Chapter 13
Data Hiding and Detection

Learning Objectives
The objectives of this chapter are to:

- Understand motivations behind data hiding techniques
- Know common data hiding techniques
- Know how to search for and recover hidden data

Much of digital forensics involves the recovery of information from electronic sources. This is often tedious work and requires great attention to detail. In previous chapters, we have studied techniques of how to recover deleted data. While criminals or mischievous computer users often cover their unlawful activities by deleting data or files which could hold them responsible, another commonly used tactic by them is to hide data. Data hiding is the act of storing information in such a way that hampers the awareness of the content and/or existence. There exist a number of techniques that allow users to hide information from other users. There are numerous reasons why data must be hidden for a period of time. Motivations will vary from person to person. For example, it can be done for matters of privacy. Some environments may be openly hostile to encrypted traffic, enforcing censorship on communications. Societies view some content as embarrassing or scandalous. However, it becomes crucial to detect hidden data and recover them from electronic devices as evidence to prosecute wrongdoers when data hiding is used to cover any illegal activities. Therefore, as a digital investigator, it is essential to understand how evidence can be hidden. In this chapter, we will focus on data hiding fundamentals. We will study data hiding techniques and also discuss analysis techniques for hidden data.

X. Lin, *Introductory Computer Forensics*,
https://doi.org/10.1007/978-3-030-00581-8_13

13.1 Data Hiding Fundamentals

Cryptography is another technique of secret writing and transforms a data in plaintext into a ciphertext, which is unreadable to anyone except authorized people. While encryption is used to ensure confidentiality, ciphertexts can be found to be identified by casual observers. A number of tools and strategies exist to aid in plaintext recovery, also known as cryptanalysis. To hide information is to conceal it from a casual observer. Data hiding differs from encryption. Data can be encrypted then hidden, only to be found by the creator or others that know of its whereabouts. Furthermore, using encryption may draw attention to a particular file. It may also raise flags in environments where encryption is frowned upon. Hiding rather than encrypting data often allows us to transmit and stores confidential information without raising the scrutiny of others. If we imagine hidden content as items which we can hide in a box. Locking such a box can be considered encrypting its contents. If the box is in our possession, with the right tools and time we can open it. If the box is hidden, we would have to search for box prior to attempting to break into it. Furthermore, if we are unaware of the box's existence which is the crux of data hiding, then accessing the contents of the box becomes much more difficult.

It is also important that we acknowledge that data can be hidden for more nefarious purposes. In the both the real and digital world, criminal activities are better performed secretly. Encrypting illicit activity is practical. However, with adequate knowledge of a cryptosystem and enough time, ciphertext can be deciphered. Here is where hiding rather than or in conjunction with encryption can be advantageous.

There are a number of illicit motivations for hiding data:

1. Financial Fraud
2. Communicating explicit or nefarious activity
3. Sales in elicit and illegal products such as drugs and weaponry
4. Instructions on developing dangerous material such as explosive devices
5. Recruitment and communication with terrorist groups
6. Child pornography

When discussing data hiding, conversations are often limited to steganography. Steganography refers to the art of electronically hiding data within another cover or media. This was derived from the Greek words *steganos* and *graphein*, which mean to *conceal or protect* and *writing* respectively. Data hiding however, is not limited to steganography. A number of alternatives that will be discussed later in this chapter, which is the focus of this chapter. The concept and techniques of steganography will later be described in Chap. 21.

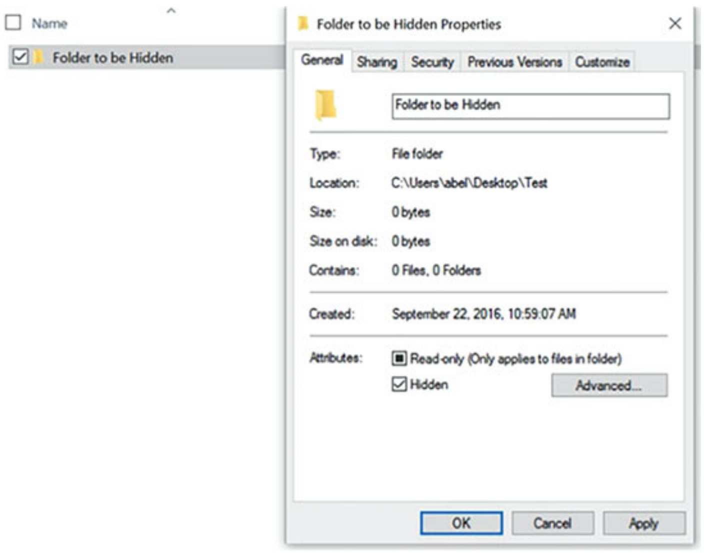
Forensic investigators are often tasked in recovering hidden incriminating evidence. As computer systems mature, new methodologies for hiding can emerge or become obsolete. Some techniques may only exist for specific operating system versions or OSes with specific patches. Keep abreast with research and vulnerabilities which can be exploited to hide data. Steganalysis or uncovering hidden data can be remarkably more difficult than cryptanalysis. Consider the earlier example of a locked boxes, a hidden box and a box which we are unaware of are increasingly more difficult to access. Similarly, we cannot attempt data recovery when we are unaware of data existence. However, by incorporating certain tools and best practices, uncovering hidden data can be more attainable.

13.1.1 Hidden Files and Folders

This is one of the more trivial techniques used to hide information. However, you are more likely to come across it in your forensic work. Most operating systems ship with features to allow users and administrators to hide files and folders. This feature allows us to hide configuration files and the like. Let us look at a simple method for hiding files on a Windows machine.

In your Windows machine, create a folder in with your file browser. Right Click on the folder to select properties. From here we select "Hidden" check box to hide your folder (Fig. 13.1). This is quite a simple method. However, it is also easy to address if we intend to search for hidden files by enabling your file explorer to show files and folders with the hidden attribute. There are however, more concrete methods for hiding your holders.

On your Windows machine open the command prompt or *cmd*. Find the path to the directory which you wish to hide. You can do by copying the file location in your file explorer. We will be using the *attrib* command to hide our folder.



Folder to be Hidden

Type: File folder
Location: C:\Users\abel\Desktop\Test
Size: 0 bytes
Size on disk: 0 bytes
Contains: 0 Files, 0 Folders
Created: September 22, 2016, 10:59:07 AM
Attributes: ■ Read-only (Only applies to files in folder)
☑ Hidden

Fig. 13.1 Simple method for hiding a folder on Windows platform

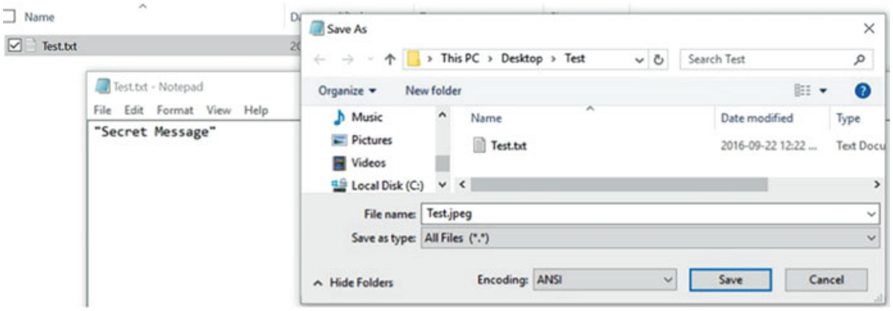
```
#To hide a folder and its contents
> attrib +s +h "C:\Users\User Name\Folder Location"
#To reveal a folder and its contents
> attrib -s -h "C:\Users\User Name\Folder Location"
```

The *attrib* command is a more concrete method for hiding folder. If others choose to display hidden folders and files, the files hidden with the above commands should remain hidden. As a digital forensic analyst you can write scripts with the *attrib* command to unveil hidden files.

13.1.2 Masks and Altering Names

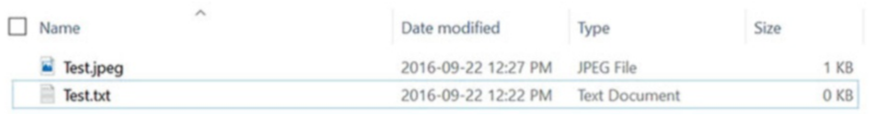
While many may go to great lengths to hide files, some may simply alter file names. A common example is to rename folders with less conspicuous titles. For instance one may hide a series of controversial files in a directory guised as work. In digital forensics it is pertinent that you remain aware of such practices. Renaming techniques are quite common with malware. They vary from changing the icon image of the file to renaming the file as *null*. There are of course more complex methods for hiding files, a few of which we will examine in this section.

Upon saving the file as an image, all attempts to preview and open the file as an image will be unsuccessful. However, you will still be able to open your document as a text file to edit its contents (Figs. 13.2 and 13.3).


"Secret Message"

File name: Test.jpeg
Save as type: All Files (*.*)
Encoding: ANSI

Fig. 13.2 Changing the file extension of our text file to hide its contents



Name	Date modified	Type	Size
Test.jpeg	2016-09-22 12:27 PM	JPEG File	1 KB
Test.txt	2016-09-22 12:22 PM	Text Document	0 KB

Fig. 13.3 The secret text file now appears as a JPEG image in your browser

13.1.3 Volume Slack

A partition or logical drive has been formatted with a file system before it can be available for data storage. Usually, an entire partition is fully occupied by the file system. However, it is possible that not the entire partition or logical drive is used, and some space is left unformatted. Under normal conditions, this unformatted space cannot be allocated to files. It is called volume slack, which is the unused space between the end of file system and end of the partition where the file system resides.

Volume slack is not accessible from the Operating System (OS) in a normal way, and can be used to hide data. Therefore, disk partition(s) must be examined carefully for volume slack. If we discover there is a size difference between the disk partition and the file system residing in the partition, we can determine that there is volume slack. Then, a further examination is needed to determine if hidden information exists inside it.

13.1.4 Slack Space

File systems organize disk space into allocation units (or clusters (FAT and NTFS for Windows) or blocks (Ext for Linux)), where each is composed of a sequence of sectors on a disk. Since the file system allocates disk space to a file in clusters but the file size is an even multiple of the cluster size, there ends up being unused space in the end of the last cluster allocated to the file. This space is called slack space, and will not be used by other files.

Slack space is not accessible from the OS in a normal way, and can be used to hide data. Also, from a security standpoint, unused space can contain previous data from a deleted file that can contain valuable information. Therefore, slack space must be examined carefully for possible hidden data or remnant of the previous file which used the space, especially, when it contains non-zero data.

13.1.5 Clusters in Abnormal States

After a disk has been in use for a while, it is inevitable that little portion of hard disk becomes unreadable. As a result, these damaged disk space should be avoid to store data. Usually, a modern OS scans a disk periodically for corrupted disk space or bad blocks (sectors), for example, using use the CHKDSK tool in Windows. These corrupted disk space is marked as bad by the OS. For example, Windows maintains a reference of the bad clusters on NTFS volumes in the file $BadClus, one NTFS metadata (or system) file. No data will be written to the clusters listed in the $BadClus since these clusters are considered as bad/faulty by the NTFS file system.

However, this functionality can be also exploited by criminals to intentionally mark a good cluster as bad in the file $BadClus and store hidden data into it.

Also, recall file systems organize disk space into clusters, where each is composed of several sectors. Apparently a single bad sector will get an entire cluster marked as bad, although the rest of the cluster is good. Thus, it is occasionally possible to recover a partial of the cluster.

Similarly, files systems keep track of the allocation status of all clusters within them. For example, in NTFS file systems, another NTFS metadata file, $BitMap, maintains the allocation status of all clusters on NTFS volumes. In reality, an allocated cluster should be associated with a file in a file system. However, it is possible that a cluster is marked as allocated or used but there is no file occupying and using it. It is called an orphaned cluster. Since orphaned clusters are flagged as being used, no data will be written to them. Apparently, data can be hidden into them.

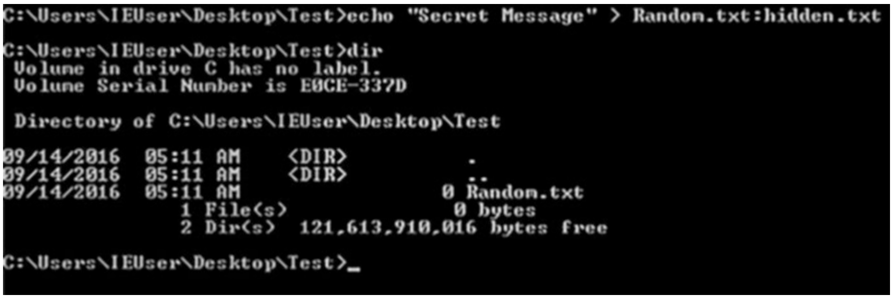
Therefore, clusters in abnormal state, including bad clusters and orphaned clusters must be examined carefully for possible hidden data.

13.1.6 Bad MFT Entries

This is a technique that is specific for NTFS file system. NTFS uses the Master File Table (MFT), which contains an entry for every file and folder on NTFS volume. Each MFT entry starts with a signature "FILE", indicating it is a good entry. When an MFT entry is corrupted, it starts with a signature "BAAD". Similar to bad clusters in a file system, NTFS will not allocate a bad MFT entry to a file or directory. Thus, criminals can exploit it to hide data into MFT entries which are intentionally marked as being bad by them.

13.1.7 Alternate Data Streams

This is another technique that is specific for NTFS file system. In the NTFS file systems, data streams contain the data for the file. A data stream is encapsulated into a $DATA attribute, also known as the data attribute. Usually, there is only one $DATA attribute for each file, and the data stream in it is referred to as the *primary data stream*. It is also called the unnamed data stream since it is a data stream without a name (or with the empty name string). Nevertheless, NTFS supports multiple data streams. Among them, only one unnamed data stream per MFT entry is allowed, which is the one in the default data attribute. The data streams in any additional data attributes must be named and dubbed alternate data streams (ADSs). There are numerous use cases for ADSs. For example, ADS is used to store summary information for files.



```
C:\Users\IEUser\Desktop\Test>echo "Secret Message" > Random.txt:hidden.txt

C:\Users\IEUser\Desktop\Test>dir
 Volume in drive C has no label.
 Volume Serial Number is E0CE-337D

 Directory of C:\Users\IEUser\Desktop\Test

09/14/2016  05:11 AM    <DIR>          .
09/14/2016  05:11 AM    <DIR>          ..
09/14/2016  05:11 AM                 0 Random.txt
               1 File(s)              0 bytes
               2 Dir(s)  121,613,910,016 bytes free

C:\Users\IEUser\Desktop\Test>_
```

Fig. 13.4 Creating a simple alternate data stream

ADS was intended to create compatibility between Macintosh Hierarchical File System (HFS) and NTFS. Unfortunately, this technique is not detectable by file browsers and information. In other ways, there are no regular ways in Windows Explorer to access ADS in a file. Apparently, with ADS, multiple hidden files can be attached to a file. Additionally, the multitude of hidden files do not affect space allocation calculations. There is however, a much darker side to ADS files. They can be used to execute malicious *.exe* files but will not show up in Windows Explorer (or the Command Prompt). Making matters worse, ADSs are simple to create. They can easily be created through scripts or via the command prompt with a colon [:] appended to the cover file [10]. The following is an example of creating an ADS file

13.1.7.1 Creating an ADS File

On your Windows machine open a command prompt. Create a new ADS file with a stored secret message using the following command.

```
echo “Secret Message” > Random.txt:hidden.txt
```

In Fig. 13.4 we store the message *Secret Message* in the file “*hidden.txt*” which is appended to “*Random.txt*”. Notice that the *dir* command does reveal the existence of hidden.txt which is hidden in an alternate data stream.

As observed above, ADS can be easily abused by criminals to hide data and malicious applications (malware). Thus, we must scan NTFS volumes thoroughly and search for ADSs. Afterwards, these ADSs are further recovered and then analyzed to determine existence of hidden data and malicious applications.

13.1.7.2 Recovering ADS Files

Next, let us attempt to recover “*hidden.txt*” and its contents.

```
# Try to locate hidden.txt in your directory
> dir
# Did you locate the file? Next let us attempt to output the contents of hidden.
txt
> notepad Random.txt:hidden.txt
# Were you able to recover your secret message?
```

13.2 Data Hiding and Detection in Office Open XML (OOXML) Documents

While there are many methods listed above to hide data, there are still several different approaches to consider. Another popular way is to hide data in some special file types. Next, we take Office Open XML (OOXML) document as an example to illustrate how to perform Data Hiding and detection in OOXML documents. Office Open XML (OOXML) is a zipped, XML based file format for representing spreadsheets, charts, presentations and word processing documents [1]. It is introduced by Microsoft into Office 2007 and has been used in office 2007/2010. The OOXML format enables the generated document to be fully compatible with other cross platform business applications. While it offers greater benefits over its predecessor, its unique internal file structure also opens the door for data hiding in Microsoft Office documents based on OOXML. Due to the popularity of Microsoft Office documents such as word or excel files, they have become a strong preference for mischievous users to cover data for hiding information because these document files could be easily ignored [11].

13.2.1 OOXML Document Fundamentals

OOXML file format consists of a compressed ZIP file, called package. The ZIP compression decreases the size of the document up to 75% and is more robust to error handling [2]. It allows an ease of managing and repairing of individual segmented files within a package. For example, you can open MS Word 2007 document that uses OOXML format, and locates the XML part that represents the body of the Word document. An updated Office document can be created by altering the part using any technology capable of editing XML. An OOXML file is based on the following: Package, part, relationship [3].

Package A package is a Zip container that holds XML and other data parts, as defined by OPC (Open Packaging Conventions) specifications [2, 4]. The package can have different internal directory structure and names depending on the type of the document. Some of the elements are shared across all MS Office applications such as document properties, charts, style sheets, hyperlinks, diagrams, and drawings. Other elements are specific to each application, such as worksheets in Excel, slides in PowerPoint, or headers and footers in Word.

A basic package contains an XML file called "[Content_Types].xml" at the root, along with three directories: "_rels", "docProps", and document type specific directory. For example, in MS Word 2007 document, the "word" directory has been created and contains the "document.xml" file, which is the starting path of the document. These folders have all the files located in the package and zipped together to form a single instance of the document. Every part in a package has a unique URI (Uniform Resource Identifier) part name along with specified content type. A part's content type explicitly defines the type of data stored and reduces ambiguity and duplication issues inherent with file extensions. Package can also include relationships that define association between the package, parts and external resources.

Parts The component parts of an MS Office document correspond to one file in a package. It can be of any type including text, image etc. [2, 4]. The extension ".rels" is reserved for storing relationship information of package parts. It is stored in "/rels" subfolders. Three names are reserved by package for organizing its files, i.e., "_rels" subfolder carrying relationship information with ".rels" file extension and file name "[Content_Type].xml". Where, "[Content_Types].xml" file provides MIME (Multipurpose Internet Mail Extensions) type information for parts used in the OOXML document. It also defines mapping based on the file extensions, along with overrides for specific parts other than default file extensions. This enables an application and third party tools to determine the contents of any part to process accurately. File "docProps/app.xml" contains application centric properties such as application type "Microsoft Office Word" etc. File "docProps/core.xml" contains OOXML document core properties such as machine name, creation and modification dates etc. File "word/document.xml" is the main part of any word document.

Relationships Relationship items specify that how a particular collection parts come together to form a document. This is achieved by verifying connection between source part and target part. For example, through a relationship, a user can identify the connection between a slide and an image that appears on that slide. Relationship files play an important role in MS Office XML formats. Every document part is referred to by at least one relationship. The use of relationships makes it possible to discover how one part relates to another part without looking at the content of the parts.

All relationships, including the relations associated with the root package, are represented as XML files. They are stored inside a package (e.g., _rels\rels). The use of relationships makes it possible to discover how one part relates to another part without looking at the content of the parts. Relationships are composed of four elements: An identifier (Id), an optional source (package or part), relationship type

(URI style expression), and a target (URI to another part). Two types of relationship files usually exist in a package. These are:

/_rels/.rels: Root level "_rels" folder contains relationship file which carries information of parts for the package. For example "_rels/.rels" file defines the starting part of the document, i.e., "word/document.xml".

[partname].rels: Each part may have its own relationships. The part specific relationship can be looked in "word/_rels" subfolder, a sibling of the file with original file name appended to it with ".rels" extension. For example, "word/_rels/document.xml.rels".

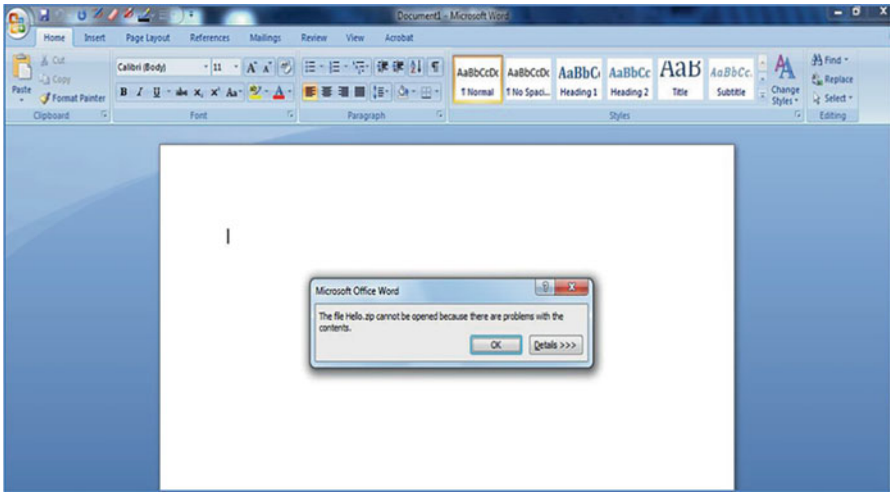
A typical package relationships file ".rels" contains XML code. For simplicity, we only present XML code for "document.xml" part as follows:

```
<Relationships
 xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
 <Relationship Id="rId1"
 Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/of
 ficeDocument" Target="word/document.xml" />
  </Relationships>
```

In above code, "Relationship Id" attribute value "rId1" is default for main document part, which is the starting part of a document. Once the document is being launched, the OOXML editor looks for an OOXML parser depending on document type. In this case, the type specifies that MS Word ML is used for MS Word document. Another attribute "Target" specifies path or location of beginning part, i.e., "document.xml".

13.2.2 Data Hiding in OOXML Documents

Data hiding in OOXML can be classified into different categories: Data hiding using OOXML relationship structure, data hiding using XML format features, data hiding using XML format features and OOXML relationship structure, data hiding using OOXML flexibility for embedded resource architecture, and data hiding using OOXML flexibility of swapping parts. We use MS Word 2007 document, as an example, to illustrate these techniques in this section. Notably, the methodology can be easily extended to any documents in MS Office 2007 or OOXML file format.



Microsoft Office Word

The file Hello.zip cannot be opened because there are problems with the contents.

Fig. 13.5 MS Office 2007 raised an error stated that problems with the contents

13.2.2.1 Data Hiding Using OOXML Relationship Structure

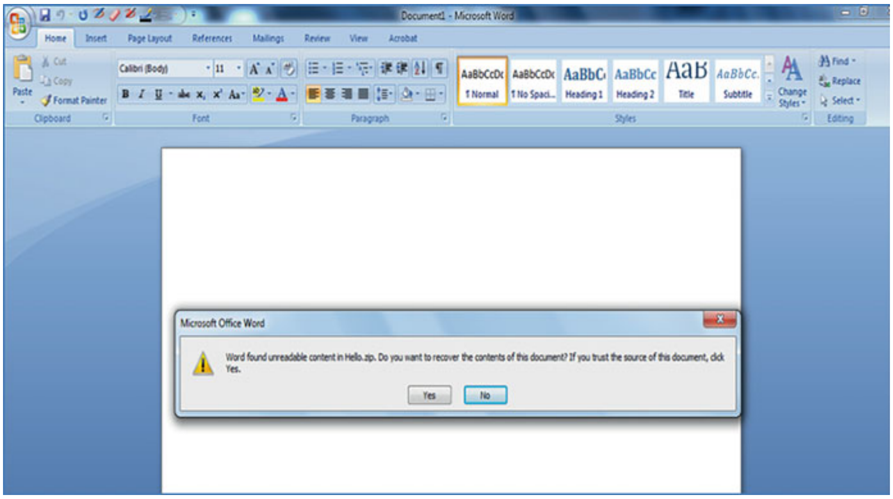
As described above, the MS Office 2007 document is composed of xml and other files. These files are known as parts and compressed together using ZIP format. These parts are also organized using the relationship information found in relationship files inside an OOXML document. To satisfy relationships within a document, all parts have to be a target of valid relationship entry. Parts, which are not the target of a valid relationship entry, are treated as an unknown part [5]. These parts are not to be ignored when reading the document by MS Office 2007 application. They raised an error that the document is corrupt as shown in Fig. 13.5. MS Office 2007 also facilitates user by giving an option to recover the document and removes the unknown parts, as shown in Fig. 13.6.

In similar way, any relationship not defined within the ECMA376 Standard is considered to be an unknown relationship. These relationship entries are accepted in OOXML document and raise no errors. Documents containing unknown relationship information are opened normally and the relationship entry can be found inside a relationship file.

Next we provide specific example of data hiding in MS Word 2007 document by using information of OOXML document relationship structure. The OOXML document works as a carrier for the hidden data.

Step 1: Unzip the OOXML document using any zip utility software. This shows that OOXML document contains several XML files and other objects.

Step 2: Insert files which wish to hide in the unzipped OOXML document archive. These files can be added to any folder or sub folder of the document.



Microsoft Office Word

Word found unreadable content in Hello.zip. Do you want to recover the contents of this document? If you trust the source of this document, click Yes.

Fig. 13.6 MS Office 2007 gives an option to recover the document by removing unknown parts

Step 3: Define types of added files into OOXML documents content type file. It is not necessary to define file types multiple times if it already exists in the content type file.

Step 4: Define relationship entry for inserted files into package relationship file, i.e., "_rels/.rels". The attributes of relationship entries are "Id", "Type", and "Target". The relationship attribute "Id" must be unique as it connects the document and the target files, whereas "Type" attribute is some character which is not defined in OOXML standard such as "a", "b" etc. The "Target" attribute contains the complete path or location from the document's root folder for the inserted files.

Step 5: all the files are zipped together with an extension of ".zip", which later is renamed to ".docx".

For example, we created a document containing some text and image and saved it as "Uparts&Rels". Next, by using WinZip utility we unzipped this document and inserted "sysinternals.zip", "mask.jpeg" and "BYE.mp3" as our hidden files in the root folder of the document. After inserting these files we updated the "[Content_Types].xml" file for inserting files types. The content type file can be found inside root folder named "[Content_Types].xml". Before opening a document, the OOXML parser validates that the files present in the package are defined in "[Content_ Types].xml" file. The code for defining hidden file types into "[Content_Type].xml" file is highlighted with existing code in Fig. 13.7.

The package level relationship file "_rels/.rels" is used for creating relationships of hidden files within a package. The relationship entry attributes such as Id, Type, and Target are defined in relationship file, as shown in Fig. 13.8.

As seen above, the relationship Id of "BYE.mp3" is "rd102" and Type is "http://schemas.openxmlformats.org/officeDocument/2006/Relationships/c". We use values that do not exist in the OOXML specifications such as "a, b, c", etc., when

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<Types
  xmlns="http://schemas.openxmlformats.org/package/2006/content-
  types">
<Override PartName="/word/footnotes.xml" ContentType="......." />
<Default Extension="jpeg" ContentType="image/jpeg" />
<Default Extension="rels" ContentType="application/vnd.openxmlformats-
  package.relationships+xml" />
 <Default Extension="xml" ContentType="application/xml" />
<Default Extension="zip" ContentType="application/zip" />
<Default Extension="mp3" ContentType="application/mp3" />
 <Default Extension="jpg" ContentType="application/jpg" />
<Override PartName="/word/document.xml" ContentType="......." />
  ............ . .
  </Types>
```

Fig. 13.7 Modified "[Content_Types].xml" file

```
 <?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

  <Relationships
  xmlns="http://schemas.openxmlformats.org/package/2006/relationships">

 <Relationship Id="rId3" Type="..." Target="docProps/app.xml" />

 <Relationship Id="rId2" Type="..." Target="docProps/core.xml" />

 <Relationship Id="rId1" Type="..." Target="word/document.xml" />

 <Relationship Id="rId100"
  Type="http://schemas.openxmlformats.org/officeDocument/2006/Relationships/a
  " Target="word/media/sysinternals.zip" />

 <Relationship Id="rId101"
  Type="http://schemas.openxmlformats.org/officeDocument/2006/Relationships/b
  " Target="mask.jpg" />

 <Relationship Id="rId102"
  Type="http://schemas.openxmlformats.org/officeDocument/2006/Relationships/c"
  Target="word/BYE.mp3" />

 </Relationships>
```

Fig. 13.8 Modified relationship file (.rels)

setting a type. After these modifications, the OOXML document is opened normally without a warning. Also, if user amends the document and updates it, the hidden data remains in the document. The MS Office application considers unknown parts and unknown relationships as valid parts and relationships of a package.

As explained earlier, hidden data must satisfy all the relationships in the package. Otherwise, it is visible in document. Also, if only an inserted file type is defined in Content Type file and its relationship is not created in relationship file, the document opens normally. Office application does not give any warning. Inserted file still exists inside the package. In this case, if user updates or makes some changes in the word document, then the inserted file will be eliminated by the application. So for keeping the existence of inserted file, its relationship needs to be created in relationship file.

This data hiding method is the natural result of an explicit relationship in OOXML. The key point of this hiding process is to assign a fresh Id to new target. This results the target being overlooked by the MS Office application. The new Id is not referenced in the relationship part. So the main source part is not aware of the new content. Then, the hidden data is not shown on screen and neither can it be eliminated by MS Office application because these hidden data has an Id and satisfies relationship structure of OOXML document. At this point, if relationships between main MS Office document file and hidden data are defined, the hidden data becomes more difficult to discover.

This data concealment approach also sidesteps the document inspection feature "Inspect document" available in MS Office 2007 applications.

13.2.2.2 Data Hiding Using XML Format Feature

XML comment feature is used to leave a note or to temporarily edit out a portion of XML code. Although XML is supposed to be self-describing data, you may still come across some instances where an XML comment might be necessary. OOXML documents do not generate any comments in XML files whereas XML comments feature can easily be used for data hiding purpose by some mischievous user. This involves technique of adding comments directly to zip archive using comment feature of zip file format and adding XML comments to the XML file [6]. In both cases, these comments are ignored by MS Office 2007 application and these comments are discarded when document is written back out.

For Example, we created one document contains some text and an image. Then, unzip it using any zip utility. A secret message is added using comments feature in one of the XML file. The secret message is as follows:

<!-- This is a secret message-->

The secret message is encoded using base64 encoding scheme. It turns to be as follows:

PCEtLSBUaGlzIGlzIGEgc2VjcmV0IG1lc3NhZ2UtLT4=

We add this data to any of the XML file. The comments cannot appear at the very top of the document in XML according to XML standard. Only the XML declaration can come first such as:

Document inspection feature of MS Office 2007 allows removing of comments from the document. Whereas, by using this approach, the document inspection feature fails to identify and remove comments embedded using base64 encoding scheme. This technique also enables to hide some file in an OOXML document using base64 encoding scheme. The quality of this data hiding technique is relatively low as XML files carry base64 encoded data. They can be easily noticed if the document is unzipped.

13.2.2.3 Data Hiding Using XML Format Feature and OOXML Relationship Structure

Ignorable attribute is also an important feature of MS Office 2007 applications [4]. Compatibility rules are very important for any associated XML element. Compatibility rules are associated with an element by means of compatibility rule attributes. These control how MS Office 2007 parser shall react to elements or attributes from unknown namespaces. The principal compatibility rule attribute is the Ignorable attribute. MS Office 2007 treats the presence of any unknown element or attribute as an error condition by default [4, 6]. However, unknown elements or attributes identified in an Ignorable attribute shall be ignored without error.

The Ignorable attribute specifies which XML namespace prefixes encountered in a markup file may be ignored by an OOXML processor. Elements or attributes, where the prefix portion of the element name is identified as "**ve:Ignorable**", will not raise an error when processed by an OOXML processor. The "**ve**" XML namespace is the recommended prefix convention to use when mapping the XAML (Extensible Application Markup Language) compatibility namespace "http://schemas.openxmlformats.org/markup-compatibility/2006". The "ve:Ignorable" attribute supports markup compatibility both for custom namespace mapping and for XML versioning.

This data concealment technique supports any kind of data such as image, audio or video file to be hidden by taking advantage of XML format feature and OOXML relationship structure. This technique coerces XAML parser to treat that element and attributes that do not exist. It will not generate an error. By default, Ignorable element is entirely ignored including its attributes and contents. This data concealment process requires the following steps.

Step 1: An OOXML document is unzipped using WinZip utility.
Step 2: Add an image which needs to be hidden in sub folder named "word/media". This sub folder usually contains all images used inside a document.
Step 3: Create metadata for hidden image inside main document file "document.xml" to make it look legitimate.
Step 4: Use ignorable attribute to define this in declaration section of main document file, i.e., "document.xml". Place this tag before and after creating metadata for hiding image.
Step 5: Amend the part relationship file "document.xml.rels" with creating relationship attributes such as "Id", Type" and "Target". The "Id" must be unique. Other attributes "Type" and "Target" contain information of image type and location.

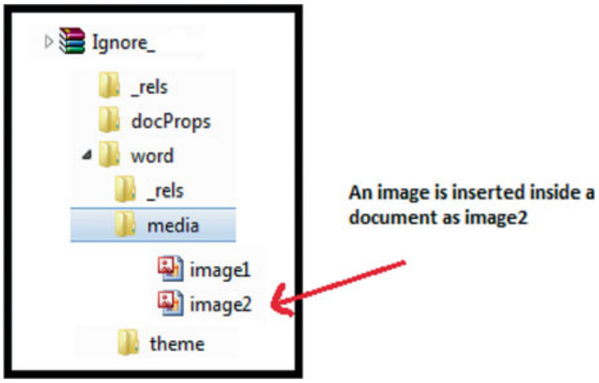


Fig. 13.9 Added Image shown with other files in OOXML document

Step 6: All the files are zipped together with an extension of ".zip", which later is renamed to ".docx".

For example, we create and save a document containing image and some text data, named as "Ignore.docx". After unzipping this document, we add an image which needs to be hidden inside a document under "word/media" subfolder as shown in Fig. 13.9.

After inserting an image in "word/media" subfolder, the main document file is updated with the metadata code for an inserted image. For simplicity, we copy the metadata code of first image inserted using MS Office application and paste it as a separate block for hidden image in "document.xml" file. Ignorable attribute is defined inside a main document declaration section to hide this image, as shown in Fig. 13.10. The code highlighted in red is used to define ignorable namespace, which markup consumer does not understand. After defining the ignorable attribute namespace, the ignorable tag is placed before and after the metadata of second image need to be hidden, as shown in Figs. 13.10 and 13.11. The hidden image is "Garden.jpeg". Its metadata tag in document.xml file is shown in Fig. 13.12.

Finally, the part relationship file "document.xml.rels" is updated with a valid relationship entry of an inserted image along with its type and target information. In this case, the highlighted relationship entry having Id "rId5" is added in relationship file with type "http://schemas.openxmlformats.org/officeDocument/2006/relationships/image" and target "media/image 2.jpeg" values. It is shown in Fig. 13.13 with highlighted code for this entry.

The image is successfully hidden by using Ignorable attribute. However, only using XML feature such as ignorable attribute does not guarantee of keeping hidden image with the document. The relationship must be created to keep this hidden image intact with the document. The content type file is not required to amend with image type if same type of image is inserted. If inserted image type is different from the first image, the content type file needs to be updated with hidden image type. The sample code for updating content type is as follows: The "[Content_Types].xml" file is located at root level in the package.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<w:document xmlns:ve="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:o="urn:schemas-microsoft-com:office:office"

xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"

xmlns:m="http://schemas.openxmlformats.org/officeDocument/2006/math"

xmlns:v="urn:schemas-microsoft-com:vml"

xmlns:wp="http://schemas.openxmlformats.org/drawingml/2006/wordprocessingDrawing"

xmlns:w10="urn:schemas-microsoft-com:office:word"

xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"

xmlns:wne="http://schemas.microsoft.com/office/word/2006/wordml"

xmlns:p1="http://schemas.openxmlformats.org/MyExtension/p1" ve:Ignorable="p1" >
```

Fig. 13.10 Beginning section of main document file “document.xml”

```
<w:sectPr w:rsidR="00401AD8" w:rsidRPr="0091776E" w:rsidSect="00DC51FF">

<w:pgSz w:w="12240" w:h="15840" />

<w:pgMar w:top="1440" w:right="1440" w:bottom="1440" w:left="1440" w:header="720"

w:footer="720" w:gutter="0" />

<w:cols w:space="720" />

<w:docGrid w:linePitch="360" />

</w:sectPr>

</w:body>

</w:document>
```

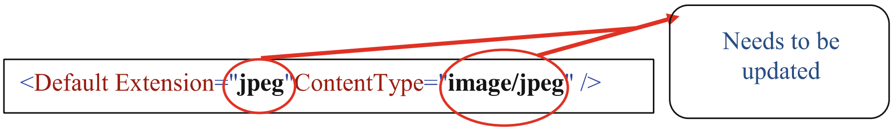
Fig. 13.11 End section of main document file “document.xml”

```
<p1:IgnoreMe>
<w:p w:rsidR="00401AD8" w:rsidRPr="0091776E"
w:rsidRDefault="0091776E" w:rsidP="0091776E">
<w:pPr>
<w:tabs>
<w:tab w:val="left" w:pos="1155" />
</w:tabs>
</w:pPr>
<w:r>
<w:lastRenderedPageBreak />
<w:drawing>
<wp:inline distT="0" distB="0" distL="0" distR="0">
<wp:extent cx="5943600" cy="4457700" />
<wp:effectExtent l="19050" t="0" r="0" b="0" />
<wp:docPr id="2" name="Picture 1" descr="Garden.jpg" />
<wp:cNvGraphicFramePr>
<a:graphicFrameLocks
xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main"
noChangeAspect="1" />
</wp:cNvGraphicFramePr>
<a:graphic
xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main">
<a:graphicData
uri="http://schemas.openxmlformats.org/drawingml/2006/picture">
<pic:pic
xmlns:pic="http://schemas.openxmlformats.org/drawingml/2006/picture">
<pic:nvPicPr>
<pic:cNvPr id="0" name="Garden.jpg" />
<pic:cNvPicPr />
</pic:nvPicPr>
<pic:blipFill>
<a:blip r:embed="rId5" cstate="print" />
<a:stretch>
<a:fillRect />
</a:stretch>
</pic:blipFill>
<pic:spPr>
<a:xfrm>
<a:off x="0" y="0" />
<a:ext cx="5943600" cy="4457700" />
</a:xfrm>
<a:prstGeom prst="rect">
<a:avLst />
</a:prstGeom>
</pic:spPr>
</pic:pic>
</a:graphicData>
</a:graphic>
</wp:inline>
</w:drawing>
</w:r>
</w:p>
</p1:IgnoreMe>
```

Fig. 13.12 Sample metadata of hidden image

```
 <?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<Relationships
    xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
<Relationship Id="rId3"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/web
    Settings" Target="webSettings.xml" />
<Relationship Id="rId7"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/the
    me" Target="theme/theme1.xml" />
<Relationship Id="rId2"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/setti
    ngs" Target="settings.xml" />
<Relationship Id="rId1"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/styl
    es" Target="styles.xml" />
<Relationship Id="rId6"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/font
    Table" Target="fontTable.xml" />
<Relationship Id="rId5"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/ima
    ge" Target="media/image2.jpeg" />
<Relationship Id="rId4"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/ima
    ge" Target="media/image1.jpeg" />
  </Relationships>
```

Fig. 13.13 Relationship file showing entry for hidden image

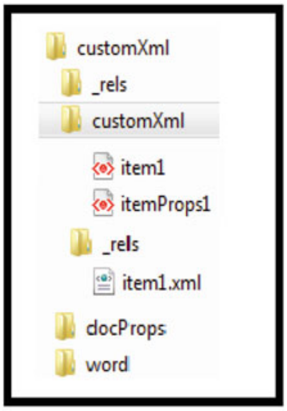


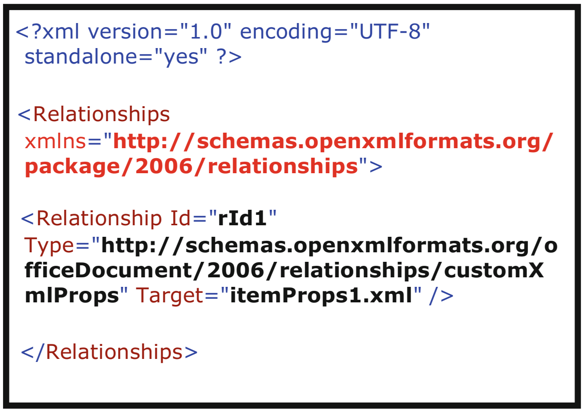
```
<Default Extension="jpeg" ContentType="image/jpeg" />
```

Data hiding using this technique is really hard to trace as it conforms all the association to the main document file. The relationship entry also exists in relationship file. Additionally, file does not raise any doubt of illegitimate data hidden inside an OOXML document with the insertion of metadata code of hidden image in “document.xml”.

13.2.2.4 Data Hiding Using OOXML Flexibility for Embedded Resource Architecture

The Custom XML feature is one of the most powerful features of OOXML documents for business scenario and document centric solutions [6]. It supports integration of documents with business process and data to get true interoperability of





```
<?xml version="1.0" encoding="UTF-8"
 standalone="yes" ?>

<Relationships
 xmlns="http://schemas.openxmlformats.org/
 package/2006/relationships">

<Relationship Id="rId1"
 Type="http://schemas.openxmlformats.org/o
 fficeDocument/2006/relationships/customX
 mlProps" Target="itemProps1.xml" />

</Relationships>
```

Document contains CustomXML data generated by MS Office.

customXml part relationship file

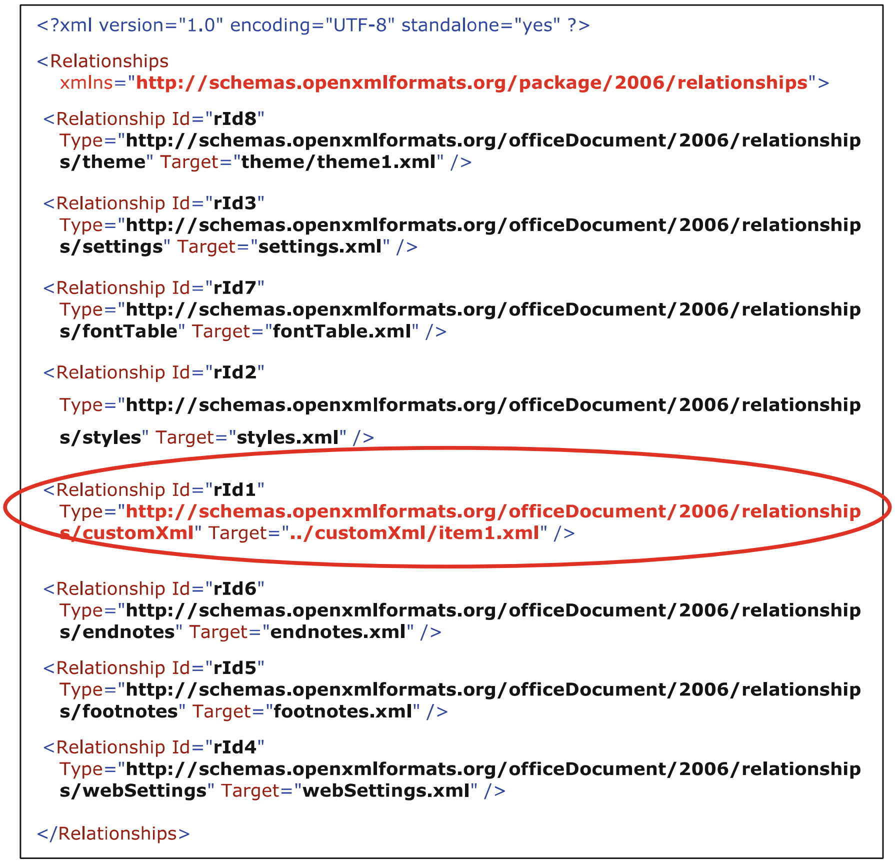
Fig. 13.14 Document Structure with Custom XML data and its part relationship code

documents. This allows you to embed business semantics in such a way that it is discoverable. Implementers not interested in using that feature can skip over it easily. You can even easily extract your data using one simple generic XSLT (Extensible Stylesheet Language Transformations). A package is permitted to contain multiple custom XML data storage parts.

The ability to embed and interweave business data into transportable and humanly readable documents is extremely useful. Take for instance the efforts to standardize the embedding of patient medical data into PDF documents, (aka PDF/H). Records For Living has been able to take advantage of Open XML's capabilities with regards to its support of custom schemas to integrate two industry standards: Ecma's Open XML and the ASTM's Continuity of Care Record (CCR). The combination is powerful: Patients can use personal health record (PHR) software to exchange live reports with their doctors in a way that is both human and machine readable [1].

This feature also empowers data concealment using OOXML flexibility for embedded resources inside MS Office 2007 document. The Custom XML data is also generated by OOXML document in some cases. The object embedding feature of OOXML document requires generation of Custom XML data for handling information of that object. The default layout of the unzip OOXML document which contains Custom XML data is shown in Fig. 13.14.

By default an OOXML document creates custom XML folder at root to store custom XML data files. This folder contains two XML files, "item1.xml" and "itemProps1.xml". The part relationship file of custom XML data is generated at "customXml" folder under "customXml/_rels" subfolder as "item1.xml.rels". It contains relationship entry for "itemProps1.xml" file, as shown in Fig. 13.15. The part relationship file of main document "document.xml.rels" is also updated with an



```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<Relationships
   xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
 <Relationship Id="rId8"
   Type="http://schemas.openxmlformats.org/officeDocument/2006/relationship
   s/theme" Target="theme/theme1.xml" />
 <Relationship Id="rId3"
   Type="http://schemas.openxmlformats.org/officeDocument/2006/relationship
   s/settings" Target="settings.xml" />
 <Relationship Id="rId7"
   Type="http://schemas.openxmlformats.org/officeDocument/2006/relationship
   s/fontTable" Target="fontTable.xml" />
 <Relationship Id="rId2"
   Type="http://schemas.openxmlformats.org/officeDocument/2006/relationship
   s/styles" Target="styles.xml" />
 <Relationship Id="rId1"
   Type="http://schemas.openxmlformats.org/officeDocument/2006/relationship
   s/customXml" Target="../customXml/item1.xml" />
 <Relationship Id="rId6"
   Type="http://schemas.openxmlformats.org/officeDocument/2006/relationship
   s/endnotes" Target="endnotes.xml" />
 <Relationship Id="rId5"
   Type="http://schemas.openxmlformats.org/officeDocument/2006/relationship
   s/footnotes" Target="footnotes.xml" />
 <Relationship Id="rId4"
   Type="http://schemas.openxmlformats.org/officeDocument/2006/relationship
   s/webSettings" Target="webSettings.xml" />
</Relationships>
```

Fig. 13.15 Part relationship file code "document.xml.rels"

entry of second custom XML file, i.e., "item1.xml" with Id "rId1" as shown in Fig. 13.15.

The steps of data hiding using Custom Xml feature are as follows:

Step 1 Create and save an OOXML document named "CustomXML.docx". It contains text and image.

Step 2 Create a folder at root named "customXml" and insert some text file needed to hide in OOXML document. The text file is in XML format, which looks legitimate Custom XML data.

Step 3 Create a subfolder named "customXml/_rels" and one relationship file for hidden text file.

Step 4 All the files are zipped together with an extension of ".zip". Later, it is renamed as ".docx".

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<w:document
xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships/cus
tomXml"
xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
<w:body>
<w:p w:rsidR="00DC51FF" w:rsidRDefault="00C2303E">
<w:r>
<w:t>Hidden Secret Message!!!!!</w:t>
</w:r>
</w:p>
</w:body>
```

Fig. 13.16 Sample code of “hiddendata.xml” file

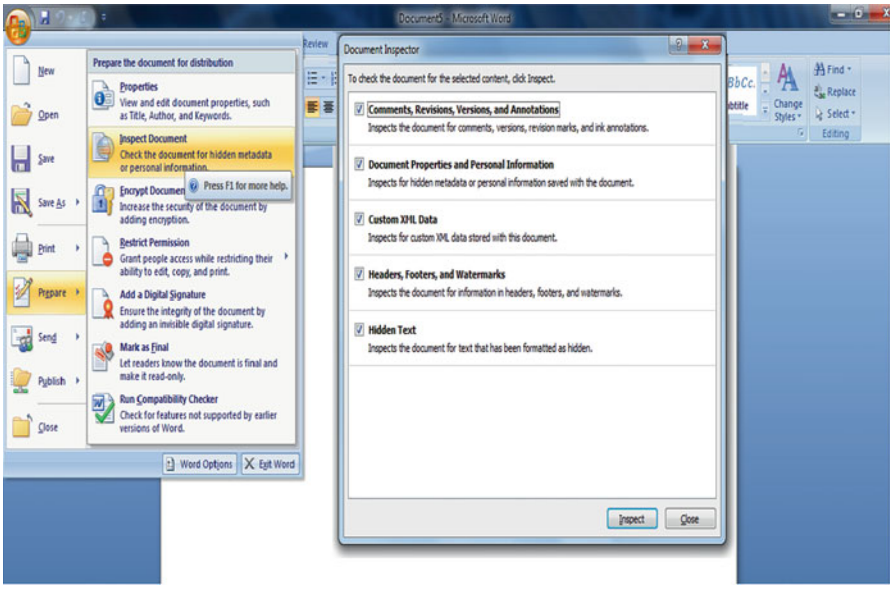
```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<Relationships

xmlns="http://schemas.openxmlformats.org/package/2006/relationships">

 <Relationship Id="rId100"

Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/custom

XmlData" Target="/customXML/test.xml" />

 </Relationships>
```

Fig. 13.17 Sample code of customXml part relationship file

For example, we use this technique to hide some text data inside MS Office document. We create “customXml” folder at root of the package and insert one text file named “hiddendata.xml”, as shown in Fig. 13.16.

Next we satisfy its relationship constraint by creating its relationship file inside a “customXml/ _rels” subfolder, named as “hiddendata.xml.rels” shown in Fig. 13.17. If we do not skip its entry in part relationship file of main document, the Inspect Document feature can easily identify the custom XML data. It allows user to discard the custom XML data.

This technique allows insertion of any text file in xml format contains hidden data with its internal sub-relationship. The key point to identify whether it is hidden data or not is to see that whether CustomXml type entry is presented in part relationship



Prepare the document for distribution

Properties
View and edit document properties, such as Title, Author, and Keywords.

Inspect Document
Check the document for hidden metadata or personal information.

Encrypt Document
Increase the security of the document by adding encryption.

Restrict Permission
Grant people access while restricting their ability to edit, copy, and print.

Add a Digital Signature
Ensure the integrity of the document by adding an invisible digital signature.

Mark as Final
Let readers know the document is final and make it read-only.

Run Compatibility Checker
Check for features not supported by earlier versions of Word.

Document Inspector

To check the document for the selected content, click Inspect.

[✓] Comments, Revisions, Versions, and Annotations
Inspects the document for comments, versions, revision marks, and ink annotations.

[✓] Document Properties and Personal Information
Inspects for hidden metadata or personal information saved with the document.

[✓] Custom XML Data
Inspects for custom XML data stored with this document.

[✓] Headers, Footers, and Watermarks
Inspects the document for information in headers, footers, and watermarks.

[✓] Hidden Text
Inspects the document for text that has been formatted as hidden.

Inspect Close

Fig. 13.18 Inspect document feature of MS Office 2007

file of main document. The hidden data under customXml folder is not linked with the main document file "document.xml".

The MS Office document gives feature that allows you to remove custom XML data associated with the document. This feature is named as Inspect document, which removes custom XML data, hidden data and other personal information from MS Office document. The snapshot of this feature is shown in Fig. 13.18. The inspect document feature functionality is to search customXml type in part relationship file of main document ("document.xml.rels"). Once it is found, the associated data using target information is deleted. Also, customXml folder with its contents is discarded. The document inspection feature of MS Office 2007 is unable to detect custom XML data in this scenario as it sidesteps the document inspection feature of MS Office 2007 application.

13.2.2.5 Data Hiding Using OOXML Flexibility of Swapping Parts

Images are the most popular cover objects used for data hiding. Many image file formats exist for different applications. The MS Office 2007 uses "png", "jpeg", "gif" and "emf" formats for storing images inside a document [7]. The flexibility of data hiding using OOXML architecture of swapping parts allows swapping of images between two OOXML documents. It supports two data hiding scenarios as follows.

Scenario 1

Step 1 The OOMXL document is unzipped using WinZip utility.
Step 2 Swap an image with the original image found in "word/media" subfolder. Ensure that the swapped image follows the same name of the original image. The inserted image is transformed according to OOXML image transformation standard before swapping.
Step 3 The content type file is needed to be updated if swapped image is of another format.
Step 4 All the files are zipped together with an extension of ".zip" and later renamed as ".docx".

Scenario 2

Step 1 The OOMXL document is unzipped using WinZip utility.
Step 2 An original image found inside "word/media" subfolder is used to embed files using any image stego software.
Step 3 All the files are zipped together with an extension of ".zip" and later renamed as ".docx".

The swapped image has to be transformed or compressed first before swapping. Otherwise, OOXML document raises an error. The best way is to add desired image into MS Office 2007 document and save the document. An image is automatically transformed by the application in this way. Later, unzip this document and swap the image with another image presented in different document, as shown in Fig. 13.19.

This facilitates mischievous users to swap images of two documents. They can embed files into an existing image by using any available image steganography tool.

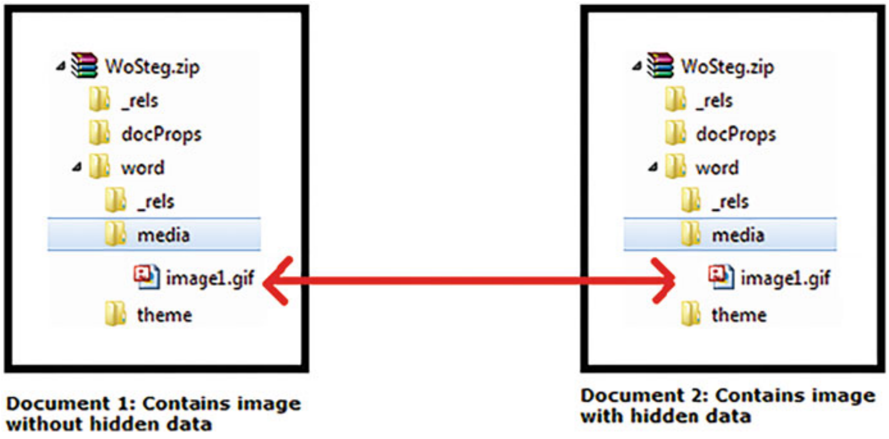


Fig. 13.19 Swapping of image between two OOXML documents

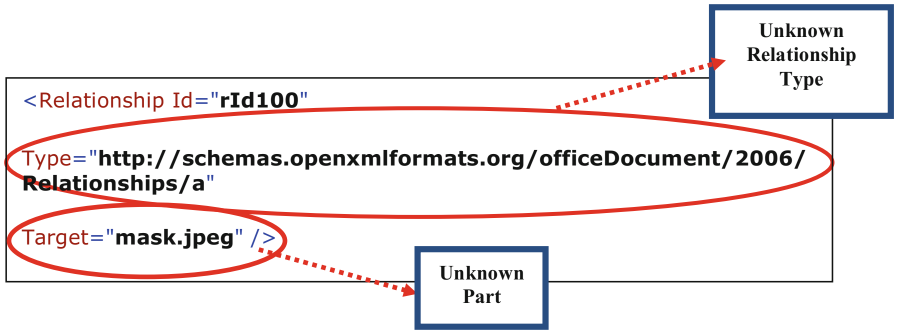
Swapping of images cannot be detected by using any feature of MS Office 2007 application. Hence, it seems that manual scrutiny of the images is required to ensure that it does not carry any hidden data. MS Office 2007 is unable to detect the changes made into an image file.

13.2.3 Hidden Data Detection in OOXML Documents

Several types of hidden data and personal information can be saved in an MS Office 2007 document [8, 9]. This information might not be immediately visible when viewing the document in MS Office 2007 application. But it might be possible for other people to detect the information. The detection logic of hidden data is entirely dependent on the data hiding techniques. It involves investigation of multiple XML files of OOXML document. In this section, we describe several hidden data detection methods in OOXML documents based on the above data hiding techniques.

13.2.3.1 Detecting Hidden Data Using OOXML Relationship Structure

This technique requires detection logic to ensure that all package parts and relationships are verifiable against the OOXML standard. This also confirms that all parts must associate with their relevant counter parts or the main document file. The properties of unknown parts and unknown relationships can be used for detecting hidden data by using this approach. It requires detection query to scan both the relationship files, package level ".rels" and part level "document.xml.rels". This identifies relationship type, which is not defined in the ECMA-376 standard. This undefined relationship "Type" is separated with its attributes such as "Id" and "Target". The "Target" attribute identifies the unknown part. Its location is shown in Fig. 13.20.



```
<Relationship Id="rId100"
Type="http://schemas.openxmlformats.org/officeDocument/2006/
Relationships/a"
Target="mask.jpeg" />
```

Fig. 13.20 Unknown relationship and unknown part

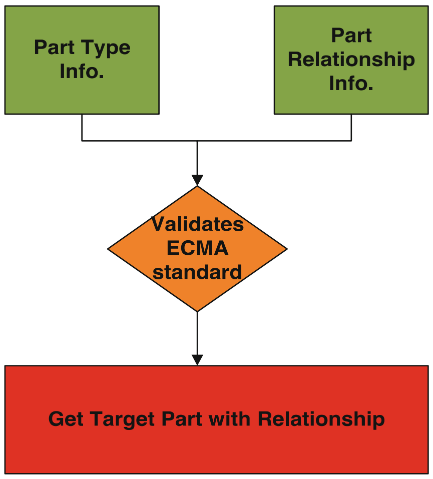


Fig. 13.21 Detection logic for unknown parts and unknown relationships technique

The detection logic of unknown parts and unknown relationships is explicitly shown in Fig. 13.21.

13.2.3.2 Detecting Hidden Data Using XML Format Feature and OOXML Relationship Structure

This logic discovers the hidden data by using ignorable attribute. In order to detect the hidden image, the query first scans main document file “document.xml” for ignorable attribute. If it is found, all the metadata code inside ignorable attribute tag is separated. As the main document file “document.xml” contains “r:embed” attribute, which carries same value of part relationship file “document.xml.rels” attribute “Id”. In this metadata code, the query looks for “r:embed” attribute value and matches it with part relationship file “Id” attribute value. The matched relationship “Id” attribute “Target” contains name and location of hidden image using ignorable attribute. The OOXML parser ignores processing of metadata tags within ignorable attribute. It is worthy mentioning that hidden image inside the document seems to be normal as its corresponding metadata also presents in main document file along with its relationship information in part relationship file.

13.2.3.3 Detecting Hidden Data Using OOXML Flexibility For Embedded Resource Architecture

The package relationship file “.rels” and part relationship file “document.xml.rels” are used to validate and ensure that all the parts inside the package are associated with main document file. The unassociated parts with main document are considered as hidden data. They can be detected by checking its relationship with the main

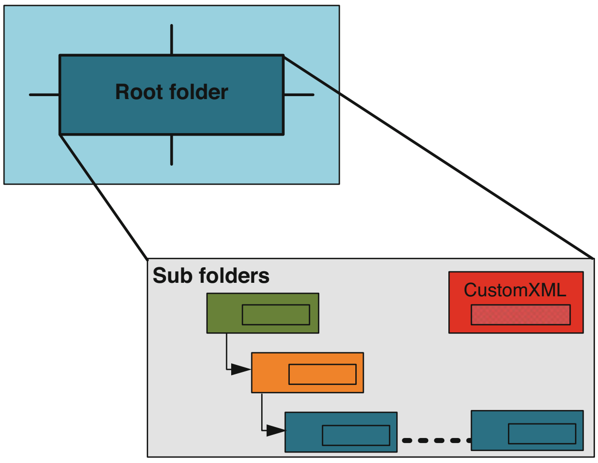


Fig. 13.22 Detection logic for CustomXml technique

document exists. The custom XML data generated by the word document is not detected as hidden data, as shown in Fig. 13.22.

From the viewpoint of computer forensics, it is very important to confirm the existence of any data, which are not examined by specific applications. In computer forensic investigations, investigators may assume that all data in electronic document files can be examined by their applications. However, it is not enough to investigate electronic document files only through their associated applications. Because data, that cannot be examined or detected by most applications, can still exist in the file.

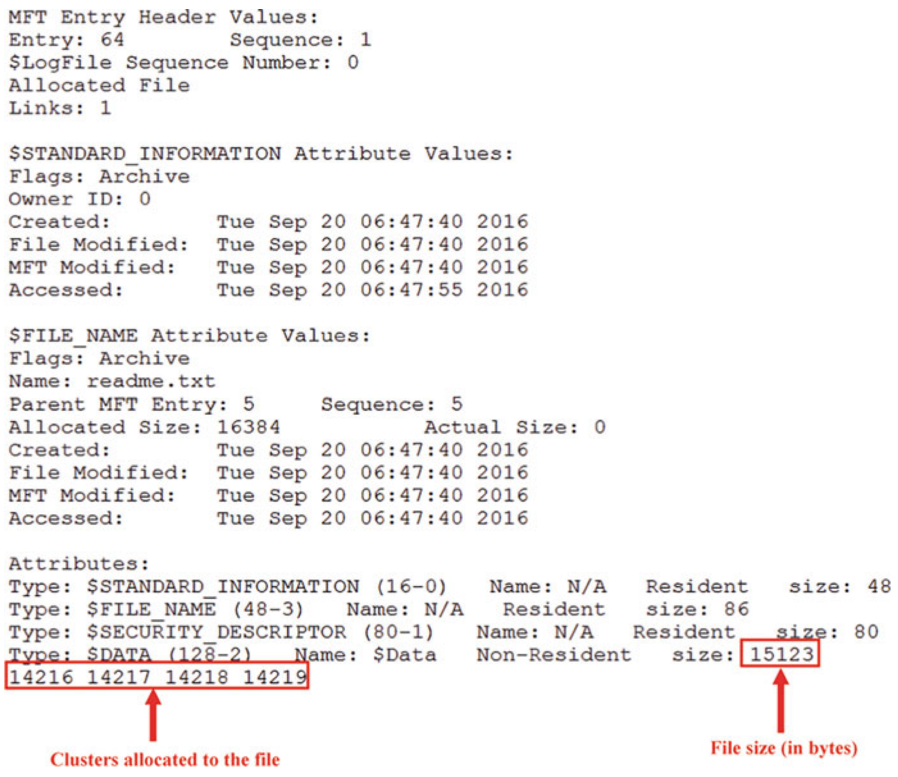
Notably, most of the descriptions focus on MS Word 2007 files in this chapter. However, in the case of MS Office PowerPoint and Excel 2007 files, these hidden data can also be detected using the same algorithms.

Review Questions

1. Which is the following statements about slack space is true? ______________

   (a) Slack space is considered unallocated space.
   (b) Slack space is only in the end of the last cluster allocated to a file.
   (c) Slack space is only in the end of the first cluster allocated to a file.
   (d) Slack space is equivalent to volume slack.

2. Describe in your own words, what is slack space?
3. Describe in your own words, what is volume slack?
4. What is orphaned cluster?
5. What is Alternate Data Stream?
6. The following figure shows an output of the TSK meta data layer tool *istat*



```
MFT Entry Header Values:
Entry: 64        Sequence: 1
$LogFile Sequence Number: 0
Allocated File
Links: 1

$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Created:        Tue Sep 20 06:47:40 2016
File Modified:  Tue Sep 20 06:47:40 2016
MFT Modified:   Tue Sep 20 06:47:40 2016
Accessed:       Tue Sep 20 06:47:55 2016

$FILE_NAME Attribute Values:
Flags: Archive
Name: readme.txt
Parent MFT Entry: 5     Sequence: 5
Allocated Size: 16384          Actual Size: 0
Created:        Tue Sep 20 06:47:40 2016
File Modified:  Tue Sep 20 06:47:40 2016
MFT Modified:   Tue Sep 20 06:47:40 2016
Accessed:       Tue Sep 20 06:47:40 2016

Attributes:
Type: $STANDARD_INFORMATION (16-0)   Name: N/A   Resident   size: 48
Type: $FILE_NAME (48-3)   Name: N/A   Resident   size: 86
Type: $SECURITY_DESCRIPTOR (80-1)   Name: N/A   Resident   size: 80
Type: $DATA (128-2)   Name: $Data   Non-Resident   size: 15123
14216 14217 14218 14219
```

Suppose that the cluster size is 4 KB. Please answer the following questions by filling in the blanks with the correct response.

(a) What is the number of clusters allocated to the file? ___________________
(b) What is the size of the file? (in bytes) ___________________
(c) There is slack space in the file? (Yes/No) _____________________
(d) If so, what is the size of slack space for the file? (in bytes) ___________________ and, which cluster contains the slack space for the file? (in bytes)___________________

13.3 Practical Exercise

The objective of this exercise is to learn how to hide data as well as recover hidden data using some open source tool.

13.3.1 Setting Up the Exercise Environment

For this exercise, we will use a data hiding tool, Bmap, which can utilize slack space to hide data. You download Bmap from the following site onto Forensic Workstation and install it

https://packetstormsecurity.com/files/17642/bmap-1.0.17.tar.gz.html

13.3.2 Exercises

Create a 1058 byte file of random data, called "myfile.dat" in the current working directory using the following command.

```
> dcfldd if=/dev/urandom of=myfile.dat bs=1 count=1058
```

where /dev/urandom is a special file (device) that serves as a pseudo random number generator, a PRNG, in Linux.

Obtain the actual file size of the file "myfile.dat" using ls command:

```
> ls -l myfile.dat
```

Q1. What is the file size of "myfile.dat"? (in bytes) ________________

Obtain the size of disk space allocated to the file "myfile.dat" using du command:

```
> du myfile.dat
```

Q2. What is the size of the disk space used by the file "myfile.dat"? (in bytes) ________________

Note that the du command measures file space in 1 KB units by default. It means that if the output displayed is 8, then the size will be 8×1024 bytes = 8192 bytes or 8 KB.

Q3. What is the size of slack space for the file? (in bytes) ________________

Next, use bmap to hide a secret message "This is Secret Message" into slack space of the file using the following command:

```
> echo "This is Secret Message" | ./bmap --mode putslack myfile.dat
stuffing block 1110943
file size was: 1058
slack size: 3038
block size: 4096
```

For simplicity, we assume that you have installed bmap in the current directory.

Note that if you view the content of the file "myfiel.dat", for example, using the cat command, you will not find the secret message you have hidden inside it.

Next, use bmap to reveal the hidden message in the slack space using the following command:

```
> ./bmap --mode slack myfile.dat
getting from block 1110943
file size was: 1058
slack size: 3038
block size: 4096
This is Secret Message
```

It can be observed that the hidden secret message has been successfully recovered.

References

1. F. Rice, "Open XML file formats," *Microsoft Corporation*, online at http://msdn.microsoft.com/en-us/library/aa338205.aspx, last accessed April 14, 2011.
2. A. Castiglione, A. De Santis and C. Soriente, "Taking advantages of a disadvantage: Digital forensics and steganography using document metadata," *Journal of Systems and Software,* vol. 80, no. 5, pp. 750-764, May 2007.
3. B. Park, J. Park, S. Lee, "Data concealment and detection in Microsoft Office 2007 files," digital investigation, no. 5, pp. 104–114, 2009.
4. "ECMA OOXML documentation," online at http://www.ecma-international.org/publications/standards/Ecma-376.htm, last accessed April 14, 2011.
5. B. Park, J. Park and S. Lee, "Data Concealment & Detection in Microsoft Office 2007 files," *Digital Investigation,* vol. 5, no. 3/4, pp. 104–114, March 2009.
6. S.L. Garfinkel and J.J Migletz, "New XML-Based Files Implications for Forensics," *Security & Privacy, IEEE*, vol. 7, no. 2, pp. 38–44, March-April 2009.

7. T. Ngo, “Office Open XML Overview,” ECMA TC45 white paper, online at http://www.ecma-international.org/news/TC45_current_work/OpenXML%20White%20Paper.pdf, last accessed April 14, 2011.
8. Microsoft, “Remove personal or hidden information,” online at http://office.microsoft.com/en-us/word/HP051901021033.aspx, last accessed April 14, 2011.
9. Microsoft, “The remove hidden data tool for Office 2003 and Office XP,” online at http://support.microsoft.com/kb/834427, last accessed April 14, 2011.
10. http://www.bleepingcomputer.com/tutorials/windows-alternate-data-streams/
11. Muhammad Ali Raffay. Data hiding and detection in office open XML (OOXML) documents. Master’s thesis, University of Ontario Institute of Technology, 2011.