Oops... Code Execution and Content Spoofing: The First Comprehensive Analysis of OpenDocument Signatures

Simon Rohlmann
Ruhr University Bochum

Christian Mainka
Ruhr University Bochum

Vladislav Mladenov
Ruhr University Bochum

Jörg Schwenk
Ruhr University Bochum

Abstract

OpenDocument is one of the major standards for interoperable office documents. Supported by office suites like Apache OpenOffice, LibreOffice, and Microsoft Office, the OpenDocument Format (ODF) is available for text processing, spreadsheets, and presentations on all major desktop and mobile operating systems.

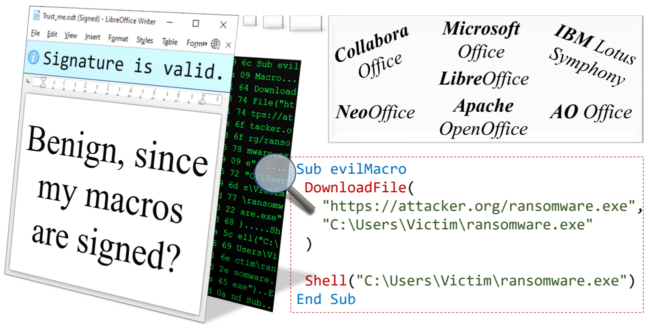
When it comes to governmental and business use cases, OpenDocument signatures can protect the integrity of a document's content, for example, for contracts, amendments, or bills. Moreover OpenDocument signatures also protect document's macros. Since the risks of using macros in documents is well-known, modern office applications only enable their execution if a trusted entity signs the macro code. Thus, the security of ODF documents often depends on the correct signature verification.

In this paper, we conduct the first comprehensive analysis of OpenDocument signatures and reveal numerous severe threats. We identified five new attacks and evaluated them against 16 office applications on Windows, macOS, Linux, iOS, Android, and two online services. Our investigation revealed 12 out of 18 applications to be vulnerable for macro code execution, although the application only executes macros signed by trusted entities. For 17 of 18 applications, we could spoof the content in a signed ODF document while keeping the signature valid and trusted. Finally, we showed that attackers possessing a signed ODF could alter and forge the signature creation time in 16 of 18 applications.

Our research was acknowledged by Microsoft, Apache OpenOffice, and LibreOffice during the coordinated disclosure.

1 Introduction

Usage of the OpenDocument Format (ODF). Major office applications, such as Apache OpenOffice, LibreOffice, and Microsoft Office, support ODF as one of the leading file formats. Apart from the undocumented private sector, organizations and governments world-wide rely on ODF documents. NATO, with its 30 member countries, uses ODF as one of its mandatory standards [26, 65]. The UK government chose ODF as the standard file format for documents in all government departments [38] and the UK Central Digital & Data Office recommends, among other methods, the use of ODF document signatures for document security [19]. The French government recommends ODF as a standard for exchanging documents between administrations and citizens [23, 34]. The Russian government allows citizens to use ODF documents when communicating with government agencies [58]. France's Inter-ministry Mutualisation for an Open Productivity Suite [25], Italy's Ministry of Defence [24], Taiwan's Ministry of Finance [27], the administration of the Spanish autonomous region of Valencia [28], among many others [84], also use ODF office applications.


Trust_me.odt (Signed) - LibreOffice Writer
File Edit View Insert Format Styles Table Form»
Signature is valid.
Benign, since my macros are signed?
Collabora Office
Microsoft Office
IBM Lotus Symphony
LibreOffice
NeoOffice
Apache OpenOffice
AO Office

```
Sub evilMacro
 DownloadFile(
   "https://attacker.org/ransomware.exe",
   "C:\Users\Victim\ransomware.exe"
 )

 Shell("C:\Users\Victim\ransomware.exe")
End Sub
```

Figure 1: If an ODF document's macro has a trusted signature, its code is automatically executed once the document is opened. We show how an attacker can execute malicious macros and manipulate the entire content by spoofing the digital signature in ODF documents.

Office Macros. In advanced use cases, such as complex calculations or automation processes, ODF documents can embed macros – a piece of program code embedded into the document. Typically, macros perform automated changes on the document's content. However, macros are not limited to the respective document and can also access the file system, start other programs, or download and execute software. Attackers can use these possibilities to load malicious code, such as ransomware, on a victim's system and execute it [15, 30, 35, 52].

ODF Digital Signatures. ODF documents can be protected with digital signatures [68, Part 2], following the XML signature standard [40]. There are two types of signatures: *document* and *macro* signatures. Document signatures ensure the document's integrity, authentication, and non-repudiation. They rely on X.509 certificates and PKIs. If a signed document was modified or signed with untrusted keys, the modification triggers a warning that the office application displays to the user. In addition, ODF can use the XML Advanced Electronic Signatures (XAdES) extension to match the European Telecommunications Standards Institute (ETSI) TS 101 903 V1.4.1 specification [29] [68, Part 2]. Macro signatures protect the integrity and authenticity of the macro code because macros are generally known to be dangerous. In their default configuration, most office applications execute macros only if a trusted entity signs them. In contrast to the document signature, macro signatures only protect the code itself. The document's content remains editable without invalidating the macro signature.

Security Risks through Office Macros. With the help of office macros, attackers can execute malicious code independently of the underlying hardware/OS. Office macros are thus typically used to start a malware attack by loading additional malware modules adapted to the OS in use. In [57], Microsoft researchers report that 98% of all office targeting attacks were based on macro code. Many of the recent ransomware campaigns were based on macros in office documents [15, 30, 35, 52].

From Signature Forgery to Code Execution. ODF macros offer the same attack potential as Microsoft Office macros: code exection. The default protection against macro-based attacks is to allow the execution if a trusted entity digitally signs the macro. Otherwise, the application blocks the macro's execution. In this paper, we show how to circumvent this protection by exploiting vulnerabilities in the signature verification and executing malicious macros without any restrictions (cf. Section 5.1). To the best of our knowledge, we are the first reporting attacks on ODF macro signatures in its current version 1.3.

From Signature Forgery to Content Spoofing. Previous work on PDF highlighted the significance of a document's signature validation Mainka et al. [51], Mladenov et al. [59], Rohlmann et al. [74]. They introduced different techniques to spoof the content presented in a PDF while keeping its digital signature valid. However, a comprehensive evaluation of digital signatures in ODF documents is still missing. In this paper, we present three novel signature forgery attacks that can be used to spoof the content of a signed ODF (cf. Section 5.2):

(1) The *Content Manipulation with Certificate Doubling* attack relies on a untrusted certificate used to sign an arbitrary ODF. The attacker adds a second, trusted certificate to the ODF. When the application processes the document, it uses both certificates for different purposes: one for the cryptographic signature verification and the other as the trust anchor.

(2) The *Content Manipulation with Certificate Validation Bypass* attack relies on a untrusted certificate used to sign an arbitrary ODF. The attacker then manipulates the certificate's internal structure stored in the ODF and disables the verification of the certificate chain. Once the ODF application processes that certificate, the manipulation results in the certificate being trusted.

(3) During the *Content Manipulation with Signature Upgrade*, the attacker forces a macro signature to be treated as a document signature. The attacker can forge arbitrary content since the macro signature cryptographically protects the macro code only.

While the general idea for (1) resembles attacks on other file formats [64], the attack (2) and (3) introduce novel techniques.

Complexity of ODF Signature Analysis. The complexity of the ODF signature analysis lies in the combination of several standards used in ODF:

- *ODF Standards*: The ODF (zip-)file consists of a collection of files. Each of the two optional signature files in this collection (document and macro signature) covers different subsets of this collection. The Content Manipulation with Signature Upgrade attack (Section 5.2.3) illustrates the potential pitfalls of this construction. Additionally, the signature files are partially signed, and contain meta information such as X.509 certificates. The standard does not cover the treatment of this meta information which leads to the successful certificate doubling attacks (Section 5.1, Section 5.2.1).
- *ASN.1 Standard*: The validation of X.509 certificates uses a Abstract Syntax Notation (ASN.1)-based data format, and is completely independent from the other validation steps. Thus, any error in the certificate verification leads to successful attacks, see Section 5.2.2.
- *XML Standards*: ODF rely on the XML signature standard [40] which is complex and builds on various other XML standards. This inclusion resulted in a large variety of attacks, ranging from XML Signature Wrapping (XSW) [54, 76] to XXE [61, 77] attacks (Section 5.3).

In this paper, we fill this gap by providing a systematic analysis of the ODF standard [68] and revealing several design issues. We show four attacks allowing an attacker to arbitrarily spoof document's content without invalidating the digital signature and one attack resulting in arbitrary code execution (ACE).

Fully Automated Evaluation. The evaluation of all attack vectors was conducted automatically using two different approaches. For content spoofing attacks, the tool Document Signature Validator (DocSV) was used to compare signature validation related strings in the process memory. In all successful cases, the strings stored in memory were identical to those stored after the validation of the un-manipulated original signature, whereas the validation of self-signed (untrusted) documents produced different strings. For macro signatures, we verified if the macro was executed after its manipulation.

Contributions. The contributions of this paper can be summarized as follows:

- We present the first comprehensive security analysis of digital signatures in ODF, both for document and macro signatures, in its current version 1.3 (Section 4).
- We describe five novel attacks against ODF signatures (Section 5). In a comprehensive evaluation, we show that 17 out of 18 analyzed ODF applications are vulnerable (Section 7).
- We present DocSV, a novel open-source tool to measure the success of signature spoofing attacks in UI applications (Section 6). DocSV can be applied to other research areas beyond ODF, for example, to Portable Document Format (PDF).

All proof of concept (PoC) files for the presented attacks and the source code of DocSV are released at `https://github.com/RUB-NDS/DocumentSignatureValidator`.

Coordinated Vulnerability Disclosure. We initiated a coordinated disclosure process to inform the affected vendors. Table 1 lists the Common Vulnerabilities and Exposures (CVE) numbers published by vendors and the associated applications that have been fixed. Microsoft Office has confirmed the vulnerability and announced a patch. We have not received any feedback from AO Office. IBM Lotus Symphony is already outside the product lifecycle.

CVE	Attack	Severity	Application	Fixed Version
2021-25633	5.1, 5.2.1	7.5 (High)	Collabora	6.2-33, 6.4.14
2021-25634	5.3	7.5 (High)	LibreOffice	7.0.2, 7.1.2
2021-25635	5.2.2	7.5 (High)		
2021-41830	5.1, 5.2.1	7.5 (High)	OpenOffice	4.1.11
2021-41831	5.3	5.3 (Med.)	NeoOffice	2017.31
2021-41832	5.2.2	7.5 (High)		

Table 1: CVEs published by the vendors. The severity score corresponds to the Common Vulnerability Scoring System (CVSS) 3.1 published within NIST's National Vulnerability Database (NVD) [66].

2 The OpenDocument Standard

The following section describes the foundations of the ODF document structure, macros, and digital signatures. The basis for this is the ODF standard published by OASIS in the current version 1.3 [68].

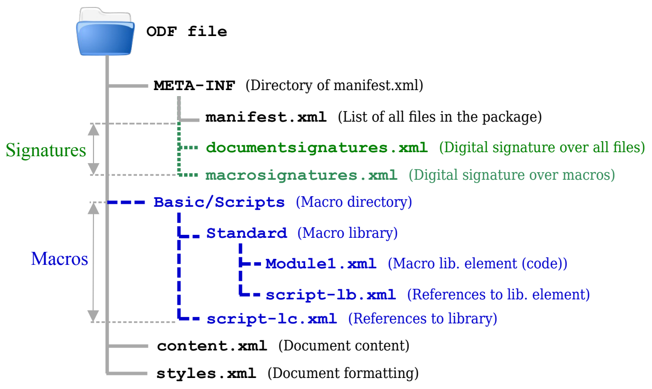

ODF file
META-INF (Directory of manifest.xml)
manifest.xml (List of all files in the package)
Signatures
documentsignatures.xml (Digital signature over all files)
macrosignatures.xml (Digital signature over macros)
Basic/Scripts (Macro directory)
Macros
Standard (Macro library)
Module1.xml (Macro lib. element (code))
script-lb.xml (References to lib. element)
script-lc.xml (References to library)
content.xml (Document content)
styles.xml (Document formatting)

Figure 2: A simplified packaged file structure of an ODF document. Besides mandatory files containing the document's content, this ODF contains files which provide integrity protection (signatures) and macros.

ODF Structure. The ODF standard defines various types of office documents. It can be used for texts, presentations/slides, spreadsheets, graphics, and charts. The prevalent method to build the document is to combine multiple files into a single zip archive. Alternatively, the document can be saved as a single Extensible Markup Language (XML) file which, however, cannot be digitally signed. In Figure 2, we depicted the typical ODF document structure. The document's starting point is the `META-INF/manifest.xml` file. It lists all files that are contained in the package. The most relevant ones are `content.xml` and `styles.xml`. They define the actual content to be presented when the document is opened. Additional information regarding the document creation and several pre-configured viewer preferences are defined in other files which are irrelevant from a security perspective. Thereby, we will not further consider them.

2.1 Macros

Macros are pieces of program code attached to a document. They are mainly used for advanced application cases, such as automatic content processing. The ODF standard is flexible in which programming language can be used. Thus, various office applications have developed their own macro functionalities [4, 20, 71, 80]. For most applications, the programming language is a *StarBasic* dialect [9, 85]. Nevertheless, macros can also be written in Python, JavaScript, or BeanShell [49, pp. 412-418] [8]. In contrast to its open-source competitors, Microsoft Office solely relies on Visual Basic for Applications (VBA) in macros [56]. Although VBA and the other Basic dialects have the same syntax, they differ in object models, and terminology [42, p. iii] [11]. Thus, VBA macros can have compatibility issues if executed in non Microsoft applications [12, 86]. Also, it is not possible to save macros in ODF documents by using Microsoft Office since a file format based on Office Open XML (OOXML) is required. For this reason, only the macro implementations that can be used within ODF documents are considered further.

Basic Macros Structure. Macros in ODF documents are organized in libraries within a `Basic` or `Scripts` directory as depicted in Figure 2. If Basic macros are used, the library is located in a sub-directory of the `Basic` directory. The file `script-lb.xml` references all macro modules, whereby each modules is a separate XML file containing the source code. Users can start individual macros manually via the user interface of the application or trigger-based by using events such as `load`, `mouseup`, and `keydown`. Such events are stored in `content.xml` in the **`<script:event-listener>`** element [7, 83]. An example is shown in Section 12.1.

Other Macro Types. If macros are created in Python, JavaScript, or BeanShell, the `Scripts` directory is used. Macro code can also be stored on the computer's local file system. This enables code reusing and cross-document execution.

Macro Execution. The ODF specification does not define any rules regarding the secure macro execution. During our analysis, we identified four security levels regarding the macro execution which are depicted in Table 2 and classified in two categories – *Run-by-default* and *Untrusted Execution*. *Run-by-default* means that macros are executed directly without any consent and *Untrusted Execution* depicts alternative execution methods if the macros are considered untrusted.

Sec. Level	Run-by-default	Untrusted Exec.
Very High	Trusted location	Forbidden
High (Default)	Trusted location or trusted signature	Forbidden
Medium	Trusted location or trusted signature	User consent
Low	All macros are trusted	No user consent

Table 2: In this paper, we concentrate on the levels running macros by default without any user consent while relying on digital signatures.

Trusted location is a trusted directory that is manually configured in the application. Document macros stored in this directory are executed automatically if triggered. Trusted signature is a successfully verified digital signature from a trusted entity. In this paper, we concentrate on the signature validation for the default security level – High. The execution of unsigned macros or macros with an invalid signature is forbidden.

2.2 Digital Signatures

There are two types of ODF signatures: *document* and *macro* signatures. They are stored in the `META-INF` directory. As depicted in Figure 2, document signatures are recommended to use the filename `documentsignatures.xml`. Macro signatures should use the filename `macrosignatures.xml`. Both files have the same internal structure. The only difference lies in the referenced files that should be protected with the digital signature.

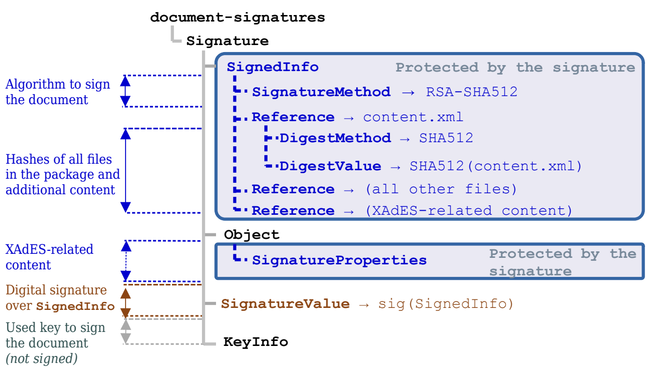

document-signatures
Signature
SignedInfo
Protected by the signature
SignatureMethod → RSA-SHA512
Reference → content.xml
DigestMethod → SHA512
DigestValue → SHA512(content.xml)
Reference → (all other files)
Reference → (XAdES-related content)
Object
SignatureProperties
Protected by the signature
SignatureValue → sig(SignedInfo)
KeyInfo
Algorithm to sign the document
Hashes of all files in the package and additional content
XAdES-related content
Digital signature over SignedInfo
Used key to sign the document (not signed)

Figure 3: An abstract overview of the structure of the XML file for digital signatures. `SignedInfo` and the XAdES-related content are protected by the signature – `SignatureValue`. The `KeyInfo`-element is not signed and is, thus, modifiable.

Document Signatures. A document signature should protect all packaged files. Thereby, all files in the ODF package should be used as an input to compute the signature. These files are referenced within the `documentsignatures.xml` file and stored with their hash value. However, some files are excluded from the computation, such as the `documentsignatures.xml` file itself and files located in the optional `external-data` directory. Three different signature states can be distinguished, which are shown in Figure 4 using LibreOffice as an example.

The signature computation follows the XML signature standard [40], and we depict the resulting XML in Figure 3. First, a hash value is computed over each referenced file. Then, the value is stored in the `DigestValue`. There may be an optional `Object` element, for example, to match the ETSI requirements for qualified XAdES signatures [68, Part 2, pp. 26-27]. In this case, the XAdES-related content is also referenced in `SignedInfo`. Finally, the signature value is computed over the `SignedInfo` element and stored in `SignatureValue`. The only unprotected elements are `KeyInfo`, which holds the certificate information necessary to validate the signature, and the XAdES-related element `xd:UnsignedProperties`.

Macro Signatures. While document signatures protect the whole ODF document, macro signatures protect only the macro code, excluding the document content. In this case, the actual document content remains modifiable, while the user can verify the origin and thus the trustworthiness of the macros. The macro signatures within ODF documents follow the same structure as the document signatures. However, only the files represented in the `Basic` or `Scripts` folder are used for signature calculation.

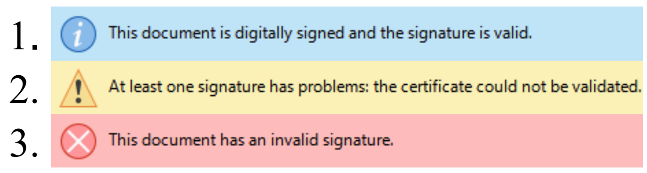

1. This document is digitally signed and the signature is valid.
2. At least one signature has problems: the certificate could not be validated.
3. This document has an invalid signature.

Figure 4: Different document signature states: (1) signature is valid and the signer is trusted, (2) signature is valid but the signer is not trusted (e.g. self-signed certificate), (3) signature is invalid.

3 Attacker Model

In this section, we describe the existing entities, the attacker's capabilities, and the winning conditions for the attacks.

Victim. The victim is a human who opens the ODF document sent by the attacker. The victim does not ignore any warnings and verifies the validation result of the digital signature.

Trusted Entity on Victim's Machine. The victim's machine maintains a list with trusted certificates and public keys. This list is automatically used by the office application. The victim does not change any setting regarding this list and does not accept untrusted key material.

Attacker Capabilities. The attacker can create and manipulate ODF documents arbitrarily. This manipulation includes, but is not limited to, adding/removing files to the ODF package (cf. Figure 2) and manipulating the content of each file in the package (e.g., the XML structure). The attacker does not possess any certificate/private key pair that belongs to a trusted entity installed on the victim's machine. Anyhow, the attacker can create a certificate/private key pair and use his own office applications to sign arbitrary ODF documents. The victim's office application treats documents signed with these keys as invalid, since they are untrusted. Due to the nature of public key cryptography, we assume that the attacker knows the trusted X.509 certificates (public key, ✹) on victim's machine. An attacker can obtain the certificate from a signed ODF, OOXML, or PDF document.

For two of five attacks, there is an additional requirement:

(1) The *Content Manipulation with Signature Upgrade* attack (Section 5.2.3) requires an ODF document with a *valid and trusted macro signature* 🗎.

(2) The *Timestamp Manipulation with Signature Wrapping* attack (Section 5.3) requires an ODF document with a *valid and trusted document signature* 🗎.

For all other attacks, the attacker does not require any signed ODF document. Finally, the attacker sends the manipulated ODF document to the victim who opens it.

Winning Conditions. The success criteria depend on the conducted attack. We distilled the following three winning conditions.

(1) </> *Macro Execution:* An attack targeting the macro execution is successful if the attacker's chosen macro code is executed on the victim's machine. Modern office applications only trigger the execution if the ODF's macro signature is valid and trusted. In this paper, macro execution is the goal of the Macro Manipulation with Certificate Doubling attack (Section 5.1).

(2) 🗎 *Content Manipulation:* An attack targeting content manipulation is successful if the attacker-chosen content is visible in the victim's office application while the digital signature remains valid and trusted. In this paper, the three attacks Content Manipulation with Certificate Doubling, Content Manipulation with Certificate Validation Bypass, and Content Manipulation with Signature Upgrade (Section 5.2.1) target the content manipulation goal.

(3) ⧗ *Timestamp Manipulation:* An attack targeting timestamp manipulation is successful if the victim's ODF application displays the attacker-chosen timestamp. The timestamp is usually not visible as part of the ODF's document content. Usually, it is shown in a dedicated menu or popup. The attack is successful if this dedicated menu or popup shows the attacker-chosen timestamp and the victim's ODF application displays a valid and trusted ODF document signature. In this paper, the XSW attack targets the timestamp manipulation (Section 5.3).

An overview of all attacks, their goals, and the attacker's capabilities/requirements is depicted in Table 3.

4 Systematic Security Analysis

After analyzing the structure of ODF documents [67, 68], processing the digital signatures [29, 40, 88] based on the relevant standard and on development toolkits [5, 6, 81, 82], we started a systematic security analysis. We summarize the considered attack concepts in three categories: partial signatures, XML-based attacks, and signing oracle. We studied related work (see Section 9) and extracted techniques that we could adapt on ODF. Except for *timestamp manipulation*, which is a classical XSW attack, no other attacks could be successfully applied to ODF. Analyzing the following three files is essential: `documentsignatures.xml`, `macrosignatures.xml`, and `manifest.xml`. The first two files state that the document has been signed and contains relevant information for the signature validation. The `manifest.xml` is the first processed file after opening a document and it is used to find all other files.

4.1 Analysis Phases

Phase 1: How are ODF signatures implemented? In the first step, we analyzed the specification with regard to digital signatures. The ODF versions 1.2 and 1.3 do not differ significantly in this point. In the next step, we analyzed which ODF package files are taken into account during the signing process and which are protected by the signature. We repeated these steps for each application. The applications are compliant to the specification and always include all existing files of the ODF package as references in the signature (except for the signature file itself). Thus, these files are included in the cryptographic signature value.

Result: We identified that all files are considered in the signature calculation, except the signature files. The signature files, `documentsignatures.xml` and `macrosignatures.xml`, can be manipulated.

Phase 2: How do the applications react to manipulations of the signed content? Since attacks cannot modify already signed content, we checked whether we can add additional files for each application. The `META-INF/manifest.xml` contains references to all files in the ODF package and its hash value is also used in the signature calculation. All applications, except Microsoft Office, consider the ODF file corrupt when the package contains files that are not referenced within `META-INF/manifest.xml`. Only files with "signatures" in the file name within the `META-INF` folder are excluded. However, only the `documentsignatures.xml` and `macrosignatures.xml` files are processed as signature files. If there are other files with "signatures" in the file name that were added later and are not referenced within the `documentsignatures.xml` file, the ODF package is not considered corrupt by the applications, but the signature is only recognized as a partial signature. Thus, the applications interpret the signature as invalid because it was not formed over all files of the ODF package according to the specification. The only exception was Microsoft Office, which accepted partial signatures and thus enabled the Signature Upgrade attack (see Section 5.2.3).

Result: Injecting new files is not possible since: (1) unreferenced files in `META-INF/manifest.xml` lead to a corrupt document; (2) `META-INF/manifest.xml` is protected by the signature and cannot be modified. The only exception are the signature files – `documentsignatures.xml` and `macrosignatures.xml`.

Phase 3: What are the possible targets of attack? The previous analysis allowed us to reduce the possible attack targets to the signature files. From a security perspective, all attacks on XML signatures are applicable. We studied the related work and extracted the following attack classes: Untrusted Keys [50], XML Signature Wrapping [54, 76], and Insecure Algorithms [33, 44]. The first two attacks can be extended to ODF documents. The remaining attack class is not applicable, since no application supports the required features.

Result: Attackers can manipulate the signature key material because of the partial signature coverage. In addition, attacks based on XSW are possible.

Phase 4: How to exploit the possible targets of attack? To create signatures, the ODF specification refers to the W3C recommendation for XML signatures [40]. We analyzed the underlying XML schema and found that the **`<KeyInfo>`** element is not signed. Also, the **`<Object>`** is partially signed and allows the inclusion of unsigned child elements. From the attacker's perspective, these elements' manipulations are possible without invalidating the digital signature.

*Manipulations in **`<KeyInfo>`**:* We analyzed the **`<KeyInfo>`** element and found that multiple **`<X509Data>`** elements are allowed and that the ODF specification does not define any restrictions on this either. In further analysis, we were thus able to develop and prove the Certificate Doubling attacks for most of the tested applications (see Section 5.1, Section 5.2.1). In the next step, we examined how the certificate is checked for trustworthiness. Here we checked whether a break in the certificate chain is detected by the application. To break the certificate chain verification, it is sufficient to replace the included signature method without damaging the certificate structure (see Section 5.2.2).

XML Signature Wrapping: The last possible point of attack is the execution of an XSW attack. The idea of XSW is to move the signed content to a place where the application does not present it when the document is opened, but the verification logic can find and process it. Instead, the application presents content created by the attacker while the signature verification remains valid.

The base of the XSW attack is the analysis of the **`<Reference>`** elements of the **`<SignedInfo>`** element in `documentsignatures.xml`. The **`<Reference>`** elements point to the signed files within the ODF and to the timestamps defined in `documentsignatures.xml`. Our experiments on moving the signed files failed due to a double check – the names and hashes of all files are contained in the **`<Reference>`** elements and the names are listed a second time in the `manifest.xml` file, which is also signed. This double-check prevents most XSW attacks, because ODF applications consider a document corrupted if additional files are added which are not listed in `manifest.xml`. Thus, only content that is not listed in `manifest.xml` can be manipulated. In ODF documents, this content are the timestamps. We show an attack in Section 5.3.

Result: In total, 132 ODF documents with attack vectors were created, 61 with XAdES compatible signatures and 61 without XAdES. The documents contain different arrangements of the attack vectors, five attack vectors each for XAdES/non-XAdES were successful and are described in detail in Section 5. We provided the attack vectors as PoC files to the vendors and the research community.

4.2 Hidden Content and Signing Oracle

In 2021, Mainka et al. [51] presented the concept of shadow attacks on PDFs. These attacks hide malicious content within the PDF document before it is signed. After the signing, a change in unsigned parts of the PDF reveals the hidden content without invalidating the digital signature. For ODF document signatures, there is no unsigned content, except in the XML signature file itself (Section 4.1). Even adding additional hidden files into the ODF package does not succeed, because revealing them requires changes in the document's meta file that is protected by the digital signature. Another approach was to add unsigned files to the ODF document after the signing, and to try to use these files to change the visible content. This approach was also not successful since all ODF applications consider documents containing files not listed in `manifest.xml` as corrupt. Repairing a corrupted file results in a removal of the ODF document signature.

5 New Attacks on ODF

In the following section, we describe five novel vulnerabilities on ODF Signatures. Each attack manipulates a signed ODF document. A victim opening such manipulated ODF sees a validly signed document.

Section	Manipulation Goal	Requirement	Manipulation Technique
5.1	</> Macro	✹	Certificate Doubling
5.2.1	🖹 Content	✹	Certificate Doubling
5.2.2	🖹 Content	-	Certificate Validation Bypass
5.2.3	🖹 Content	✹ 🗎	Signature Upgrade
5.3	⧗ Timestamp	✹ 🗎	XML Signature Wrapping

Table 3: We present five novel attacks on ODF signatures. Each attack can manipulate different parts of the ODF (i.e., content, macro, or timestamp) with attacker-selected values. The signature verification remains valid in all cases. All but the Content Manipulation with Certificate Validation Bypass attack require the attacker to possess the trusted signer's public certificate ✹. Some attacks require either an ODF with a valid document signature 🗎 or macro signature 🗎. In total, we used four different techniques for achieving these goals.

We categorized the attacks into three groups according to their manipulation goal (i.e., macro, content, or timestamp) as depicted in Table 3. In summary, one attack allows to embed and execute the attacker's macro code stealthily (Section 5.1) on the victim's machine. Three attacks forge the visible document content and display different content than the signed one (Section 5.2). One attack can change the XAdES timestamps (Section 5.3).

The Certificate Validation Bypass and the Signature Upgrade attack are novel and do not rely on any previous work. The Certificate Doubling and XSW attacks were previously applied on Single Sign-On protocols like SAML. We extended these techniques on ODF documents and demonstrated their applicability.

5.1 Macro Manipulation with Certificate Doubling

In general, macros in office documents have the highest potential of damage, since a victim only needs to open the document to execute arbitrary code on his system. To minimize the potential risk, office applications can use macro policies. For example, in Apache OpenOffice and LibreOffice, the policy level *High* is set by default in the macro security settings. This policy level allows macro execution only if the macro is signed by a trusted source. Otherwise, macros are disabled and not executed. Trusted sources are maintained separately from the operating system's certificates store and must be approved by the user. Once the author of the signed macro is included in this list, the macro execution is unrestricted.

</> Manipulation Goal. The Macro Manipulation with Certificate Doubling attack allows an attacker to alter an ODF's signed macro code without invalidating its signature. The attacker's goal is to create an ODF document that contains valid signed macro code, with trusted certificate data, from a third person or organization. The attack is based on the partial signature protection leaving the `<KeyInfo>` unprotected.

Attack Requirements. For the Macro Manipulation with Certificate Doubling attack, the attacker only requires the public X.509 certificate of an entity trusted to execute macros ✹. The attacker does not require a signed and trusted ODF document.

Manipulation Technique. In this attack, the attacker manipulates a self-signed ODF document so that it contains two certificates. One certificate is only used to represent the signer's identity (e.g., name or public key) and as such, the certificate's public information is sufficient for the attacker. The other certificate is used for the cryptographic computation of the signature. For this computation, the attacker uses his own private key and the entire document is no longer signed, but only the files contained in the macro folders.

Manipulation Steps.

(1) The attacker creates an ODF document with *arbitrary macro code* and digitally signs the macros. The attacker uses his own private key for this purpose. For example, a self-signed certificate / private key combination is sufficient.

(2) The attacker manipulates the document's macro signature file `META-INF/macrosignatures.xml` as follows:
   (a) The attacker duplicates the entire `<X509Data>` object within this signature file.
   (b) The attacker exchanges the value of the `<X509Certificate>` object within the second `<X509Data>` object with a public certificate from an entity trusted to execute macros ✹.

During our evaluation, we systematically analyzed the doubling of the `<X509Certificate>` object in Step 2. (e.g., vice versa or tripling), but the only working solution was to use the trusted certificate ✹ as the second `<X509Certificate>` element.

Impact. Suppose the attacker uses a developer certificate of the victim's organization. In this case, the abused certificate is on the victim's trusted list and the macro code is executed without any confirmation once the document is opened. The execution takes place in the background, so that the victim does not notice it. There is a variety of ways to execute malicious code on the user's computer. For example, many ODF applications provide a complete Python environment inside their program folder, which the attacker can use for malicious code development. One possibility is to create a Python file via macro code and execute it via the Python runtime of the office application. The function `ctypes.windll.user32.ShowWindow(ctypes.windll.kernel32.GetConsoleWindow(), 0)` prevents the victim from seeing a console window. The malicious code can thus be executed entirely in the background. By offloading the execution to Python, it is not interrupted even when the document is closed. We provide a PoC of *harmless* ransomware that hashes all files instead of encrypting and deleting them. Delegating the execution to PowerShell commands is also possible, but scripts are restricted by default on Windows client systems [55]. In addition to the malicious code contained directly in the document, it can also be outsourced and downloaded from the Internet. We identified an interesting behaviour in the execution of downloaded executable `*.exe` files under Windows systems. For security reasons, the download and execution of `*.exe` files via a browser require additional permissions and user interaction. However, if an `*.exe` file is downloaded and executed via the macro code, it will be executed directly. For this purpose, we also provide a PoC ODF file that reloads a harmless `*.exe` file from the Internet and is executed via the macro code when the manipulated ODF file is opened.

5.2 Content Manipulation

The second desired goal of the attacker is the content manipulation. The following three attacks manipulate a signed ODF document so that it displays arbitrary attacker-selected content while forging the document's signature of a trusted entity on the victim's machine.

5.2.1 Content Manipulation with Certificate Doubling

Manipulation Goal. The attacker can use the Content Manipulation with Certificate Doubling attack for spoofing the content of a signed ODF document. When the victim opens the document, the result of the signature verification is valid and the document seems to be signed by a trusted entity.

Attack Requirements. The attacker only requires the public X.509 certificate of a trusted entity on the victim's machine ✹. The attacker does not require a signed and trusted ODF document.

Manipulation Technique. First, the attacker manipulates a self-signed ODF document so that it contains two certificates. The manipulation steps roughly correspond to those of the Macro Manipulation with Certificate Doubling attack.

Manipulation Steps.

(1) The attacker creates an ODF document with *arbitrary selected content*. The attacker digitally signes the document using the own private key. For example, a self-signed certificate / private key combination is sufficient.

(2) The attacker manipulates the document's signature file `META-INF/documentsignatures.xml` as follows:
   (a) The attacker duplicates the entire `<X509Data>` object within the signature file.
   (b) The attacker exchanges the value of the `<X509Certificate>` object in the second `<X509Data>` object with a public certificate from a person trusted by the victim ✹.

The document generation is comparable to Macro Manipulation with Certificate Doubling but with two main differences. In Step 1, the attacker signs the entire document content. In Step 2, the attacker manipulates the document signature data file (`META-INF/documentsignatures.xml`) instead of the macro signature file (`META-INF/macrosignatures.xml`).

Impact. When the victim opens the attacker's manipulated ODF document, the office application represents the information of the second certificate as the signing entity ✹. The victim can manually investigate all certificate data of the supposed signer and the office application successfully verifies the certificate's trustworthiness. However, during the signature verification, the office application uses the public key of the first `<X509Data>` element (i.e., the attacker's certificate), while the trustworthiness of the signer is determined using the second `<X509Data>` (i.e., the certificate of the trusted entity) element.

5.2.2 Content Manipulation with Certificate Validation Bypass

Manipulation Goal. The goal of the Content Manipulation with Certificate Validation Bypass attack is to create a valid signed and trusted ODF document with arbitrary content. The owner and issuer data contained in the certificate, which is displayed to the victim, can be chosen arbitrarily.

No Attack Requirement. There are no further requirements for this attack, because the attacker creates and signs a document with his own private key. The attacker requires neither a signed or trusted ODF document nor the public X.509 certificate of a trusted entity on the victim's machine.

Manipulation Technique. To verify the trustworthiness of a signature in ODF documents, it is crucial that the system trusts the signer's certificate. Normally, the Windows Certificate Store or the Mozilla Certificate Store for macOS and Linux are used for this purpose. The Content Manipulation with Certificate Validation Bypass attack aims to disrupt the verification of this trust verification and thus force a victim's office application to classify arbitrary certificates as trusted. We focused on manipulating the X.509 certificate stored as an ASN.1 blob. We studied the fields in this blob which are responsible for the certificate verification and created multiple attack vectors covering public key injection, algorithm manipulation, and signature exclusion. Only the algorithm manipulation was successful and will be described further.

Manipulation Steps.

(1) The attacker creates an ODF document with arbitrary content and signs it using his own private key.

(2) The attacker manipulates the signature file `META-INF/documentsignatures.xml` in the signed document as follows:

  (a) The attacker manipulates his own X.509 certificate by setting the therein defined signature algorithm to an invalid value. That algorithm is defined with an Object Identify (OID). For example, he can cut off the trailing bytes of the OID `1.2.840.113549.1.1.11` (`sha256WithRSAEncryption`) and set the OID value to `1.2` (`member-body`). For this purpose, an ASN.1 Editor can be used.[1] This truncation breaks the ODF application's internal certificate validation, turning a formerly untrusted signer into a trusted signer.

  (b) The attacker exchanges the value of the `<X509Certificate>` object with the manipulated certificate from the previous step.

Impact. If the victim opens the manipulated ODF document, the signature is recognized as valid because the private key used for signing the ODF document matches the public key of the certificate it contains. The manipulation of the certificate simultaneously disrupts the verification of the chain of trust, so that the signer is treated as trusted. The attacker can also manipulate further parameters in his own certificate (Step 2a), such as its subject or issuer. By this means, the attacker can impersonate an arbitrary entity. Only the public key must remain unchanged to prevent influencing the cryptographic signature verification.

5.2.3 Content Manipulation with Signature Upgrade

Manipulation Goal. The Content Manipulation with Signature Upgrade attack manipulates an ODF document with a macro signature in such a way that the ODF application treats that signature as a document signature. In this way, the attacker can arbitrarily set the content of the ODF document while the application shows a valid document signature. The attack abuses the partial signature coverage of the files contained in the ODF document when only the macros are signed.

Attack Requirement. The attacker needs an ODF document with validly signed macros.

Manipulation Technique. ODF applications use two types of digital signatures. The first type is document signatures, the second type is macro signatures. From a technical perspective, the only difference between these two types is the file name (`documentsignatures.xml` vs `macrosignatures.xml`) and the selection of the files in the ODF package which are included in the signature calculation with their hash values. While document signatures include all contained documents (for exceptions, see Section 2), macro signatures only include the files within the macro directory. Despite their otherwise identical implementation, the two signature variants, thus, differ significantly in the content to be protected.

The attacker can use this macro signature together with the included macros to create ODF documents with arbitrary content and display the included macro signature as the document signature.

Manipulation Steps.

(1) The attacker creates an unsigned ODF document with arbitrary content.

(2) From the macro signed ODF document, the attacker extracts the signature file `META-INF/macrosignatures.xml`, as well as the entire macro directory (i.e., `Basic` or `Scripts`).

(3) The attacker renames the `macrosignatures.xml` to `documentsignatures.xml` and inserts it together with the macro directory into the ODF document of Step 1.

Impact. We assume that the victim trusts the certificate of the original signer of the macro. In that case, the ODF application recognizes the signature contained in the manipulated document as a valid and trusted document signature, even if the signature does not protect the actual visible content of `content.xml` in any way. Since macros are typically treated as dangerous, companies could sign specific macros that are necessary for their business cases (e.g., fill templates, processing of large tables or specific calculations in spreadsheets). An attacker who has access to that file can upgrade the macro signature to forge arbitrary signed content (e.g., agreements or contracts).

5.3 Timestamp Manipulation with Signature Wrapping

Manipulation Goal. The attacker's goal is to change the timestamp of an ODF signature without invalidating it. This change allows the attacker, for example, to forge the creation time of a particular contract, which could lead to legal consequences.

Initially, our goal was to manipulate the entire content of the document. Our analysis showed that we cannot manipulate

[1] www.codeproject.com/Articles/4910/ASN-1-Editor

any external files such as `manifest.xml` or `content.xml` due to the limitations which we described in Section 4.

Attack Requirement. The attacker needs an ODF document with a valid and trusted document signature 🗎.

Manipulation Technique. The certificate's data in the **`<X509Data>`** element and the **`<Object>`** element play a unique role in the representation of signature information within the ODF application. For instance, the **`<Object>`** element holds the timestamp of the signature creation time, which is displayed alongside the signer's data. By using XSW attacks, we can forge timestamps.

Manipulation Steps.

(1) The attacker manipulated the signature file `META-INF/documentsignatures.xml` as follows:

(2) The attacker duplicates the entire **`<Object>`** element within the signature file.

(3) The attacker changes the `ID` attribute of the **`<SignedProperties>`** element within the second **`<Object>`** element. For example, the attacker increases the value by 1.

(4) The attacker selects an arbitrary time and sets the timestamp in the second **`<Object>`** element to that value.

- In the case of a regular ODF signature, the attacker manipulates the timestamp hold in the **`<dc:date`** `xmlns:dc="http://purl.org/dc/elements/1.1/">` element.
- In the case of a XAdES signature [29], the attacker manipulates the timestamp hold in the **`<xd:SigningTime`** `xmlns:xd="http://uri.etsi.org/01903/v1.3.2#">` element.

Impact. If the victim trusts the signer's certificate, a valid and trusted signature will be displayed when opening the manipulated ODF document. However, the manipulated timestamp with the modified ID is used to display the signing time.

6 DocSV: A Novel Evaluation Technique of Office Documents

Verifying an ODF macro signature attack is straightforward. If the victim's machine executes the malicious macro, the code execution itself can be used to inform the attacker about the attack's success. For document signatures, the automatic verification of document signatures is challenging since there is no feedback telling the attacker whether the attack was successful. During the research, we created 132 ODF documents with different attack vectors. The evaluation effort of the resulting documents is multiplied by the number of analyzed applications (in this paper: 18). Thus, an automation is necessary to make the evaluation results effective and independent of the number of applications to be tested.

In general, there are two approaches used for the validation: the creation and comparison of screenshots or the use of OCR text recognition. A screenshot comparison is only possible when a ground truth image exists. This limitation leads to two adverse side effects: the configuration overhead increases and the comparison is applicable only on documents with the same content. In addition, the comparison reacts sensitively to changes in the document to be analyzed, for example, popups. Kuchta et al. [46] located inconsistencies in PDF readers and Rohlmann et al. [74] implemented a tool called PDF Tester,[2] which can automatically evaluate attacks on PDF signatures. However, screenshot comparison is slow and requires a ground truth image. Another approach is using OCR text recognition. Besides being slower than the screenshot-approach, OCR cannot entirely exclude misinterpretations of the texts [14]. Thus, both variants do not provide optimal evaluation results.

6.1 Memory-based Evaluation of Documents

In this section, we present a new tool called DocSV, which is faster, more reliable than existing analyzing techniques, vendor-independent, and compatible with the primarily used document formats – ODF, OOXML, and PDF. The basic idea behind DocSV is the following: Whenever a document signature validation succeeds or fails, the document processing application must generate a string to display this result to the user (Figure 4). DocSV searches the process memory for these (known) strings, and deduces the result of the signature validation from the string found.

We defined the following requirements for DocSV:

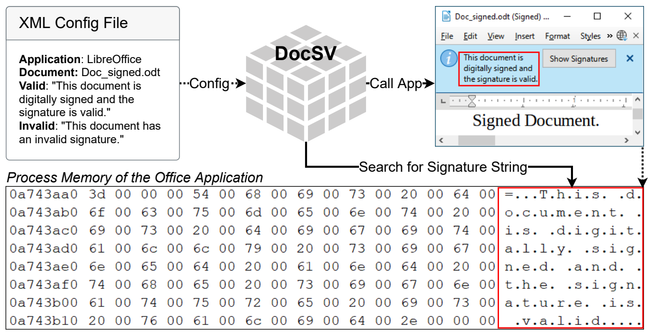
(1) The analysis should require less configuration overhead and fewer resources than previous techniques.

(2) The approach should allow the easy inclusion of new office applications.

(3) The analysis should be independent of the document's content.

Design Decisions. DocSV should be as generically compatible as possible with all standard office file formats and corresponding applications. Even though some vendors, such as Apache OpenOffice or LibreOffice, allow control through APIs, this does not apply to many other vendors. Thus, the use of vendor-specific interfaces would not have been appropriate and would violate our requirements. To circumvent the disadvantages of optical processing of screenshots and to remain application-independent (i.e., avoid vendor-specific APIs [3, 79]), we decided to rely on the analysis of the process memory of the individual applications. For this purpose, DocSV is written in C++ and capable of accessing the application memory. Using the memory-based analysis, we achieved reliable and reproducible results explained further in Section 6.2. In addition to ODF applications, we were able to successfully test the correct functionality for OOXML applications such as Microsoft Office and PDF applications,

[2] https://github.com/RUB-NDS/PDF-Tester


XML Config File
DocSV
Config
Call App
Doc_signed.odt (Signed) ...
File Edit View Insert Format Styles
Show Signatures
Search for Signature String
Process Memory of the Office Application

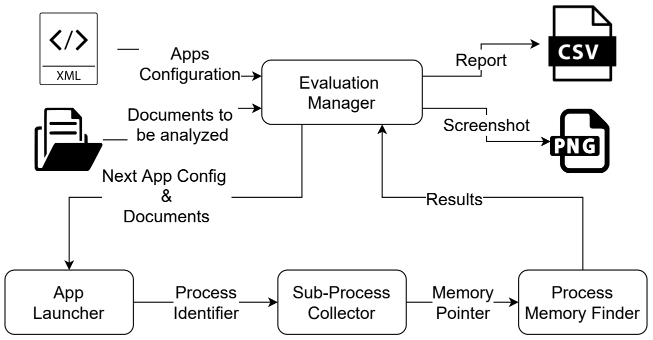
```
Application: LibreOffice
Document: Doc_signed.odt
Valid: "This document is digitally signed and the signature is valid."
Invalid: "This document has an invalid signature."
```

This document is digitally signed and the signature is valid.

Signed Document.

```
0a743aa0  3d 00 00 00 54 00 68 00 69 00 73 00 20 00 64 00  =...T.h.i.s. .d.
0a743ab0  6f 00 63 00 75 00 6d 00 65 00 6e 00 74 00 20 00  o.c.u.m.e.n.t. .
0a743ac0  69 00 73 00 20 00 64 00 69 00 67 00 69 00 74 00  i.s. .d.i.g.i.t.
0a743ad0  61 00 6c 00 6c 00 79 00 20 00 73 00 69 00 67 00  a.l.l.y. .s.i.g.
0a743ae0  6e 00 65 00 64 00 20 00 61 00 6e 00 64 00 20 00  n.e.d. .a.n.d. .
0a743af0  74 00 68 00 65 00 20 00 73 00 69 00 67 00 6e 00  t.h.e. .s.i.g.n.
0a743b00  61 00 74 00 75 00 72 00 65 00 20 00 69 00 73 00  a.t.u.r.e. .i.s.
0a743b10  20 00 76 00 61 00 6c 00 69 00 64 00 2e 00 00 00   .v.a.l.i.d.....
```

(a) The DocSV analysis tool automatically opens documents in various office applications based on a configuration file. It then evaluates the signature status of the document by analyzing the application process's memory.



(b) DocSV consists of four main modules: Evaluation Manager, App Launcher, Sub-Process Collector, and Process Memory Finder. The modules are used to classify whether the signature validation in a document is successful and to generate a report with these results.

Figure 5: The idea behind DocSV is to analyze the office application's process memory to determine whether the signature validation of documents (ODF, OOXML, and PDF) is correct or not.

including Adobe Acrobat and Foxit PDF Reader. DocSV consists of four modules: Evaluation Manager, App Launcher, Sub-Process Collector, and Process Memory Finder, see Figure 5b.

Evaluation Manager. On its startup, DocSV receives a configuration file. It contains, among other information, the office application, a folder containing all documents that it should analyze, and a list of predefined strings with possible states of the signature validation. The Evaluation Manager is responsible for starting the analysis process for each configured application by calling the App Launcher. For documentation purposes, this module takes a screenshot before closing the office applications and stores it on the system together with a CSV file documenting the entire analysis process.

App Launcher. The App Launcher receives the path of the application that it should start. It automatically launches the office application and opens the documents in the document directory one by one. During our evaluation, we determined that the App Launcher should proceed only when the application is fully loaded. Since the operating system does not provide any reliable APIs to detect this, DocSV waits a time specified in the configuration file before starting the process memory analysis. This time can be determined depending on the applications to be analyzed. Usually, 2-6 seconds is sufficient. We established this value based on our empirical observations regarding the loading time of different office applications.

Sub-Process Collector. Depending on the application, the office application can start further sub-processes. For instance, Adobe Reader creates a sub-process for each opened document. Thus, DocSV first determines and collects all subprocesses and includes these in the analysis based on the primary process.

Process Memory Finder. All processes belonging to one application are analyzed for the presence of the defined signature status strings (see Figure 4). In case of a hit, the analysis is terminated. The selection of the appropriate character encoding plays a major role for the successful analysis. For example, in Apache OpenOffice and LibreOffice the signature states are encoded in UTF-16, while Adobe Acrobat uses UTF-8. The encoding can be selected in advance in the configuration file. This configuration is made for each application separately.

6.2 Advantages and Disadvantages

In this section, we provide a comparison to the other two evaluation approaches based on screenshots and OCR. We show that the memory-based analysis is the most sufficient technique for the analysis of office documents since the configuration overhead is minimal, the recognition rate is high, and resource consumption is less than other approaches.

Test Environment. We first created a pool with 60 PDF documents separated in three categories each containing 20 unsigned documents, 20 validly signed documents, and 20 manipulated documents. Thus, we can determine the detection rate of the different techniques. We executed the tests on a PC with Intel Core i7-8565U CPU, 40GB RAM, and Windows 10. We installed PDF Tester. PDF Tester supports three comparison modes: (1) *PDF Tester Comparer* uses a pixel-based screenshot comparison; (2) *PDF Tester OCR* applies OCR-based text recognition; (3) *PDF Tester OCR improved* optimizes the screenshot before analysis. Additionally, we considered the screenshot analysis introduced by Kuchta et al. [46] which relies on the Complex-wavelet structural similarity index (CW-SSIM) algorithm. This algorithm compares differences in the images that are perceptible to the human eye. Please note that only the source code of PDF Tester is publicly available. Thus, we used it to open the documents and create the screenshots which we also used for the CW-SSIM evaluation.

Direct Comparison. Table 4 depicts a direct comparison between the resource consumption of all three evaluation techniques. DocSV is the fastest analyzing tool in our evaluation. The slowest methods are the OCR-based analysis and the CW-SSIM since the images are transformed first and then compared (OCR-based) or multiple measurements on small image pieces and computations of the corresponding Structural Similarity Indices (SSIM) are executed (CW-SSIM). The detection rate of screenshot-based analysis and DocSV is high. The OCR comparison is less correct with $33.\overline{3}\%$ success rate in the normal mode. Even the improved OCR recognition is in $16.\overline{7}\%$ incorrect and falsely classify the validation status.

Content-independent Analysis. The memory-based approach requires only a configuration of the status messages regarding the signature validation. This configuration is made once per office application and can afterwards analyze documents with arbitrary content. In comparison, the screenshot-based analysis requires the configuration of one valid document as a ground truth. This ground truth, however, bases on the specific document content. The OCR-based evaluation produces similar results than the memory-based approach and requires the same configuration overhead.

Tool	Detect-ion	Adobe Acrobat	Foxit Reader	Libre Office	Total
PDF Tester Comparer	Time	2:04	2:04	2:04	**6:12**
	Rate	100%	100%	100%	**100%**
CW-SSIM	Time	10:20	10:16	10:12	**30:48**
	Rate	95%	100%	100%	**$98,\overline{3}\%$**
PDF Tester OCR	Time	2:07	1:52	2:24	**6:23**
	Rate	50%	0%	50%	**$33,\overline{3}\%$**
PDF Tester OCR *improved*	Time	8:02	7:37	8:43	**24:22**
	Rate	100%	100%	50%	**$83,\overline{3}\%$**
DocSV	Time	1:18	1:02	1:06	**3:26**
	Rate	100%	100%	100%	**100%**

Table 4: The results of the comparison between screenshot-based analysis, OCR evaluation, and memory-based analysis shows that DocSV is the most efficient tool which also provides a high detection rate. Time values are provided in minutes and seconds.

Limitations of DocSV. The configuration of strings as recognition markup for the validation results could lead to false results if the same string occurs in the document's content. For our test documents, we control the content and made sure that none of the validation strings were present. During our evaluation, we discovered another problem that occurs if the application provides multiple predefined signature status strings in the process memory. This is the case, for example, with Foxit PDF Reader where both strings for valid and invalid signatures are contained in the memory dump. However, we observed that, the signature status "Signature is valid" occurs twice in an actually valid signed document, while it appears only once in an invalid or unsigned document. As a result, for these type of applications we defined a minimum number of found signature status strings in the configuration file. For the other applications, this adjustment was not necessary, since the predefined signature status strings differed from the actually displayed signature state based on their encoding (UTF-16 vs. UTF-8).

7 Evaluation

In this section, we describe which office applications are vulnerable to which of our five attacks. We discuss the used test environment and the specifics of individual office applications in connection with the attacks. The results in Table 5 show that 17 out of 18 office applications are vulnerable to at least one attack. Only for Digital Signature Service (DSS) developed by Nowina Solutions, we could not detect any vulnerability.

Test Environment. We use three different system landscapes to represent a test situation that is as close as possible to real-world use cases. The *attacker system* consists of a Windows 10 virtual machine (VM). It contains a combination of the private key and self-signed public certificate of the attacker, and the public certificate of a entity classified as trusted by the victim. The second system is the *signing system*, consisting of a Windows 10 VM with the private key and public certificate of the trusted entity. The third system depicts the *victim's landscape* and it spans VMs with Windows 10 and Ubuntu 20.04.3 LTS, as well as physical systems in the form of macOS Catalina, Android 10, and iOS 15. All systems trust the public certificate of the signer system, but not the attacker's certificate.

Tested Applications. We used the references [13, 37, 70, 90, 91] as a basis for the selection of office applications to be evaluated. In the first step we checked the applications for general ODF support. In the next step, we analyzed whether the applications processed the included and unmanipulated ODF signatures. All applications that met these requirements and whose development had not already been discontinued were included in the evaluation (see Table 5). IBM Lotus Symphony development was discontinued in 2012 and features were merged with Apache OpenOffice [2, 10]. We included Lotus Symphony in the evaluation to demonstrate how long some of the vulnerabilities have been present in the code and are still detectable in today's office applications. AndrOpen Office in version 4.8.7, is excluded from the evaluation since it does not show an included signature for any of our unmanipulated test documents. The online variant of Collabora in version 6.4.11 could not perform correct signature validation for any of our unmanipulated signed test documents. Any signatures included were always detected as invalid. Thereby, we used an older variant in version 6.0-18, in which this bug

			Attacks on OpenDocument Signature				
Application	**Version**	**OS**	**Macro Manipulation with Certificate Doubling** Section 5.1	**Content Manipulation with Certificate Doubling** Section 5.2.1	**Content Manipulation with Certificate Validation Bypass** Section 5.2.2	**Content Manipulation with Signature Upgrade** Section 5.2.3	**Timestamp Manipulation with Signature Wrapping** Section 5.3
Apache OpenOffice	4.1.8	Windows	●	●	●	○	●
Collabora Office	6.2-20210530	Windows	●	●	●	○	●
IBM Lotus Symphony	3.0.1 fp2	Windows	●	●	○	○	●
LibreOffice	7.0.4.2	Windows	●	●	●	○	●
Microsoft Office 2019	16.0.10374.20040	Windows	⊘	○	○	●	○
Apache OpenOffice	4.1.8	macOS	●	●	○	○	●
Collabora Office	6.2-20210530	macOS	●	●	○	○	●
LibreOffice	7.0.4.2	macOS	●	●	○	○	●
NeoOffice	2017.27	macOS	●	●	○	○	●
Apache OpenOffice	4.1.8	Linux	●	●	○	○	●
Collabora Office	6.2-20210530	Linux	●	●	○	○	●
IBM Lotus Symphony	3.0.1 fp2	Linux	●	●	○	○	●
LibreOffice	7.0.4.2	Linux	●	●	○	○	●
Collabora Office	6.4.11-2	iOS	⊘	●[1]	○	○	●[1]
AO Office	4.1.6	iOS	⊘	●	○	○	●
Collabora Office	6.4.3	Android	⊘	●[1]	○	○	●[1]
Collabora Online (CODE)	6.0-18	Online	⊘	●[1]	○	○	●[1]
Digital Signature Service	5.9	Online	⊘	○	○	○	○
$\sum$ Applications that are *vulnerable* ●, max 18			12	16	3	1	16

● Vulnerable: Application is vulnerable to this attack.
○ Secure: Application is not vulnerable to this attack.
⊘ Non-verifiable: Attack cannot be tested with this application.

[1]No certificates view available, but valid signature and valid certificate verification is displayed.

Table 5: Evaluation results. We could identify macro execution in all applications supporting macros (i.e., in 12 of 18). In all cases except the Digital Signature Service, we could spoof signed content (i.e., in 17 of 18).

does not exist.

Certificate Doubling Attacks. Macros and Content Manipulation with Certificate Doubling attacks are based on the fundamental problem of incorrectly processing the XML signatures. However, both attacks have entirely different manipulation goals: in the first variant, arbitrary malicious code is supposed to be executed on the victim's system, and in the second variant, the victim is supposed to be fooled into thinking that a document is signed by a trusted entity. We show that 12 out of 18 applications are vulnerable for the first variant, and 16 out of 18 applications are vulnerable for the second variant. We could, thus, execute arbitrary macro code in all ODF applications that can generally process macro code in ODF documents. Our PoC examples show that the attacker could cause significant damage (e.g., compromise all user files) by exploiting this vulnerability. For the content manipulation variant, the Collabora iOS, Android, and online variants show correct signature validation and certificate verification for the attacker-manipulated ODF documents. Still, they do not provide a feature to view more detailed information about the signer. Microsoft Office and DSS specified the attacker's certificate as the creator of the signature and are, thus, not vulnerable to this attack.

Content Manipulation with Certificate Validation Bypass. Only the Windows versions of Apache OpenOffice, Collabora Office and LibreOffice are vulnerable to the attack. 15 of the 18 applications classify the signature as invalid due to the manipulated certificate of the signer.

Content Manipulation with Signature Upgrade. This problem of partial signatures, which is the root cause for this attack, has been recognized in the ODF specifications 1.2 and 1.3. These specifications require that a document signature is always created over all files within the ODF package [67, Part 1, p. 99] [68, Part 3, p. 98] (for exceptions, see Section 2). Following this recommendation, 17 out of 18 applications comply and evaluate the macro signature disguised as a document signature as invalid. Only Microsoft Office accepts this type of signature and is vulnerable.

Timestamp Manipulation with Signature Wrapping. We identify 16 of 18 applications as vulnerable to the Timestamp Manipulation with Signature Wrapping attack. Only Mircosoft Office and DSS are secure. Surprisingly, there is not even a logical check of the timestamp for the 16 applications. This missing check allows the attacker to form arbitrary date such as 66/66/6666.

8 Countermeasures

This section discusses mitigations for our attacks. We categorized them by their manipulation techniques.

Manipulation Technique: Certificate Doubling. Both Certificate Double attack techniques have the same root cause. They abuse a logical inconsistency during the validation of the signature. This validation consists of two steps.

(1) The signature must be cryptographically verified. For example, in RSA signatures, the signature's hash and its RSA value are verified.

(2) The origin of the used key material must be verified. In the case of an XML signature, this typically means to ascertain the trustworthiness of the X.509 certificate.

A Certificate Doubling attack abuses a binding between these two steps. It provides two X.509 certificates within the XML signature. One certificate is trusted (Step 2) but fails during the cryptographic verification (Step 1). The other provides the opposite. In the case of LibreOffice, the inconsistency lies in the `XSecParser class` of the `xmlsecurity` module [87]. The signature validation extracts two certificates but uses them for different purposes (cf. Step 1 and Step 2). A mitigation for the attack is to enforce the passing of each certificate to both validation steps. Only if one certificate passes both steps, the signature can be handled as valid. Another proper way would be to execute the validation steps subsequently but remove a certificate in the event of the XML signature document object not passing a step. If this process leaves no certificates, the signature must be handled as invalid.

Manipulation Technique: Certificate Validation Bypass. The Certificate Validation Bypass technique abuses unspecified input value during the certificate validation process. The root cause for the attack is that certificates are initially considered valid. In case that the certificate trust verification returns invalid (Step 2), the initial value changes. If the application cannot finish Step 2 of the verification, the signature validity status remains valid. The attack causes this interruption by using an unspecified input at a specific place in the X.509 certificate. The effective mitigation (e.g., implemented by LibreOffice and OpenOffice) is to set the default validity of each certificate to invalid during the startup so that disruption during its processing does not result in trusted certificates.

Manipulation Technique: Signature Upgrade. The Signature Upgrade technique can be traced back to an incomplete implementation of the ODF specifications. These specifications requires that for document signatures all files of the ODF package are included in the signature calculation [68, Part 3, p. 98] [67, Part 1, p. 99] (exceptions see: Section 2). Thus, an effective countermeasure is to treat partial signatures as invalid in general.

Self-signed Certificates. Four attacks use the attacker's self-signed certificates. However, deprecating self-signed certificates in the ODF specification would not be a useful countermeasure. For the attacks, the attacker could also use certificates issued by a trusted certificate authority (CA) and perform the attacks in the same way.

Manipulation Technique: XSW. A reliable method to prevent XSW is to apply the *see-what-is-signed* approach [76]: the signature validation removes all parts of the ODF document that are not part of the signature computation. With respect to the Timestamp Manipulation with Signature Wrapping attack, all unsigned timestamps will be ignored and making the spoofing impossible. This protection applies to both ODF signature variants – XAdES and non-XAdES signatures. The mitigation would also prevent additional XSW variants that could possibly be applied to other parts, for example, the content.

9 Related Work

Security of Office Documents. In the years 2006 to 2009, different researchers [21, 31, 32, 47, 72] analyzed the security of OpenOffice.org respectively OOXML and proposed different attack and obfuscation techniques to stealthy execute malicious code. The authors highlighted security issues in the design of OpenOffice.org version 2.x to 3.x. The analyses also identified problems with digital signatures, which have been addressed in recent ODF standards. Our work, in contrast, is the first to analyze ODF signatures systematically, focusing on the latest standard. Malware in the context of VBA macros has been discussed by various researchers [22, 36, 48, 92]. Their focus lied in convincing the victims to allow macro execution. Another attack technique is called VBA stomping which abuses the so-called performance code in VBA macros [16, 18, 69, 75]. This attack cannot be adapted to ODF since it does not use precompiled code. In 2020 Müller et al. [62] presented a systematic analysis of ODF and OOXML documents. They investigated, inter alia, macro attacks but did not consider signatures.

Attacks on PDF Documents. In 2019, Mladenov et al. [59] presented attacks on digitally signed PDF documents. They targeted a similar attack goal (content spoofing). On an abstract level, they used comparable techniques (signature wrapping, universal forgery) for two of their attacks. However, they did not consider code execution attacks and concentrated on the PDF file format. In 2021, Mainka et al. [51] presented shadow attacks on PDF that are not applicable to ODF documents (cf. Section 4.2). In 2021, Rohlmann et al. [74] abuses PDF annotations for attacking PDF signatures. Annotation-based attacks on ODF are not applicable because ODF annotations are stored in `content.xml` which is protected by the signature. Various research groups conducted content masking attacks based on polyglots [1, 17, 73]. These attacks are not applicable on ODF documents due to its compression into a zip archive and the strict internal structure of the document containing different files.

XML-based Attacks. In 2002, Klein [45] and Steuck [78] used for the first time XML features to carry out efficient Denial-of-Service attacks. In the following years, additional malicious features have been systematically evaluated and further vulnerabilities were discovered [43, 61, 77]. We used their attack vectors and adapted the attacks on ODF documents. The first XSW attacks were published by McIntosh and Austel [54] in 2005. In the following years, several other XSW attacks and countermeasures were presented [39, 41, 60, 76, 89]. In 2019, Munoz and Mirosh [64] demonstrated a key confusion attack on SAML which is comparable to our Certificate Doubling attacks. The difference is that a symmetric key, instead of a trusted certificate, is added to the `<KeyInfo>` element to bypass the signature validation.

10 Discussion and Future Work

Future Research Directions. ODF is only one format in the family of XML-based file formats. When considering the transferablity of XML-based attacks to these formats, future research in various file formats is imaginable. There are further XML-based document formats, for example, the Open Packaging Conventions (OPC) family, which includes – inter alia – OOXML, 3D Manufacturing Format (3mf), and Autodesk AutoCAD Design Web Format (DWFX). Although these formats similarly support digital signatures, and thus directly relate to our research, further attacks, for example, ACE [36, 48, 63], should also be considered.

Proprietary Features. We identified proprietary features to be a desired source of attacks, as shown by the partial document signatures feature that lead to the Content Manipulation with Signature Upgrade attack. We were not the first to come to this conclusion, as the discoveries in recent years show. For example, the performance code feature in Microsoft Office led to the VBA stomping attack [16, 18, 69, 75]. Such proprietary features are not well documented. They require a thorough analysis of each implementation. A generic approach to deal with them is missing.

Memory-based Attack Verification. By using DocSV, the attack's success can be automatically evaluated on GUI applications, such as Microsoft Office (OOXML) or AutoCAD. In its current state, we only use DocSV to analyze the signature verification result, but other attacks could extend its scope. For example, it could be used to detect inconsistencies in the *presentation* of the document, but the evaluation of content masking [53] and polyglot [1, 17, 73] attacks would greatly benefit from this approach.

11 Conclusion

In this paper, we conducted the first comprehensive analysis of digital signatures in ODF documents. We described five attacks, three of which forged content and one of which forged the signature timestamp. Another attack even led to code execution on the victim's computer system. Using a novel memory-based evaluation approach for documents named DocSV, we showed that 17 out of 18 ODF applications were vulnerable.

Attacks' Root Causes. The discovered issues that our attacks abuse raise from an imprecise or incomplete description in the ODF standard and its subsequent standards. For example, the ODF signatures rely on using XML signatures. These can have multiple certificates, but the ODF standard does not describe what an implementation should do in such cases, which lead to Macro Manipulation with Certificate Doubling and Content Manipulation with Certificate Doubling attacks. Another example is the incomplete description of naming signature files in the ODF package. The renaming of `macrosignates.xml` to `documentsignatures.xml` was the foundation for the Content Manipulation with Signature Upgrade attack. These gaps leave developers on their own to properly address such cases, although all malicious documents presented in this paper remain compliant to the ODF/XML standards.

Lessons Learned. Document signatures and macro signatures are hard to implement flawlessly. To the best of our knowledge, we can find one reason in the inappropriate handling of partial signatures. Compared to classical file signatures, where the signature protects every byte, documents are more flexible. Particularly in the case of ODF, document and macro signatures leave unprotected gaps in the ODF package. The gaps in combination with imprecise or incomplete descriptions in the standards open the potential for severe threats, as shown in this paper. For countering these threats, future standards should be more precise. In addition, we see the demand for automatic attack evaluation in the field of document security. DocSV could serve as a foundation for this automation.

Acknowledgment

We would like to thank Steve Martin for his contributions in developing the initial attacks in his master's thesis, David Dankelmann for his idea of using memory analysis for signature validation, and Daniele Antonioli for shepherding this paper. Simon Rohlmann was supported by the German Federal Ministry of Economics and Technology (BMWi) project "Industrie 4.0 Recht-Testbed" (13I40V002C). This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.

References

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

[2] Apache Software Foundation. Merging Lotus Symphony: Allegro moderato, . URL https://blogs.apache.org/ooo/entry/merging_lotus_symphony_allegro_moderato.

[3] Apache Software Foundation. The Apache OpenOffice API Project, . URL https://www.openoffice.org/api/.

[4] Apache Software Foundation. Apache OpenOffice BASIC Programming Guide, . URL https://wiki.openoffice.org/wiki/Documentation/BASIC_Guide.

[5] Apache Software Foundation. Security/Digital Signatures, . URL https://wiki.openoffice.org/wiki/Security/Digital_Signatures.

[6] Apache Software Foundation. Apache OpenOffice Basic IDE, . URL https://wiki.openoffice.org/wiki/Documentation/DevGuide/Basic/OpenOffice.org_Basic_IDE.

[7] Apache Software Foundation. Service Events, . URL https://www.openoffice.org/api/docs/common/ref/com/sun/star/document/Events.html.

[8] Apache Software Foundation. Scripting Framework, . URL https://wiki.openoffice.org/wiki/Documentation/DevGuide/Scripting/Scripting_Framework.

[9] Apache Software Foundation. FAQs, . URL http://www.openoffice.org/FAQs/faq-overview.html.

[10] Apache Software Foundation. Symphony contribution, . URL https://wiki.openoffice.org/wiki/Symphony_contribution.

[11] Apache Software Foundation. The Structure of Spreadsheet Documents, . URL https://wiki.openoffice.org/wiki/Documentation/BASIC_Guide/Structure_of_Spreadsheets.

[12] Apache Software Foundation. VBA interoperability in OpenOffice, . URL https://wiki.openoffice.org/wiki/VBA_interoperability_in_OpenOffice.

[13] Apple Inc. App Store. URL https://www.apple.com/app-store/.

[14] Mehdi Assefi. Ocr as a service: An experimental evaluation of google docs ocr, tesseract, abbyy finereader, and transym. *International Symposium on Visual Computing*, pages 735–746, 12 2016.

[15] Australian Cyber Security Centre (ACSC). Update on Petya ransomware campaign. URL https://www.cyber.gov.au/acsc/view-all-content/advisories/update-petya-ransomware-campaign.

[16] Vesselin Bontche. pcodedmp.py - a vba p-code disassembler. URL https://github.com/bontchev/pcodedmp.

[17] Francesco Buccafurri, Gianluca Caminiti, and Gianluca Lax. Fortifying the dalì attack on digital signature. In *Proceedings of the 2nd International Conference on Security of Information and Networks*, SIN '09, page 278–287, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605584126. doi: 10.1145/1626195.1626262. URL https://doi.org/10.1145/1626195.1626262.

[18] Pieter Ceelen and Stan Hegt. Ms office file format sorcery. In *Troopers*, March 2019. URL https://github.com/outflanknl/Presentations/blob/master/Troopers19_MS_Office_file_format_sorcery.pdf.

[19] Central Digital and Data Office. Using Open Document Formats (ODF) in your organisation. URL https://www.gov.uk/guidance/using-open-document-formats-odf-in-your-organisation#securing-your-odf-compliant-solution.

[20] Collabora Ltd. Programming with Collabora Office Basic. URL https://help.collaboraoffice.com/latest/en-US/text/sbasic/shared/01000000.html?DbPAR=BASIC.

[21] David de Drézigué, Jean-Paul Fizaine, and Nils Hansma. In-depth analysis of the viral threats with OpenOffice.org documents. In *Journal in Computer Virology*, volume 2, pages 187–210, dec 2006. doi: 10.1007/s11416-006-0020-2. URL https://doi.org/10.1007/s11416-006-0020-2.

[22] Didier Stevens. Tampering with Digitally Signed VBA Projects, 2020. URL https://blog.nviso.eu/2020/06/04/tampering-with-digitally-signed-vba-projects/.

[23] European Commission. France's RGI v2 recommends ODF, . URL https://joinup.ec.europa.eu/collection/open-source-observatory-osor/news/frances-rgi-v2-recommends-odf.

[24] European Commission. Italian military to save 26-29 million Euro by migrating to LibreOffice, . URL https://joinup.ec.europa.eu/collection/open-source-observatory-osor/news/italian-military-save-26-2.

[25] European Commission. MIMO: a working group of French ministries to certify a LibreOffice release, . URL https://joinup.ec.europa.eu/collection/open-source-observatory-osor/document/mimo-working-group-french-ministries-certify-libreoffice-release.

[26] European Commission. EU: NATO makes ODF one of its mandatory standards, . URL https://joinup.ec.europa.eu/collection/egovernment/news/eu-nato-makes-odf-one-its.

[27] European Commission. Taiwanese government standardises on true ODF document format, . URL https://joinup.ec.europa.eu/collection/open-source-observatory-osor/document/taiwanese-government-standardises-true-odf-document-format.

[28] European Commission. Valencia region government completes switch to LibreOffice, . URL https://joinup.ec.europa.eu/collection/open-source-observatory-osor/news/valencia-region-government-co.

[29] European Telecommunications Standards Institute (ETSI). *XML Advanced Electronic Signatures (XAdES)*, Jun. 2009. URL https://www.etsi.org/deliver/etsi_ts/101900_101999/101903/01.04.01_60/ts_101903v010401p.pdf.

[30] European Union Agency for Law Enforcement Cooperation (Europol). World's most dangerous malware EMOTET disrupted through global action. URL https://www.europol.europa.eu/newsroom/news/world%E2%80%99s-most-dangerous-malware-emotet-disrupted-through-global-action.

[31] Eric Filiol. Openoffice v3.x security design weaknesses. In *Black Hat Europe*, April 2009. URL https://www.blackhat.com/presentations/bh-europe-09/Filiol_Fizaine/BlackHat-Europe-09-Filiol-Fizaine-OpenOffice-Weaknesses-slides.pdf.

[32] Eric Filiol and Jean-Paul Fizaine. Openoffice security and viral risk – part one. In *Virus Bulletin Journal*, September 2007. URL https://www.virusbulletin.com/virusbulletin/2007/09/openoffice-security-and-viral-risk-part-one.

[33] James Forshaw. Exploiting XML Digital Signature Implementations, 2013. URL https://conference.hitb.org/hitbsecconf2013kul/materials/D2T1%20-%20James%20Forshaw%20-%20Exploiting%20XML%20Digital%20Signature%20Implementations.pdf.

[34] French government. Arrêté du 20 avril 2016 portant approbation du référentiel général d'interopérabilité. URL https://www.legifrance.gouv.fr/jorf/id/JORFTEXT000032438896.

[35] G DATA CyberDefense AG. Ransomware on the Rise: Buran's transformation into Zeppelin. URL https://www.gdatasoftware.com/blog/2020/06/35946-burans-transformation-into-zeppelin.

[36] Jacob Gajek. Macro malware: dissecting a malicious word document. *Network Security*, 2017(5):8–13, 2017. ISSN

1353-4858. doi: https://doi.org/10.1016/S1353-4858(17)30049-1. URL https://www.sciencedirect.com/science/article/pii/S1353485817300491.

[37] Google Ireland Limited. Google Play. URL https://play.google.com/store.

[38] Government Digital Service (GDS). Open document formats selected to meet user needs. URL https://www.gov.uk/government/news/open-document-formats-selected-to-meet-user-needs.

[39] Abhinav Nath Gupta and P. Santhi Thilagam. Detection of xml signature wrapping attack using node counting. In V. Vijayakumar and V. Neelanarayanan, editors, *Proceedings of the 3rd International Symposium on Big Data and Cloud Computing Challenges (ISBCC – 16')*, pages 57–63, Cham, 2016. Springer International Publishing. ISBN 978-3-319-30348-2.

[40] Frederick Hirsch, David Solo, Joseph Reagle, Donald Eastlake, and Thomas Roessler. XML signature syntax and processing (second edition). W3C recommendation, W3C, June 2008. URL http://www.w3.org/TR/2008/REC-xmldsig-core-20080610/.

[41] Paul Höller, Alexander Krumeich, and Luigi Lo Iacono. Xml signature wrapping still considered harmful: A case study on the personal health record in germany. In Audun Jøsang, Lynn Futcher, and Janne Hagen, editors, *ICT Systems Security and Privacy Protection*, pages 3–18, Cham, 2021. Springer International Publishing. ISBN 978-3-030-78120-0.

[42] James M. Thompson. *Porting Excel/VBA to Calc/StarBasic*, Jun. 2004. URL http://www.openoffice.org/documentation/HOW_TO/various_topics/VbaStarBasicXref.pdf.

[43] Sadeeq Jan, Cu D. Nguyen, and Lionel Briand. Known XML Vulnerabilities Are Still a Threat to Popular Parsers and Open Source Systems. In *2015 IEEE International Conference on Software Quality, Reliability and Security*, pages 233–241. IEEE, August 2015. doi: 10.1109/qrs.2015.42. URL http://ieeexplore.ieee.org/document/7272938/.

[44] Kelby Ludwig. Duo Finds SAML Vulnerabilities Affecting Multiple Implementations. URL https://duo.com/blog/duo-finds-saml-vulnerabilities-affecting-multiple-implementations.

[45] Klein. Multiple vendors XML parser (and SOAP/WebServices server) Denial of Service attack using DTD, 2002. URL http://www.securityfocus.com/archive/1/303509.

[46] Tomasz Kuchta, Thibaud Lutellier, Edmund Wong, Lin Tan, and Cristian Cadar. *On the correctness of electronic documents: studying, finding, and localizing inconsistency bugs in PDF readers and files*, volume 23. Empirical Software Engineering, 2018. ISBN 1066401896. doi: 10.1007/s10664-018-9600-2.

[47] Philippe Lagadec. OpenDocument and Open XML security (OpenOffice.org and MS Office 2007). In *Journal in Computer Virology*, volume 4, pages 115–125, may 2008. doi: 10.1007/s11416-007-0060-2. URL https://doi.org/10.1007/s11416-007-0060-2.

[48] Philippe Lagadecl. Advanced vba macros attack & defence. In *Black Hat Europe*, December 2019. URL https://www.decalage.info/files/eu-19-Lagadec-Advanced-VBA-Macros-Attack-And-Defence.pdf.

[49] LibreOffice Documentation Team. *Getting Started Guide*, Jun. 2021. URL https://documentation.libreoffice.org/assets/Uploads/Documentation/en/GS7.1/GS71-GettingStarted.pdf.

[50] Christian Mainka, Vladislav Mladenov, Florian Feldmann, Julian Krautwald, and Jörg Schwenk. Your software at my service: Security analysis of SaaS single sign-on solutions in the cloud. In *Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security*, October 2014. URL https://dl.acm.org/doi/10.1145/2664168.2664172.

[51] Christian Mainka, Vladislav Mladenov, and Simon Rohlmann. Shadow Attacks: Hiding and Replacing Content in Signed PDFs. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, 2021. URL https://www.ndss-symposium.org/ndss-paper/shadow-attacks-hiding-and-replacing-content-in-signed-pdfs/.

[52] Malwarebytes Inc. Ransom.Locky. URL https://blog.malwarebytes.com/detections/ransom-locky/.

[53] Ian Markwood, Dakun Shen, Yao Liu, and Zhuo Lu. PDF Mirage: Content Masking Attack Against Information-Based Online Services. In *26th USENIX Security Symposium (USENIX Security 17), (Vancouver, BC)*, pages 833–847, 2017.

[54] Michael McIntosh and Paula Austel. XML signature element wrapping attacks and countermeasures. In *Proceedings of the 2005 Workshop on Secure Web Services*, SWS '05, page 20–27, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595932348. doi: 10.1145/1103022.1103026. URL https://doi.org/10.1145/1103022.1103026.

[55] Microsoft Corporation. about Execution Policies, . URL https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_execution_policies?view=powershell-7.1.

[56] Microsoft Corporation. Office VBA Reference, . URL https://docs.microsoft.com/en-us/office/vba/api/overview/.

[57] Microsoft Defender Security Research Team. New feature in Office 2016 can block macros and help prevent infection. URL https://www.microsoft.com/security/blog/2016/03/22/new-feature-in-office-2016-can-block-macros-and-help-prevent-infection/.

[58] Ministry of Digital Development of the Russian Federation. в россии вступил в действие национальный стандарт открытых офисных приложений OpenDocument (ODF). URL https://digital.gov.ru/ru/events/27931/.

[59] Vladislav Mladenov, Christian Mainka, Karsten Meyer zu Selhausen, Martin Grothe, and Jörg Schwenk. 1 trillion dollar refund: How to spoof PDF signatures. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 1–14. ACM, 2019. doi: 10.1145/3319535.3339812. URL https://doi.org/10.1145/3319535.3339812.

[60] Subrata Modak, Koushik Majumder, and Debashis De. Vulnerability of cloud: Analysis of xml signature wrapping attack and countermeasures. In Debotosh Bhattacharjee, Dipak Kumar Kole, Nilanjan Dey, Subhadip Basu, and Dariusz Plewczynski, editors, *Proceedings of International Conference on Frontiers in Computing and Systems*, pages 755–765, Singapore, 2021. Springer Singapore. ISBN 978-981-15-7834-2.

[61] Timothy D Morgan and Omar Al Ibrahim. XML Schema, DTD, and Entity Attacks . Technical report, may 2014. URL https://vsecurity.com//download/papers/XMLDTDEntityAttacks.pdf.

[62] Jens Müller, Fabian Ising, Christian Mainka, Vladislav Mladenov, Sebastian Schinzel, and Jörg Schwenk. Office document security and privacy. In *14th USENIX Workshop on Offensive Technologies (WOOT 20)*. USENIX Association, August 2020. URL https://www.usenix.org/conference/woot20/presentation/muller.

[63] Jens Müller, Dominik Noß, Christian Mainka, Vladislav Mladenov, and Jörg Schwenk. Processing Dangerous Paths - On Security and Privacy of the Portable Document Format. In *Proceedings 2019 Network and Distributed System Security Symposium*. Internet Society, February 2021.

[64] Alvaro Munoz and Oleksandr Mirosh. SSO Wars: The Token Men-

ace, 2019. URL https://www.blackhat.com/us-19/briefings/schedule/index.html#sso-wars-the-token-menace-15092.

[65] NATO. NATO Interoperability Standards and Profiles - Agreed Interoperability Standards and Profiles. URL https://nhqc3s.hq.nato.int/Apps/Architecture/NISP/volume2/ch03s02.html.

[66] NIST. National Vulnerability Database. URL https://nvd.nist.gov/.

[67] OASIS Open. *Open Document Format for Office Applications (OpenDocument) Version 1.2*, Sep. 2011. URL http://docs.oasis-open.org/office/v1.2/.

[68] OASIS Open. *Open Document Format for Office Applications (OpenDocument) Version 1.3*, Apr. 2021. URL https://docs.oasis-open.org/office/OpenDocument/v1.3/.

[69] Harold Ogden, Kirk Sayre, and Carrie Roberts. Vba stomping advanced malicious document techniques. In *DerbyCon*, December 2018. URL https://github.com/clr2of8/Presentations/blob/master/DerbyCon2018-VBAstomp-Final-WalmartRedact.pdf.

[70] OpenDoc Society. About OpenDocument Format. URL https://opendocumentformat.org/aboutODF/.

[71] Planamesa Inc. Using Macros. URL https://neowiki.neooffice.org/index.php/Using_Macros.

[72] Henrich C. Pöhls and Lars Westphal. Die "untiefen" der neuen xml-basierten dokumentenformate. In *Black Hat Europe*, 2008. URL http://henrich.poehls.com/papers/2008_Poehls_Westphal_2008_DFN-CERT-WS_Untiefen_der_XML-Dokumentenformate.pdf.

[73] Dan-Sabin Popescu. Hiding malicious content in PDF documents. *CoRR*, abs/1201.0397, 2012. URL http://arxiv.org/abs/1201.0397.

[74] Simon Rohlmann, Vladislav Mladenov, Christian Mainka, and Jörg Schwenk. Breaking the Specification: PDF Certification. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1485–1501, Los Alamitos, CA, USA, May 2021. IEEE Computer Society. doi: 10.1109/SP40001.2021.00110. URL https://doi.ieeecomputersociety.org/10.1109/SP40001.2021.00110.

[75] Kirk Sayre and Carrie Roberts. Advanced malware vba stomping. In *Sp4rkCon*, May 2019. URL https://github.com/clr2of8/Presentations/blob/master/Sp4rkCon2019-VBAstomp.pdf.

[76] Juraj Somorovsky, Andreas Mayer, Jörg Schwenk, Marco Kampmann, and Meiko Jensen. On breaking saml: Be whoever you want to be. In *21st USENIX Security Symposium*, Bellevue, WA, August 2012. URL https://www.usenix.org/system/files/conference/usenixsecurity12/sec12-final91.pdf.

[77] Christopher Späth, Christian Mainka, Vladislav Mladenov, and Jörg Schwenk. SoK: XML parser vulnerabilities. In *USENIX Workshop on Offensive Technologies (WOOT)*, August 2016. URL https://www.usenix.org/system/files/conference/woot16/woot16-paper-spath.pdf.

[78] Gregory Steuck. XXE (Xml eXternal Entity) Attack, 2002. URL http://www.securiteam.com/securitynews/6D0100A5PU.html.

[79] The Document Foundation. LibreOffice 7.2 API Documentation, . URL https://api.libreoffice.org/.

[80] The Document Foundation. Programming with LibreOffice Basic, . URL https://help.libreoffice.org/latest/en-US/text/sbasic/shared/01000000.html?DbPAR=BASIC.

[81] The Document Foundation. Applying Digital Signatures, . URL https://help.libreoffice.org/latest/en-US/text/shared/guide/digitalsign_send.html.

[82] The Document Foundation. LibreOffice Basic IDE, . URL https://help.libreoffice.org/latest/en-US/text/sbasic/shared/01050000.html?DbPAR=BASIC.

[83] The Document Foundation. Document Event-Driven Macros, . URL https://help.libreoffice.org/latest/ro/text/sbasic/shared/01040000.html.

[84] The Document Foundation. LibreOffice Migrations, . URL https://wiki.documentfoundation.org/LibreOffice_Migrations.

[85] The Document Foundation. BASIC, . URL https://wiki.documentfoundation.org/BASIC.

[86] The Document Foundation. Working with VBA Macros, . URL https://help.libreoffice.org/latest/lo/text/sbasic/shared/vbasupport.html.

[87] The Document Foundation. XSecParser Class Reference, . URL https://docs.libreoffice.org/xmlsecurity/html/classXSecParser.html.

[88] W3C. Extensible Markup Language (XML) 1.0 (Fifth Edition). URL https://www.w3.org/TR/xml/.

[89] Gerard Wawrzyniak and Imed El Fray. New xml signature scheme that is resistant to some attacks. *IEEE Access*, 8:35815–35831, 2020. doi: 10.1109/ACCESS.2020.2975034.

[90] Wikipedia. OpenDocument software, . URL https://en.wikipedia.org/wiki/OpenDocument_software.

[91] Wikipedia. List of word processors, . URL https://en.wikipedia.org/wiki/List_of_word_processors.

[92] Yu Kaijun. Upgrade signed Office VBA macro projects to V3 signature, 2021. URL https://developer.microsoft.com/en-us/sharepoint/blogs/upgrade-signed-office-vba-macro-projects-to-v3-signature/.

12 Appendix

12.1 Macro Example

```
Sub Macro1
    MsgBox("Hello world!")
End Sub
```

Listing 1: Basic code displaying a message box.

```
<script:event-listener script:language="ooo:script"
    script:event-name="dom:load" xlink:type="simple"
    xlink:href="vnd.sun.star.script:Standard.Module1
    ↪ .Macro1?language=Basic&amp;location=document"
/>
```

Listing 2: Once the document is opened (`dom:load`), "Macro1" is triggered via the `xlink:href` in the file `content.xml` (see Listing 1).