BABEŞ BOLYAI UNIVERSITY, CLUJ NAPOCA, ROMÂNIA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION COMPUTER SCIENCE IN GERMAN

# Cloud based malicious PDF Detection using Machine Learning

– Diploma thesis –

**Supervisor**
Assoc. Prof. Christian SĂCĂREA

**Author**
Viorel GURDIŞ

2020

**Abstract**

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Context

The informational technology progress, that is in continous growth, brings a lot of benefits along with new responsabilities. There are plenty of applications that we use everyday across the entire World Wide Web. We don't even realize how much of our personal information is transfered to the virtual environment. That being said, it's important to be prepared for cybersecurity threats that can misuse our sensitive data. The problem is that the cyber attackers develop a lot of *hacking*[1] techniques, which are becoming increasingly difficult to detect. The popular software applications are the best target to inject malicious behavior. This happened with the Adobe PDF format. PDF documents are well known, trustworthy files and they are a global solution for sharing information. There are a lot of PDF readers and even browsers and email applications have support to open these files for viewing. It became so convenient to work with this format, that users interact with PDF documents without noticing any possible danger. However PDF is a often used attack vector. The large number of discovered PDF vulnerabilities and also the support of embedding Javascript code into documents are just some of the most exploited methods.

## 1.2 Motivation

Under the guise of seeming harmless, PDF documents are used on a daily basis across numerous public institutions, private companies and for personal purposes. Most of the people don't consider that these files could be dangerous and just copy the documents to their computers and access them. It is enough for one PDF to be malicious and the entire network affiliated to an institution becomes compromised. The cyber attackers succeed in achieving their goals, but

---

[1]attempt to gain unauthorized access to data in a system

the victim institution requires huge resources, both financially and time wise, to restore the integrity and security of their infrastructure. A measure to combat the described situation is having a real time protection installed on the computer. The most common solution, antivirus applications, work by signature matching, which is effective for detecting previously identified malware[2]. This means that all the antivirus applications require permanent updates in order to keep their malware databases up-to-date. Many users opt out of security solutions because of the multitude of hardware requirments they need. In this thesis we will go through a new approach that implies transfering complex detection algorithms from user's computers to a remote service, whose only task is analyzing suspicios uploaded PDF documents. Consequently the computers that will use this solution get rid of additional workload while still maintaining the system secure.

## 1.3 Paper structure and original contributions

The main contribution of this thesis is the research, design and implementation of alternative security solution oriented on multiplatform use and performance efficiency. For achieving this, the work was split into three important parts:

1. **Real time monitoring system on Windows**, whose goal is to notify the user when a specific file type, in our case PDF, is downloaded. The main focus when designing this software was to keep its functionality basic, so that it would require minimal computer's resources. As soon as it detects a new such file, it submits it to the Cloud for further analysis.

2. **Development of an intelligent algorithm for PDF classification** implies studying the PDF format skeleton, researching popular attack vectors using PDFs, extracting the most relevant features from a document and feeding them to the most fit Machine Learning classification algorithm. A part of this effort was also spent for searching the dataset, based on which the classification model was trained to correctly detect malicious PDF files.

3. **Cloud based application for remote scanning**, which is a generic application that can integrate models such as the developed one for PDF classification and can use them for analyzing the uploaded files. It offers an user friendly dashboard for visualizing the scanned files and also provides the possibility of submitting an URL containing a PDF file. The application will care about downloading and analyzing the file, showing the user the scanning results. This is a perfect option for users that require the Clouds features from a smartphone.

---

[2]any software intentionally designed to cause damage to a computer, server, client, or computer network

The remaining parts of this thesis are structured in the following manner: Chapter 2 contains the research made in the past years on malicious PDFs and Cloud Computing as an antimalware solution. Chapter 3 introduces the reader into fundamental mechanisms used for implementing the final application, as well as a brief history of malware in PDF. In Chapter 4 we present a Proof of Concept for developed classification model based on recreating a pseudo attack using malicious PDF. The 5th Chapter is putting each component together and presents the overall design of the application. Chapter 6 is meant for our final conclusions and we also show some of the future ideas for improving the application.

# Chapter 2

# Related work

Beginning with the first reported occurences of malware, the security researchers have made a huge effort to prevent the harmful behavior. Over the years malware has evolved in different forms and of course the antivirus industry has developed more complex detection solutions. Since PDF documents became a good target for cybercriminals, more and more specialists pay attention to the analysis of dangerous PDFs. Integration of so many analysis tools in a single antivirus software has a negative impact on computers performance. For this reason, the cybersecurity field attaches importance to Cloud Computing. In the following sections we will analyze other academic and industrial approaches for transfering antimalware engines to the cloud and some docummented detection techniques of malicious PDFs.

## 2.1 Cloud based malware detection

In the field of Cloud solutions for malware detection, there is a sparse amount of shared academic works. As compensation, in the antivirus industry there is a fast growing interest for Cloud Computing, which has higher computational power and could therefore run more advanced and complex detection algorithms.

An important example is **VirusTotal** [3], a popular Cloud application developed by Hispasec Sistemas, that aggregates more than 50 antivirus engines and makes them publicly available for scanning uploaded files. From the user perspective, this is an excellent application, that can extract metadata of the submitted file and can identify any dubious signal. The provided result represents a comparison between analysis verdicts of cybersecurity market leaders. Of course this is an advantage in terms of the scanning result precision and respectively a gain regarding spent computational time. On average it takes aprox. 55 seconds to upload and analyze a 400KB file. VirusTotal is also helping to maintain the global cybersecurity at a high level, by sharing all the submitted suspicious files with the security researchers. Thereby the antivirus engines will be permanently improved.

**Sandbox Analyzer** is another antimalware cloud solution developed by Bitdefender [5]. It's approach is a bit different, because it ensures the security on a private network, where the Bitdefender product is installed, thus not being available for public access. It's operating principle is also specific, by preventing the execution of threats on an endpoint and automatically sending of harmful files to the Cloud. After extra analysis in the Cloud, the Sandbox Analyzer can take remediation action based on the verdict. In that way, malicious files get disinfected, quarantined or deleted. One of the benefits for this solution is in-depth analysis of malicious files in an isolated environment, rather than on user's machine. Thereby the risk for performance implication, as well as the risk of accidental run of malware on an endpoint machine are eliminated. At the core of the Sandbox Analyzer there are Machine Learning algorithms and dynamic behavior analysis techniques that are constantly improved to detect fresh threats.

## 2.2 Detection of malicious PDF

**PJScan**, presented in the paper of Laskov and Srndic [10], is a Machine Learning approach that trains One-Class Support Vector Machine ($OCSVM$) to classify PDF files. This approach is focused on static analysis of embedded Javascript code, as it is known for the ability of integrating malicious behavior in PDF documents. The authors used *n-gram* analysis to extract lexical features. The obtained sequence of tokens served later as input for the machine learning algorithm. The trained model can correctly classify malicious samples containing Javascript code. However PJScan has a lower accuracy, because it is not able to detect obfuscated parts and there are also some samples containing other types of malicious payload, such as SWF[1].

Another example of static analysis of PDF Structure is **PDF Tools** by Didier Stevens [16]. It represents a suite of tools for scanning, parsing and dumping PDF files. This approach focuses on identifying the fundamental elements of the format, such as Streams, Cross Reference Tables etc. The advantage of PDF Tools is their ability of name obfuscation handling and their simplicity. Because of their high speed performance, PDF Tools are largely used in cybersecurity research.

A completely new approach mentioned in the research paper of Fettaya et al. [7] describes an algorithm that uses a Convolutional Neural Network ($CNN$) to detect malicious PDF files. The trained model, based on a single convolutional layer with a global max pool and a linear layer doesn't require any data preprocessing. Instead of this, the model is trained using as input the binary representations of the files. It is worth noting that the described algorithm could be efficiently used for distinguishing various families of malware. The rate of correct detections achieves 94%, the algorithm being also capable to classify aprox. 80% of the malware into different categories.

---

[1] Adobe Flash file format used for multimedia

# Chapter 3

# Scientific Problem

## 3.1  Problem definition

The purpose of the research in this thesis consists of studying some alternative methods to replace classic file scanning techniques. Particularly, our target is to demonstrate that Machine Learning algorithms can be used as an efficient mechanism for malicious PDF files detection. Additionally, we want to implement a framework that could remotely analyze suspicious PDF files by applying earlier mentioned performant algorithms. This framework should have several advantages:

- Take on the complicated task of applying classification model for uploaded files and give the analysis result to the users.

- Designed to be deployed as a Cloud Application on a powerfull server that can handle multiple requests at the same time, freeing users from caring about having high specifications for their computers.

- Guarantee the privacy of the scanned documents. The algorithm shouldn't require the documents content in order to decide whether they are malicious or benign. Also, after the analysis process, none of the documents should be stored on the Cloud.

- Keep the detection algorithms up-to-date. It won't be the users responsability anymore to regularly update the software antivirus installed on their computers.

- Provide a generic implementation for analyzing uploaded files. The support for a new file extension to be scanned, could be effortlessly added by adding a classifier Machine Learning model for the wanted file format.

Integrating both Machine Learning and Cloud Computing into Cybersecurity should provide a significant progress to this field.

In order to fully automate the process of detecting malicious PDF files, our application provides a tool (currently has support only for Microsoft Windows) that independently uploads detected files to the Cloud Service, waits for analysis results and takes appropiate protective measures. In the following we will introduce some operating system concepts that helped us achieve the expected behavior for our application.

## 3.2   Background processes in Microsoft Windows

In computing, a background process is a process[1] that runs independently and doesn't require any user involvement. Usually it has the task of system monitoring and sending any types of notifications to the user. Each operating system has its own principle of implementation for background processes. We will have an in-depth look at background processes in Microsoft Windows, which are called **Windows Services**. According [18] these services are started by a central utility - *Services Control Manager* at the boot-time[2] of the computer and they don't have any graphical interface. For security and safety reasons, specifically to prevent user applications from accessing Windows Services that could run with high privileges, Windows OS creates a new session for each logged in user, reserving *Session 0* for non-interactive services. The mentioned features make Windows Services suitable for long-running functionality, whose operation doesn't interesect with other users who work on the same computer. The support for creating a service is provided by **.NET Framework**, its features being possible to access by using one of Microsoft's high-level programming language C#.

## 3.3   File System monitoring

In the field of operating systems, a file system is aimed to store data in form of files in a long-term storage. Before being stored on physical devices, data that is created and saved by applications in User-Mode, goes through a chain of drivers from Kernel-Mode. This principle is adopted in many operating systems, inclusively in Microsoft Windows (see Figure 3.1). In order to track every change on the computer's file system, we need a monitoring mechanism. Throughout the development of Windows, serveral technologies have been implemented for integration by software engineers, including: File System Filter Drivers, Update Sequence Number Journal (*USN Journal*), Event Tracing for Windows (*ETW*), *FileSystemWatcher* Class from .NET. In this section we are focusing on two of the most popular Windows file system monitoring mechanisms: Minifilter drivers managed by *Filter Manager* and *FileSystemWatcher*.

---

[1]a container for a set of resources used when executing the instance of the program
[2]the period when a computer is starting

**Windows Filter Manager** is a Kernel-Mode driver that intercepts every file system I/O operation and can extend the functionality of the requested operation through each minifilter driver that is registered to it (see Figure 3.1). The order of attachement of each minifilter is identified by an altitude[3], thus avoiding interleaving the functionality of callback routines of each minifilter. Windows Driver Kit (*WDK*) provides the necessary SDK[4] for developing minifilter drivers, which are most suitable for Backup agents, Encrpytion managers and Antivirus filters. For more details consult [14].



Figure 3.1: Simplified I/O stack from Windows [14]

Compared to the previous described technology, which is complex for development and mantaining, **FileSystemWatcher** is a class from .NET that makes file system monitoring straightforward to implement. According to [4], this class represents a wrapper over Win32 SDK *ReadDirectoryChangesW* function, which is also used by Windows Explorer to monitor folder changes. By being provided by .NET, it is clear that FileSystemWatcher can be used in User-Mode for development (see Figure 3.2).

---

[3]unique identifier assigned by Microsoft that determines when a minifilter is loaded
[4]collection of software development tools

Figure 3.2: Relationship between .NET and Windows OS [18]

To configure a working instance of this class, we need to specify a couple of properties, such as *Path* (for indicating the folder to monitor), *NotifyFilters* (for specifieng what kind of file changes to monitor) and *EnableRaisingEvents* (to start/stop monitoring process). Also we need to attach the corresponding events that we want to receive (see Table 3.1).

Table 3.1: Events supported by FileSystemWatcher

| Event | Occuring time |
|-------|---------------|
| Created | When a new file is created or is moved into monitored folder |
| Changed | When a file has its content or file attributes modified |
| Deleted | When a file is removed or is moved out of monitored folder |
| Renamed | When the name of the file gets changed |

Earlier we mentioned that minifilter drivers live in Kernel-Mode, while FileSystemWatcher provided by .NET lives in User-Mode. One more argument in favor of FileSystemWatcher, besides the high level object-oriented programming language C# that we have at our disposal over unmanaged[5] C language, is that it's safer to run applications in User-Mode. Each application runs in isolation in User-Mode and in case of a crash, it won't affect other applications or even worse the OS itself, what can happen with a Kernel-Mode driver. Even the smallest error that occurs in a driver can cause BSOD[6].

---

[5]code executed directly by OS; does not provide any security to the application

[6]"Blue Screen of Death", error screen displayed by Windows in case of fatal system crash

## 3.4 Analyzing PDF File Structure

In this section we are going to understand PDF structure which is a foundation stone of identifying backdoors[7] in this file format.

PDF was first developed by Adobe in the 90s with many version being released afterwards. Its initial purpose was to allow to present various types of data, including text, images, webpage links etc. regardless of the environments it was opened in. As explained in [9], PDF files consist of objects, which are of eight types: Boolean values, Numbers (integer and real), Strings, Names, Arrays, Dictionaries, Streams and the Null object. Each PDF document should begin with a header that identifies it as a *PDF* and includes the version number `%PDF-1.7`. The document should also end in a certain way - with the signature `%%EOF` (see Figure 3.3). After header the objects are declared, that can contain necessary information for rendering the document. One of the most important objects is the stream, that can store an unlimited size sequence of bytes. Usually streams are used for storing text, but they can also store executable code, i.e., checkbox for agreement of *Terms and Conditions*. Streams can contain the optional entry `/Filter`, which is aimed for compression and encryption of data inside streams. Next comes the Cross-Reference Table (*Xref*), which determines the location of existing objects in PDF. The last section of the file is *Trailer*, that contains the metadata of the file, e.g., size of the file, number of objects, the id of the root object, etc. Hence it becomes clear that PDF documents are parsed from the end to the start.
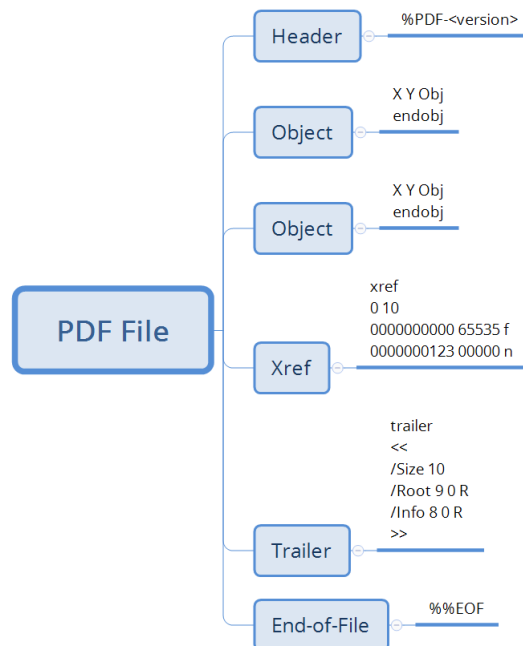


Figure 3.3: Structure of PDF Format [11]

---

[7]a malicious method by which a system bypass is hidden

## 3.5 Malware in PDF

Over the years PDF format has evolved and now it has support not just for text and images, but also for editable forms, animations and even 3D graphics. All of these features made PDF a great solution for information sharing. Meanwhile it also became a good target for cybercriminals, malware being easily embedded into PDF documents and widespread through different types of communication, e.g., HTTP, emails, etc. (see Figure 3.4)



Figure 3.4: Top malicious file types - H1 2019 [15]

Malicious PDF can be classified into three categories: *reader exploitation*, *feature abuse*, *phishing attacks*. Every year numerous vulnerabilities related to most used PDF readers, e.g., Adobe Reader, Acrobat Reader, Chrome, are reported. Before these weaknesses are patched[8], attackers create exploits to execute arbitrary code, when PDF is opened with one of the vulnerable application. Phishing attacks are more difficult to detect, because in the context of PDF documents, they aren't infected, instead the social engineering is used in order to psychologically enforce someone to click on malicious links written in the documents. Another frequently used attack vector involves misusing the PDF extra-functionalities, such as running malicious code when file is opened, stealing sensitive information and sending to remote addresses, downloading infected executables and so on. Pursuant to [12], it's usually the fault of embedded Javascript code, whose intentions could be wrong, but as seen in Table 3.2 there are also several standard PDF entries, which seem to be interesting in terms of malware hunting. However, when searching for them we should not forget about the possibility of obfuscation using hex code, in which, for example, `/Launch` can turn into `/L0x61unc0x68`. Encrypting the streams and strings is another often applied "trick" to complicate the analysis work for both antiviruses and researchers. A PDF document from which it is impossible to copy text or which can not be printed, is such a obfuscated sample.

---

[8]applying changes to fix bugs or errors in a software program

Table 3.2: Standard PDF Entries whose functionality can be abused

| Entry | Functionality |
|---|---|
| `/JavaScript` | Sets JavaScript code to be executed |
| `/OpenAction`, `/Names`, `/AcroForm`, `/Action`, `/AA` | Defines a script or an action to be automatically run |
| `/Launch` | Runs a program or opens a document |
| `/URI` | Accesses a resource by its URL |
| `/SubmitForm`, `/GoToR` | Can send data to indicated URL |
| `/ObjStm` | Hides objects inside Streams |

## 3.6 Machine Learning for Malware Detection

Machine Learning (*ML*) algorithms are increasingly used in various fields of science due to their fast adaptability to the new arising problems. Cybersecurity is one of these areas, where the complexity and the assortment of attacks are raising daily. It becomes convenient to put ML into practice because of the large amount of collected data (application logs, network traffic scans, etc.), which can be processed using statistics techniques and can be transformed into models for making future predictions based on the new incoming inputs. This solution reduces the costs for continuous analysis, leaving the hard work on powerful computers, whose high performance don't require huge investments nowadays. Also it helps businesses to more efficiently prevent potential threats and automate routine tasks, allowing security teams to focus on implementing new detection techniques to advance ML algorithms.

As presented in [6], Machine Learning's use cases in security can be split into two types: *anomaly detection* and *pattern recognition*, thus these solutions are used to solve following specific security issues: spam detection (filters are applied on emails, webpages in order to detect patterns of usual phishing attacks), malware catching (polymorphic malwares such as Trojan[9], Spyware[10], Ransomware[11] etc. are recognized based on their behavior), account takeover spot (such kind of anomaly is detected, because of multiple tries when gaining access to someone's account) and many others. For more detailed information about Machine Learning techniques used in Cybersecurity consult Table 3.3.

---

[9]malware that is disguised as benign software to avoid detection
[10]malware that is aimed to collect information without permission of the user
[11]malware that restricts access to personal data

Table 3.3: Application of Machine Learning to Cybersecurity problems

| Technique | Explanation | Algorithm | Cybersecurity problem |
|---|---|---|---|
| `Classification` | Separation into different categories | K-Nearest Neighbors (`KNN`), Support Vector Machine(`SVM`), Random Forest | Spam filtering |
| `Regression` | Prediction of future values based on previous ones | Linear Regression (`LR`) | Fraud detection |
| `Association Rule Learning` | Recommendation based on past experience | Equivalence Class Clustering and bottom-up Lattice Traversal (`ECLAT`) | Incident response (suggest a policy to apply in case of a particular incident) |
| `Clustering` | Grouping into unknown categories based on similar features | `K-Means` | Detection of polymorphic malwares |
| `Generative Model` | Generation based on known distribution | Markov Chains | Vulnerability scanning |

Many of the popular tech giants already apply Machine Learning solutions into their security departments. According to [8], Google is using ML to analyze threats against Android smartphones. Monthly new malicious applications are detected and removed from Google Play (Android application market). Microsoft integrates ML algorithms into Windows Defender Advanced Thread Protection (Microsoft cybersecurity cloud platform) in order to spot malware. Of course a lot of companies are determined to use Machine Learning to combat cybersecurity attacks, however artificial intelligence should be just a tool to suppliment human efforts, rather than replacing them.

# Chapter 4

# Proposed approach

For solving the problem of detecting malicious PDF documents, we came with a Machine Learning based approach. We took into account the importance of keeping the documents content confidentiality, as well as the need for ensuring the speed of analysis process. This solution will be next integrated into a Cloud framework responsible for remote documents scanning.

## 4.1   Dataset

The first step in building the optimal *ML* model was to create a dataset which would train the classifier. Our target was to collect malicious and benign PDF samples used in real life scenarios. Circa 70% of the clean and malicious documents were downloaded from the online malware repository *Contagio* [13], which provides samples collected from various open sources. Nevertheless, for more recent malicious PDFs we used VirusTotal [3], Hybrid Analysis [1] and VirSCAN [2], that allow searching files by their hash (*MD5*, *SHA1*, *SHA256*). Some of them provide access to the entire file collection so that we can order them by upload date and search for the most recent uploaded harmful files. Many clean samples were collected from public sources using Google Search Operators, i.e., `filetype:pdf`, for restricting search results of PDF files. Additionally, we completed the dataset with more than 100 clean interactive PDF files, that contain embedded 3D Widgets, incorporated JavaScript games etc. These clean samples should train the ML model, so as to correctly classify files even in corner cases.

Table 4.1: Dataset of PDF samples

| Category | Source |
|---|---|
| benign | Contagio, Google Queries, Tetra4D, PdfScripting |
| malicious | Contagio, VirusTotal, Hybrid Analysis, VirSCAN |

## 4.2 Proof of Concept

In the following subsections we will touch upon algorithms and techniques we have used to achieve the purpose of this thesis, namely detecting malicious PDF documents using a Machine Learning based solution. Figure 4.1 represents the entire process which was followed while developing the *ML* model. Each process phase will be fathomed in the following.
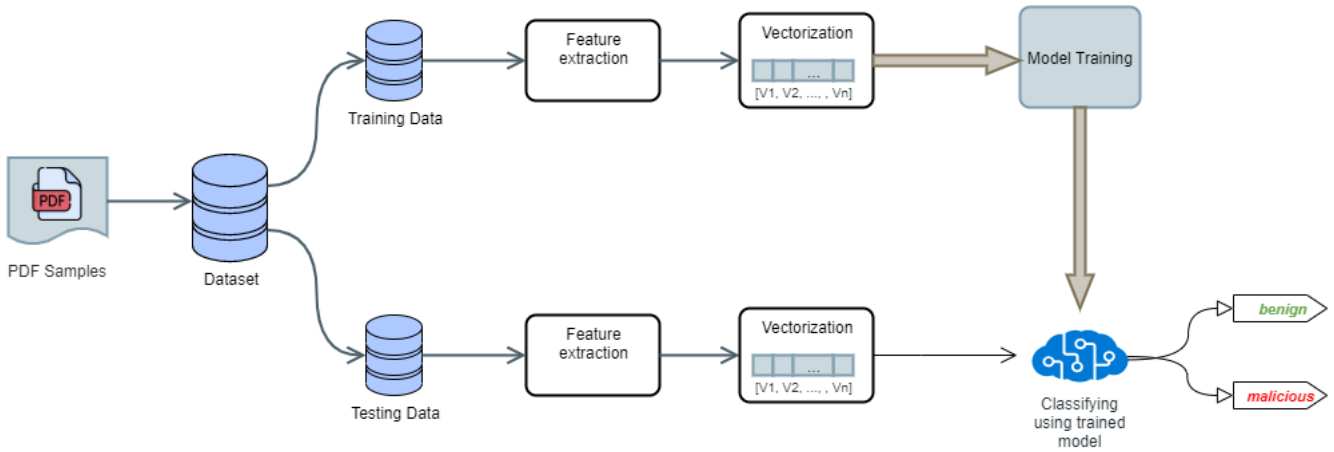


Figure 4.1: Machine Learning model creation

### 4.2.1 Feature Selection

As we already know, Machine Learning is based on mathematical concepts, hence the algorithms consist of performing mathematical operations to identify patterns in data. In order to use our dataset as input for ML algorithms, first of all we need to bring the data in an appropiate form. Therefor the most relevant data from a PDF document should be extracted and transformed into a vectorized form. As already seen in Table 3.2, there are several standard PDF entries which can be used for malicious intentions. For featurizing PDF documents we have used *PDFiD* from *PDF Tools* suite [16], which is a Python script that selects 22 features from a PDF file, including keywords commonly found in malicious documents, name obfuscations etc. The output of the script is a list of PDF entries and their occurences (see Figure 4.2). Before turning this featurized form of the PDF file into input for ML algorithms, the data should be normalized to [0, 1] range. This is an important step in order to avoid ML issues when features are on drastically different scales (see [17]). The vectorized form of the PDF document is built by applying the *Min-Max Normalization* (4.1) strategy on the array of features extracted by *PDFiD*.

$$z = \frac{x - min(x)}{max(x) - min(x)} \tag{4.1}$$

```
root@bt:/pentest/forensics/pdfid# python pdfid.py testpdf/apple1.pdf
PDFiD 0.0.11 testpdf/apple1.pdf
PDF Header: %PDF-1.4
obj 49
endobj 49
stream 15
endstream 15
xref 1
trailer 1
startxref 1
/Page 1
/Encrypt 0
/ObjStm 0
/JS 0
/JavaScript 0
/AA 0
/OpenAction 0
/AcroForm 0
/JBIG2Decode 0
/RichMedia 0
/Launch 0
/Colors &gt; 2^24 0
```
```
root@bt:/pentest/forensics/pdfid# python pdfid.py testpdf/evil_apple1.pdf
PDFiD 0.0.11 testpdf/evil_apple1.pdf
PDF Header: %PDF-1.4
obj 57
endobj 57
stream 16
endstream 16
xref 2
trailer 2
startxref 2
/Page 2
/Encrypt 0
/ObjStm 0
/JS 1
/JavaScript 1
/AA 1
/OpenAction 1
/AcroForm 0
/JBIG2Decode 0
/RichMedia 0
/Launch 1
/Colors &gt; 2^24 0
```

Figure 4.2: Example of results of *PDFiD* applied on a benign PDF file (left) and on a malicious one (right)

### 4.2.2 Classification Techniques

### 4.2.3 Performance Evaluation

### 4.2.4 Experiment

[19]

## 4.3 Used technologies

### 4.3.1 Microsoft .NET Framework

As previously mentioned in sections 3.2 and 3.3, our application is going to have a help tool for Windows based computers that will automatically detect new downloaded PDF documents, upload them to our Cloud Analyzer and take action (keep/delete) on that files corresponding to their category (benign/malicious). This tool will take the form of a Windows Service which will start running at the boot-time of the computer in the Session 0 (see section 3.2) and will communicate with an application responsible for notifying users through Windows notifications (see Figure 4.3).
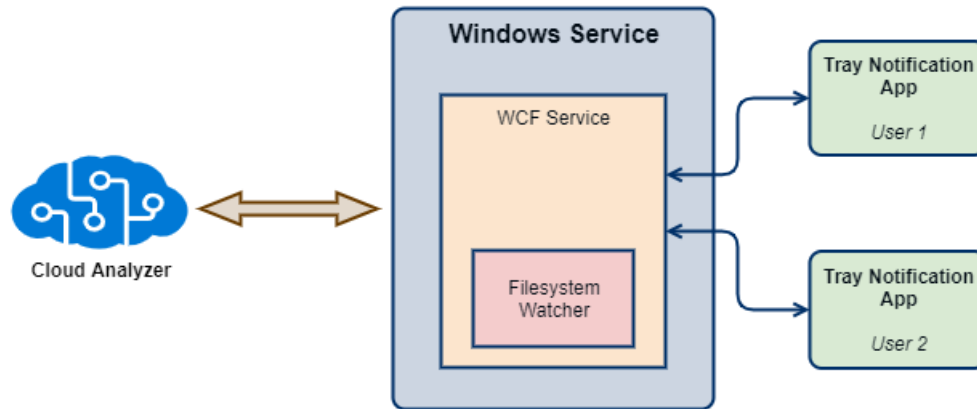
Figure 4.3: Architecture's outline of the help tool for Windows

Microsoft **.NET Framework** provides all the necessary Classes for creating the described tool. This framework represents a solution for standardization of application development on Windows operating system. Prior to the introduction of .NET Framework, the developers mostly used COMs[1] as a solution for code reuse. However, COM requires developers to write boilerplate code in order to turn their business logic into a reusable component, including memory clean up, reference counting etc, whence result a lot of errors. .NET Framework allow developers to focus on writing the business logic, while it is dealing with the memory management and is providing consistent exception-handling mechanisms. For example, in our application we implemented several .NET interfaces to achieve the expected behavior without having to manage a lot of low-level details. To create a Windows Service, we derived our class from `System.ServiceProcess.ServiceBase` and we had to override it's `OnStart`, `OnStop` methods. Because a service runs in Session 0, it means that it does not have support for user interface. Yet the application is intended to push Windows notifications and to ensure this functionality we had to treat our tool as a client-server application. The Windows service establishes an interprocess communication with the client *Tray Notification App* when a user logs in. For this mechanism the WCF[2] framework is integrated, which allows sending data as asynchronous messages from one endpoint to another. WCF was chosen as communication layer because it provides the *duplex* message exchanging pattern, where two processes establish a secure TCP connection and can send data in both directions. To make this working, it is necessary to just define used entities for communication as *Data Contracts*, that will later serialize the metadata by the WCF serialization engine. Another important component that lies at the core of the WCF service is the monitor class that inherits from .NET `System.IO.FileSystemWatcher`. It's behavior was described in section 3.3 and basically we use it to monitor user's specified folders and to send events to the users via the WCF communication. At the top level, in order to catch

---

[1]Component Objcet Model (COM) is a model of reusable software introduced by Microsoft in 1993

[2]Windows Communication Foundation (WCF) is a set an API in the .NET Framework for building connected, service-oriented applications

sent events from the Windows Service, a hidden WPF[3] application starts communication with the service at the user's login. When a PDF document is downloaded into monitored folder the Windows Service throws an event that is displayed to the user in form of a Windows System Tray Notification.

## 4.3.2   Scikit-learn Machine Learning Library

It would be extremely inefficient to always reimplement each Machine Learning algorithm that is required for solving a specific problem. **Scikit-learn** is one of the well known and largely used ML libraries, that came across as a solution for code reusability and performance optimization. This open-source library was created above the already prepared "ecosystem" for Data Science - *Python*. Even Python is just a programming language, there are a plenty of created libraries using Python, that are used on a daily basis by scientists. In terms of efficiency there are no doubts, that Scikit-learn would give way to other ML libraries. And all this beacause of it's underlying technologies. At the core it integrates:

- *Cython* - a compiled language that can bind compiled libraries, reaching high performance of the CPU;

- *Numpy* - optimized Python module for working with large multidimensional arrays;

- *Scipy* - a large collection of efficient algorithms for linear algebra, interpolation etc.

Till now, the Scikit-learn developers have supplied the library with various supervised and unsupervised algorithms, such as: regression, clustering, classification algorithms, as well as: Random forest, Support Vector Machine etc. It is a perfect solution for both academic and commercial projects, because of the simple API that it provides, thus the generic implementation of the ML algorithms can be adapted to personal purposes. Also, Scikit-learn integrates well with other Python libraries, that consistently simplify the data analyisis process. Some of these libraries are:

- *Pandas* - offers optimized data structures (*DataFrames*) for manipulation of large datasets. We could simpy apply *Min-Max Normalization* (see 4.2.1) on entire loaded dataset from a raw CSV file.

- *Matplotlib* - used for data visualization and plotting. Learning Curves of the classifier can be better analyzed from a visual representation.

---

[3]Windows Presentation Foundation (WPF) is a UI framework for desktop applications from .NET Framework

### 4.3.3 Python Flask

As soon as the heart of our application came to life - the ML model for detecting malicious PDFs we needed a way to make it publicly accessible. **Flask** is a microframework for web applications that helped us to make it possible. Comparing to other web frameworks, Flask is focused more on simplicity and extensibility, making the development much more oriented on solving own business case. This leightweight framework has no in-built database abstraction layer or object-relational mapping and neither any form validation or authentication, instead it supports third-party libraries that provide all enumerated functionalities and even more. Flask core includes basic functionality required by all web applications and gives an abstraction for creating request-response mechanism based on *Werkzeug* library, which provides a collection of utilities for WSGI[4] applications. Other implementation details can be easily integrated based on personal preferences. For example we used a NoSQL database in our application, having the possibility to choose from a large variaty of available mechanisms for data storage.

The created web service respects the requirements of a RESTful architectural design, Flask framework providing a simple mechanism for specifying routing and HTTP methods. For demonstration purposes we have listed below a minimalistic example of a REST HTTP Method (processes a file sent over a POST request and returns a JSON result) showing the Flask simplicity. One of the major advantages of this framework is the compatibility supported by nowadays giants of Cloud Computing. Our created web framework that exposes methods for uploading and PDF files and analyzing them, as well as visualizing history of scanned files, can be later deployed to Cloud services such as: Google Cloud Platform, IBM Bluemix etc.

```python
from flask import Flask, jsonify, request
app = Flask(__name__)

@app.route('/api/file-upload', methods=['POST'])
def upload_file():
  file = request.files['file']
  file_status = file_controller.upload(file)
  response = jsonify({'message': file_status})
  response.status_code = 200
  return response

if __name__ == "__main__":
    app.run()
```

---

[4]Web Server Gateway Interface (WSGI) is a specification that describes the communication pattern between a web server and web applications

### 4.3.4   ReactJS Framework

In order to use the web service in a user friendly way, a modern graphical user interface (*GUI*) is needed. The contemporary approach for developing a web GUI is to use a frontend JavaScript Framework. The range of choice is extremely large, e.g. *Angular*, *Vue.js*, *Ember.js* etc., but finally we decided for **ReactJS**. Being deveveloped and mantained by Facebook, ReactJS has a fast growing community and is one of the trending JavaScript Frameworks for 2020. The need of using a framework for web GUI arises because of necessity for time-efficient development experience, standardized methods of programming, faster performance and robust memory management. As other frameworks, ReactJS is aimed to render data to the $DOM^5$ and to eliminate the subsequent page refreshes in a complex web application that constantly updates its data. This mechanism is provided by the concept of *Virtual DOM* used in React, which involves constructing the representation of a web page in a virtual memory, where it uses the *Reconciliation Algorithm*[6] to perform only the necessary updates before rendering the page into the browser.

The extensibility of React code is assured by *Components*, that are rendered to specific elements in the DOM. These can be declared in two ways: functional and class-based, the main difference being in the "state" which can be hold only by a class-based component. The performance is influenced positively by the lifecycle methods in React Components. There are multiple lifecycle methods for controlling the state of a web page, but two of them that should be definetely took into consideration are `componentDidMount`, that is called once the data is fully loadded from a remote source, and `render` which is called every time the component's state gets updated.

Usually React requires the use of additional libraries for common functionalities like routing, state management and layout styling of the UI. For these purposes we used:

- *React Router* - is a routing library that keeps the user interface in sync with the corresponding URL. It manages the redirection, lazy code loading and dynamic route matching.

- *Redux* - is a state management tool, that keeps tracking of each component's state and store them into a global place, so that data becomes more accessible from any component.

- *Semantic-UI-React* - is a collection of UI standards defined as a singular style. Instead of recreating the layout of commonly used web elements (Buttons, Grids, Modals, Input Fields etc.), it is more efficient to use a library with all of those elements already created.

---

[5]Document Object Model (DOM) is the representation of a web page that can be manipulated using JavaScript

[6]is a diffing algorithm that compares the types of root elements and decides which element should be rerendered

# Chapter 5

# Application

In this chapter we are going to have an in-depth look at the architecture of our application - **ExtWatcher**, that represents a solution for the enunciated problem in this thesis.

## 5.1 Design

*ExtWatcher* is a software application assembled from multiple components using different technologies (see Figure 5.1). In the following we will introduce you into it's implementation design starting from the low level and we will explain what role each component plays in the context of providing a truly effective security solution.
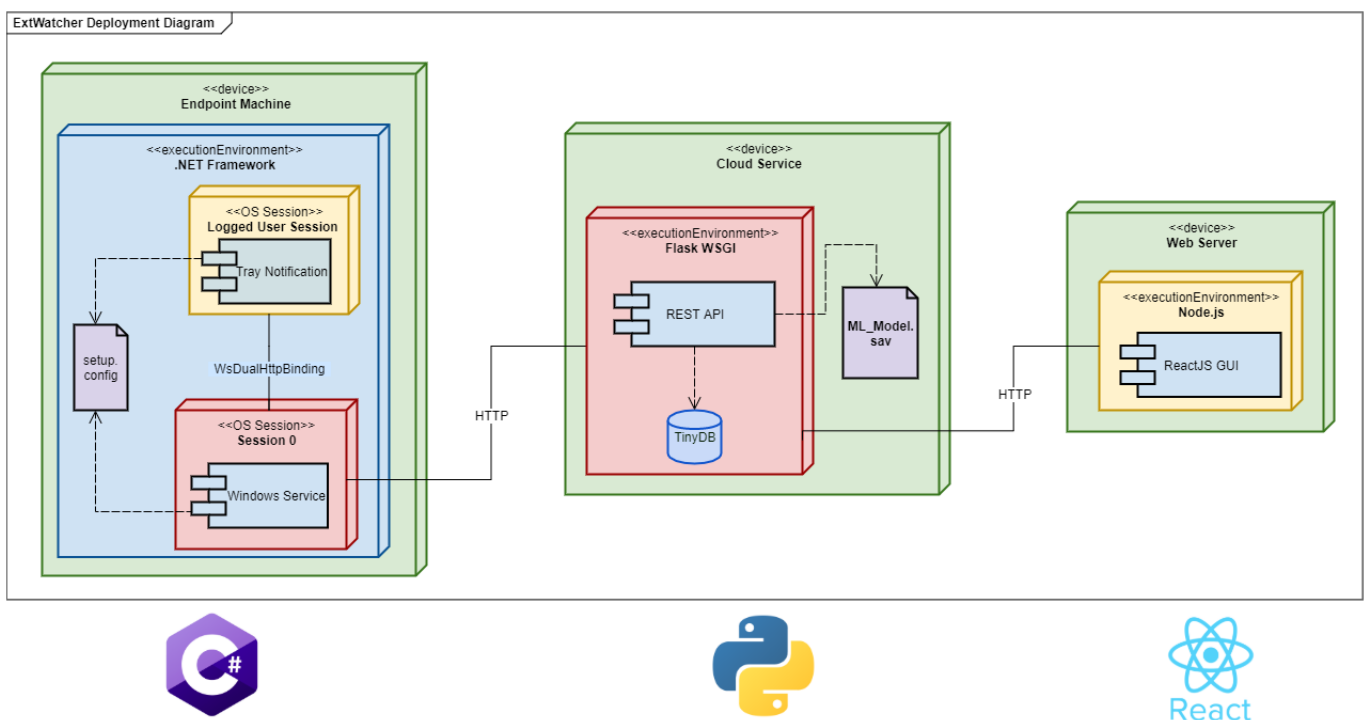


Figure 5.1: Deployment Diagram of the application and used technologies

The core of the entire software application is the *Cloud Analyzer*, which provides the security measure against malicious PDF documents. ExtWatcher has two different, independent approaches for getting access to the Cloud Analyzer. The first one implies automatic file detection and requires the installation of *ExtWatcher Service*, which will establish a connection through HTTP to the Cloud Analyzer and will remove instantly detected malicious files from the user's computer. The second approach is an *out-of-the-box* functionality, user being able to analyze a file by accessing the Cloud Analyzer Dashboard (web graphical user interface) and entering the URL of the file. The dashboard submits the URL to the Cloud Analyzer and the later downloads the file, classifies it and sends back the verdict. This solution is fit for mobile devices, as it requires no additional installation, ExtWatcher Service having support just for Windows OS at the moment.

In order to properly work, our application requires permanent Internet connection, because the core of analysis is remotely located. The advantage of this is the fact that cloud servers are better at applying Machine Learning models for malware detection, thereby reducing the workload of personal computers.
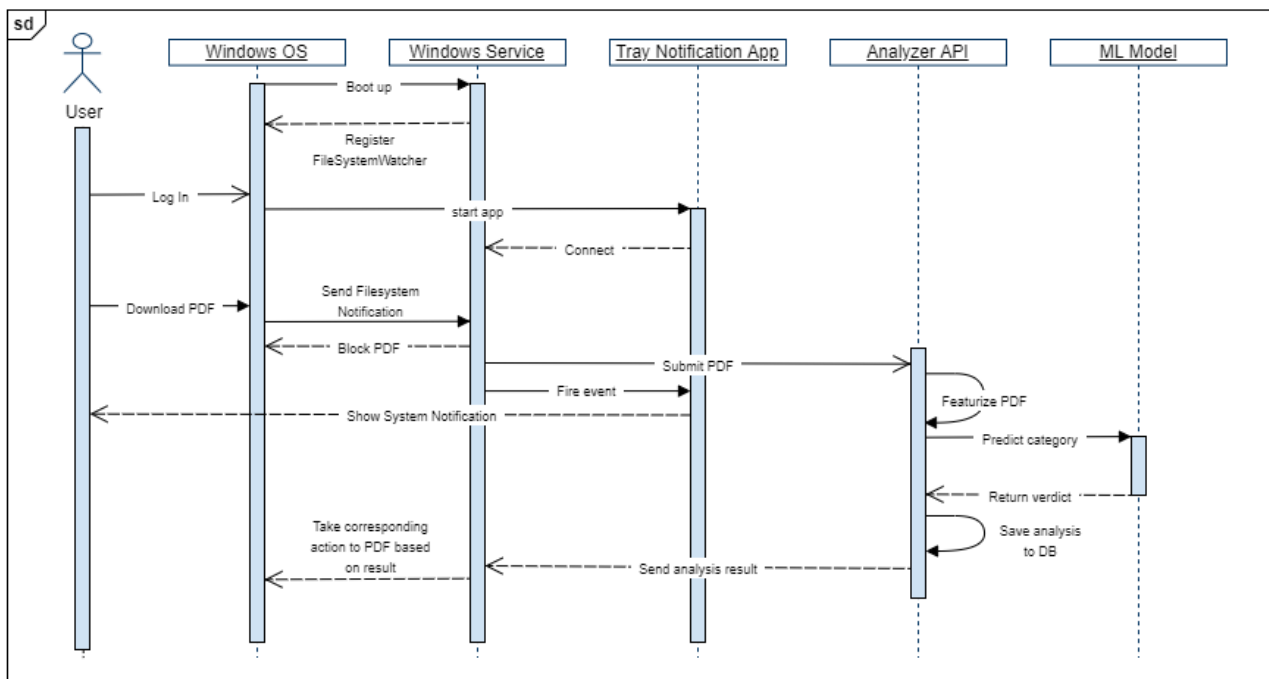
## 5.2 ExtWatcher Service



Figure 5.2: Sequence Diagram showing interaction between User, Windows Service and Cloud Analyzer
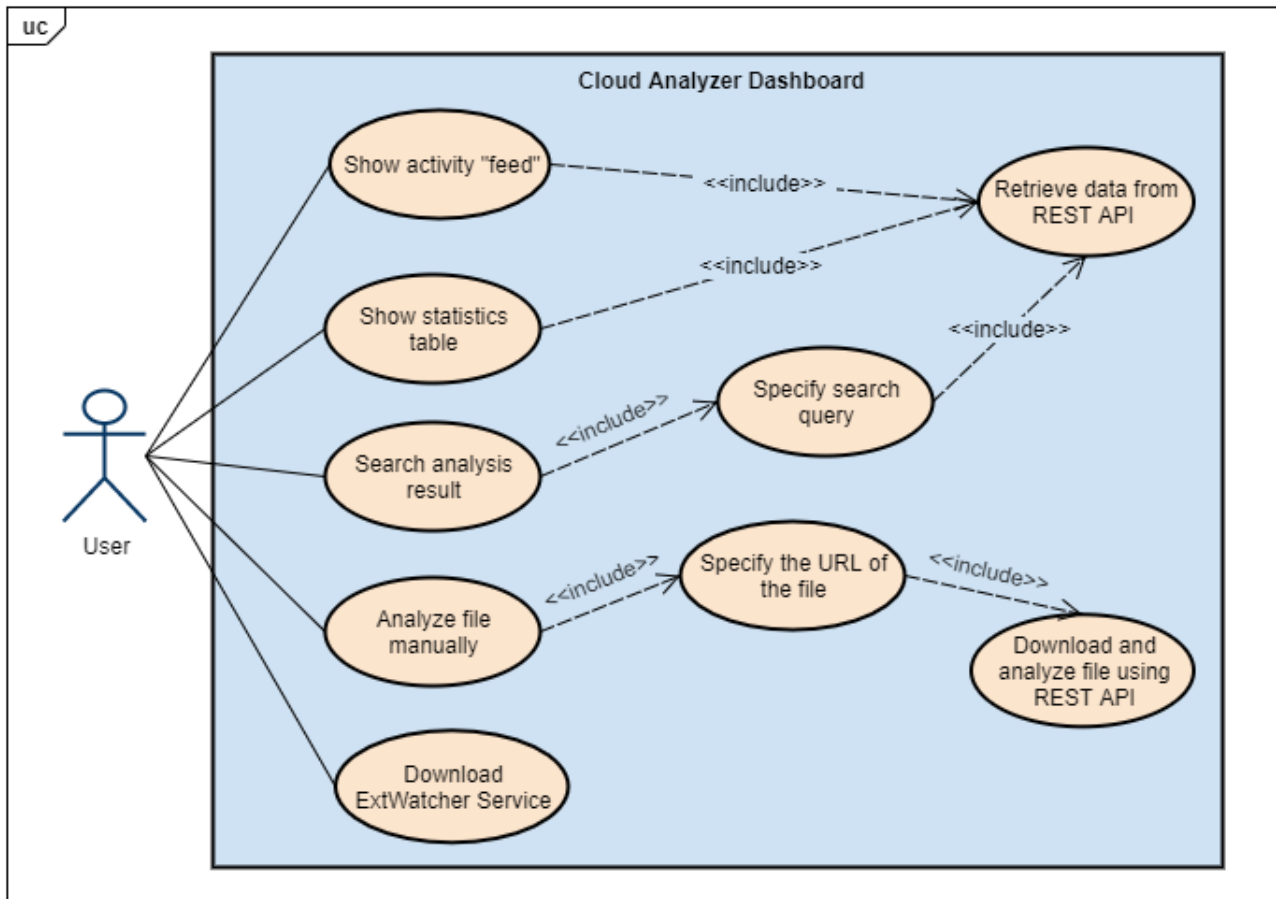
## 5.3 Cloud Analyzer

## 5.4 Dashboard Interface



Figure 5.3: Use Case Diagram showing interaction between User and Cloud Analyzer using web GUI

# Chapter 6

# Conclusion and future work

# Bibliography

[1] Hybrid analysis. link. [Online; Accessed 20-April-2020].

[2] Virscan. link. [Online; Accessed 20-April-2020].

[3] Virustotal. link. [Online; Accessed 1-April-2020].

[4] Tom Archer and Nishant Sivakumar. *Extending MFC Applications with the .NET Framework.* Addison-Wesley, 2003.

[5] Bitdefender. Sandbox analyzer. link. [Online; Accessed 1-April-2020].

[6] Clarance Chio and David Freeman. *Machine Learning and Security.* O'Reilly Media, Inc., 2018.

[7] R. Fettaya and Y. Mansour. Detecting malicious pdf using cnn. *International Conference on Learning Representations*, 2020.

[8] Gordon Gottsegen. Machine learning cybersecurity: How it works and companies to know, January 2020. [Online; Accessed 11-April-2020].

[9] Adobe Systems Incorporated. *PDF Reference, sixth edition: Adobe Portable Document Format Version 1.7.* Adobe, 2006.

[10] P. Laskov and N. Srndic. Static detection of malicious javascript-bearing pdf documents. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 373–382. ACM, 2011.

[11] LogRhythm. Detecting potentially malicious javascript embedded within a pdf file using logrhythm netmon. link. [Online; Accessed 11-April-2020].

[12] Xakep Magazine. Looking for exploits in pdf-documents on our own. link, 2014.

[13] Contagio malware dump. Malicious files for signature testing and research. link. [Online; Accessed 20-April-2020].

[14] Microsoft. Microsoft docs. link. [Online; Accessed 1-April-2020].

[15] Check Point Research. Cyber attack trends: 2019 mid-year report. link. [Online; Accessed 11-April-2020].

[16] Didier Stevens. Pdf tools. link, 2008. [Online; Accessed 1-April-2020].

[17] Emmanuel Tsukerman. *Machine Learning for Cybersecurity Cookbook.* Packt Publishing, 2019.

[18] Pavel Yosifovich, Alex Ionescu, Mark E. Russinovich, and David A. Solomon. *Windows Internals.* Microsoft Press, 2017.

[19] Lenny Zeltser. link. [Online; Accessed 1-April-2020].