**4.1: System overview**

In this chapter, we describe the proposed framework for self-healing version aware ransomware protection system. The main goal of the proposed framework is to serve as a version control system and protection against ransomware attacks for XML based documents. To achieve this goal, we employs two main mechanisms; the first mechanism is version control system to keep track of old versions of the files using the concept of hard links under Linux environment. While protection from ransomware is achieved by employing access control techniques that ensure complete protection of the file versions from modification or deletion in such a way that allows for direct recovery from ransomware attacks. Here, we point out that the novelty of our proposed framework relies on the way we combine well known techniques from access control theory and version control mechanisms in addition to some OS based techniques (mainly the use of hard links) in order to achieve the desired self-healing version aware ransomware protection of XML documents.

Figure 4.1 depicts the overall framework architecture. In this framework, all XML documents in a predefined directory goes through the version control module at the time of file closing in order to maintain the latest version of each document. Access control module which is activated by by invoking root daemon service to perform write protection for the taken version snapshot that is already pointed to the original file. The rest of this Chapter is organized as follows: In Section 4.2, we describe the details of the proposed framework. In Section 4.3, we illustrate the recovery process from ransomware attack. In Section 4.4. We discuss main limitations and highlight challenges. We first describe the version control stage in the proposed system. Then, we mention the need of adding links from the versioned copy to the original file, after that we describe the protection process for the copied version, then we describe how to recover original file in case of ransomware attack that could be encrypt the original file or delete it, so we described how to recover and how the file will be self-healed. Finally, describing the implementation challenges and limitations. The description of the framework components are build up by first presenting a flowchart of the system process (Figure 4.1), the proposed system is consists of three parts, namely: Version Control System, Link version to the original file, and Version Protection. The framework is created under Ubuntu 18.0 Linux operating system. Our framework starts by calling a root daemon service that loops over the liber office documents in a predefined directory. The process starts by changing the (.odt) file extension to (.zip), then extracting the document archive. After that, it will take a snapshot of the metadata of the file and adding new xml file to the file archive that contains the absolute URL of the original file. Then, recompress the version and change the extension to (.odt), and finally perform the write protection. The steps above are performed in the following sections.

Version Control

XML

Documents

Root Daemon Service

Access Control

Protected versions of the XML documents

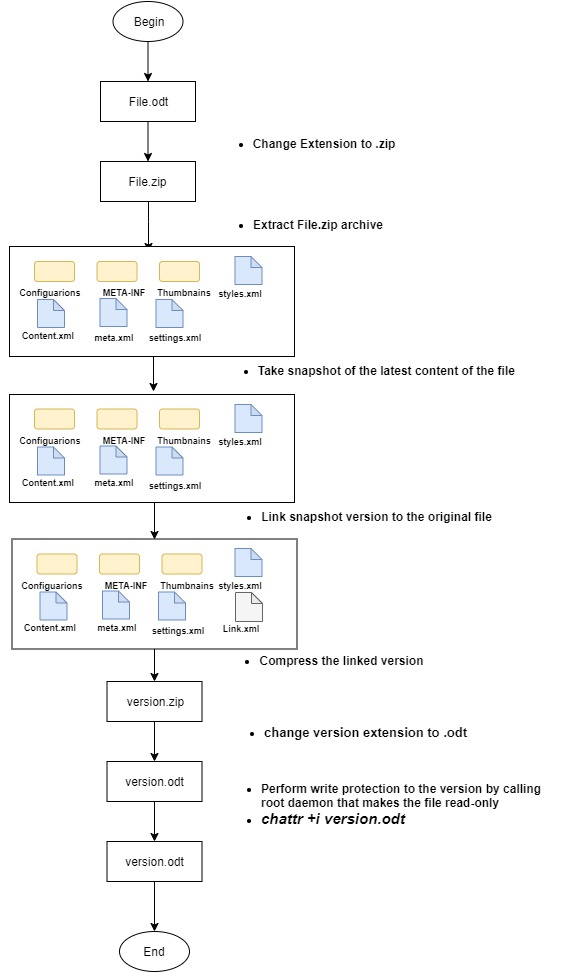


Figure 4.1: system workflow

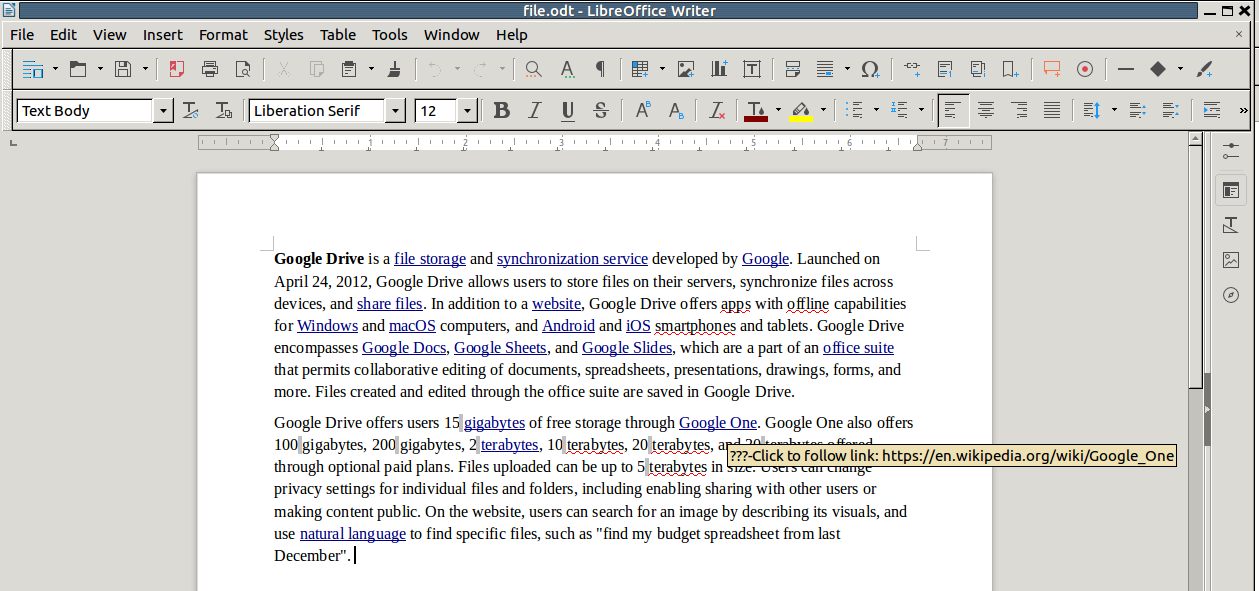
**4.2 System components**

**4.2.1 Version Control Module**

The version control module is supposed to maintain a copy of the XML file at time of closing the file such that we can retrieve the latest version in case of any corruption or system failure. This can be achieved by adding a special plugin for Microsoft word or Libr Office. As part of this thesis work we have implemented a custom plugin for Microsoft word …. Add more about this plugin.

**Step1: Extract File**

In this step, we convert the (.odt) file to (.zip) extension, which is already compressed structure. Then we perform an extraction to this zipped file in a predefined directory. We repeat this process for all existing (Figure 4.2)



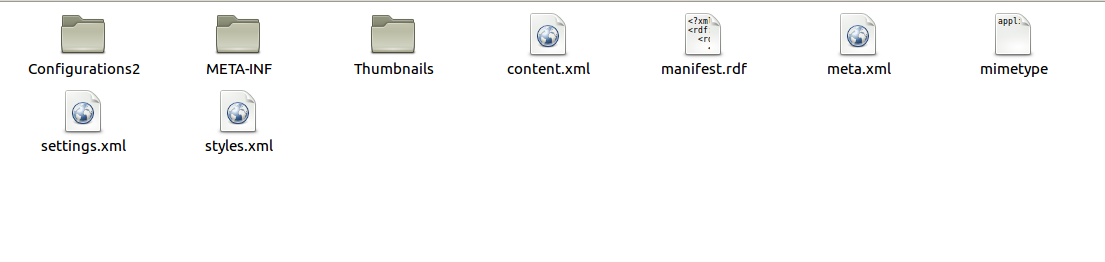


Figure 4.2: Extract (.odt) file

**Step2: Performing snapshots**

This part will employs the version control system by keeping a snapshot of the latest version of the file, by copying the metadata of the file to a predefined directory that stores the versions.

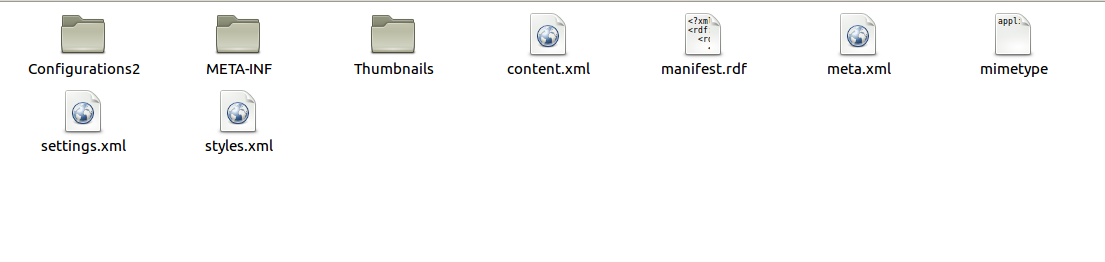


Figure 4.3: take snapshot of the file

**Step3: Adding Links to the original file**

In this step, we add a link to the version structure (copied version) that points to the original file. This part will guarantee that each version is belongs to the original file. When the original file is corrupted or deleted, this link will provide a hint and a pointer to it. This is done by adding a new xml file to the version structure that includes the absolute URL for the location of the original file.

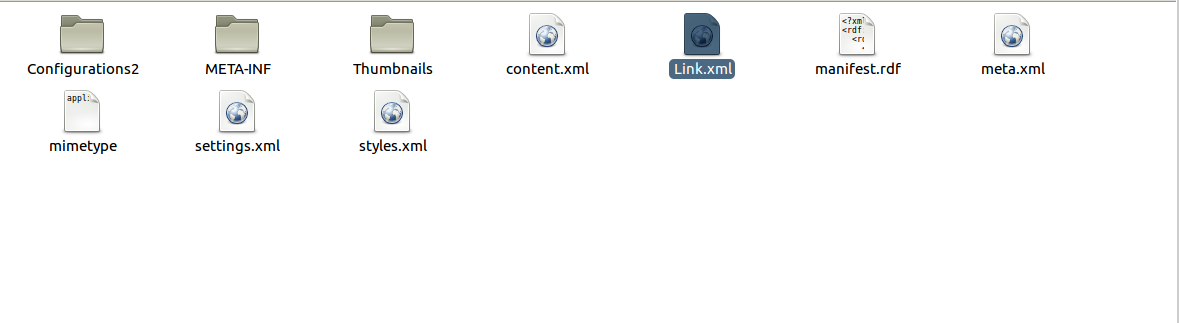


Figure 4.4: Adding Link to the original file (Link.xml)

**Access Control Module**

**Step4: Perform write-protection**

This components is the second main component in the proposed framework, through this process we make the root permissions to protect the copied snapshot from corruption or deletion by using sticky bit concept under Linux environment. To achieve this we use **chattr** command in Linux (Change Attribute) which is a command line in Linux that is used to set/ unset certain attributes to a file in Linux system to secure accidental deletion or modification of important files and folders , even though root users.

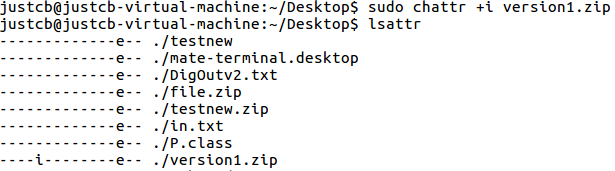


Figure 4.5: perform write/delete protection using chattr command

If we try to write to the file or delete the file, we got a message that the file is write-protected and the operations are not permitted, so the file is immutable as shown in figure 4.6

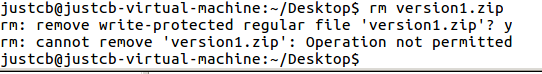


Figure 4.6: the file is immutable when trying to write or delete

**4.3 Recovery from Ransomware Attack**

The framework ultimate goal as mentioned earlier, is to preserve the files in case of ransomware attacks take place. The result of attack will corrupt the file or even delete it. But we ensured that the file is self-healed as the protected version will not be affected and points to the original file. Based on our solution the protected snapshots will be not affected and can be recovered under root privileges. The recovery process is done by removing the sticky bit attribute and make sure that the file extension is .odt

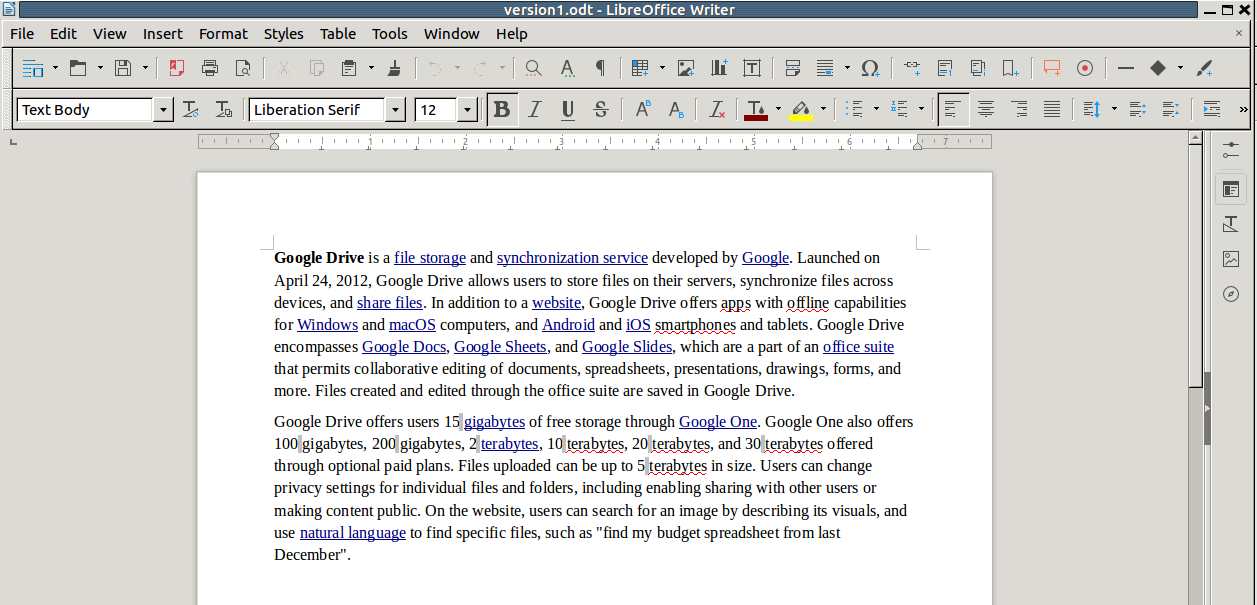
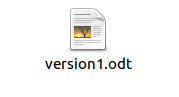


Figure 4.7: recovery process

**4.4: Implementation challenges and limitations**

Throughout this theses, we conduct several experiments to achieve our goal to keep a protected version of our xml based files. Unfortunately, we faced many challenges during the implementation. I can summarize these challenges in the following section:

* We start our work under Windows environment, we decided to consider MS Word documents to be our use cases which is XML based structure. We did the following steps, first we implement a plugin to be added to MS Word files ribbon using MS Visual studio project. The main goal for this plugin is to work as version control system for the working documents by keeping a version or full snapshot of the active word document when fie closing. We make this process as a background process. Figure 4.8 shows the added plugin to MS word 2013 ribbon.



Figure 4.8: MS word 2013 custom plugin

* We aimed to build the version control system to be a distributed version control system that ensures portability and doesn’t depends on a centralized repository, the implemented plugin ensures portability since it keeps the versions inside the file structure. Figure 4.9 shows example of the kept versions (history folder) inside the file itself after extraction.

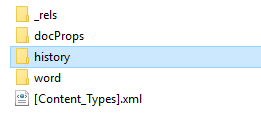


Figure 4.9: history versions folder inside file structure

* We implement a windows service that runs in the background to call the plugin function when file closing that iterate for the word documents that exist in the hard disk or a specified directory. This service implicitly open the word file and close it, by closing it will take a snapshot inside the file itself, so by taking the snapshot the plugin creates the version and adds it inside the file structure, So we are taking about a portable version control system, the version is saved and added to the file structure.
* The second challenge is to perform the permission to the version each time the service is called. MS word files are compressed structure files; which means its compressed structure as shown in the following example, the version is part of the file structure. The main challenge when perform the read-only permissions on the specified version we lose these permissions while compressing or zipping the file again. This is because a file attributes in windows environments.
* So we decided to move to Linux environment, to perform our experiments. The approach still almost the same with some modifications. The plugin is implemented under windows and it doesn’t for Liber Office files under Linux.