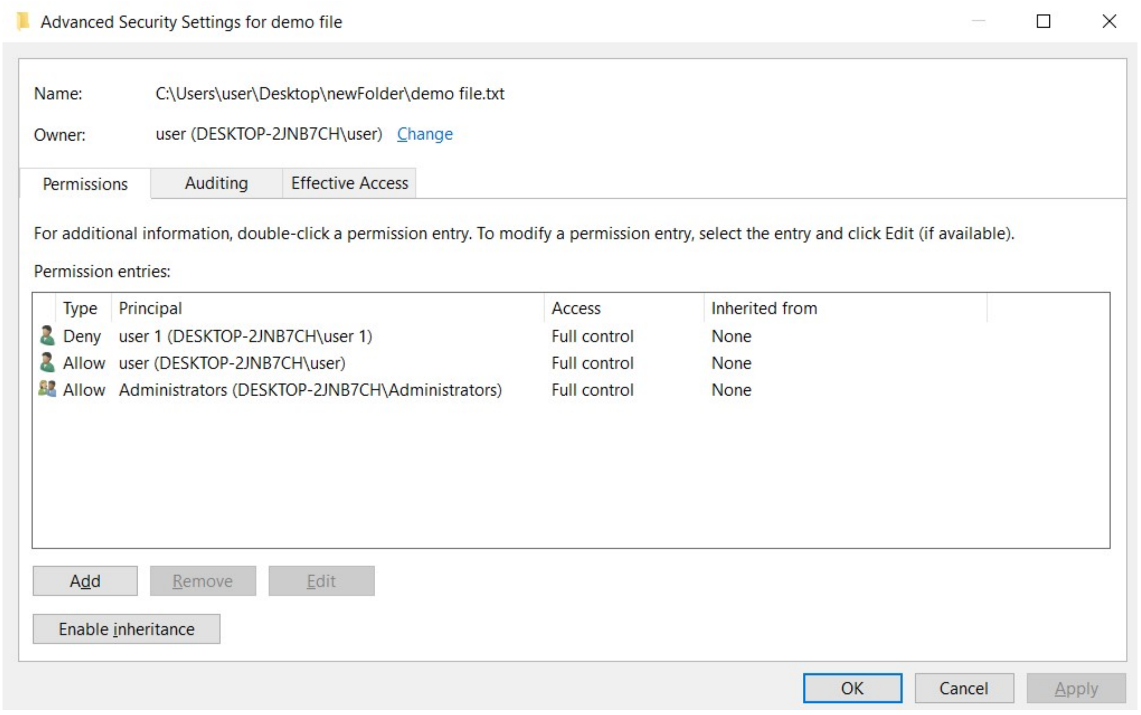
2.1 Built in protection

It's important to know that usually, operating systems do not have a built in mechanism that include a password protection method. However, there is a file permission policy that counts as a level of protection from other users.

2.1.1 Windows

File protection in Windows can be achieved through DACLs (discretionary access control lists). As a short explanation, a Windows object can have a DACL which contains ACEs(access control entries) that define what type of access any user or group of users can have. [2]

Usually, DACLs are used by defining the users and groups that can have access because the ones that are not included will have restricted access by default. If a DACL does not contain any ACEs the system will deny any type of access.



Name: C:\Users\user\Desktop\newFolder\demo file.txt

Owner: user (DESKTOP-2JNB7CH\user) Change

For additional information, double-click a permission entry. To modify a permission entry, select the entry and click Edit (if available).

Permission entries:

Type	Principal	Access	Inherited from
Deny	user 1 (DESKTOP-2JNB7CH\user 1)	Full control	None
Allow	user (DESKTOP-2JNB7CH\user)	Full control	None
Allow	Administrators (DESKTOP-2JNB7CH\Administrators)	Full control	None

Figure 2.1: Advanced Security Settings dialog

Let us consider a new text file named "demo file.txt" on a Windows 10 OS that has 2 users, namely user" and user 1, which both have administrator permissions. To restrict access for user1 I have to go to file's properties **Security->Advanced**.

In figure 2.1 we can see the "Permission Entries" which maps to DACL we mentioned earlier whilst list's entries map to ACEs. I added the Administrators and user to have full control, and also I set user 1's permissions to deny any type of action. In this example we can notice that order in which permissions are set matters because user 1 cannot access the file, even if that user takes part from Administrator's group.

At the moment only the current user and other possible users in the Administrator group could have access to the file. Still, we are not fully protected when it comes for example to exploits. We could consider that we are logged in with user 1, which currently does not have any kind of permission for the demo file. There is the possibility of an exploit having first user's permissions and overwriting, encrypting or deleting the file which would be disastrous.

2.1.2 Linux

Almost the same mechanism exists in Linux, but it's more trivial. In my examples I will talk about Ubuntu, but the mechanism is almost the same for most of the Linux distributions.

2.2 Existing products for Windows

As a similar tool, a first one that draws attention, is iBoysoft's File Protector for Windows, coming as a 7-day free trial or a payed version. As a main functionality, this tool allows the user to select the desired folders or files to be protected, along with the protection type (i.e. read/write/delete).

The application starts with a message box, demanding a password. After it was inserted correctly, the application starts, displaying on the main screen a list of files and folders the user has chosen to be protected.

In figure 2.2 we can see the main screen of the application, where we have a list of protected files, buttons for adding files or folders, and also the menu on the right.

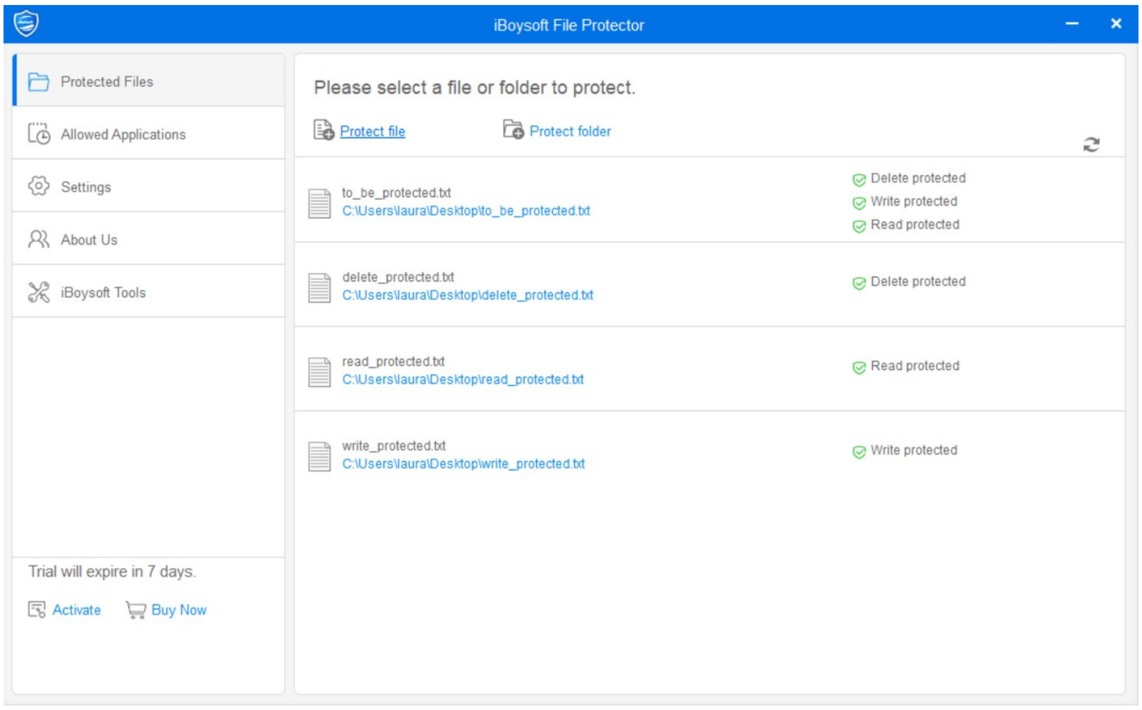
Another feature of this tool is the allowed applications. There, the user can choose which applications that can have rights, in spite of the protection selected for the file or folder. After adding an application, the user should also add the files or folders that application can have access to.

If for example a text file was added to the list with read protection, that file cannot be opened by the user, which is a drawback for this application. There is one thing that can be done, and that is giving permissions to other applications to perform actions upon the desired files.

There is also the settings tab which contains two important sections, one where you can change the password, another one where you can choose the default protection types that are selected when adding a file or folder.

As a conclusion, the only time a password is required, is when the application is started.

*find another soft like this



Please select a file or folder to protect.

to_be_protected.txt C:\Users\laura\Desktop\to_be_protected.txt	Delete protected Write protected Read protected
delete_protected.txt C:\Users\laura\Desktop\delete_protected.txt	Delete protected
read_protected.txt C:\Users\laura\Desktop\read_protected.txt	Read protected
write_protected.txt C:\Users\laura\Desktop\write_protected.txt	Write protected

Figure 2.2: iBoysoft File Protector for Windows

2.3 Used Technologies

There are three main technologies that I want to talk about in the next subsections, namely Windows Filter Manager, which is important as a concept to be understood, C and .NET where I will explain my choice. //TODO

2.3.1 Windows Filter Manager

concepts : Altitude

2.4 Design and Implementation

2.4.1 Minifilter High Level Overview of Design

//TODO * describe the structure of the minifilter driver

2.4.2 IRP Filtering

IRPs (I/O request packets) are Windows kernel mode structures that are used to incorporate .

* what type of IRPs exist
* which IRPs I filter
* how i filter the IRPs

2.4.3 Monitoring and Denying Access

//TODO
*diagram + code of how and where exactly it happens

2.4.4 Encountered Challenges

//TODO
* process of installation of a minifilter driver on a test machine

2.4.5 Inter Component Communication

//TODO
* make some diagrams to make it more clear

2.5 Testing Approach

In order to achieve a reliable software, the application has been tested using both black box and white box testing methods, which will be described in the following sections.

2.5.1 Driver Verifier

First of all I will describe some basic functionalities of Driver Verifier, that I have also used to test the minifilter.

As the name suggests, Driver Verifier is a tool that can monitor Windows kernel drivers. It can detect major problems, such as memory leaks, memory corruptions or deadlocks. This tool is developed by Microsoft, making it's first appearance in Windows XP as a command line utility. Over the time this tool has gained more checks as well as a graphical user interface namely Driver Verifier Manager, that allow a programmer to select the preferences for driver's testing.

2.5.2 Static Code Analysis

In this section I will talk about source code annotation language (SAL). This is used to make the code more explicit, in terms of behavior, parameters, return values, or other things* to make it more understandable. In other words, programmer's work is to use annotations considering the desired functionalities, and compiler's work is to verify if the written code corresponds with the annotations.

Another benefit of SAL is that specifications in the form of code comments and documentation are validated by the annotations. This is an advantage because in some cases the documentation or comments can be vague, making it hard for a programmer to integrate the code correctly. Moreover, if SAL is used, the compiler can validate that the code written meets the documentation and specifications.

All SAL annotations have a specific "look", and that is `_Annotation_name_`. For example, some parameter most used SAL annotations are `_In_` and `_Out_`. `_In_` makes sure the compiler interprets data as being input data that cannot be modified in function's scope. `_Out_` allows the programmer to use the parameter as an output one, as long as the space for that parameter has been allocated by the programmer.

As for behavior, a simple example is `_Check_return_`. If this annotation is used, the caller must inspect the returned value. In case the function has a void return type, an error will be shown at compilation time. A similar example is `_Must_inspect_result_`, which contains the functionality of `_Check_return_` and adds that any output parameter should also be used after the function call.

After these examples anyone should be convinced that SAL is beneficial from multiple reasons, and that is also why I used it in my own code.

//TODO My code examples

3. Further Work

//TODO
* make file blocking more efficient
* make this project at a bigger scale

4. Conclusion

Bibliography

[1] Bottazzi G., Italiano G.F., Spera D. “Preventing Ransomware Attacks Through File System Filter Drivers”. In: Milan, Italy, Feb. 2018. URL: `http://ceur-ws.org/Vol-2058/paper-08.pdf`.

[2] Microsoft. *DACLs and ACEs*. `https://docs.microsoft.com/en-us/windows/desktop/secauthz/dacls-and-aces`. 2018.

[3] Rienhardt F. “Kernel-based monitoring on Windows (32/64 bit)”. In: Dec. 2015. URL: `https://bitnuts.de/KernelBasedMonitoring.pdf`.

[4] Yosifovich P., Ionescu A., Russinovich M.E., Russinovich D.A. *Windows Internals, Part 1: System architecture, processes, threads, memory management, and more*. Seventh Edition. Redmond, Washington: Microsoft Press, 2017.