DEPARTMENT OF COMPUTER SCIENCE

KAISERSLAUTERN UNIVERSITY OF TECHNOLOGY

EXPLORATION AND COMPARISON OF WIN32 API TO IMPROVE DATA USAGE CONTROL

Master Thesis

By

Dilan Shaminda Pela Widanele Gedara

2018

Supervisors:

Prof. Dr.-Ing. habil. Peter Liggesmeyer

Kaiserslautern University of Technology

Patricia Kelbert

Fraunhofer IESE

Robin Brandstädter

Fraunhofer IESE

Abstract

Computers have become an inherent part of our daily lives today. The Windows operating system achieved high popularity over the past years. Microsoft Office applications including Word, Excel, and PowerPoint will smooth the day-to-day works which have become the most acquainted Office application suite in organizations. Windows API enables developers to investigate the functionalities and capabilities unique to each version of the Windows operating system.

Accidental data leakage is a major data breach, and this has become a critical issue during the last few years. Even though the Windows operating system provides different access levels, and organizations train employees to prevent data leakages, still there is a need for application-level additional security layer to prevent accidental data leakages happening by employees.

The thesis investigates the use of Win32 API functions to address the specific use-cases. The use-cases defined in the thesis are related to the accidental data leakages. Furthermore, this thesis describes the steps taken for the implementation of the prototype solutions, and finally, the solutions are evaluated regarding security and performance with suggestions for future improvements.

Keywords: Win32 API**,** Accidental data leaks, Windows

Erklärung

Hiermit erkläre ich, Dilan Shaminda Pela Widanele Gedara, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen wurden, habe ich durch die Angabe der Quelle, auch der benutzten Sekundärliteratur, als Entlehnung kenntlich gemacht.

Kaiserslautern, den 22. September 2018

Dilan Shaminda Pela Widanele Gedara

Acknowledgements

First and foremost, I would like to offer my sincerest gratitude to my supervisors, Patricia Kelbert and Robin Brandstädter, who have supported me whenever I ran into a trouble spot or had a question about my research or writing. Also, I would like to thank for their excellent feedback and advice throughout the last six months.

I would also like to thank the Security department at Fraunhofer IESE for providing me a workspace and appropriate hardware. Also, for the staff in the Security department especially, Raj Shah and Arghavan Hosseinzadeh da Silva for the help and advice they gave me throughout the thesis.

Next, I would like to thank Prof. Dr. Peter Liggesmeyer for allowing me to write this thesis at Fraunhofer IESE.

Finally, I would like to thank my family, and friends, who always encourage me in everything I do, and who have supported me incessantly throughout my course of studies.

Table of Contents

List of Figures

List of Tables

1 INTRODUCTION

With the progression of modern computers, enterprise sectors have started using computers to store, manipulate and share sensitive data. Sensitive or confidential data are referred to as data that must be protected from unauthorized parties for legal or ethical reasons. Organizations store confidential data in various domains such as protected healthcare information, confidential personal information, customer record information, and student education records etc. [1]. Companies are failing to protect confidential personal and co-operate data, and this situation is growing in a fast phase [2]. Therefore, Organizations should control confidential data and must prevent them being leaked to the external parties.

The focus of this thesis is to prevent accidental data breaches Occurred by the employees. Accidental data exposure is harmful, and the consequences experienced by the organizations are severe and progressively increasing. Thus, the thesis explores Win32 API[1] to prevent accidental data leaks in MS[2] Office applications by addressing the following use-cases: Detect pasted confidential data in MS Word, Excel, and PowerPoint applications, clear Office Clipboard history, External device monitoring application, confidential file printing detector application and handling the confidential file in DDE message passing.

1.1 MOTIVATION

According to the latest statistics on a global market share of operating systems on desktop computers worldwide, the Microsoft Windows operating system holds 82.88 percent and stays on top [3]. Also, Microsoft Office 365 has become the most used office application in the enterprise sector. Microsoft Office 365 includes Microsoft Word, Excel and PowerPoint applications which are the focus of the thesis.

Windows API enables developers to explore the power of the Windows operating system. This provides access to the low-level Windows features securely. Another advantage of Windows API is, that it gives the accessibility to most of the operating system related tasks directly. Example: creating windows, display graphics, opening files

[1] API stands for Application Programing Interface
[2] MS is the acronym for Microsoft

etc. Accidental data leaks can happen at any point in time by the employees. Moreover, it might cost the organization big time depending on the value of the data being leaked to the external parties. Restricting the access to the data or defining organizational policies or rules will not help to overcome the problem entirely. Restricting the access to the data degrades the ability of innovation and the creativity of the employees.

Hence, there is a need for a mechanism which protects confidential data from being leaked to the outside of the organization while providing full access to the resources.

1.2 PROBLEM ANALYSIS

Organizational policies and rules elucidate the agreed behaviours of employees within the organization. These policies govern the internal organization and their duties, responsibilities and most importantly the limitations of each role. From the organizational perspective, it is crucial to maintain the confidential information within the organization. It improves the quality and the trustworthiness between the customers and the organization by protecting confidential information not being accessed by external users. Data leaks can happen in different ways. It might happen on purpose, accidentally, reluctantly or by carelessly. Malicious mischief cannot be prevented nor stopped. Modern technology amplifies the impact of accidental data leaks. For example: Imagine a situation where an employee sends the wrong data per email to a mailing list causing immediate colossal damage.

As stated in the article written by B. Rossi [4] "20% of businesses reported losing data from a software vulnerability incident, while 22% reported losing data from an accidental leak by staff". According to the global survey of IT professionals, 29% of all businesses reported having lost confidential data due to accidental leak by employees.

Now, this became worst, and currently, this is the most significant source of losing data. SolarWinds 2015 Federal Cybersecurity Survey[3] listed most common accidental insider threats [5]:

1. Phishing Attacks (42% of respondents identified as a most common cause)
2. Data copied to insecure devices (44%)

[3] The full report available at https://www.solarwinds.com/resources/survey/solarwinds-federal-cybersecurity-survey-summary-report-2015

3. Accidentally deleting, corrupting or modifying data (41%)

4. Using personal devices that are against company IT policy (37%)

5. Poor password management (37%)

6. Device loss (36%)

7. Incorrect use of approved personal devices (33%)

8. Not applying security updates (31%)

9. Incorrect disposal of hardware (28%)

10. The insecure configuration of IT assets (24%)

The survey states, 44% of accidental *insider threats* are caused by data copied to insecure devices. For this thesis, we chose to focus on accidental copy/paste events of sensitive data to external sources. Among other office package applications, the MS Office package enables users to make documents, worksheets, databases, presentations etc. Some fictive situations of how data leak can happen while using that software are illustrated below.

Imagine a database with thousands of customers' information stored in spreadsheet files. The information may contain their names, social security numbers, financial details such as net worth and investment portfolios etc. An employee could copy data from one of the spreadsheets which contain those confidential data and paste it into another document which is not marked as a confidential file. Then he/she would further edit the document. A few days later, he/she can share it with external people, has probably forgotten that he/she added confidential information in the file. Also, the information copied from the confidential file could be pasted to an email mistakenly and sent to a third-party user. Eventually, an employee can copy that confidential file to external storage or external network.

1.3 POSSIBLE SOLUTIONS

As discussed in the problem analysis chapter, accidental data leaks became a critical issue in the organizations. Locking-down the secret information is not an answer to the problem. Because information is needed to make better decisions, come up with innovations, to develop more robust strategies. Good information sources enable

employees to work with increased efficiency. Therefore, organizations should ensure the access to the information while protecting the sensitive information. There is a tradeoff between enabling information and securing them at the same time. One possible solution is to train employees how to handle sensitive data. Awareness of possible security features which are already offered by the software applications to prevent those incidents being happened is also essential. However, this will not overcome the situation entirely. Because we are prone to make mistakes. Though we can define access levels (administrator, standard user, and guest) using the default features in windows operating system is sometimes not enough to stop these activities from happening. An application-level security layer is needed to enforce policy on the windows system.

1.4 GOALS AND CONTRIBUTION

This thesis investigates how to improve data usage control using Win32 API in the Windows operating system. Mainly, the thesis addresses how to control accidental data leaks happening in the Microsoft Word, Excel, and PowerPoint applications. For that, the thesis describes the Win32 API functions to demonstrate the power of Win32 API and furthermore how to use them to provide solutions for the use-cases defined previously. By exploring the win32 API, we will get information about the capabilities of the API and the possible alternatives. Another benefit is, that the developers and researchers get an idea beforehand, and it will be beneficial for them. The prototype applications developed can be extended in the future with more functionalities.

1.5 STRUCTURE OF THE THESIS

This thesis comprises of six chapters. Introduction chapter has given a short overview of the topic of the thesis and discussed the current problem, possible solutions, and thesis goals and contributions.

The second chapter describes the related works as well as the Windows API functions required to address the defined use-cases in the thesis. Furthermore, the thesis introduces Wrapper APIs, followed by the existing APIs like Managed Windows API and EasyHook.

The third chapter identifies the requirements for the use-cases defined for the thesis with all the implementation aspects and describes the steps and essential functions used in the implementation.

The fourth chapter explains the testing methodology used for evaluating the quality of the prototype applications.

Chapter fifth evaluates the concepts and solutions with regard to the requirements and security. Additionally, it also provides an initial performance evaluation with suggestions for future improvements.

Finally, chapter six summarizes and assesses the results achieved in the thesis.

2 RELATED WORK

The following chapter gives an overview of the Data Loss Prevention (DLP) software product “Office 365 security and compliance center” [6]. Also, Windows API that is related and inline to the subject of this thesis. As this thesis covers data usage control and Windows operating system applications’ security, this chapter will be structured to introduce related work in these respective areas.

2.1 Data Loss Prevention Software

Data loss prevention or DLP is a set of processes and techniques which ensure that confidential data were not leaving an organization. The software products, which use DLP strategies, classify and protect confidential data by identifying the transgressions of policies defined by the organization.

2.1.1 Office 365 security and compliance center

Microsoft Office 365 for business uses data loss prevention (DLP) policy in the Office 365 security and compliance center [6]. Office DLP policies define how sensitive information can be shared with other parties and how users can identify, monitor and protect confidential information. Office DLP policy provides the following features to protect sensitive information.

- Can identify confidential information in storages like Exchange online (a hosted email form business), SharePoint Online, and One Drive for Business. In these defined locations a user can identify and monitor sensitive information such as a credit card number is being mentioned in any document.
- DLP supports prevention of sensitive information being shared accidentally. For example, by Office 365 DLP policy one can define a policy which ensures the information compliance to the Health Insurance Portability and Accountability Act (HIPAA). It identifies any health-related records that shared with outside the organization and blocks the access to the document.
- Monitors and protects the defined confidential information in Word 2016, Excel 2016 and PowerPoint 2016 desktop versions. If DLP policies are enabled, it monitors the information shared between the Office 365 applications, SharePoint Online, and OneDrive for Business. This feature only available in MS Office business premium package.

These DLP policies ensure the prevention of accidentally published sensitive information to sites like social. They also manage data uniformly throughout the organization. But this solution does not meet the requirements of the defined use-cases for the thesis. Because Office DLP monitors specific sensitive data such as financial data, credit card numbers, social security numbers and health records defined by the policies. However, use-case 1 requires monitoring and detecting any confidential content in MS Word, Excel and PowerPoint applications.

2.2 Windows API

API stands for Application Programming Interface. An API provides the functionality to interact with the features of an application, service or a software component. The Windows API provides the ability to explore the power of the Windows operating system by accessing the lower level features and the capabilities of the Windows operating system. Different Windows versions have a unique set of features and capabilities. By using the native Windows API, a developer can access features which are not always provided by the wrapper libraries and wrapper APIs (See section 2.3). That is one advantage of the API. Microsoft provides, for example, the Clipboard class [7], which is particularly interesting for our copy/paste use case. This class provides methods to retrieve and set data on the system Clipboard in Windows forms (language: C#) application environment.

Below is the list of functions available C# Clipboard class and its description [7].

Table 2.2.1: Available functions in C# Clipboard class

Available functions in the Clipboard class	Description
Clear()	Empty the Clipboard
ContainsAudio()	Returns true if the Clipboard contains an audio file in WaveAudio format
ContainsData(String)	Returns true if the Clipboard contains data with the specified Clipboard data format
ContainsFileDropList()	Checks whether the Clipboard holds data in the format of File Drop or it can be converted into a FileDrop format
ContainsImage()	Checks whether the Clipboard holds data in the

	format of Bitmap or it can be converted into that format
ContainsText()	Returns true if the Clipboard contains data in Text or Unicode text format. This depends on the operating system
ContainsText(TextDataFormat)	Indicates whether there is a text data object in the specified textDataFormat value
GetAudioStream()	Gets an audio stream from the Clipboard data
GetData(String)	Get Clipboard data in the specified format

**Full list can be found in the appendix section

When handling different formats of data, it is convenient for developers to use the functions from the Clipboard class. For example, for audio files, it will automatically convert it to a stream before setting the byte array on to the Clipboard. If a developer uses native Windows API functions to do that it would be much more work compared to the usage of SetAudio(Byte []) function. Table 2.2.2 shows the available functions in the Windows API Clipboard class [8].

Table 2.2.2: Available functions in the Windows API Clipboard

Windows API Clipboard functions	**Description**
AddClipboardFormatListener(HWND)	Register the current window in the Clipboard format listener list
ChangeClipboardChain(HWND, HWND)	Removing the window from the Clipboard chain list
CloseClipboard	Closes the Clipboard
CountClipboardFormats	Gets the number of Clipboard formats in the current object of the Clipboard
EmptyClipboard	Remove the data from the Clipboard and assign the ownership to the current window which currently opened the Clipboard
EnumClipboardFormats(UNIT)	Enumerate through the Clipboard formats in the current object of the Clipboard. The function returns the next available Clipboard format
GetClipboardData(UNIT)	Gets the Clipboard data in the specified Clipboard format

GetClipboardFormatName(UNIT, LPTSTR, int)	Gets the name of the Clipboard format in the specified Clipboard type
GetClipboardOwner	Gets the window handler of the current owner of the Clipboard
GetClipboardSequenceNumber	Retrieves the sequence number. This number changes whenever the Clipboard content is changed or emptied
GetClipboardViewer	Returns the handler of the first window in the Clipboard chain
GetOpenClipboardWindow	Returns the handler of the window that has open the Clipboard
GetPriorityClipboardFormat	Gets the first available Clipboard format. Clipboard formats are stored in an ordered list
GetUpdatedClipboardFormats(PUINT, UNIT, PUINT)	Gets the currently supported Clipboard formats
IsClipboardFormatAvailable(UNIT)	Returns true if the Clipboard contains data in the specified Clipboard format
OpenClipboard	Opens the Clipboard
RegisterClipboardFormat(LPCTSTR)	Register a new Clipboard format by the given name
RemoveClipboardFormatListener(HWND)	Remove the window from the Clipboard format listener list
SetClipboardData(UNIT, HANDLE)	Sets data to the Clipboard in the specified Clipboard format
SetClipboardViewer(HWND)	Adds the specified window to the system Clipboard change.

Below steps should be applied to achieve the same functionality given by the SetAudio(Byte[]) function.

1. Register the current window in the Clipboard chain
2. Get the handler of the next window in the Clipboard chain
3. Override the *WndPrc*[4] function and listen to the WM_CHANGECBCHAIN and WM_DRAWCLIPBOARD messages

[4] WndProc method processes Windows messages

4. Passes the message to the next window in the Clipboard
5. Empty the Clipboard
6. Retrieves the *WaveAudio* stream
7. Sets the stream to the Clipboard
8. Remove the window from the Clipboard chain list (optional)

The table 2.2.2 shows the list of functions available in the Windows Clipboard API. There is no function available in the C# Clipboard class which gives the information about the ownership of the Clipboard. This fails to fulfil the requirement $\mathbb{R}_2$ of the use-case 1 (see table 3.3.1). Nevertheless, as shown in table 2.2.2 the Windows API has a function called *GetClipboard Owner*, which delivers precisely what we need.

All the Windows API functions can be accessed by using C or C++ language and must import the specific DLL (Dynamic-link library) when using them in a C# environment.

2.2.1 Data Access and Storage API

The Data Access and Storage API is a set of Windows APIs that enables controlling how users can access or retrieve data in the Windows operating system environment. It also provides functionalities for exchanging data between applications. Sections 2.2.1.1, 2.2.1.2 and 2.2.1.3 explain the APIs available for access and store data in the Windows environment.

2.2.1.1 System Clipboard

Clipboard is a user-driven set of functions and messages to transfer data between different kinds of applications or within the same application. It has four different commands: Cut, Copy, Paste and Delete [9]. A window must not transfer any data to or from the Clipboard without any acknowledgement of the user. Clipboard data format can be any data format, and each of the registered Clipboard formats is recognized by an unsigned integer value. The *RegisterClipboard format* [8] function can be used to register a new format, and it will return a value in the type of UNIT which would be in the range of 0xC0000 to 0xFFFF [8]. Clipboard formats can be categorized like below:

- **Standard Clipboard Formats**

  Clipboard formats defined by the system. Table 2.2.3 shows the standard Clipboard formats in Windows system [8].

Table 2.2.3: The standard Clipboard formats

Format value	Description
CF_BITMAP	Bitmap format
CF_DIB	A memory object containing a BITMAPINFO structure
CF_DIBV5	A memory object containing a BITMAPV5HEADER structure
CF_DIF	Data Interchange Format
CF_DSPBITMAP	Bitmap display format associated with a private format

**Full list can be found in the appendix

- **Registered Clipboard Formats**

  Different applications support different types of data objects. These formats are not supported by the standard Clipboard formats, and if an application uses a standard format to hold this format (ex: rich text format), formatting information would be lost. Different applications can use the same registered Clipboard format if they use the same name (case-insensitive [8]).

- **Private Clipboard Formats**

  Application-specific data formats can be used without registering with the system. Clipboard uses a value between CF_PRIVATEFIRST to CF_PRIVATELAST to identify private Clipboard formats. Since the system does not automatically remove the data handles, private Clipboard formats need to release the resources using WM_DESTROYCLIPBOARD message when they are no longer needed.

- **Multiple Clipboard Formats**

  An application or a window can contain several Clipboard formats at the same, as long as each piece is in a different format. *CountClipboardFormats* [8] function can be used to find out how many Clipboard formats are currently using on the Clipboard. Clipboard arranges the copied information in the following manner: The format which has the most information will be at the top, followed by less descriptive formats. An example situation for this scenario could be: copying some text from notepad++ (supports RTF[5]) with different font colours and

[5] RTF stands for Rich Text Format and it is a method of encoding formatted text

pasting it to notepad (supports text). In this case, notepad will retrieve Clipboard object in the most descriptive format it recognizes. Otherwise, some formatting information is lost.

- **Synthesized Clipboard Formats**

  This is when the system automatically converts between data formats when the requested type of data is not on the Clipboard. The available data formats to the requested formats conversion are shown in table 2.2.4 [10].

Table 2.2.4: Clipboard formats conversion

Clipboard Format	Conversion Format
CF_BITMAP	CF_DIB
CF_BITMAP	CF_DIBV5
CF_DIB	CF_BITMAP
CF_DIB	CF_PALETTE
CF_DIB	CF_DIBV5
CF_DIBV5	CF_BITMAP
CF_DIBV5	CF_DIB
CF_DIBV5	CF_PALETTE
CF_ENHMETAFILE	CF_METAFILEPICT
CF_METAFILEPICT	CF_ENHMETAFILE
CF_OEMTEXT	CF_TEXT
CF_OEMTEXT	CF_UNICODETEXT
CF_TEXT	CF_OEMTEXT
CF_TEXT	CF_UNICODETEXT
CF_UNICODETEXT	CF_OEMTEXT
CF_UNICODETEXT	CF_TEXT

- **HTML Clipboard Format**

  Clipboard uses a special format called HTML Clipboard format for transferring fragments of selected HTML text. Clipboard uses CF_HTML format for storing fragments of HTML text and its context as ASCII. This format allows applications to examine the all preceding surrounding HTML tags with their attributes. Interpretation of HTML fragments is depending on the application.

A window stores the data on the Clipboard when cut and copy operations are executed and is able to retrieve data when paste operation is executed. Only one window can open the Clipboard at a time. To get or retrieve data from the Clipboard, first, it should open the Clipboard by *OpenClipboard* function [8]. By using the *GetOpenClipboardWindow* [8] function, one can find out which window has opened the Clipboard. The window which owns the Clipboard must close it by calling the *CloseClipboard* function [8].

When a window writes information to the Clipboard, it must clear the previously added content in the Clipboard. When clearing the resources from the Clipboard, it sends a WM_DESTROYCLIPBOARD message to the previous Clipboard owner and then assigns the ownership to the current window which opened the Clipboard. This ownership lasts until the window closes the Clipboard or another window clears the Clipboard to place new data. After successfully emptying the Clipboard, all the formats will be placed from the most descriptive to least descriptive format and it uses the function *SetClipboardData* to specify the format identifier along with a global memory handle [8].

2.2.1.2 Dynamic Data Exchange (DDE)

Dynamic Data Exchange is a protocol for sharing information between different applications using a set of messages and guidelines. The protocol uses client and server architecture. An example scenario of this DDE is, a table in Microsoft word can be automatically updated by the data changes in a Microsoft Excel sheet. DDE uses shared memory like in interprocess communication (IPC) to share data. This DDE protocol can be used for sharing one-time data and for continuously interchanging between applications. DDE communicates by using message parsing which is the base of Windows architecture (message-based) and yet it is more suitable for transferring information automatically. DDE protocol uses two parameters, *wParam* and *lParam* for exchanging data. For a large piece of data transferring, DDE has defined how to use those two parameters by means of global atoms [11] and shared memory objects. Once the link is being established between applications, DDE will pass the data automatically without any involvement of the user. So, this method is better when less user participation is needed for data exchange. DDE can be used for real-time data transferring, creating compound documents, execute data queries between applications etc.

When sharing data between a client and a server, the DDE protocol identifies the units of data by application, topic and item names. By using the application name (name of the server application) and the topic, any DDE conversation can be identified uniquely. This unique identifier is determined by the client and the server at the beginning of the conversation. This application name and the topic cannot be changed through the time of the conversation. A window cannot have multiple conversations with another window. A window may be the main window of the application or a hidden window whose only purpose is to process DDE messages. For applications that operate on file-based documents, the topic is usually a filename. For other applications, the topic is an application-specific name. A DDE data item exchanges information between the server and the client. Data item values can be passed from client to server or from server to client with any standard Clipboard formats [10] or with any registered Clipboard formats. System topics are a set of predefined topics that provide information of general interest to DDE clients. These predefined topics are supported by the default processing of the *DdeServerManager*.

In DDE, an application (the client) can establish two types of data links: “warm data link” and “hot data link”. A data link is the way of notifying the client whenever a change occurs in the server side. If a client establishes a warm data link, the server only notifies the changes if the client has requested for it. In a hot data link, the server will immediately send the data when a new update happens on the server side. The client can make one or more permanent data links with the server. A DDE conversation can be ended by either the client or server at any time. Below are the steps or the events associated with a normal DDE conversation:

1. As in a normal client-server application, the client initiates the conversation, and the server responds to a valid request.
2. After establishing a successful connection between the client and the server, the applications exchange data by the following methods:
    - When the client requests, the server sends the data to the client.
    - The client sends data to the server application without any request from the server application.
    - At a request of a client, the server application performs a command.
3. The DDE conversation can be terminated by either the client application or the server application.

A client or a server must handle the DDE messages of a conversation asynchronously. However, it can switch between conversations as of a need. So, it is not necessary at all to process all messages asynchronously for all DDE conversations.

DDE uses atom tables to pass data between applications rather than passing the string value directly. Atom tables are implemented as hash tables [11]. A number of buckets can be changed by using the *InitAtomTable* function [12]. Default bucket size for local atom table is 37. An atom table stores a string (*atom name*) and a 16-bit number (*atom*) which returns to the application. Later, this 16-bit value can be used to access to that string.

Mainly there are two types of atom tables. Global atom table and local atom table. Global atom table is responsible for sharing item-name and topic-name. Applications which use DDE, pass the atom table to other DDE applications and those applications will use the atoms (unique values throughout the system) to get the specific string from the table. The Windows system provides several atom tables. Each atom table serves a different purpose. An application cannot directly access to those atom tables. But applications use those atom tables by calling different functions available in the API like *RegisterClipboardFormat*, *RegisterClass* or *RegisterClassEx* function etc. Applications which create their own private data formats must use the global atom table to hold the name. This will avoid the conflict of having the same name of formats by other applications or by the system.

Local atom table can only be used by the application which creates it. The advantages of using a local atom table are:

- Can reduce the memory usage by sharing the atom table among different structures which require the same string.
- Fast searching and comparison of strings by using atoms rather than using the actual string.
- Efficiently manage a large number of strings within an application.

2.2.1.3 Distributed File System (DFS) and DFS Replication

Distributed File System (DFS) is a client-server application which provides the ability to logically group shared folders located in multiple servers into a single hierarchical namespace. DFS supports two types of namespaces. Stand-alone DFS namespaces and domain-based namespaces. Stand-alone namespaces are used with single host server while domain-based namespaces are used with multiple host servers. Namespace

component provides the service of location transparency, and Distributed File System Replication component provides redundancy service. In the situation of a failure or overload, DFS maintains the availability of data by logically grouped multiple host servers under one root folder. This gives a virtual view of the shared folders to the users. Windows Server 2008 R2 Standard operating system can only host a single stand-alone namespace. However, the following operating systems can host both the multiple domain-based and single stand-alone namespaces servers [13].

- Windows Server (Semi-Annual Channel)
- Windows Server 2016
- Windows Server 2012 R2
- Windows Server 2012
- Windows Server 2008 R2 Datacenter/Enterprise

- **DFS Namespaces**

  DFS Namespaces provide an overview of the logically grouped folders on a network. Multiple namespaces can be created by the system administrator. In the user's perspective, these namespaces appear as a single shared folder with subfolders.

- **DFS Replication**

  This service used to synchronize the folders on multiple servers continuously. DFSR is a multi-master replication engine. Synchronizing the folders into multiple servers ensures the availability of the data, and thus it gives users reliable access to files.

  Below diagram illustrates the idea of DFS namespace servers [13]

Figure 2.2.1: DFS logical folder hierarchy

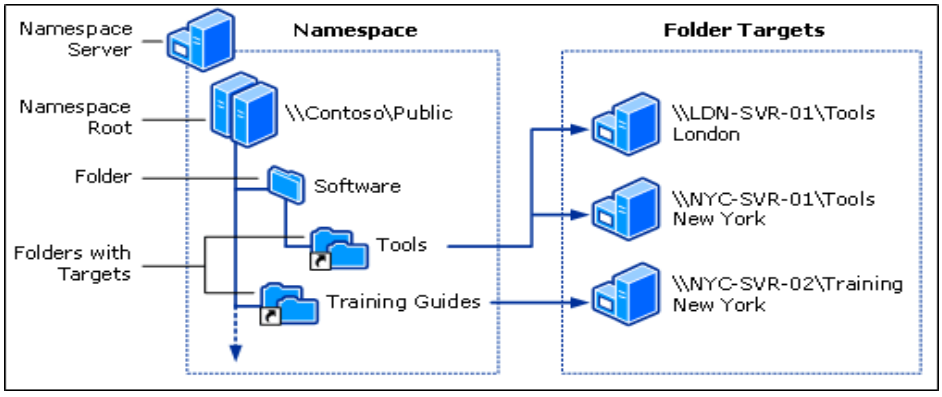


Figure 2.2.1 shows the tree-like structure of the DFS shared resources on a network. A Namespace server can be either a member server or a domain controller, and it hosts a namespace. Namespace root defines the initial point of the namespace and figure 2.2.1 shows a domain-based namespace (example: Contoso). A folder can be with a target or without a target. Target folders provide the actual content. When user access to a target by using a DFS link, it redirects the user to the first available target set. This ensures the availability of the data when some servers fail. Folder targets store the actual data and the content. When a user wants to access the \\Contoso\Public\Software\Tools the user will be redirected to either \\LDN-SVR-01\Tools or \\NYC-SVR-01\Tools based on the user's site information [13].

2.2.2 Security and Identity

The Security and Identity APIs enable applications to use Windows security features such as password authentication at login, privileged access control, authentication management, and security auditing. Following subsections explain Authentication and Authorization services available in Windows API.

2.2.2.1 Authentication

Authentication is the process of checking whether a user can access a system or not. It compares the username and the password that the user has entered with a given authorized list and if there is a match, the user will be granted to access the system with an allocated permission set. Below is the list of authentication technologies supported by Microsoft [14].

- **LSA Authentication**

  LSA stands for Local Security Authority. LSA authority used by applications to authenticate and log users to the system. Functionalities provided by the LSA let developers create and call authentication and combined security support provider packages. Authentication packages provide the following services: Inspecting login data, establish a new login session for login identifiers and passing security information to LSA [14].

- **Credentials Management**

  Credential Management API provides a way to interact and manage username and password. When an application needs some special permission, which is not provided by the credentials obtained while logging on to the system, these

functions can be used to call Windows account information and get the necessary information [14].

- **Smart Card Authentication**

  There are three main parts in the Smart Card Authentication based on PC/SC standards. PC/SC is a set of standards for integrating smart cards into computers to work together. Essential parts of the Smart Card Authentication are resource manager which interacts with Windows API, a user interface for resource management and service providers which enable access to specific services.

- **Network Provider API**

  This interface facilitates with functions which enable the Windows operating system to interact with networks. The advantage of the API is, the Windows operating system can communicate with different types of networks without any knowledge of network specific implementation details.

- **Security Support Provider Interface (SSPI)**

  The idea of SSPI functions is, an application can interact with different security models without changing the interface to the security system. SSPI does not have any access to create login credentials since it is handled by the operating system. SSPI is available in both the kernel-mode and user-mode.

2.2.2.2 Authorization

Authorization is the right granted an individual to use the system and the data stored on it. Sections like Access Control, Access Control Editor, Client/Server Access Control, Access Control for Application Resources, Mandatory Integrity Control and User Account Control are providing information about authorization [15]. Authorization is typically set up by a system administrator and determines the resources to which a user may gain access. Microsoft authorization technologies include the Authorization Manager and the Authz API.

- **Access Control –** Access control refers to security features which can access the resources in the Windows operating system [15]. This enables the application to control access to resources provided by the application or to set who have authority to access to specific features of the application.
- **Access Control Editor -** Access control editor consists of two main parts.
  - A basic security property page – This provides an interface to edit access control entries of an object's discretionary access control list (DACL).

DACL defines the who can access or who are denied access to securable objects.

- An advanced security property sheet – This contains a set of property pages which provides the functionality edit the object's system access control list (SACL).

- **Client/Server Access Control -** A server provides different services to one or many clients, which initiates requests for services like save and retrieve information from the database, provide access to network resources and control access to its services.
- **Access Control for Application Resources –** The Authorization Manager API (**aka** AzMan) and MMC snap-in are role-based access control frameworks which provide control to application resources. Access control for application resources can be in two types: Role-based Access Control and ACL (access control list)-based Access Control.
- **Mandatory Integrity Control -** Mandatory Integrity Control (MIC) is a classical computer science concept from the 1970s that's finally getting its first commercial implementation in Windows Vista. Mandatory Integrity Control (MIC) provides a mechanism for controlling access to securable objects [15].
- **User Account Control (UAC) –** This provides users with the ability to perform common tasks as standard users without switching the accounts, log off, or use "Run As". UAC feature was first added on Windows Server 2008 and Windows Vista. The advantage of this UAC is it reduces the number of applications that run with elevated privileges.

2.3 Wrapper APIs

This section describes the available wrapper APIs. A wrapper API provides a set of components that enable the access to the Windows API calls from the managed code environment. As described in section 2.2, Windows API functions can be directly accessed by the unmanaged code. The interaction between the managed code and unmanaged code has to be handled carefully. The reason is Data types, error-handling mechanisms, creation and destruction rules, and design guidelines are different between managed and unmanaged object models [16].

Common language runtime (CLR)

Common language runtime (CLR) is responsible for managing and execution of the code segments written in the .NET Framework. CLR makes the development process more manageable by supporting several services. Below is the list of services offered by the CLR.

- Memory management
- Security
- Transform the source code into Common intermediate language (CIL)
- Translate from intermediate language to machine code
- Promote the interaction between the managed code and the unmanaged code
- Exception handling
- Application memory isolation
- Thread management

Managed code

Managed code is running under the CLR. All the codes written in .NET Framework runs on the CLR and known as managed code. However, the exceptional case is, in visual .NET C++ the developer can either choose it to run in managed or unmanaged code. Managed code is first translated into CIL and then to machine code. This transformation (source code to CIL) is handled by the CLR. The advantage is CLR handles memory allocation, garbage collection, reference checking, etc.

Unmanaged code

Unmanaged code directly compiles to the machine code and no intermediate representation like in managed code. COM (Component Object Model) components, COM+ services, the Windows API are some types of unmanaged codes. There are three main technologies to interact with the managed and unmanaged code. COM interop, C++ interop and P/Invoke [16].

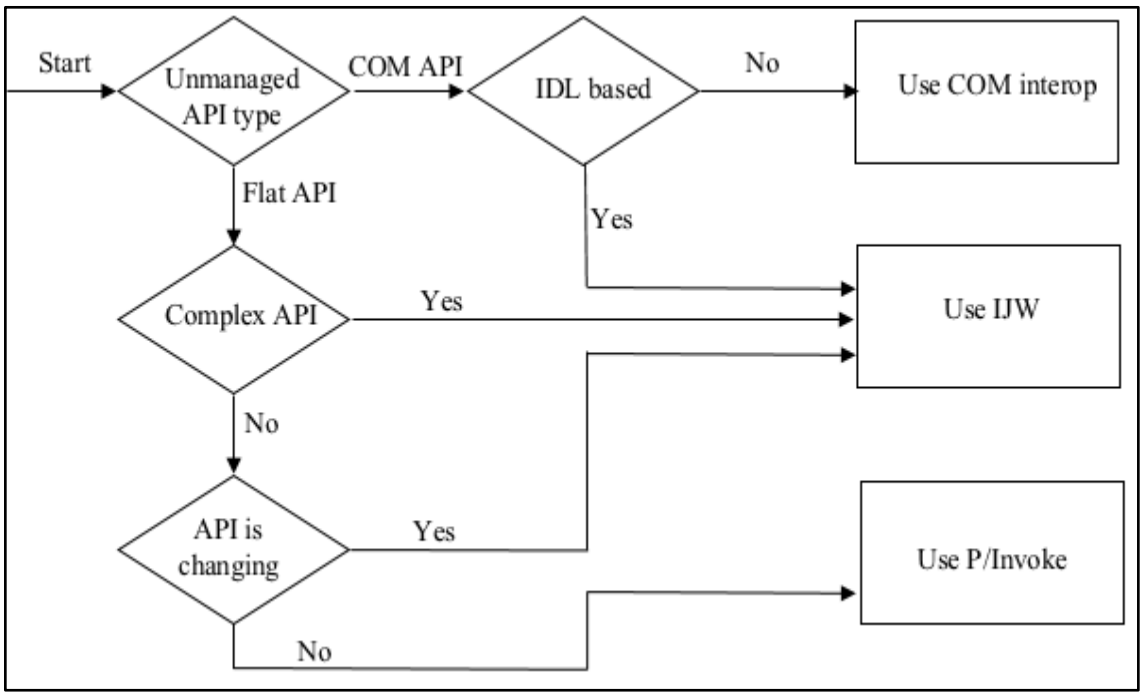
- **COM interop** - COM components were fundamental for Windows programming for many years. COM interop is included in the CLR, and it enables the interaction between managed code and COM components. COM components are located in the unmanaged memory, and the .NET Framework objects are located

in the managed memory. .NET Framework objects are managed by the CLR, and it can be moved at the runtime as necessary. However, the COM interop can be used by adding the *System.Runtime.InteropServices* namespace in .NET Framework.

- **C++ interop (Implicit Invoke) -** C++ interop or It Just Works (IJW) is a specific feature in C++ that enables flat APIs and COM APIs to be used directly by any .NET Framework supported language. This technology is more potent than COM interop because it provides performance enhancement, better type safety, easy to implement. Unlike in Platform Invoke, in C++ interop it is unneeded to specify how function parameters should be marshalled.
- **Platform Invoke (P/Invoke)** - Platform Invoke enables calling any unmanaged code functionality by directly importing the required DLL (ex: kernel.dll, user2.dll etc.). The correct function signature must be declared to invoke the function. This technology is further described in section 2.3.3 with an example.

The figure 2.3.1 shows the decision tree for choosing the technologies available for calling unmanaged APIs [16].

Figure 2.3.1: Decision Tree for Calling Unmanaged APIs [16]



Depending on the unmanaged API type and the interop technologies, the mechanism must be chosen. Figure 2.3.1 gives a great overview of deciding about choosing the correct technology for calling unmanaged APIs. As shown in the figure 2.3.1, Platform

Invoking is the best option rather than C++ interop to call simple unmanaged flat APIs. The CLR takes care about loading the DLLs and marshalling the parameters. Platform Invoke technology has been used to access the functionalities of the Windows API for the development of the thesis prototypes.

Marshalling

Marshalling is the process of organizing or converting managed data into a format that is prescribed in the receiving unmanaged environment, or vice-versa. CLR marshalling service can be accessed under *System.Runtime.InteropServices*[6] and it is responsible for that the data is correctly transferred. Table 2.3.1 contains the list of data types used in the Win32 API, unmanaged C language type functions and managed class names in .NET Framework [17]. Same size types can be substituted.

Table 2.3.1: List of data types in managed and unmanaged environment

Unmanaged type in Win32 API	Unmanaged C language type	Managed class name	Description
HANDLE	void*	System.IntPtr	A platform-specific type. Used to represent a pointer or a handler. 32 bits on 32-bit Windows operating systems, 64 bits on 64-bit Windows operating systems
BYTE	unsigned char	System.Byte	Unsigned integers with values ranging from 0 to 255. 8 bits
SHORT	short	System.Int16	Represents a 16-bit signed integer
WORD	unsigned short	System.UInt16	Represents a 16-bit unsigned integer
INT	int	System.Int32	Represents a 32-bit signed integer
LONG	long	System.Int32	Represents a 32-bit signed integer

[6] This namespace provides functionalities that support COM interop and platform invoke services

Blittable types

Most of the data types used in the Win32 API and .NET Framework have common representation and do not require explicit handling by interop marshaler. Those are called blittable types. Structures that are returned from platform invoke calls must be blittable types. Platform invoke does not support non-blittable structures as return types.

Below is the list of some blittable types from the System namespace in .NET Framework [18]. The full list can be found in the appendix section.

- System.Byte
- System.SByte
- System.Int16
- System.UInt16
- System.Int32

Non-Blittable Types

Representation of the Non-Blittable types between the managed and the unmanaged memory is dissimilar and do require marshalling. Object references are not blittable. When accessing the unmanaged functions in the managed environment, the conversion between the non-blittable data types must be done correctly. Table 2.3.2 shows Non-blittable types from the System namespace in .NET Framework [18].

Table 2.3.2: Non-blittable types from System namespace

Non-blittable type	Description
System.Array	Converts to a C-style array or a SAFEARRAY
System.Boolean	Converts to a 1, 2, or 4-byte value with true as 1 or -1
System.Char	Converts to a Unicode or ANSI character
System.Class	Converts to a class interface
System.Object	Converts to a variant or an interface
System.Mdarray	Converts to a C-style array or a SAFEARRAY

System.String	Converts to a string terminating in a null reference or to a BSTR
System.Valuetype	Converts to a structure with a fixed memory layout
System.Szarray	Converts to a C-style array or a SAFEARRAY

**Class and object types are not supported by Platform Invoke interop.

2.3.1 Managed windows API

A managed window API is a set of .NET components which enable access to the Windows API by managed code environment. This wrapper API provides a high-level interface for the low-level API by wrapping all P/Invoke calls (see section 2.2.3) with error checking. It also contains example tools (the list can be found in the appendix) built using the API for reference, source code, binary (DLL) to download. This managed API covers below functionalities.

- Clipboard notifier - Specifies a component that monitors the system Clipboard for changes.
- Unique code-point range - The Unicode range of codepoints supported by a font.
- Crosshair - Creates a new crosshair control.
- General Window settings
- Extended file information - Provides methods for getting additional information about files, like icons or compressed file size.
- ListView and TreeView controls
- Hotkey accessor
- Sounds and audio
- Accessibility
- Keyboard key - This class contains utility methods related to keys on the keyboard.
- Low-level system hooks
- Creating a new memory chunk that points to existing memory

Below code, segment demonstrates how to use *ClipboardNotifier* class in Managed Windows API. *ManagedWinapi* is a binary file and can include to the project as a DLL file. This code example sets the current application to the chain of Clipboard viewers. Inside the *ClipboardNotifer* constructor, it registers itself to the Clipboard chain. Whenever there is a change in the system Clipboard, the application receives

WM_DRAWCLIPBOARD message. It also handles the message passing to the next application in the Clipboard chain. This is necessary to be done. Otherwise it breaks down the Clipboard chain. The advantage is, all these functionalities are handled inside the *ClipboardNotifier* class. Thus, the developer who uses the Managed Windows API does not have to handle these operations explicitly.

```
1  Class Program {
2   Private static ManagedWinapi.Clipboard Notifier clipNotifier;
3   static void Main(string[] args) {
4    clipNotifier = newManagedWinapi.ClipboardNotifier();
5    clipNotifier.Clipboard Changed += ChangeOccured;
6   }
7   Protected static void ChangeOccured(object sender, EventArgs e) {
8    Console.WriteLine("Clipboard changed!");
9   }
```

ClipboardNotifer class also provides the functionality to remove the window from the Clipboard chain. Below is the list of method signatures used inside the *ClipboardNotifier* class and the usage of each of the method signature has been explained subsequently.

SetClipboardViewer

This function registers the current window to the chain of Clipboard viewers. After successfully registering the window to the Clipboard chain it receives WM_DRAWCLIPBOARD message when the Clipboard changes. Below is the P/Invoke method signature for the *SetClipboardViewer* function. The function returns a handler to the next window in the Clipboard viewer chain in the successful call.

```
1  [DllImport("User32.dll", CharSet = CharSet.Auto)]
2  Private static extern IntPtr SetClipboardViewer(IntPtr hWndNewView);
```

ChangeClipboardChain

This *ChangeClipboardChain* function is responsible for removing the current window from the Clipboard chain applications. Parameter *hWndRemove* is the handler of the window that must be removed from the Clipboard chain. The second parameter *hWndNewNext* is the handle to the window that follows the *hWndRemove* window in the Clipboard viewer chain. This parameter is the output from the *SetClipboardViewer* function. Below code, segment shows the P/Invoke method signature for the

ChangeClipboardChain function. The function returns true if the window has been successfully removed from the Clipboard chain.

```
1  [DllImport("user32.dll")]
2  Private static extern bool ChangeClipboardChain(IntPtr hWndRemove, IntPtr
3  hWndNewNext);
```

SendMessage

Each window which is registered in the Clipboard chain receives WM_DRAWCLIPBOARD message. All the windows which receive this message must send the WM_DRAWCLIPBOARD message to the next window in the chain.

```
1  [DllImport("user32.dll", CharSet = CharSet.Auto)]
2  private static static int SendMessage(IntPtr hwnd, int wMsg, IntPtr
3  wParam, IntPtr lParam);
```

2.3.2 EasyHook

Easy Hook was firstly introduced in 2008. Easy Hook provides a mechanism to hook unmanaged code from the pure managed environment. Easy Hook makes hooking a simple task. Following is the list of features available in Easy Hook [19].

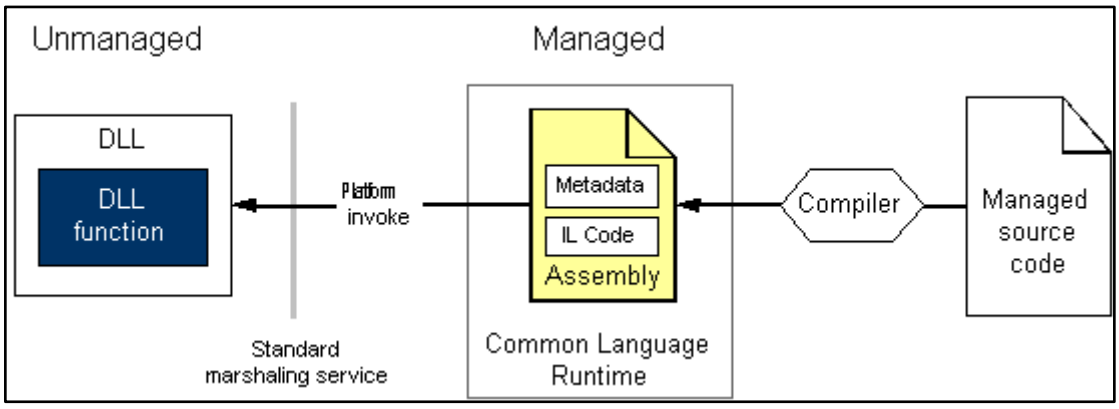
- Supports "Thread Deadlock Barrier" technology which is unique to EasyHook. This technology handles many significant problems when hooking unknown APIs.
- Can use in .NET Remoting, WPF and WCF as well.
- Can write managed hook handlers for unmanaged APIs.
- Support for hooking COM interfaces.
- Good documentation, pure managed hooking API.
- No resource or memory leaks are left in the target
- EasyHook libraries can be used without any .NET framework installed.
- Hooks are installed and removed stably.

2.3.3 P/Invoke

As explained in section 2.3 under the interaction technologies between managed and unmanaged code, Platform Invoke or P/Invoke enables calling functions that are implemented in unmanaged DLLs from a managed code environment. P/Invoke is the only option available for calling unmanaged functions when the source code for the DLL is not available. This service allows invoking an exported function and marshals the

parameters as needed at runtime. Figure 2.3.2 illustrates the process of locating and invoking an unmanaged DLL function [20].

Figure 2.3.2: The process of locating and invoking an unmanaged DLL function



As shown in the figure 2.3.2 invoking an unmanaged function in a DLL file can be mainly divided into following actions.

- Locate the DLL using metadata provided
- Loads the DLL file into the memory which contains the specified function
- Retrieve the address of the function and pushes the function arguments onto the stack. Marshalling the data is required depending on the argument type. Table 2.3.1 shows the list of data types in managed and unmanaged environment. Pushing arguments onto the stack by locating the function address in memory only happens in the initial call to the function.
- The final step is to hand over the control to the unmanaged function.

Below is a P/Invoke method signature for function *PostMessage* which posts a message to the message queue associated with the thread. There are several steps included calling an exported DLL function. All those steps and the attributes used in the method signature are explained subsequently.

```
1  [return: MarshalAs(UnmanagedType.Bool)]
2  [DllImport("user32.dll", SetLastError = true, CharSet = CharSet.Auto)]
3  public static extern bool PostMessage(IntPtr hWnd, uint Msg, IntPtr wParam
4  , IntPtr lParam);
```

To consume an exported DLL function, below sequence of steps are involved.

1. Identify the required function in the DLL file

   Most commonly used DLL files in Windows API are GDI32.dll, Kernel.dll, and User32.dll. To identify the required function, as shown in the above code

segment, the function name (ex: ***PostMessage***) and its location (ex: ***User32.dll***) are needed. Windows API functions may contain two versions of each function based on the character set property. It can be either 1-byte character ANSI[7] version or 2-byte character Unicode version. Further information about character sets (represented by *CharSet* field) is described in a below subsection under the topic *CharSetAttribute*.

2. Create a separate class to hold method signatures

   This step is not mandatory but using a separate class to hold method signatures leads to encapsulate platform functionality. To make it more convenient for use in developing environment DLL function names can be renamed. To ensure the renamed function calls the correct function in the DLL, you must explicitly specify the entry point for the renamed function.

3. Create a static entry point to export the DLL function

   Use the *DLLImport* attribute to identify the function. This exported function must be marked with *static* and *extern* modifiers.

4. Call the DLL function

   Call the declared method in managed code by passing expected parameters.

Pinvoke.net

When invoking unmanaged DLL from managed environment, developers must pay attention to use correct method signatures. This includes correct DLL name, string formatting between managed to unmanaged, marshalling of data types from managed to unmanaged and vice-versa. This is very time consuming and prone to make runtime failures if the method signature is incorrect. Pinvoke.net is a wiki which helps developers to modify and add Platform Invoke signatures, user-defined types, code examples and other relevant information regarding Windows API. So, the developers can easily find the required method signatures. Also, they can contribute to the community by providing method signatures and example codes. This provides an add-in to Visual Studio 2010 and 2013 version which makes easy to insert P/Invoke signatures. This wiki can be accessed under the www.pinvoke.net URL.

[7] ANSI is a character encoding schema in Windows operating system

3 IMPLEMENTATION

The following chapter addresses implementation aspects of the defined use-cases and describes the win32 API functionalities used for the development. Furthermore, it explains all the methodologies carried out and the reasons for choosing the proposed solution among other solutions.

3.1 Use-case 1 - Detect pasted confidential data in MS Office applications

The following chapter describes the concept of detecting paste event of confidential data in MS Office applications (MS Word, Excel and PowerPoint).

Accidental data leaks can happen at any time by the employees. As described in section 1.2 this has become a significant issue. MS Word, Excel and PowerPoint are mostly used day to day applications at present. A user can share a newly created MS Word, Excel or PowerPoint file which contains confidential data accidentally with an external user. The question is how to prevent those kinds of leaks at the application level. Some of the possible solutions were discussed in section 1.3. However, it would be better if we can handle them automatically by a software application. We can notify the user that the document contains confidential information and ask to save the file as a confidential file. The challenge is how to track that information in the Windows environment.

Requirements

In the requirement section, properties and requirements of the use-case are identified and described informally. Based on these requirements different solution approaches, as well as resulting implications, are discussed, eventually leading to a working concept. As described in the description section above, the main idea of the thesis is to prevent accidental data leaks happen by the employees. In general, the use-case can be divided into two parts. Collecting copied content from confidential files and detecting pasted content on MS Word, Excel and PowerPoint documents. Table 3.1.1 shows the functional requirements for collecting copied content from confidential files.

Table 3.1.1: Functional requirements for use-case 1

ID	Description
$\mathbb{R}_1$	The system should collect the copied content from the confidential applications (Microsoft Word, Excel and PowerPoint).
$\mathbb{R}_2$	The system should only collect information if the copied source is confidential.

$\mathbb{R}_3$	When starting the system should clear the system Clipboard.
$\mathbb{R}_4$	When starting the application, the system should clear the collected content from the application itself.
$\mathbb{R}_5$	*When starting the application, the system should clear the collected content from the Microsoft Office Clipboard.
$\mathbb{R}_6$	The system should be able to store the copied source of the content.
$\mathbb{R}_7$	The system should be able to store the copied Date of the content.
$\mathbb{R}_8$	The system should be able to store the copied Time of the content.
$\mathbb{R}_9$	The system should be able to store the username of the current Windows user.
$\mathbb{R}_{10}$	The system should be able to store the copied text.
$\mathbb{R}_{11}$	When the application closes, the system should be able to store the collected data on the disk.
$\mathbb{R}_{12}$	When the application starts, the system should be able to restore the collected data from the disk.
$\mathbb{R}_{13}$	The system will store the Clipboard history data up to the maximum of 24 elements.
$\mathbb{R}_{14}$	When the Clipboard history count exceeds 24 elements, the system will clear the Clipboard data from the application.
$\mathbb{R}_{15}$	*When the Clipboard history count exceeds 24 elements the system will clear the Clipboard data from the Microsoft Office Clipboard.

*$\mathbb{R}_5$ and $\mathbb{R}_{15}$ requirements apply to the use-case clearing the office Clipboard data. It will be discussed later in the corresponding section for the use-case "Clear MS Office Clipboard data".

Table 3.1.2 shows the functional requirements for MS Office add-ins which detect pasted confidential data in MS Word, Excel and PowerPoint applications.

Table 3.1.2: Functional requirements for Office add-ins

ID	Description
$\mathbb{R}_{16}$	When the user saves the document, the system should scan the document.
$\mathbb{R}_{17}$	The system must store the detected confidential data in memory.
$\mathbb{R}_{18}$	When the search is finished, the system must show a message to the user to save the file as a confidential file.

$\mathbb{R}_{19}$	When the user accepts to save the file as a confidential file, the system will save the file as a confidential file.
$\mathbb{R}_{20}$	When the user ignores to save the file as confidential, the system should log the event with the matching confidential data.
$\mathbb{R}_{21}$	The system should include username, date, time and matched confidential texts data to the log file
$\mathbb{R}_{22}$	When the document is marked as confidential, the system does not need to search the document for confidential content.

Attempt 1 - Use .NET Clipboard Class

.NET Framework is a software framework developed by Microsoft. It provides a large number of class libraries which can be used in development. Microsoft has introduced Clipboard class with .NET Framework 3.0 under the namespace of *System.Windows.Forms* [7]. This class library facilitates exchanging data with and from the system Clipboard. A full list of features can find in the appendix. Example of copying a simple text to the system Clipboard using a console application is shown below.

```
1 using System;
2 using System.Windows.Forms;
3 namespace WordAddInClip {
4 Class Program {
5 [STAThread] private static void Main(string[] args) {
6 Console.WriteLine("Copying a text to the Clipboard ");
7 Clipboard.SetText("Simple text");
8 Console.Read();
9 }}}
```

The major drawback of the Clipboard class provided by the .NET Framework is, it unable to fulfil the requirements $\mathbb{R}_1$, $\mathbb{R}_2$, $\mathbb{R}_5$, and $\mathbb{R}_6$. Clipboard class does not provide any method to get the source of the copied content (fail to fulfil $\mathbb{R}_6$). So, it eventually fails to fulfil the requirements $\mathbb{R}_1$ and $\mathbb{R}_2$. There is no method provided by the Clipboard class to clear the Microsoft Office Clipboard which is a major requirement ($\mathbb{R}_5$) of the use-case.

As stated in the Microsoft developer network under the Clipboard class section, “Paste operations need to be user initiated (Ctrl-V, Paste Menu)” due to security purposes [7]. Thus, Clipboard class cannot be used to detect paste event in Windows environment.

Since this is a major requirement of the use-case, we decided to move further and do more research on alternative solutions.

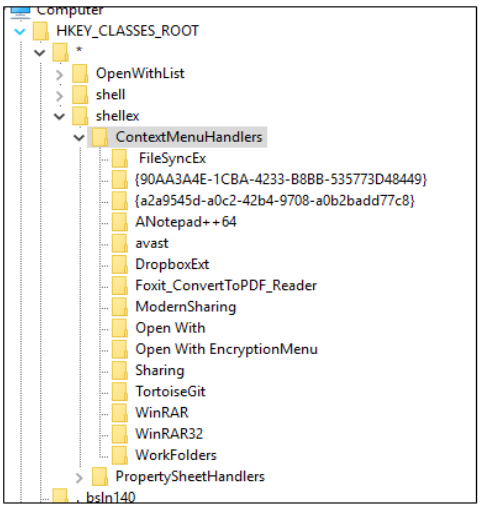
Attempt 2 - Listening to Windows messages to detect paste operation

Next attempt was to detect any notification generated by the Windows operating system when paste operation take place. As stated in the Microsoft documentation "An application sends a **WM_PASTE** message to an edit control or combo box to copy the current content of the Clipboard to the edit control at the current caret position. Data is inserted only if the Clipboard contains data in CF_TEXT format" [21]. The **WM_PASTE** message is an internal application message. It sends a notification to the active control window when choosing the paste option from the Edit menu of an application, the context menu of a control window, or upon the Ctrl-V / Shift-Ins accelerators. This message is handled in application level, and such application-level messages are not forwarded to Windows or other applications. This attempt failed due to no notification is generated by the Windows operating system for the paste operation.

Attempt 3 - Extend "Paste registry key" in shell context menu

The context menu is a graphical user interface (GUI) that enables control to expose functionality of the operating system or an application which the context menu belongs. An application can extend the application context menu and/or operating system context menu by adding registry key entries. Shell context menu can be found in the registry key under HKEY_CLASSES_ROOT\shellex\ContextMenuHanlders. The idea was to extend the "Paste registry key" in the context menu. Figure 3.1.1 shows sample registry key items in HKEY_CLASSES_ROOT\shellex\ContextMenuHanlders.

Figure 3.1.1: Shell Context menu registry key handlers



As shown in the figure 3.1.1 there is no paste registry key in the registry. Shell context menu items are free text, and they are not Windows registry standard items. The idea is, an application can create their own context menu which replaces the “paste” item and link it to perform a different task rather than performing the default task. So, attempt 3 failed due to no registry key found in the windows registry.

Attempt 4 - Detect keystroke

Most of the applications use Ctrl +V as a common key combination to perform a paste operation. However, there is no standard way to perform the paste operation. Any application can develop this functionality in many ways. Microsoft Excel provides 14 additional paste options referred to as *Excel’s paste special.* Table 3.1.3 shows the available special paste options in Excel 2016 and a description of each option [22].

Table 3.1.3: Special paste options in Excel 2016

Shortcut	Operation	Keystroke
P	Paste	Ctrl + Alt + V then P
F	Formulas	Ctrl + Alt + V then F
O	Formulas and number formatting	Ctrl + Alt + V then O
K	Keep source formatting	Ctrl + Alt + V then K
B	No borders	Ctrl + Alt + V then B
W	Keep source column widths	Ctrl + Alt + V then W
T	Transpose	Ctrl + Alt + V then T
V	Values	Ctrl + Alt + V then V
A	Values and number formatting	Ctrl + Alt + V then A
E	Values and source formatting	Ctrl + Alt + V then E
R	Formatting	Ctrl + Alt + V then R
N	Paste link	Ctrl + Alt + V then N
U	Picture	Ctrl + Alt + V then U
I	Link picture	Ctrl + Alt + V then I

To detect all these key combinations in Microsoft, Excel 2016 application a program must hook the keystrokes shown in the table 3.1.3. A Hook is a mechanism to detect

messages, mouse actions and keystrokes by an application. This is for only Microsoft Excel application. We must also consider Microsoft Word and PowerPoint applications as well. The initial idea was to hook all the possible key combinations to detect paste operation. As stated in the Microsoft documentation "Hooks tend to slow down the system because they increase the amount of processing the system must perform for each message. You should install a hook only when necessary and remove it as soon as possible" [23]. So, concerning the performance of the application and as well as the entire system, this idea was not implemented. Besides these complications there is another drawback to the use of hook procedures: the application unable to capture other possible paste options such as shell context menu paste option and pasting from edit menu of an application.

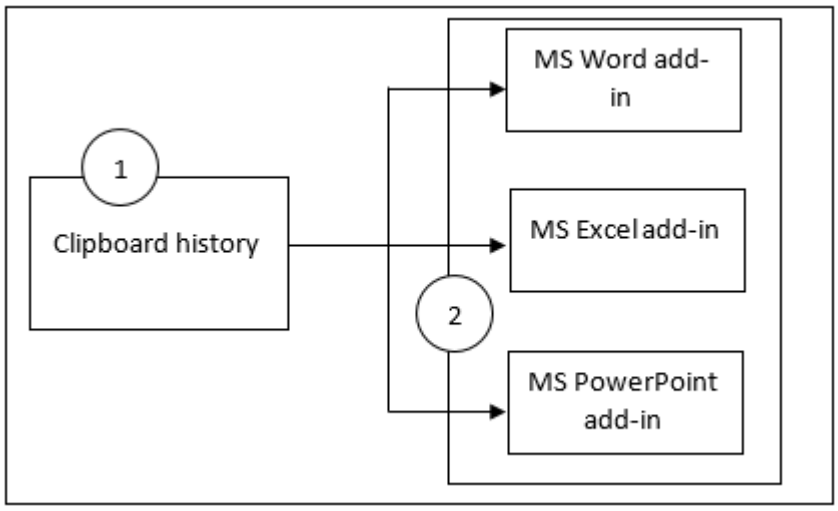
Attempt 5 – Access Microsoft Office Clipboard

Microsoft Word, Excel and PowerPoint have a shared Clipboard tool which stores the copied content and perform a paste operation. The idea was to find a way to access the Office Clipboard to get the history of the copied data and as well as detect paste event inside Microsoft Word, Excel and PowerPoint applications. Microsoft does not provide any documentation or any API to access the Office Clipboard content. Office Clipboard works the same as the solution proposed by this thesis. It registers the Office application to the Clipboard viewer chain with *SetClipboardViewer*, and it is notified any changes happen in the system Clipboard.

Attempt 6 - Working solution

Since all the previous attempts failed to fulfil all the requirements stated in tables 3.1.1 and 3.1.2, attempt 6 came as an alternative working solution. Below diagram shows the conceptual view of the proposed solution for the use-case. This provides a high-level overview of the solution and gives the reader a first impression of the general architecture.

Figure 3.1.2: The high-level architecture of the proposed solution



As stated earlier, the system consists of two primary components as illustrated in Figure 3.1.2. The first part "Clipboard History" is responsible for collecting and maintaining a Clipboard history. Component number 1 is responsible for fulfilling the requirements $\mathbb{R}_1$ to $\mathbb{R}_{15}$ except for the requirement $\mathbb{R}_5$. Below steps help to understand the process of the program.

A. Read Clipboard history data from the disk.
B. Empty the system Clipboard.
C. Add the application to the Clipboard viewer chain.
D. Listen to the WM_DRAWCLIPBOARD[8] message generated by the Windows operating system.
E. If any changes occurred in the system Clipboard:
    a. Get copied content from the system Clipboard.
    b. If the copied content format is in one of the following formats, go to step iii. (Valid formats: CF_TEXT, CF_UNICODETEXT). Otherwise, go to step D.
    c. Get the window handle of the current owner of the Clipboard.
    d. Get the *processId* of the current owner of the Clipboard.
    e. Get the window title of the source file of copied content.
    f. If the source file is Microsoft Word, Excel or PowerPoint, save Window title, username, copied content, date and time in the disk.

[8] This message is sent to the first window in the clipboard viewer chain

g. Add saved data in step vi to the application grid view.

h. Send added data to Word, Excel and PowerPoint add-ins.

i. Go to step D.

Figure 3.1.3: Class diagram of use-case 1 "Clipboard History" application

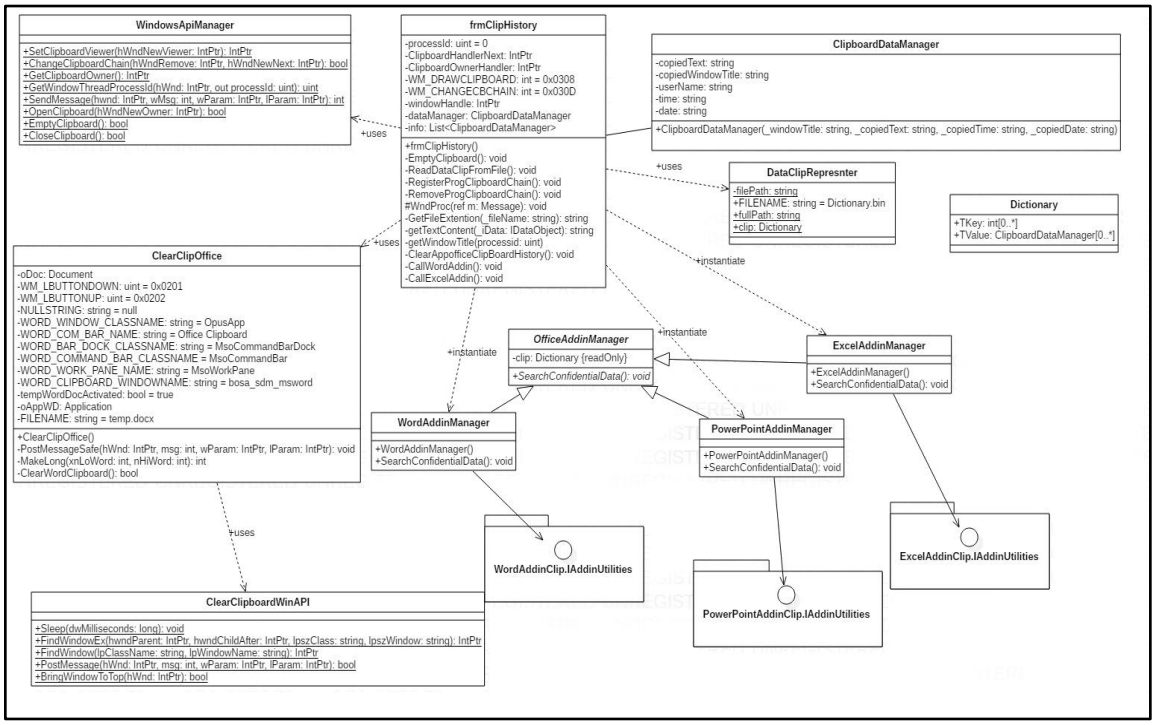


Figure 3.1.3 shows the class diagram of the “Clipboard History” application. The following code example shows how to use the *DllImportAttribute* attribute to import the Win32 *SetClipboardViewer* function. The code example then calls the imported method inside the construction.

```
1  using System;
2  using System.Runtime.InteropServices;
3  namespace Clipboard Prototype {
4  Public partial class frmClipHistory: Form, {
5  [DllImport("user32.dll", CharSet = CharSet.Auto)]
6  public static extern IntPtr SetClipboardViewer(IntPtr hWndNewViewer);
7  public frmClipHistory() {
8  InitializeComponent();
9  SetClipboard Viewer(this.Handle);
10 }}}
```

WordAddinManger, ExcelAddinManager and PowerPointAddinManager classes require WordAddinClip.dll, ExcelAddinClip.dll and PowePointAddinClip.dll respectively to

send Clipboard history data to add-ins. Below code, segment is responsible for getting the COM add-in in the Microsoft Office Word application and returns the *IAddinUtilities* object that is the basis for the specified “WordAddinClip” object.

```
1  word = new Microsoft.Office.Interop.Word.Application();
2  object addinName = "WordAddInClip";
3  COMAddIns comAddins = word.COMAddIns;
4  COMAddIn comAddin = comAddins.Item(addinName);
5  WordAddInClip.IAddinUtilities comObj = (WordAddInClip.IAddinUtilities) comAddin.Object;
```

COMAddIns interface is a collection of *COMAddIns* objects that provides information about a COM add-in registered in the Windows Registry. This *comAddins* object at line 3 in the above code example contains all the COM add-ins that are registered in the Microsoft Word application. *comAddin* object in line number 4 represents the COM add-in which has the name “WordAddInClip”. When exposing add-in classes to out-of-process clients, a race condition can occur leading to *comAddin* object to having a *null* value. The race condition might occur when the out-of-process client holds a collection of *COMAddIns* and *COMAddIn* objects before the host application (In above code example host application is Microsoft Word) loads the add-in. To overcome this problem below code segment can be used. Each second it returns the *IAddinUtilities* object from the “WordAddinClip” until the *comObj* is not *null*.

```
1  while (comObj == null) {
2  comObj = (WordAddInClip.IAddinUtilities) comAddin.Object;
3  System.Threading.Thread.Sleep(1000);
4  }
```

Detecting pasted content on Microsoft Office applications

Microsoft office add-in platform allows building custom add-ins to extend the functionality of Office applications. It also allows interacting with the content of the MS Office document. To detect confidential data in Microsoft Office Word, Excel and PowerPoint, three different add-ins were created. These add-ins are responsible for scanning the office document for confidential data with the data in the “Clipboard History” program. Below steps show the process of the office add-ins created for search confidential data.

A. Collect data sent by “Clipboard History” application.
B. Save collected data in the XML file.

C. When the user clicks on saves, save as, or close button starts scanning the document.

D. Retrieve data from the XML file.

E. Search entire document for matching the content.

F. If any matching content found, add it to the memory.

G. Highlight the matching content.

H. After finish searching the document,

   a. If there any matching content, ask the user to save the document as a confidential file.

   b. If no matching content found no actions would be taken.

   c. When the user selects to save the file as a confidential file:

      i. Change the title of the document by adding "_confidential" text at the end of the document title.

      ii. Add custom property tag to the document.

      iii. Save the current document.

      iv. Close the current document.

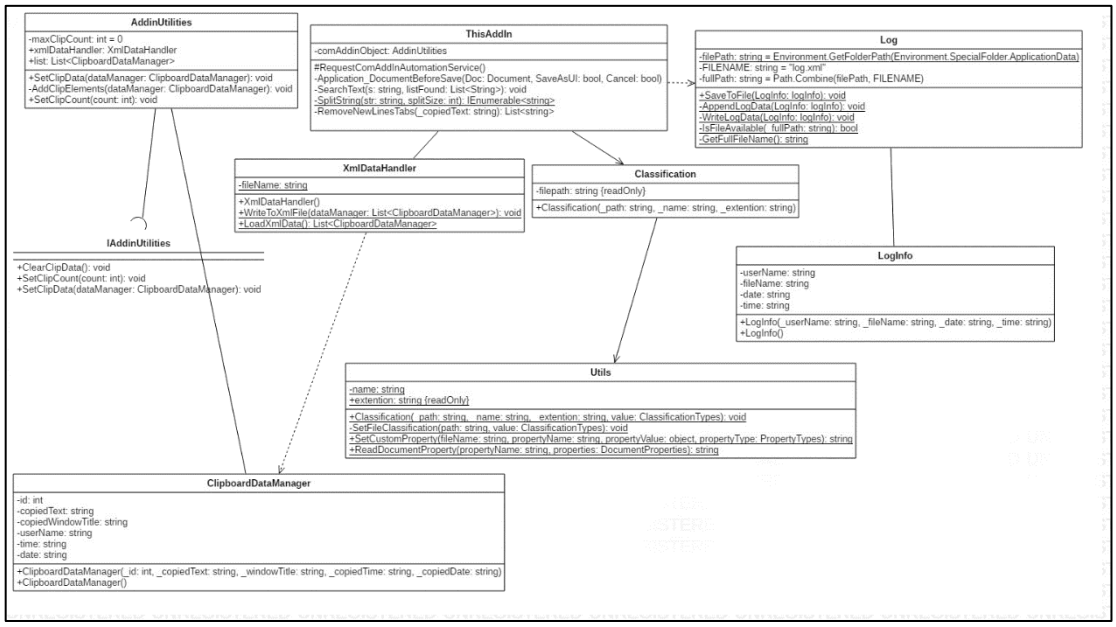
      v. Replace the new document with the old document.

      vi. Open the new document.

   d. When the user rejects to save as a confidential document:

      i. Save details in a log file including file name, username, date, time.

Figure 3.1.4: Class diagram for Microsoft Word add-in



**Class diagrams of MS Excel and PowerPoint add-ins can be found in appendix

As shown in the figure 3.1.4 *ThisAddin* class contains a protected override method *RequestComAddInAutomationService* under *Microsoft.Office.Tools* namespace. This method is important because it returns an object in our add-in that can be used by other third-party applications. The requirement for exposing the *comAddinObject* object to outside is, it must be public, it must be visible to COM, and it must expose the *IDispatch* interface[9]. Otherwise, the application will throw an *InvalidCastException* when calling the required add-in.

```
1  Private AddinUtilities comAddinObject;
2  protected override object RequestComAddInAutomationService() {
3  if (comAddinObject == null) {
4  comAddinObject = new AddinUtilities();
5  }
6  return comAddinObject; }
```

3.2 Use-case 2 - Clear Office Clipboard

The following chapter explains the mechanism for clearing Microsoft Office Clipboard data. Microsoft Office applications use their own Clipboard tool. It shares the Clipboard data among Microsoft Word, Excel and PowerPoint applications. It collects the copied data up to 24 items from the Office documents and other applications. Office Clipboard contains options to paste or delete individual items by clicking on each item, paste all data to the document by clicking on "Paste All" button and empty the Clipboard by clicking on "Clear All" button. This tool keeps a history of copied texts, data from a workbook or datasheet, image etc.

The only requirement for this use-case is to clear the Office Clipboard data. Two approaches have been taken to fulfil the above requirement. Both the approaches are explained in the following sections.

Attempt 1 – Use of MSO.DLL shared Office component

As stated in the article "Office Clipboard API" [24] Microsoft office Clipboard tool attached to the Clipboard viewer chain and receives the changes occur in the system Clipboard. Microsoft Office Clipboard tool uses MSO.DLL shared Office component to keep a list of items in shared memory [24]. Author of the article has provided binary of the project implemented and as well as the source code which was written in C++ language. The given program can provide the details of MSO.DLL details, number of

[9] This interface allows client applications to use the methods and properties exposed by IAccessible interface

items on the Office Clipboard and clear Office Clipboard. Below code sample can be used to call the MSOClipHelper.DLL functionalities by C#.

```
1  class Program
2  {
3  static void Main(string[] args)
4  {
5  MSOClipHelperLib.OfficeClipHelperFactory clip = new MSOClipHelperLib.OfficeClipHelperFactory();
6  MSOClipHelperLib.OfficeClipHelper clipi = clip.OfficeClipHelper;
7  Console.WriteLine("count" + clipi.NumClipItems);
8  Console.ReadLine();
9  }}
```

This program gives *"Can't get module handle for MSO.DLL"* COMException at the line number 6 and fails to execute further.

Attempt 2 – Working solution using Win32 API

A mechanism to clear the Office Clipboard programmatically by using Windows API functions was described by A. Bhatia [25]. Each window in the system has a unique window class name. This class name distinguishes the windows from other registered classes. Table 3.2.1 shows the list of window class names used in some of the Microsoft Office applications [26].

Table 3.2.1: Window class names of Microsoft Office applications

Application	Class name
Access 97	OMain
Access 2000	OMain
Access XP	OMain
Excel 97	XLMAIN
Excel 2000	XLMAIN
Excel XP	XLMAIN
FrontPage 2000	FrontPageExplorerWindow40
FrontPage XP	FrontPageExplorerWindow40
Outlook 97	rctrl_renwnd32
Outlook 98	rctrl_renwnd32

Outlook 2000	rctrl_renwnd32
Outlook XP	rctrl_renwnd32
PowerPoint 95	PP7FrameClass
PowerPoint 97	PP97FrameClass
PowerPoint 2000	PP9FrameClass
PowerPoint XP	PP10FrameClass
Project 98	JWinproj-WhimperMainClass
Project 2000	JWinproj-WhimperMainClass
Visual Basic Editor	wndclass_desked_gsk
Word 97	OpusApp
Word 2000	OpusApp
Word XP	OpusApp

FindWindow

The *Findwindow* function retrieves a handler to the top-level window whose class name and window name match the specified strings. Below code shows the platform Invoke signature of the *FindWindow* function.

```
1  [DllImport("user32.dll", SetLastError = true)]
2  Public static extern IntPtr FindWindow(string lpClassName, string lpWindowName);
```

FindWindow accepts two string type parameters namely: *lpClassName* (Class name of the window) and *lpWindowName* (Title of the window). Parameter *lpWindowName* can be null if the window title is unknown. Below code block shows the example usage of the *FindWindow* function.

```
1  IntPtr how = FindWindow("OpusApp", null);
2  if (hnw != IntPtr.Zero) { //Continue }
```

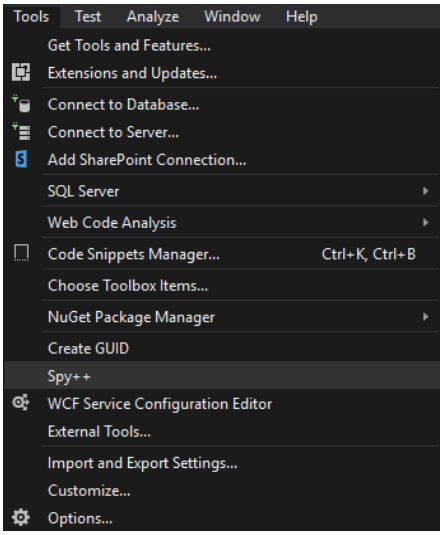
To get a handler to Microsoft Office Clipboard window, we must know the class name of that window. Following list represents the class names required to get access to the Office Clipboard window.

- WORD_WINDOW_CLASSNAME = "OpusApp";
- WORD_COM_BAR_NAME = "Office Clipboard ";

- WORD_BAR_DOCK_CLASSNAME = "MsoCommandBarDock";
- WORD_COMMAND_BAR_CLASSNAME = "MsoCommandBar";
- WORD_WORK_PANE_NAME = "MsoWorkPane";
- WORD_CLIPBOARD _WINDOWNAME = "bosa_sdm_msword";

Spy++ utility provides a graphical view of the system's processes, threads, windows, and window messages. Microsoft Visual Studio 2012, 2013, 2015 and 2017 versions include Spy++ tool as shown in figure 3.2.1.

Figure 3.2.1: Spy++ utility in Microsoft Visual Studio 2017

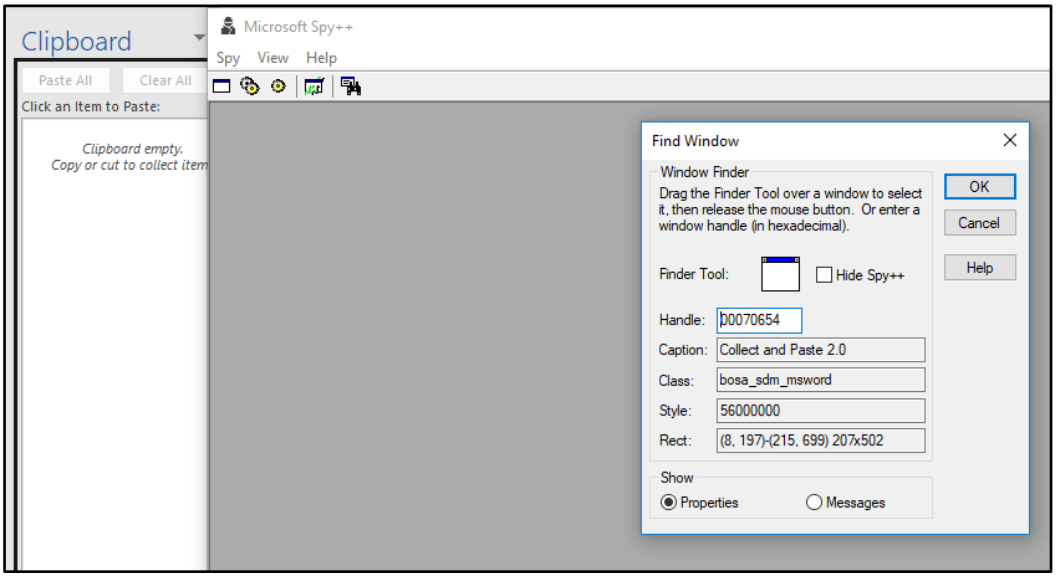


Spy++ tool helps to perform the following functionalities [27]:

- Display a graphical tree of relationships among system objects.
- The tool can be used for the search for specified windows, threads, processes, or messages and view properties of them.
- Select a window, thread, process, or message directly in the view.
- Finder Tool is helpful to select a window by the mouse pointer.
- Set message options by using complex message log selection parameters.

Window Finder is a helpful tool which allows users to find details such as a handle, caption, class, style and window rectangle details. A user can drag the finder tool over a specific window to get window details. Figure 3.2.2 shows the details of the Microsoft Word Clipboard window.

Figure 3.2.2: Details of the Microsoft Word Clipboard window



As shown in the figure 3.2.2, Find Window tool returns the class name of the MS Office Clipboard window when hovering the mouse over the window. The returned class name for the corresponding window is "bosa_sdm_msword". This class name has been used in the proposed solution to access to the MS Office Clipboard window.

3.3 Use-case 3 - External Device Monitor

As explained in section 3.1, an employee can accidentally share the confidential data with external users. In the same section, I explained the proposed solution to protect confidential data in Microsoft Word, Excel and PowerPoint applications. Now the problem is what if an employee copies a confidential file and accidentally pastes it into an external USB (Universal Serial Bus) device? Following subsections explain the requirements in an informal way and as well as the proposed solution to overcome the "confidential data sharing to external devices" situation.

Requirements

Table 3.3.1 illustrates the functional requirements identified for the External Device Monitor application.

Table 3.3.1: The functional requirements of use-case 3

ID	Description
$\mathbb{R}_1$	When starting the system, the system should detect all USB devices already plugged into the computer.
$\mathbb{R}_2$	The system should detect new USB devices when inserting into the computer.

$\mathbb{R}_3$	The system should monitor all the detected USB drivers.
$\mathbb{R}_4$	The system should detect when confidential files are being copied to USB drivers.
$\mathbb{R}_5$	When the system detects that a confidential file is being copied to an external drive, it should show a warning message “Do you want to copy this confidential file?” with Yes/No options.
$\mathbb{R}_6$	If the user selects the option “Yes”, the system will save username, file path, date and time to log.xml file.
$\mathbb{R}_7$	If the user selects the option “No”, the system will delete the file from the external device.

Windows Management Instrumentation

Windows Management Instrumentation (WMI) is a set of specifications for management data and operations on Windows-based operating systems [28]. WMI is the Windows application of Web-Based Enterprise Management (WBEM). WMI facilitates with setting and changing system properties, changing permissions for users and user groups, scheduling processes, write scripts to automate administrative tasks etc. WMI uses the industry standard (Common Information Model (CIM)) to represent systems, applications, networks, devices, and other managed components.

WMI Query Language (WQL)

The WMI Query Language (WQL) is a subset of standard American National Standards Institute Structured Query Language (ANSI SQL). WQL is a way of accessing WMI data in Windows. WMI Tester (wbemtest.exe) is a popular tool installed in Windows executes WQL queries.

Win32_DiskDrive

The Win32_DiskDrive class is a representation of a physical disk drive in a computer system. It provides following important properties about a physical disk drive.

Table 3.3.2: Important properties of Win32_DiskDriver WMI class

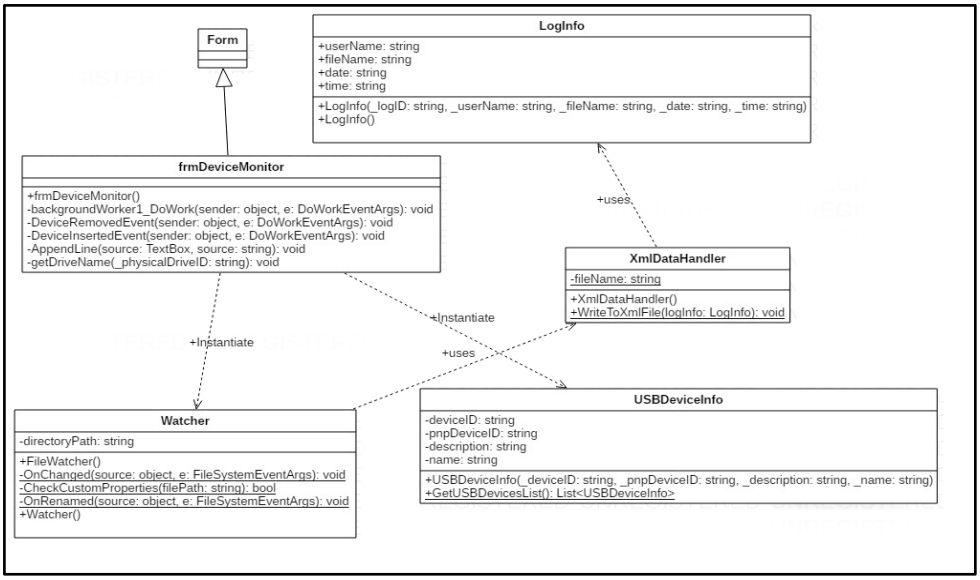
Data Type	Property Name	Data Type	Property Name
uint32	Availability	string	CapabilityDescriptions[]

uint32	BytesPerSector	string	Caption
uint16	Capabilities[]	string	CompressionMethod

InterfaceType and *MediaType* are useful properties to filter out the USB devices from the other physical disk drivers. Possible interface types are SCSI (Small Computer System Interface), HDC (Hard Disk Controller), IDE (Integrated Drive Electronics), USB (Universal Serial Bus) and 1394[10]. Available media types are External hard disk media, Removable media ("Removable media other than floppy"), Fixed hard disk ("Fixed hard disk media"), Unknown ("Unknown drive format"). Following WQL uses the above two properties two finds plugged USB devices in the computer. "Select * from Win32_DiskDrive where InterfaceType='USB' AND MediaType='Removable media'"

Figure 3.3.1 shows the class diagram of the External Device Monitor application. Important functions used for implementation are discussed in the following subsections.

Figure 3.3.1: Class diagram of External Device Monitor application



FileSystemWatcher

FileSystemWatcher is a .NET class which can be used under the *System.IO* namespace to monitor changes in a specified directory or a file inside a directory. This class can be used to listen to the file system change notifications such as a file changed, created,

[10] The IEEE 1394 interface is an electronic standard that is used to connect computers

deleted and renamed. The *Filter* property allows several options when monitoring files. To monitor all files, we can set the *Filter* property to an empty string ("") or can set wildcards ("*.*"). To monitor specific file in the system can set the name directly as the *Filter* property. Example: To monitor the changes in Doc.txt, can set the *Filter* property to "Doc.txt". To monitor all the text files can set the *Filter* property to "*.txt". The drawback of the *FileSystemWatcher* is, it cannot monitor multiple file types at the same time. The solution to that problem is shown in the below code segment.

```
1 private static void OnChanged(object source, FileSystemEventArgs e)
2 {
3 FileInfo fileInfo = new FileInfo(e.FullPath);
4 (fileInfo.Extension.Equals(".docx") || fileInfo.Extension.Equals(".pptx") ||
  fileInfo.Extension.Equals(".xlsx"))
5 {
6 //Continue code
7 } }
```

In the above code segment after the *OnChanged* event is fired, it filters out the unwanted file formats at line number 4. It only accepts Microsoft word documents, PowerPoint files, and Excel files. The system can be further extended by including more file formats at line number 4.

To monitor directory changes in the Windows environment, it is possible to use Windows API functions rather than using the *FileSystemWatcher* class in .NET Framework. Below section explains corresponding functions to be used to monitor a specified directory.

Windows API provides *FindFirstChangeNotificationA* function which allows creating a change notification handler to the given directory and setting up the change notification filters to the object. Table 3.3.3 shows the available filter conditions in the function.

Table 3.3.3: Available filter conditions in Find First Change Notification a function

Filter value	Description
FILE_NOTIFY_CHANGE_FILE_NAME 0x00000001	Any changes happen in the specified directory or subdirectories causes a change notification. Change in a file name includes renaming, creating, or deleting a file name.

FILE_NOTIFY_CHANGE_DIR_NAME 0x00000002	Creation or deletion of the specified directory or subdirectories causes a change notification
FILE_NOTIFY_CHANGE_ATTRIBUTES 0x00000004	Any attribute change in the specified directory or subdirectories causes a change notification.
FILE_NOTIFY_CHANGE_SIZE 0x00000008	Any file size change in the specified directory or subdirectories causes a change notification. The operating system only updates the file size when it is actually written to the disk.
FILE_NOTIFY_CHANGE_LAST_WRITE 0x00000010	Operating system detects last write time when it is actually written to the disk. Any changes to the last write time of the files in the specified directory or subdirectories cause a change notification.
FILE_NOTIFY_CHANGE_SECURITY 0x00000100	Any security descriptor changes in the specified directory or subdirectories cause a change notification.

After a successful invoke of the *FindFirstChangeNotificationA* function, it returns a handle which is an input to the wait functions. The wait function does not return until the specified conditions are met. Below is the list of wait functions available and a suitable "wait function" must be chosen depending on the need [29].

- Single-object Wait Functions
- Multiple-object Wait Functions
- Alertable Wait Functions
- Registered Wait Functions
- Waiting on an Address
- Wait Functions and Time-out Intervals
- Wait Functions and Synchronization Objects
- Wait Functions and Creating Windows

FindNextChangeNotification

If the wait condition is satisfied, the *FindNextChangeNotification* function continuous monitoring the registered directory and its subdirectories. To obtain the changes occurs in the watched directory or subdirectories *ReadDirectoryChangesW* function can be used

with the *FindNextChangeNotification* function. The *Readdirectorychangesw* function does not report changes to the specified directory itself.

3.4 Use-case 4 - Confidential File Printing Detector Application

Section 3.3 described how to handle the copy of a confidential file to an external device. Apart from copying a file to a USB device, employees can print a confidential file without remembering the file contains confidential data. We can stop it by restricting access to the printers. But it is not a good approach since taking printouts is essential in day-to-day work. The following subsection describes the requirements for the use-case informally.

Requirements

Table 3.4.1 shows the functional requirements for managing printers for printing confidential files. Following subsections describe the Windows API functions used for the proposed solution and as well as the working prototype.

Table 3.4.1: Functional requirements for managing printers

ID	Description
$\mathbb{R}_1$	The system should monitor a specified printer.
$\mathbb{R}_2$	The system should pause the printer only if the printing source is confidential.
$\mathbb{R}_3$	When the system detects a confidential file is being printed from the specified printer, the system should show a warning message “Do you want to print this confidential document?” with Yes/No options.
$\mathbb{R}_4$	If the user selects the option “Yes”, the system will prompt a panel to enter the password.
$\mathbb{R}_5$	If the password is correct, the system will resume the print job with current parameters.
$\mathbb{R}_6$	If the password is correct, the system will clear the password textbox and hide the password panel.
$\mathbb{R}_7$	If the password is incorrect, the system will delete the print job from the print queue.
$\mathbb{R}_8$	If the user selects the option “No”, the system will delete the print job from the print queue.
$\mathbb{R}_9$	The system should freeze the printer queue window when the system detects a confidential file is being printed from the specified printer.

$\mathbb{R}_{10}$	The system should unfreeze the printer queue window when the user successfully entered the password.
$\mathbb{R}_{11}$	The system should unfreeze the print system tray when the system successfully deletes the file from the print system tray.

The proposed solution addresses all requirements mentioned in the table 3.4.1, and following subsections describe the used Windows API functions and data structures.

FindFirstPrinterChangeNotification

This Windows API function creates an object which provides a change notification when one of the specified changes occurs in the specified printer or print server. This function returns a handle to the object which can later be used with a wait operation. Wait operations are described in section 3.3. When the wait operation succeeded, we can use *FindNextPrinterChangeNotification* function to retrieve information about the change notification. *FindFirstPrinterChangeNotification* function accepts four parameters namely: *hPrinter*, *fdwFilter*, *fdwOptions* and *pPrinterNotifyOptions*. All the parameters are explained below.

- *hPrinter* - This is the handle to the printer that we want to monitor. A handle can retrieve for a specific printer or a print server by calling *OpenPrinter* [30] or *AddPrinter* functions [30].

- *fdwFilter* - This parameter specifies the set of conditions to be true to return the wait function. This change notification occurs when one or more specified conditions are met. This parameter can be set to 0 when *pPrinterNotifyOptions* parameter is not null. Table 3.4.2 shows the possible values for the *fdwFilter* parameter. The full table can be found in the appendix section. Proposed solution only interested in newly added print jobs. So, this flag was set to PRINTER_CHANGE_ADD_JOB = 0x100

Table 3.4.2: Available bit flags for change notification object

Value	Specific flags
PRINTER_CHANGE_FORM	PRINTER_CHANGE_ADD_FORM = 0x10000 PRINTER_CHANGE_SET_FORM = 0x20000 PRINTER_CHANGE_DELETE_FORM = 0x40000
PRINTER_CHANGE_JOB	PRINTER_CHANGE_ADD_JOB = 0x100 PRINTER_CHANGE_SET_JOB = 0x200 PRINTER_CHANGE_DELETE_JOB = 0x400 PRINTER_CHANGE_WRITE_JOB = 0x800
PRINTER_CHANGE_PORT	PRINTER_CHANGE_ADD_PORT = 0x100000 PRINTER_CHANGE_CONFIGURE_PORT = 0x200000 PRINTER_CHANGE_DELETE_PORT = 0x400000

- *fdwOptions* - This determines the category of printers: both 2D and 3D printers or only for 3D printers. This can be set to zero if we are only interested in 2D printers.
- *pPrinterNotifyOptions* - This holds a pointer to PRINTER_NOTIFY_OPTIONS structure. It specifies the set of printer or job information fields to monitor for change.

FindNextPrinterChangeNotification

This function retrieves information about changes that had occured with the changed notification object for a specified printer or printer server. The change notification object was created by the *FindFirstPrinterChangeNotification* function with the set of changes to be monitored. All the parameters required for the function are explained below.

- *hChange* - Handle obtains by calling the *FindFirstPrinterChangeNotification* function.

- *pdwChange* - This parameter indicates the changes that occurred in the print notification object to cause the most recent notification. Can set one or more of the following available bit flags. These possible bit flags are described in the table 3.4.2.
- *pPrinterNotifyOptions* - This parameter requires a pointer to PRINTER_NOTIFY_OPTIONS structure. By passing the structure with the *FindNextPrinterChangeNotification* function, it retrieves data for current monitored job information fields. This parameter is optional and can set to NULL.
- *ppPrinterNotifyInfo* - This parameter is also optional and can set to NULL when no information is required. This is a pointer to a pointer variable which is needed to retrieve PRINTER_NOTIFY_INFO structure.

Figure 3.4.1: Class diagram of confidential file printing detector application

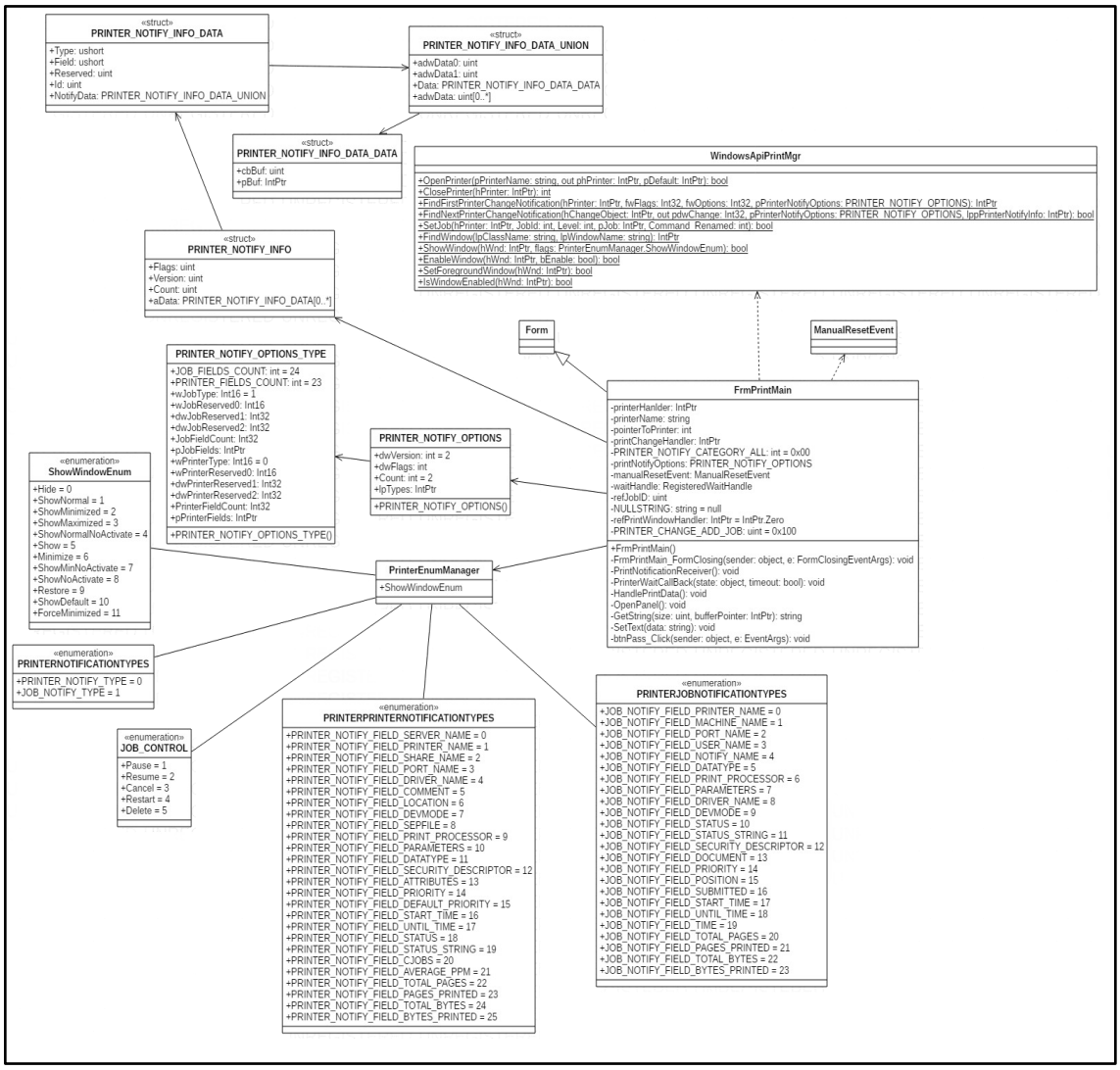
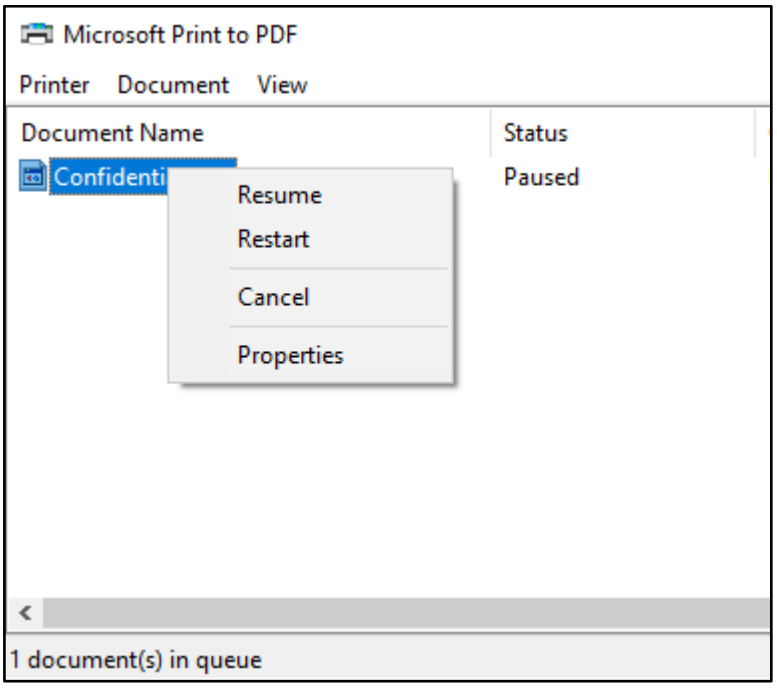


Figure 3.4.1 shows the class diagram with all the structures and enums used for development. The application pauses the detected print job in order to authenticate the user. After successfully pausing the print job, the user will be requested to provide credentials to proceed further. Even though the application pauses the current print job for authentication, the user can resume the print job using the printer queue window. As illustrates in the figure 3.4.2 printer queue window provides options for users to resume or restart the print job. This window must be inaccessible from the user until the authentication process finished.

Figure 3.4.2: Printer queue windows



By using the *FindWindow* function which was explained in section 3.2, we can get a handle to the window shown in the figure 3.4.2. The class name associated with the printer queue window is *PrintUI_PrinterQueue*. Window title name can be different depending on the printer. Below are the steps performed to disable the printer queue window.

- Get the handle to the printer queue window by using the *FindWindow* function.
- Restore the window from the system tray by using the *ShowWindow* function. *ShowWindow* function accepts visibility options such as Hide, ShowNormal, ShowMinimized, ShowMaximized, Maximize, ShowNormalNoActivate, Show,

Minimize, ShowMinNoActivate, ShowNoActivate, Restore, ShowDefault and ForceMinimized.

- Get the Window to the foreground and make it active.
- Disable the Window by using *EnableWindow* function [30].

3.5 Use-case 5 - Handling confidential file in DDE message passing

Applications can use different techniques to transfer data between each other. As explained in section 2.1.1.2 the Dynamic Data Exchange protocol is an interprocess communication method which allows applications to exchange data between them. The advantage of DDE is, once the links between the two applications are successfully created, the data can be passed between those two applications without any interaction of the user. Imagine a situation where two Excel files have been linked via DDE and passing data. The question arises when we mark the server application as a confidential file. How to notify the user about the DDE link which connects the server application with other applications and makes sure the data passing is secured at the client end? The following algorithm illustrates how to notify the user regarding existing DDE connections in a confidential Excel file.

Algorithm 3.5.1: Notify the user regarding existing DDE connections

```
1.  A ← get existing Excel connections
    //Workbook.LinkSources(Type)
2.  if A ≠ null then
3.      for A = 0 to A.Length – 1
4.          //DDE links use the "Server|Document!Item" syntax
5.          Filter only DDE links
6.          Notify user regarding existing DDE linked document names
7.          Inform user to mark the linked document as a confidential document
8.          Re-establish the DDE link //Workbook.ChangeLink(String,
    String, XlLinkType)
9.      end
10. end
```

As shown in the algorithm 3.5.1 the system can notify the users regarding the existing DDE links. This functionality can be implemented as an add-in for an Excel application.

The algorithm 3.5.1 ensures a confidential Excel file will not take part of any DDE connection with a non-confidential file.

4 TESTING

The following chapter explains the testing methods used to evaluate the quality of the prototype applications built. Only the significant functions of the use-case 1 application are manually tested due to the time constraints. Furthermore, this chapter explains a test approach to be performed as future work.

4.1 Unit Testing

Black-box testing also known as Behavioral Testing is a software testing method typically involves in running through every possible input to verify that it results in the right outputs using the software as an end-user would. Mainly Unit testing has been conducted as a black-box testing method while developing the prototype applications. Manual test cases have been written and tested for main functions in the prototype applications. This ensures the important individual units of software are tested. Following steps need to be carried out for Unit testing.

- Examine the initial requirements of the system.
- Determine valid inputs to check whether the system under test processes them correctly. Similarly, some invalid inputs must be used to check whether the system generates expected error messages.
- Write test cases with the selected inputs.
- The test cases are executed.
- Compares the actual outputs with the expected outputs.
- Defects if any are fixed and re-tested.

Below is the list of test cases used to test the use-case 1 prototype application developed.

Table 4.1.1:Example test cases used to test the use-case 1 prototype application

TC ID	Testcase Description	Expected Result	Test Data	Actual Result	Status (Pass/ Fail)
1	Verify that the collection of information copied to the system and source is confidential	The system should be only allowed to collect information if the copied source is confidential	A confidential Word file (Test_confidential.docx)	The system should add the copied item to the list	Pass
2	Verify that system should be able to store the copied source of the content	The system should be able to store the copied source of the content	Copy text from a confidential Word file (Test_confidential.docx)	The system should add the copied content into the list under the “View Data” column	Pass
3	Verify that confidential file name is matched and view in File Name field	The confidential file name should be displayed in File Name field	Copy text from a confidential Word file (Test_confidential.docx)	The system should add the “Test_confidential.docx” title into the list under the “File Name” column	Pass
4	Verify that system should be able to store the copied Date of the context	If it is a confidential file, the copied date should be shown in the Clipboard history window	Copy text from a confidential Word file (Test_confidential.docx)	The system should add the copied Date into the list under the “Date” column	Pass
5	Verify that system should be able to store the copied time of the content	If it is a confidential file, copied time should be shown in Clipboard history window	Copy text from a confidential Word file (Test_confidential.docx)	The system should add the copied time into the list under the “File Name” column	Pass
6	Verify that system should be able to store username of the current window user	If it is a confidential file, username should be shown in Clipboard history window	Copy text from a confidential Word file (Test_confidential.docx)	The system should add the current user’s name into the list under the “File Name” column	Pass
7	Verify that non-confidential files content is not copied to the Clipboard history	If the document is not confidential, then the file should not be copied to the Clipboard history	Copy text from a non-confidential Word file (Test.docx)	The system should not add the copied content information to the list	Pass

Table 4.1.1 shows the example test cases developed to test the use-case 1 application. All the other functions are tested while developing the application. These test cases ensure the main functions of the application are working according to the requirements.

Below is the list of requirements identified as important requirements of the prototype applications developed in the thesis and not tested manually writing test-cases due to time constraints.

- Verify that the system only searches for confidential data in Word, Excel and PowerPoint files only if the file is a non-confidential file (Applicable to add-ins developed for MS Word, Excel and PowerPoint)
- Verify that the system searches for confidential data in Word, Excel and PowerPoint files when the user triggers the save event (Applicable to add-ins developed for MS Word, Excel and PowerPoint)
- Verify that the system searches for confidential data in Word, Excel and PowerPoint files when the user wants to close the file (Applicable to add-ins developed for MS Word, Excel and PowerPoint)
- Verify that the system successfully saves the file by modifying the title and by adding the custom property tag (Applicable to add-ins developed for MS Word, Excel and PowerPoint)
- Verify that the system detects already inserted USB devices when the system starts (Applicable to External Device Monitor application)
- Verify that the system detects newly inserted USB devices while the application is running (Applicable to External Device Monitor application)
- Verify that the system adds a record in the log file for all copied confidential files (Applicable to External Device Monitor application)
- Verify that the system successfully deletes the file if the user no longer needs to copy the file to a USB device (Applicable to External Device Monitor application)
- Verify that the system only detects when a confidential file is being printed by the specified printer (Applicable to Confidential File Printing Detector Application)
- Verify that the user entered the correct password (Applicable to Confidential File Printing Detector Application)

5 EVALUATION

In this chapter, concepts and implementation are evaluated for the following aspects: the compliance with the requirements, the security of the system and the ethical use of Windows API and 3rd party applications.

The main objective of this thesis is to prevent accidental data leaks in Microsoft Office applications. As described in the Implementation section, several applications have been developed and tested to meet the requirements. For the use-case 1: "Detect pasted confidential data in MS Office applications" three add-ins were implemented to monitor the MS Word, MS Excel, and MS PowerPoint applications. Main objectives of this task were to find out pasted confidential data within the application. The open issue could be that the user might change the pasted content by adding some additional content. Then the application may not be able to detect the pasted content, and this situation could occur depending on the changes, which were applied to the content. The problem has not been addressed in the thesis due to the time constraints and limitation of the default search functionality provided by the Microsoft Interop library. A possible solution to overcome this problem is to use a string-matching algorithm to match a portion of a given string.

The proposed solution for the use-case 1 ensures that the user is notified when a confidential text is found in MS application. However, a user can fool or circumvent the system to gain some benefits by taking confidential information outside the organization. Microsoft Office applications provide options to disable or remove add-ins. This situation cannot be overcome because this functionality cannot be disabled, and it is originally provided by the MS Office Applications. Users can circumvent the provided add-in functionality by removing the string "confidential" from the title of the file and by removing the custom property tag added by the add-in. This leads to a situation where the file contains confidential data, but it was not marked as a confidential file. These security issues are beyond the control of the system, intentional, and cannot be prevented.

As described in section 4.1, as per the requirement $\mathbb{R}_{17,}$ the provided add-ins start searching the document for confidential data when the user triggers the save operation. The user can circumvent the functionality provided by the use-case 4 described under the section 4.3. The user can print the document with confidential data directly without

performing save operation. In this situation, the provided solution fails to detect the document. A solution for this issue is to extend the add-ins to search the document for confidential data not only for saving operation, but also when the print operation is performed by the user.

MS Word, Excel and PowerPoint application add-ins failed to save the file as a confidential file when the initial "save as" operation performs by the user. The reason for this is the following: in the initial save the directory name of the file is empty. Thus, the add-ins fail to generate the file path which is essential in the save operation. Also, the scanning event happens before the user performs the save operation. So, at the initial save the user can change the file name and save it in a different location even after the application detects confidential content. This problem can be solved by saving the file into a known directory in the initial save operation or by creating a custom event to save the document again after the initial save operation is completed by the user. I would like to suggest this workaround as future work, and this was not addressed in the implementation due to time constraints.

The solution provided in section 4.2 to clear the MS Office Clipboard is an unsatisfactory solution. The issue with this solution is: it seeks for the MS Word Clipboard windows by their class names, and the following assumptions has been made during the developments:

Assumption 1: MS Word has been successfully installed in the system;

Assumption 2: The class names used to navigate to the MS Word Clipboard are the same for all the released versions of MS Word;

At least one of those mentioned assumptions were failed, the proposed solution will not be able to clear the MS Word Office Clipboard items. Since MS Word, Excel and PowerPoint applications share the same storage to store the Office Clipboard data, it is enough to clear the content from one of those applications.

The Office Clipboard can hold up to 24 items. If you copy the 25^{th} item, the first item on the Office Clipboard is deleted. The proposed solution in use-case 1 also holds up to 24 items. When it reaches the defined maximum number of items, it clears the local application history and as well as the MS Office Clipboard history. But, due to unavailability of an API or documentation to interact with the MS Office Clipboard, the

proposed system fails to clear the collected items of “Clipboard History” application when the user manually clears the Office Clipboard.

As described in section 4.3, external device monitor application provides an extra confirmation when copying a confidential file to a USB device. This application can track a confidential file being copied to a USB device by cut and paste, copy and paste, drag and drop or send to option available under the windows shell context menu. As per requirement $\mathbb{R}_7$ in external device monitoring application, the file will be deleted if the user does not want to keep the copied file in the USB device. A possible problem due to this requirement is, the file can be lost if the user performed the cut operation and then paste it to the USB device. A possible solution to overcome this issue is to keep a backup of the copied file in the memory and restore it to a known location like “Document” folder. Before deleting the file, the user must be notified regarding the new location of the file. This functionality is not implemented due to time constraints and would like to suggest as future work.

The user can circumvent the provided functionality by removing the string “confidential” from the title of the file and removing the custom property tag. This is a deliberate attempt to take confidential data outside the premises and cannot be prevented by the provided solution. Due to the limitations provided by the technology used for the implementation of the external device monitoring application, the notification is generated only after it is being copied to the USB device. Even at this point, the system detects the file copied is confidential; the user can unplug the USB device to circumvent the process.

The solution provided to detect printing confidential documents application only supports monitoring a given printer. Normally, an organization has more than one printer at operation. So, the proposed solution having lack of functionality to monitor all available printers and employees can use another printer other than the specified printer in the application and print a confidential document. This functionality was not implemented due to the time constraints and the constraints of the used technique to detect the printer queue, the proposed solution unable to get the source of the printing file. Otherwise, the system could redirect the detected file to a specific printer which can be authenticated with the smart card, instead of requesting the user to authenticate it using a password.

5.1 Ethical usage of Windows API and third-party applications

Windows API enables developers to explore the power of the Windows operating system. By using Windows API functionalities, developers can build applications which interact with low-level features of the operating system. Windows API provides functionalities that non-ethical developers could use for their own goals. Some of those functionalities cannot be accessed by the already available frameworks such as .NET framework. For example, *EnableWindow* function in Windows API enables or disables mouse and keyboard input for a specific window. This function can be used even to disable the mouse and keyboard interaction to the *Taskbar* window which has the class name "MSTaskListWClass". This scenario of disabling the Task Manager window can be considered as a non-ethical use of the *EnableWindow* function. Because *Taskbar* is an essential window in the Windows operating system to quickly access to applications when the user is on the desktop.

As explained in section 4.2 under the "Clear Office Clipboard" topic, the proposed method described a mechanism to access the Microsoft Word Clipboard window and then invoke the "Clear All" button to delete all the entries from the Office Clipboard. This method can be applied to currently active Word application. Invoking the "Clear All" button happens in the background without any acknowledgement of the user. The dangerous side of this is that a window can be accessed via its class name and, we can invoke any button in the background on that window if the location of the button is known.

Microsoft.Office.Interop.Excel.dll, *Microsoft.Office.Interop.Word.dll*, and *Microsoft.Office.Interop.PowerPoint.dll* interop assemblies enable third-party applications to access the active Excel, Word and PowerPoint applications respectively. Developers can use these assemblies to interact with the above applications and modify the content of the active document. Example non-ethical use of this as follows. Imagine a word document is opened by a user and working on it. An application running on Visual Studio environment can get a reference to the active document. By iterating each paragraph, the application can copy the entire content of the active Word document to the program and use it for their own purposes.

5.2 Performance Evaluation

Considering the performance of the developed prototypes, the execution time of the critical functions and memory usage by the applications must be evaluated. Identified bottlenecks during the performance evaluation are discussed in subsequence sections.

As the Windows operating system allows applications to run in the background as services, they could influence the analysis by background computations and additional memory consumption. Therefore, all the unnecessary applications were closed while measuring the performance of the prototype applications. All measurements are performed on an HP Pavilion g6 with an Intel Core i5- 3210M 2.50GHz, 4GB main memory laptop running Windows 10, 64-bit operating system.

Performance of the prototype application developed for use-case 1 is evaluated by analyzing the execution time of the *WndProc* method, mean time taken to start the application with and without object reference to the COM components and average memory usage by the application.

The critical or the main method of the proposed solution is the *WndProc* method. This method is responsible for listening to the operating system messages and notifying the application when the *WM_DRAWCLIPBOARD* message occurs in the system. Apart from notifying the application regarding the change happens in the system Clipboard, this *WndProc* method also responsible for following sub-tasks:

- Retrieve Clipboard owner details
- Retrieve source name of the copied content
- Filter out non-confidential files and files other than MS Word, MS Excel, and MS PowerPoint applications
- Search for duplicate entries with the locally stored copy of Clipboard data
- Add an entry to the UI and save the newly copied content locally

Since the *WndProc* method carries out major tasks of the application, it is important to evaluate the performance of the method. Table 5.2.1 shows the running time of the *WndProc* method concerning the number of pages copied to the system Clipboard. The data is collected by increasing number of pages gradually to check whether there is any correlation between a number of copied pages vs execution time of the *WndProc* method. A word document (Test.docx) has been used with 2,300 pages generated using

=lorem(m,n) command in Microsoft Word 2016. Parameter *m* indicates the number of paragraphs and parameter *n* indicates a number of sentences in each paragraph. Table 5.2.1 contains the data collected for measuring the execution time of the *WndProc* method. This generated *Test.docx* document has been used for all the following performance evaluation attempts.

Table 5.2.1: Execution time of WndProc method

Number of pages copied	A number of words copied (Approx.)	The execution time of the *WndProc* method (milliseconds)
1	629	54
5	3218	306
10	6435	438
50	31980	342
100	63909	1415
250	159821	1221
500	320284	1454
750	479886	2513
1000	639033	3859
1500	959306	7322

The average execution time to copy 500 pages *WndProc* method takes 2271.24 milliseconds. The data has been plotted in a scatter plot graph as below. X-axis contains the number of pages copied, and the Y-axis contains the execution time for the *WndProc* method in milliseconds.

Figure 5.2.1: WndProc execution time respect to the number of pages copied

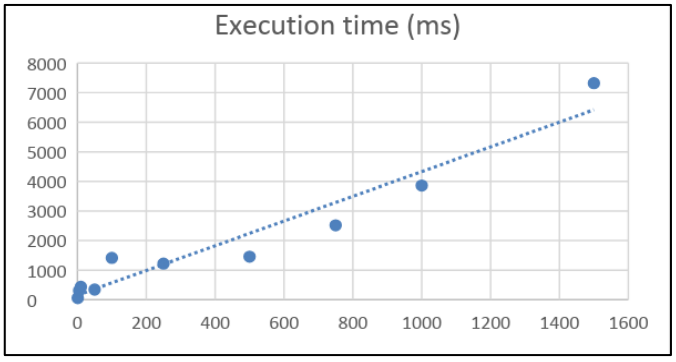


Figure 5.2.1 shows that there is a positive correlation between variables X and Y in the graph. This indicates the *WndProc* method execution time increases with the number of copied pages to the system Clipboard. From the figure 5.2.1, we can say that Execution time is directly proportional to the number of pages copied from the document.

The second performance metric to be evaluated for the use-case 1 is the mean time to start the application UI. Before starting the application UI, it is responsible for the following subtasks to be performed:

- Initialize the components
- Read the locally stored Clipboard history data and load them to the application UI
- Register the application in the Clipboard chain
- Get COM object reference to invoke Word add-in
- Get COM object reference to invoke Excel add-in
- Get COM object reference to invoke PowerPoint add-in

The goal of this evaluation is to improve the starting time of the application by using alternative methods to share data between the application and add-ins created to detect pasted content on MS Word, Excel and PowerPoint applications. Currently, the application takes few seconds to load the UI, and table 5.2.2 shows the application starting time with getting COM object references and without getting COM object references.

Table 5.2.2: Application starting time with COM object references and without COM object references

Application start time (milliseconds) with COM object reference	Application start time (milliseconds) without COM object reference
15506	143
9319	118
9191	169
9333	147
14367	147
9810	161
9558	191
9771	180

9503	139
9486	188

The mean starting time with getting COM object references is 10,584.4 milliseconds. The mean of the second column of the table 5.2.2 is 185.3 milliseconds. Thus, the result clearly shows the application starting time was delayed due to the COM object references at the startup. To reduce the loading time of the application following alternatives can be suggested as future work.

- Use threads to get COM object references instead of invoking them from the main thread
- Use shared DLL file to store the copied content from the Clipboard
- Use a JSON or XML file in a known location to share copied data between the application and add-ins.

The maximum amount of physical memory usage by the application was monitored to analyze the performance. The following data was collected when the application was in idle state, and no any explicit operation was performed after the application was loaded into the memory. These data were collected in debugging mode of the project. Thus, actual performance data can be varied from the performance data listed below.

$$X = 29, 27, 29, 28, 30, 28, 29, 27, 28, 28$$

The average memory usage of the application denoted by $X = \frac{\sum_{i=1}^{n} X}{n}$, where n is the number of samples, X is memory usage in Megabytes for each sample.

$$\mu = \frac{\sum_{i=1}^{n} X}{n} = 28.3\ MB$$

The above analysis shows the maximum memory usage of the use-case 1 application is 28.3 MB. The application only uses 0.708% of the physical memory from the laptop which used to test the application.

Add-in applications developed for searching confidential data has been evaluated by the mean execution time and CPU usage of the search method. Due to time constraints, only the word add-in search functionality was evaluated." For testing the performance, *Test.docx* Word document was used, and the following text has been used as an input to

the search operation. "*Video provides a powerful way to help you prove your point. When you click Online Video, you can paste in the embed code for the video you want to add. You can also type a keyword to search online for the video that best fits your document.*" The input text's character length is 240. This is important because the maximum length of a string for finding and replace function is 255 characters. For the strings, more than the maximum length must be separated into 255-character chunks and execute each using the search function. So, the used input ensures that the search operation scans the Word document only once. The generated Word document only contains text, and no additional styles have been added to the document. Below is the data collected for measuring the execution time (milliseconds) of the find and replace method which starts searching from the first page of the document.

$$Xt = 551{,}617{,}685{,}672{,}621{,}585{,}595{,}597{,}607{,}655$$

The mean loading time of the application denoted by $Xt = \frac{\sum_{i=1}^{n} Xt}{n}$, where n is the number of samples, Xt is the time taken to execute search the function in milliseconds.

$$\mu = \frac{\sum_{i=1}^{n} Xt}{n} = 618.5\ miliseconds$$

The mean execution time in the search operation is 618.5 milliseconds. In other words, the proposed solution takes approximately 0.6 seconds to scan the document to find a matching confidential data.

As per the requirements $\mathbb{R}_2$ and $\mathbb{R}_4$ in use-case 3, the system should be able to detect newly inserted USB devices and monitor the detected drivers all the time. Thus, the application built for monitoring the confidential files being copied to external devices needs to run from the startup. Below the list of data shows the maximum memory (Megabytes) usage by the application during each trial.

$$X = 23{,}20{,}23{,}22{,}20{,}20{,}22{,}20{,}23{,}20$$

The average memory usage of the application denoted by $X = \frac{\sum_{i=1}^{n} X}{n}$, where n is the number of samples, X is memory usage in Megabytes for each sample.

$$\mu = \frac{\sum_{i=1}^{n} X}{n} = 21.3\ M$$

The evaluation chapter showed the open issues related to the prototype applications. Also, the suggestions to overcome the current security issues were discussed and this information bears potential for future development. The mean execution time of the copy function of the use-case 1 application and search function in the add-in developed for MS Word indicates the speed of the systems developed. The maximum average memory consumption was 28.3 MB and 21.3 MB for the applications “Detect pasted confidential data in MS Office applications” and “External Device Monitor” respectively. These performance results show the applications can run under a low memory system and does not require a high capacity of physical memory to process the application.

6 CONCLUSION

This master thesis focused on data usage control in the Windows operating system environment. Accidental data leaks caused by unintentional human actions. This became a major cause of sensitive information loss. The possible solutions discussed in section 1.3 limits access to the data for employees. The main question addressed by the thesis was, “How to protect confidential data while enabling full access to the data for employees?”.

Windows API provides an interface to developers to explore the features and capabilities of the Windows operating system. Related work section discussed the Windows API functionalities in detail to give an insight about the API features used in the development. The existing functionalities provided by the .NET framework to interact with the system Clipboard was not sufficient. Therefore, the Clipboard Windows API has been used for development of prototype applications.

The use-cases were defined by analyzing the possible scenarios where accidental data leaks could occur. The defined use-cases provide an application-level additional security layer for protecting confidential data. The provided solution for protecting confidential data in MS Office applications contain two parts. Since the existing technologies were insufficient to fulfil the use-case 1 requirements, the thesis proposed a unique solution to notify the users regarding confidential data in MS Word, Excel or PowerPoint applications.

The proposed solution for clearing the Office Clipboard does the intended work under the assumption made during the implementation. These assumptions were discussed in the evaluation chapter. The key part of the implementation is invoking the “clear all” button from the third-party application. This solution indicates how a third-party application can interact with an MS Word document without any acknowledgement of the user. This gives developers to think about the ethical usage of Windows API functions. Also, this gives users to re-think about using unknown third-party applications. Windows API is powerful; hence the provided functionalities must be used ethically.

Wrapper APIs enable developers to use the Windows API functions without engaging directly with unmanaged functions. Managed Windows API and EasyHook APIs were

discussed in sections 2.2.1 and 2.2.2 respectively. Compared to Managed Windows API, EasyHook wrapper API contains better documentation with sample codes for developers. EasyHook wrapper API is mainly focusing on low-level hook procedures while Managed Windows API supports different types of components developed using C# language.

The implemented solutions were evaluated under the aspects of compliance with requirements, security, and performance. The developed systems can be considered as secure, under the assumptions were made during the evaluation. Performance evaluation indicates that the proposed solutions memory usage and execution time of the critical functions. Thus, the proposed solutions do not require the high capacity of memory to run the applications.

This is, in conclusion, the thesis contributed valuable information related to Windows API. Essential functions used during the implementation were discussed in detail. The information provided will be beneficial for developers and researchers to get an idea beforehand doing further research on the topic. Thesis addressed all the defined use-cases successfully, and all the implemented prototypes are tested in a real environment. In summary, the contributed solutions complied with all introduced requirements and achieved the desired results by answering the main research question.

7 References

[1] "SENSITIVE DATA," Information Technology Services, [Online]. Available: https://its.unc.edu/security/sensitive-data/ . [Accessed 3 September 2018].

[2] R. Kuhn and S. Liu, "Data Loss Prevention," *IT Professional,* vol. 12, no. 2, pp. 10-13, 2010.

[3] "Operating System Market Share Worldwide - August 2018," StatCount, [Online]. Available: http://gs.statcounter.com/os-market-share. [Accessed 01 September 2018].

[4] B. Rossi, "Accidental data leaks by staff now a primary security weak point," 29 October 2014. [Online]. Available: http://www.information-age.com/accidental-data-leaks-staff-now-primary-security-weak-point-123458600. [Accessed 04 June 2018].

[5] D. Jodarski, "The Accidental Insider, Mitigating Unintentional Data Leaks," [Online]. Available: https://www.bigskyassociates.com/blog/the-accidental-insider-mitigating-unintentional-data-leaks. [Accessed 4 June 2018].

[6] S. Howard and D. Vangel, "Overview of data loss prevention policies," Microsoft, 29 06 2018. [Online]. Available: https://support.office.com/en-us/article/overview-of-data-loss-prevention-policies-1966b2a7-d1e2-4d92-ab61-42efbb137f5e. [Accessed 20 07 2018].

[7] "Clipboard Class," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/system.windows.forms.clipboard(v=vs.110).aspx. [Accessed 22 May 2018].

[8] "Clipboard Reference," Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/windows/desktop/dataxchg/clipboard-reference. [Accessed 25 May 2018].

[9] "About the Clipboard," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/enus/library/windows/desktop/ms649012(v=vs.85).aspx. [Accessed 17 April 2018].

[10] "Clipboard Formats," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/ms649013(v=vs.85).aspx. [Accessed 18 April 2018].

[11] "About Atom Tables," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/ms649053(v=vs.85).aspx. [Accessed 14 May 2018].

[12] "Data Exchange," Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/windows/desktop/api/_dataxchg/. [Accessed 27 May 2018].

[13] J. Gerend, J. Flores, L. Poggemeyer, W. Gries and J. Tobin, "DFS Namespaces overview," Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/windows-server/storage/dfs-namespaces/dfs-overview. [Accessed 4 July 2018].

[14] "About Authentication," Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/windows/desktop/SecAuthN/about-authentication. [Accessed 22 April 2018].

[15] "About Authorization," Microsoft, 31 May 2018. [Online]. Available: https://docs.microsoft.com/en-us/windows/desktop/SecAuthZ/about-authorization. [Accessed 2 July 2018].

[16] "An Overview of Managed/Unmanaged Code Interoperability," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/ms973872.aspx#manunman_clr. [Accessed 11 June 2018].

[17] "Platform Invoke Data Types," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/ac7ay120(v=vs.100).aspx. [Accessed 20 July 2018].

[18] "Blittable and Non-Blittable Types," Microsoft, 30 March 2010. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/framework/interop/blittable-and-non-blittable-types. [Accessed 20 July 2018].

[19] "EasyHook," [Online]. Available: https://easyhook.github.io/. [Accessed 6 June 2018].

[20] "Consuming Unmanaged DLL Functions," Microsoft, 30 March 2017. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/framework/interop/consuming-unmanaged-dll-functions. [Accessed 10 July 2018].

[21] "WM_PASTE message," Microsoft, 31 May 2018. [Online]. Available: https://docs.microsoft.com/en-us/windows/desktop/dataxchg/wm-paste. [Accessed 15 July 2018].

[22] S. Cheusheva , "Excel Paste Special: copy values, comments, column's width and more," 22 June 2018. [Online]. Available: https://www.ablebits.com/office-addins-blog/2016/10/20/excel-paste-special-shortcuts-features. [Accessed 15 July 2018].

[23] "Hooks Overview," Microsoft, 31 May 2018. [Online]. Available: https://docs.microsoft.com/en-us/windows/desktop/winmsg/about-hooks. [Accessed 16 July 2018].

[24] "Office Clipboard API," 1 January 2010. [Online]. Available: http://www.benf.org/excel/officeclip. [Accessed 12 June 2018].

[25] A. Bhatia, "Clear Office Clipboard Programmatically," 1 March 2014. [Online]. Available: https://ankushbhatia.wordpress.com/2011/03/01/clear-office-clipboard-programmatically/. [Accessed 4 May 2018].

[26] "Window class names used in Microsoft Office," [Online]. Available: http://users.skynet.be/am044448/Programmeren/VBA/vba_class_names.htm. [Accessed 22 July 2018].

[27] "Introducing Spy++," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/dd460756.aspx. [Accessed 10 May 2018].

[28] "Windows Management Instrumentation," Microsoft, 31 May 2010. [Online]. Available: https://docs.microsoft.com/en-us/windows/desktop/wmisdk/wmi-start-page. [Accessed 2 August 2018].

[29] "Wait Functions," Microsoft, 31 May 2010. [Online]. Available: https://docs.microsoft.com/en-us/windows/desktop/Sync/wait-functions. [Accessed 6 August 2018].

[30] "Print Spooler API Reference," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/dd162863(v=vs.85).aspx. [Accessed 7 August 2018].

[31] "Microsoft Disables DDE Feature in Word to Prevent Further Malware Attacks," 15 December 2017. [Online]. Available: https://www.bleepingcomputer.com/news/microsoft/microsoft-disables-dde-feature-in-word-to-prevent-further-malware-attacks/. [Accessed 8 May 2018].

[32] "Remote Desktop Protocol," Microsoft, 31 May 2010. [Online]. Available: https://docs.microsoft.com/en-us/windows/desktop/termserv/remote-desktop-protocol. [Accessed 5 August 2018].

Appendix

A. Clipboard class

The following sections describe the available functions in .NET Clipboard class and available formats.

A.1 .NET Clipboard class

Microsoft has introduced the .NET Clipboard class with the .NET framework 3.0. This class provides a set of static methods to interact with the system Clipboard. For security purposes, the paste operation needs to be user initiated. Table A.1 shows the full list of available functions in .NET Clipboard Class [4].

Table A.1: Available functions in the .NET Clipboard class

Available functions in the Clipboard class	Description
Clear()	Empty the Clipboard
ContainsAudio()	Returns true if the Clipboard contains a audio file in WaveAudio format
ContainsData(String)	Returns true if the Clipboard contains data with the specified Clipboard data format
ContainsFileDropList()	Checks whether the Clipboard holds data in the format of FileDrop or it can be converted into a FileDrop format
ContainsImage()	Checks whether the Clipboard holds data in the format of Bitmap or it can be converted into that format
ContainsText()	Returns true if the Clipboard contains data in Text or Unicode text format. This depends on the operating system
ContainsText(TextDataFormat)	Indicates whether there is a text data object in the specified textDataFormat value
GetAudioStream()	Gets an audio stream from the Clipboard data

GetData(String)	Get Clipboard data in the specified format
GetDataObject()	Get the currently available data from the Clipboard
GetFileDropList()	Gets the collection of file names
GetImage()	Gets an image from the Clipboard
GetText()	Gets a text in the format of Text or UnicodeText from the Clipboard. This depends on the operating system
GetText(TextDataFormat)	Get text data from the Clipboard in the specified TextDataFormat value
SetAudio(Byte[])	Empty the Clipboard. Sets a byte array in a format of WaveAudio.
SetAudio(Stream)	Empty the Clipboard and sets a stream in a WaveAudio format
SetData(String, Object)	Empty the Clipboard and sets data to the Clipboard in the specified Clipboard format
SetDataObject(Object)	Empty the Clipboard and sets non-persistent data on the Clipboard
SetDataObject(Object, Boolean)	Empty the Clipboard and sets data on it. The boolean value indicates whether the data should remain on the Clipboard after the application closed
SetDataObject(Object, Boolean, Int32, Int32)	Sets the data on the Clipboard. Can set the number of attempts and the delay between the attempts. Also, can indicates whether the data should remain on the Clipboard after the application closed. Useful when the Clipboard is occupied by another application or a thread
SetFileDropList(StringCollection)	Empty the Clipboard and places the collection of file names in the FileDrop

	format
SetImage(Image)	Empty the Clipboard and add an image in the format of Bitmap
SetText(String)	Empty the Clipboard and places a text in the format of Text or UnicodeText on the Clipboard. This depends on the operating system
SetText(String, TextDataFormat)	Empty the Clipboard and places a text in the specified TextDataFormat value

A.2 Standard Clipboard Formats

Clipboard supports a set of predefined Clipboard formats that can be used to identify the data in Clipboard operations. Data formats describe the data objects used for transferring data to and from the system Clipboard. Table A.2 shows the standard data formats available in .NET Clipboard class.

Table A.2: Standard data formats in .NET Clipboard class

Format	Format Value
Bitmap	Bitmap
CommaSeparatedValue	CSV
DIB	Device Independent Bitmap
DIF	Data Interchange Format
EnhancedMetafile	Enhanced Metafile
FileDrop	File Drop Format
HTML	HTML Format
Locale	Windows locale (culture) Format
MetafilePicture	Metafile Picture Format.

OemText	OEM Text Format
Palette	Palette Data Format
PenData	Pen Data
RIFF	Resource Interchange File Format Audio Format
RTF	Rich Text Format
Serializable	Any type of serializable data objects
StringFormat	String Class Format
SymbolicLink	Symbolic Link
Text	Text
TIFF	Tagged Image File Format
UnicodeText	Unicode Text
WaveAudio	Wave Audio Format
XAML	Extensible Application Markup Language data Format
XamlPackage	Extensible Application Markup Language package data Format

A.3 System standard Clipboard Formats

Below table A.3 shows the standard Clipboard formats supported by the Windows API Clipboard. Only the main Clipboard formats are listed in the table.

Table A.3: Standard Clipboard formats in Windows API

Format value	Description
CF_BITMAP	Bitmap format

CF_DIB	A memory object containing a BITMAPINFO structure
CF_DIBV5	A memory object containing a BITMAPV5HEADER structure
CF_DIF	Data Interchange Format
CF_DSPBITMAP	Bitmap display format
CF_DSPENHMETAFILE	Enhanced metafile display format
CF_DSPMETAFILEPICT	Metafile-picture display format
CF_DSPTEXT	Text display format
CF_SYLK	Microsoft Symbolic Link (SYLK) format
CF_TEXT	Text format
CF_TIFF	Tagged image file format
CF_UNICODETEXT	Unicode text format
CF_WAVE	Audio file format

B. Dynamic Data Exchange messages

Since DDE is a message-based protocol, client and server communication is conducted by passing certain defined DDE messages.

Table B.1: Summary of DDE messages

DDE Message	Description
WM_DDE_ACK	Acknowledge message send for success or unsuccess message passing
WM_DDE_ADVISE	This message is used to establish a permanent data link.
WM_DDE_DATA	Sends data to the client application
WM_DDE_EXECUTE	Send series of commands as a string to the server application
WM_DDE_INITIATE	Initiates a conversation between the client and server applications
WM_DDE_POKE	Sends data to the server application

WM_DDE_REQUEST	Requests the server application to provide the value of a data item
WM_DDE_TERMINATE	Terminates a conversation
WM_DDE_UNADVISE	Terminates a permanent data link

DDE feature was disabled from the Microsoft Word because it was vulnerable and exposed to malware writers to distribute malware. This feature can be enabled by changing the registry key values as shown in below steps [27].

1. In the Registry Editor navigate to \HKEY_CURRENT_USER\Software\Microsoft\Office\version\Word\Security AllowDDE(DWORD)
2. Set the DWORD value based on your requirements as follows:
   AllowDDE(DWORD) = 0: To disable DDE. This is the default setting after you install the update.

   AllowDDE(DWORD) = 1: To allow DDE requests to an already running program but prevent DDE requests that require another executable program to be launched.

   AllowDDE(DWORD) = 2: To fully allow DDE requests.

C. Marshalling data types

Marshaling is the process of organizing or converting managed data into a format that is prescribed in the receiving unmanaged environment, or vice-versa. This is a run-time activity handled by the CLR. Most of the data types have a common representation in managed and unmanaged environment. These data types are known as blittable data types and those data types do not require any explicit handling by interop marshaler. Below list shows the available Blittable data types in the .NET Framework.

- System.Byte
- System.SByte
- System.Int16
- System.UInt16
- System.Int32
- System.UInt32

- System.Int64
- System.UInt64
- System.IntPtr
- System.UIntPtr
- System.Single
- System.Double
- One-dimensional arrays of blittable types (Example: an array of integers)

Data type is represented in different ways in managed and unmanaged code environments. When invoking functions in unmanaged code from managed code, it is important to know the matching types between them. Table C.1 shows the full list of available data types in unmanaged Win32 API, unmanaged C language environment and .NET framework.

Table C.1: List of data types in unmanaged and managed environment

Unmanaged type in Win32 API	Unmanaged C language type	Managed class name	Description
HANDLE	void*	System.IntPtr	A platform-specific type. Used to represent a pointer or a handler. 32 bits on 32-bit Windows operating systems, 64 bits on 64-bit Windows operating systems
BYTE	unsigned char	System.Byte	Unsigned integers with values ranging from 0 to 255. 8 bits
SHORT	short	System.Int16	Represents a 16-bit signed integer
WORD	unsigned short	System.UInt16	Represents a 16-bit unsigned integer
INT	int	System.Int32	Represents a 32-bit signed integer
LONG	long	System.Int32	Represents a 32-bit signed integer

BOOL	long	System.Int32	Represents a 32-bit signed integer
DWORD	unsigned long	System.UInt32	Represents a 32-bit unsigned integer
ULONG	unsigned long	System.UInt32	Represents a 32-bit unsigned integer
CHAR	char	System.Char	Decorate with ANSI
WCHAR	wchar_t	System.Char	Decorate with Unicode
LPSTR	char*	System.String or System.Text.StringBuilder	Decorate with ANSI
LPCSTR	Const char*	System.String or System.Text.StringBuilder	Decorate with ANSI
LPWSTR	wchar_t*	System.String or System.Text.StringBuilder	Decorate with ANSI
LPCWSTR	Const wchar_t*	System.String or System.Text.StringBuilder	Decorate with ANSI
FLOAT	Float	System.Single	Represents a single-precision floating-point number. 32 bits
DOUBLE	Double	System.Double	Represents a double-precision floating-point number. 64 bits

D. Managed Windows API Tools

Managed Windows API contribute useful tools which were developed by using the API. Following is the name of available tools in Managed Windows API. These tools can be used as a reference for exploring the functionalities of the Managed Windows API.

- AOExplorer

- ClipHancer
- ContentsSaver
- DeskIconeRestore
- GuessEXE
- NeatKeys
- QuickMacro
- ShootNotes
- TreeSizeSharp
- VolumeFader
- WinternalExplorer

E. FindFirstPrinterChangeNotification *fdwFilter* parameter values

The FindFirstPrinterChangeNotification function responsible for creating a change notification object for a specified printer which needs to be monitored by the application. The parameter *fdwFilter* indicates the conditions which cause the FindFirstPrinterChangeNotification object to trigger while monitoring a printer. Table E.1 shows the list of valid conditions for *fdwFilter* parameter.

Table E.1: Values for fdwFilter parameter

Value	Specific flags
PRINTER_CHANGE_FORM	PRINTER_CHANGE_ADD_FORM = 0x10000 PRINTER_CHANGE_SET_FORM = 0x20000 PRINTER_CHANGE_DELETE_FORM = 0x40000
PRINTER_CHANGE_JOB	PRINTER_CHANGE_ADD_JOB = 0x100 PRINTER_CHANGE_SET_JOB = 0x200 PRINTER_CHANGE_DELETE_JOB = 0x400 PRINTER_CHANGE_WRITE_JOB = 0x800
PRINTER_CHANGE_PORT	PRINTER_CHANGE_ADD_PORT = 0x100000 PRINTER_CHANGE_CONFIGURE_PORT = 0x200000 PRINTER_CHANGE_DELETE_PORT = 0x400000
PRINTER_CHANGE_PRINT_PROCESSOR	PRINTER_CHANGE_ADD_PRINT_PROCESSOR = 0x1000000 PRINTER_CHANGE_DELETE_PRINT_PROCES

	SOR = 0x4000000
PRINTER_CHANGE_PRINTER	PRINTER_CHANGE_ADD_PRINTER = 1 PRINTER_CHANGE_SET_PRINTER = 2 PRINTER_CHANGE_DELETE_PRINTER = 4 PRINTER_CHANGE_FAILED_CONNECTION_PRINTER = 8
PRINTER_CHANGE_PRINTER_DRIVER	PRINTER_CHANGE_ADD_PRINTER_DRIVER = 0x10000000 PRINTER_CHANGE_SET_PRINTER_DRIVER = 0x20000000 PRINTER_CHANGE_DELETE_PRINTER_DRIVER = 0x40000000
PRINTER_CHANGE_ALL	Notify if any changes occur in the specified printer
PRINTER_CHANGE_SERVER	Windows 7: Notify of any changes to the server

F. Class diagrams of MS Excel and PowerPoint add-ins

Figure F.1 shows the class diagram of MS Excel add-in.

Figure F.1: Class diagram for Microsoft Excel add-in

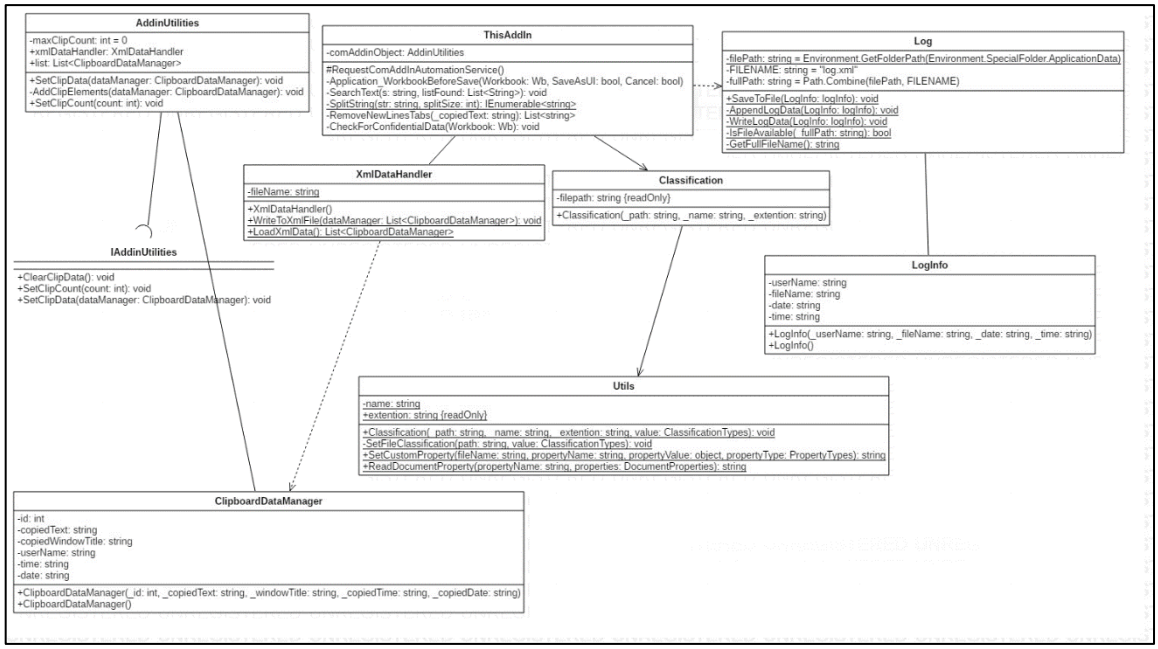
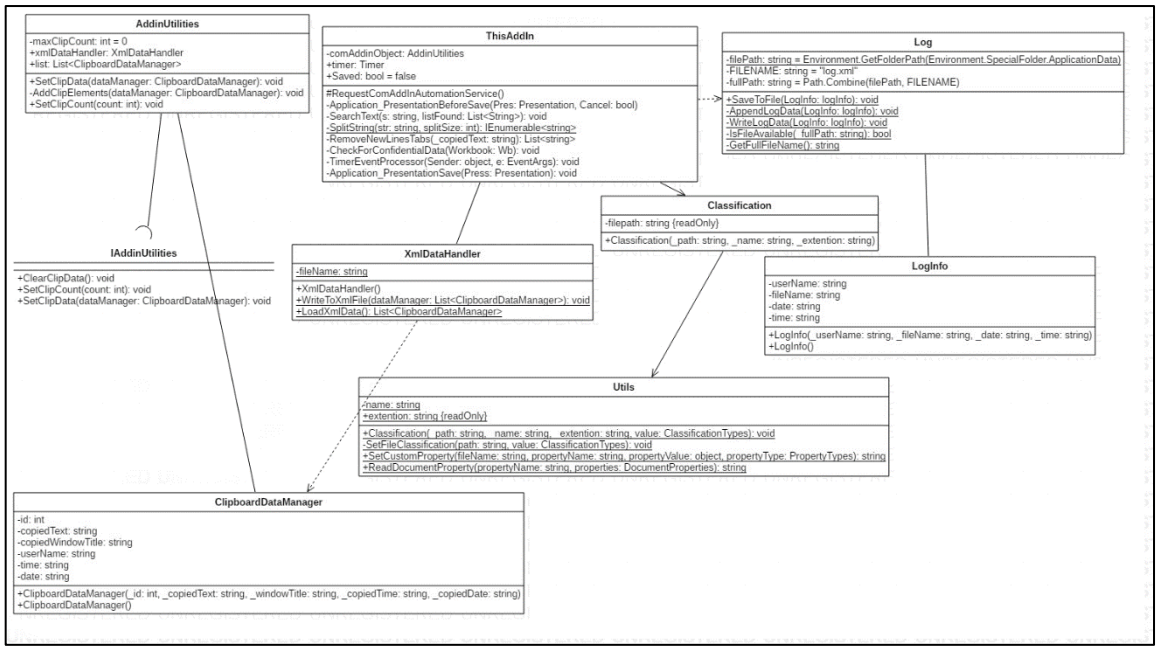


Figure F.2 shows the class diagram for Microsoft PowerPoint add-in

Figure F.2: Class diagram for Microsoft PowerPoint add-in



Acronyms

MS - Microsoft

API - Application Programming Interface

IT - Information Technology

DLL - Dynamic Link Library

DFS - Distributed File System

DDE - Dynamic Data Exchange

QOS - Quality of a Service

RDP - Remote Desktop Protocol

RSA - Rivest, Shamir, and Adelman

NLB - Network Load Balancing

GDI - Graphics Device Interface

LSA - Local Security Authority

SSPI - Security Support Provider Interface

DACL - Discretionary Access Control List

MMC - Microsoft Management Console

ACL - Access Control List

MIC - Mandatory Integrity Control

UAC - User Account Control

CA - Certification Authority

PKI - Public Key Infrastructure

CRL - Certificate Revocation List

CLR - Common Language Runtime

CIL - Common Intermediate Language

COM - Component Object Model

IJW - It Just Works

USB - Universal Serial Bus

WMI - Windows Management Instrumentation

CIM - Common Information Model

WBEM - Web Based Enterprise Management

WQL - WMI Query Language

HDC - Hard Disk Controller

IDE - Integrated Drive Electronics

SCSI - Small Computer System Interface

CPU- Central Processing Unit

UI - User Interface

JSON - JavaScript Object Notation

XML - Extensible Markup Language