Die „Untiefen“ der neuen XML-basierten Dokumentenformate

Henrich C. Pöhls Lars Westphal
University of Hamburg, Dept. of Informatics
Research Group Security in Distributed Systems (SVS)
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
(poehls|9westpha)@informatik.uni-hamburg.de

1 Einleitung

Der Austausch elektronischer Dokumente gehört inzwischen zum Alltag. Die Dokumentenformate haben sich inzwischen von geschlossenen proprietären Formaten zu offenen XML-basierten Formaten gewandelt.

So neu sind die XML-basierten Dokumentenformate keineswegs. Das *Open Document Format* (ODF) von OpenOffice.org wurde schon im Mai 2005 als OASIS-Standard [24] verabschiedet. Und auch schon in Microsoft Office 2003 waren erste Ansätze erkennbar, doch erst seit Microsoft Office 2007 haben sie durch DOCX nochmals an Bekanntheit gewonnen. Im November 2006 wurde ODF als ISO/IEC DIS 26300 [14] in die Riege der ISO-Standards eingereiht. Microsoft versucht derzeit, *Office Open XML* (OOXML), wie das Dokumentenformat offiziell heißt, ebenfalls zu standardisieren und zwar auch als ISO Standard. Die nicht unumstrittene Entscheidung [10] dazu fällt voraussichtlich im Februar 2008. Welches Format sich in Zukunft durchsetzen oder für den jeweiligen Anwendungsfall besser geeignet ist, wollen wir hier aber nicht weiter thematisieren. Dies käme eher einem Blick in die Kristallkugel gleich. Stattdessen werden wir einige sicherheitsrelevante Eigenschaften und problematische Eigenheiten diskutieren, die man an den bereits existierenden Formaten und den dazugehörigen Implementationen schon jetzt beobachten kann.

1.1 Genereller Aufbau, Struktur und Anwendungen im Vergleich

Wir werden im Folgenden auf einige sicherheitsrelevante Probleme und Besonderheiten beider Formate näher eingehen. Dabei haben wir uns die Formate näher angesehen, allerdings weniger im Hinblick auf das was in den Standards steht, sondern den Fokus auf den

Output der beiden Textverarbeitungen gelegt. Untersucht wurden dabei Microsoft Word 2007 in der Version 12.0.6015.5000[1] und OpenOffice.org Writer in der Version 2.3.0. Wir werden daher im folgenden von DOCX-Dateien sprechen, wenn sich die untersuchten Eigenheiten auf ein MS Word-Dokument im OOXML-Format beziehen, beziehungsweise diese Begriffe synonym verwenden. Generell können beide Formate natürlich auch von anderen Anwendungen erstellt und bearbeitet werden, dies ist ja der größte Vorteil der Offenlegung der Formate. So gibt es bereits Bibliotheken für gängige Programmiersprachen, mit denen beispielsweise OOXML-Dokumente bearbeitet werden können (z.B. OpenXML4J für Java [25]).

Zum Aufbau: Beide Formate sind sogenannte *Containerformate*. Das bedeutet, sie enthalten mehrere mit ZIP [28] komprimierte Dateien. Dies ermöglicht, unterschiedliche Inhalte in verschiedenen Dateien innerhalb des Containers abzulegen. Dadurch erhöht sich die Modularität und es trägt zum Teil auch zu einer besseren Übersicht bei.

Der Container von OOXML gliedert sich grob in drei Bereiche [8] [7, erstes Buch]:

- *Part items*: Entsprechen einen Dokumententeil, z.B. enthält `/word/document.xml` den eigentlichen Dokumentinhalt.
- *Content Type items*: Enthalten zusätzlich hinzugefügte Objekte (z.B. Bilder).
- *Relationship items*: Verbinden mehrere Part und Content Type items zu dem vollständigen Dokument. `/_rels/.rels}` "verlinkt"das Hauptdokument `document.xml` mit Metainformationen (z.B. `/docProps/core.xml`) oder einer digitalen Signatur.

Bei OpenOffice besteht der Container aus XML-Dateien und optionalen Inhalten. Die wichtigen XML-Dateien sind:

- `content.xml`: Enthält den Textkörper, den eigentlichen Dokumentinhalt.
- `styles.xml`: Beschreibt die Stile (Formatierungen), die im Dokument verwendet werden.
- `meta.xml`: Besteht aus Angaben zu Autor und Dokument.
- `settings.xml`: Speichert die Einstellungen zum Dokument (nicht-globale Einstellungen).
- `/META-INF/manifest.xml`: Führt alle Dateien innerhalb des Containers auf.

Zu den optionalen Dateien gehören Vorschaubilder, weitere Einstellungen und eingebettete Objekte. Diese werden in Unterordner abgelegt.

Die beiden Container sind sich prinzipiell sehr ähnlich, allerdings macht bei OOXML die lose Kopplung der verschiedenene Dateien über verschiedene Relationship items es nicht immer einfach, die relevanten XML Dateien zu finden.

[1] Microsoft Office in der Version 12.0.6017.5000

1.2 Betrachtete sicherheitsrelevante Eigenschaften

Es handelt sich bei den betrachteten Formaten beziehungsweise deren Anwendungen um Standardprodukte für den Arbeitsalltag mit elektronischen Dokumenten. Diese elektonischen Dokumente werden von ihren Anwendern ganz selbstverständlich bearbeitet, verschickt und wieder mit der Standardanwendung betrachtet. Dabei sind viele Eigenschaften, z.B. die Container-Eigenschaft, komplett transparent für den Nutzer. Dies ist auf der einen Seite von Vorteil, da es unnötige Komplexität versteckt, auf der anderen Seite ist dies aber auch ein Nachteil, denn die Komplexität führt zu neuen sicherheitsrelevanten Problemen, die vom durchschnittlichen Anwender nicht erkannt werden. So bieten beide Anwendungen die Möglichkeit, das Dokument digital zu signieren oder zusätzliche Objekte (wie Grafiken oder Daten) einzufügen. Gerade für einen durchschnittlichen Nutzer sollten die Sicherheitsfunktionen trotz Transparenz benutzbar sein. Allerdings treten in der Praxis Probleme auf, welche ohne ein tieferes Verständnis des Aufbaus der Dokumentenformate nicht zu verstehen oder vorherzusehen sind.

Wir werden im Rahmen dieser Arbeit folgende sicherheitsrelevante Eigenschaften beider Formate näher beleuchten:

- **Ungewollte Metainformationen**: Informationen, die nicht zu den sichtbaren Inhalten des Dokumentes gehören, aber dennoch erzeugt oder erhalten geblieben sind. Sie sind im Dokument gespeichert, ohne das der Nutzer es direkt sieht. Diese Informationen könnten vertraulich sein, daher stellt die mangelnde Transparenz hier ein Sicherheitsproblem dar.
- **Digitale Signaturen**: In beiden Office-Anwendungen kann man Dokumente mit einer digitalen Signatur versehen. Wir haben beispielhaft untersucht, welche relevanten Bestandteile eines signierten Dokuments verändert werden können, ohne die Signatur zu invalidieren. Es ist wichtig, dass der Nutzer weiß, was genau dem Schutz der digitalen Signatur unterliegt und was nicht. Auch werden wir kurz die grafische Benutzerschnittstelle (GUI) in beiden Anwendungen betrachten.
- **Ungewollte Container-Inhalte**: Die Container-Eigenschaft der Dokumentenformate erlaubt den Transport beliebiger Dateiformate (Binary, Script etc). Wir haben untersucht, welchen Einschränkungen dies bei OOOXML und dem OODF unterliegt.
- **Verbergen von Informationen**: Wir zeigen, wo sich Informationen gezielt im XML-Code des Dokumentes verstecken ließen (Covert Channels oder Information Hiding).

Diese Auswahl an Problemen oder Eigenschaften erhebt nicht den Anspruch auf Vollständigkeit, es werden beispielsweise keine Viren oder maliziöse, aktive Inhalte untersucht. Bei Interesse an diesem Thema sei auf Abschnitt 3 zu themenverwandten Arbeiten hingewiesen. Wir haben bewusst diese Auswahl getroffen, um speziell auf zwei Dinge hinzuweisen: Erstens sind es nicht immer die intentional maliziösen Inhalte, die Probleme verursachen, unauthorisierte Personen dürfen über harmlos aussehende Dokumente nicht an vertrauliche Informationen gelangen oder diese darin verstecken. Zweitens schafft die fehlende Transparenz bei steigender Komplexität neue sicherheitsrelevante Probleme.

2 Untersuchte „Untiefen“

2.1 Ungewollte Informationen

Die Dokumentenformate erlauben es, Metainformationen über das Dokument anzulegen bzw. die Applikationen generieren diese Informationen automatisch. Zusätzlich zu den herkömmlichen Metainformationen über das Dokument selber und dessen Autor werden auch Informationen über eingebettete Objekte generiert und die Objekte selber abgelegt. Im Folgenden geben wir einen Einblick in die zusätzlichen Informationen, die im Container zum Dokument und in eingebetteten Objekten gespeichert sind. Diese Untersuchung von Meta- und Umgebungsinformationen erhebt aber keinen Anspruch auf Vollständigkeit, denn obgleich die Standards offen sind, müsste man zusätzlich die Implementationen der Standards in ihrer Gänze untersuchen um jedes mögliche XML-Element innerhalb aller Dateien innerhalb des Containers zudurchdringen. Eine solche Untersuchung wäre sehr aufwändig. Für weitergehende Untersuchungen zu verschiedenen Teilproblemen sei auf Abschnitt 3 verwiesen.

2.1.1 Metainformationen des Dokumentes

In OOXML werden die Metainformationen in zwei Dateien abgelegt, nämlich `/docProps/core.xml` und `/docProps/app.xml`. Die erste hält sich zum größten Teil an die von der Dublin Core Initiative [12] vorgegebenen Bezeichner und beschreibt mit diesen: *Titel*, *Betreff*, *Autor*, *Beschreibung*, *Erstellungs-* und *Änderungsdatum*. Zusätzlich werden noch in einem OOXML-eigenen Namensraum *Schlüsselwörter*, *Revisionsnummer*, *Kategorie* sowie der *Autor der letzten Änderung* festgehalten. Diese Werte sind auch über eine entsprechende GUI erreichbar, wobei die Datumswerte und der Wert des *Autors der letzten Änderung* automatisch generiert werden. Weitere Metainformationen werden in der Datei `app.xml` gespeichert. Hier werden Informationen wie *Seiten-*, *Wort-* und *Zeichenanzahl* sowie die verwendete *MS Office-Anwendung* und ihre *Versionsnummer* gespeichert.

Vielleicht ist die Diskussion über einen Globally Unique Identifier (GUID) in Microsoft Office-Dokumenten noch im Gedächnis [20] [22] [26]. Diese GUID lässt in Office 2003 und vorherigen Versionen durch Einbeziehung der MAC-Adresse Rückschlüsse auf den für die Erstellung benutzten Rechner zu. In OOXML haben wir eine solche GUID nicht gefunden, allerdings wird eine ganze Reihe von zusätzlichen Informationen gesammelt, wenn man eine DOCX-Datei signiert. In `/_xmlsignatures/sig1.xml` wird die Signatur gemäß dem XML Signature Standard [6] abgelegt. Zusätzlich zu den signaturrelevanten XML-Elementen enthält die Datei im Zweig `SignatureInfoV1` weitere Informationen. Wie in Abbildung 1 ersichtlich, werden dort auch *Windowsversion*, *Anzahl der Bildschirme* und deren *Auflösung* gespeichert. Es ist fragwürdig, warum diese Information im Rahmen des Signaturerstellungsprozesses gespeichert und im Dokument festgehalten wird. Ihre Speicherung ist aber zumindest, wenn auch erst nach mehreren Dialogen, über die GUI einsehbar (vgl. Abb. 1).

OpenOffice-Dokumente enthalten ebenfalls Metainformationen. Diese werden in der Datei `meta.xml` gespeichert. Hier werden ebenfalls DublinCore-Elemente genutzt, nämlich

```
<SignatureInfoV1 xmlns="http://schemas.microsoft.com/office/2006/digsig">
        <SetupID/>
        <SignatureText/>
        <SignatureImage/>
        <SignatureComments>abcde</SignatureComments>
        <WindowsVersion>5.1</WindowsVersion>
        <OfficeVersion>12.0</OfficeVersion>
        <ApplicationVersion>12.0</ApplicationVersion>
        <Monitors>2</Monitors>
        <HorizontalResolution>1280</HorizontalResolution>
        <VerticalResolution>1024</VerticalResolution>
        <ColorDepth>32</ColorDepth>
        <SignatureProviderId>{00000000-0000-0000-0000-000000000000}</SignatureProviderId>
        <SignatureProviderUrl/>
        <SignatureProviderDetails>9</SignatureProviderDetails>
        <ManifestHashAlgorithm>http://www.w3.org/2000/09/xmldsig#sha1</ManifestHashAlgorithm>
        <SignatureType>1</SignatureType>
</SignatureInfoV1>
```

Abbildung 1: Word 2007 - Zusätzlich gesammelte Signierungsinformationen

Titel, *Betreff*, *letzter Autor* und *Speicherungs-Datum*. In einem zusätzlicher Namensraum speichert OpenOffice unter anderem das *Erstellungsdatum* und den *initialen Autor* sowie die *OpenOffice-Versionsnummer*. Und auch hier werden Statistikinformationen über den Text (z.B. *Anzahl Seiten*, *Zeichen*, *Wörter*, *Objekte*) gespeichert. Allerdings wird auch die *Anzahl der Bearbeitungen* und die *Bearbeitungsdauer* mit in den Metadaten abgelegt. Die meisten Felder sind über einen GUI-Dialog einsehbar und editierbar. Die Datumsfelder und Autoreninformationen werden automatisch ausgefüllt, die Felder *initialer Autor* und *Erstellungsdatum* sind einsehbar, werden allerdings von den OpenOffice-Anwendungen nicht mehr verändert. Es gibt zusätzlich einen Button in der GUI, um das Speichern der vom Benutzer erstellten Metainformationen zu unterdrücken bzw. diese zu entfernen (vgl. Abb. 2). Allerdings betrifft dies nur die vom Benutzer zusätzlich eingegebenen Informationen. Es verbleiben weiterhin folgende Informationen im Dokument: *OpenOffice-Versionsnummer*, *initialer Autor*, *letzter Autor*, die *Erstellungs-* und *Änderungszeitpunkte* sowie die Statistikinformationen.

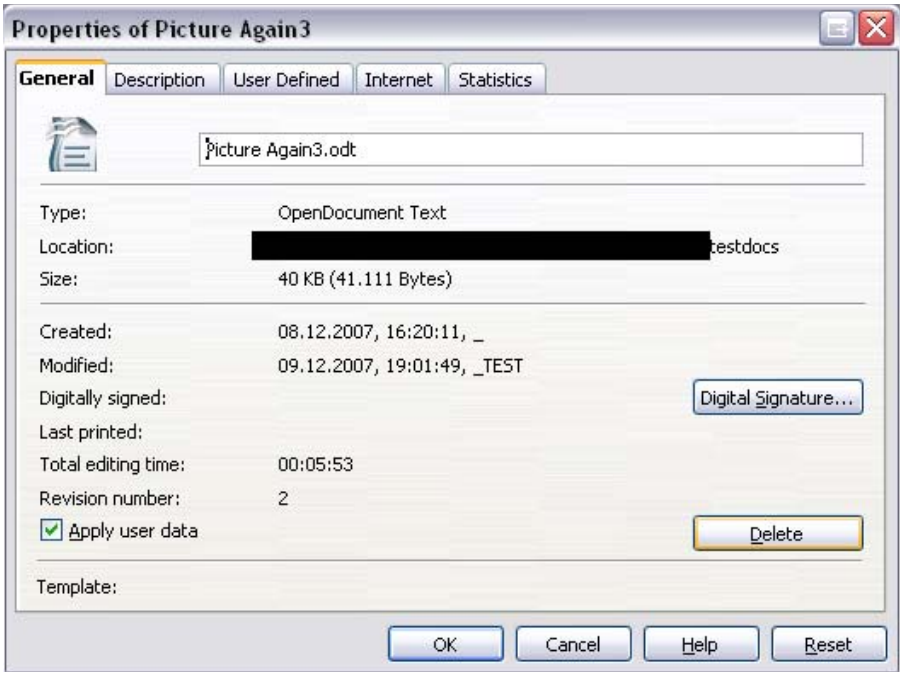

Abbildung 2: Writer - GUI zum Entfernen von benutzergenerierten Metainformationen

2.1.2 Metadaten eingebetteter Objekte

Die Container-Eigenschaft ermöglicht es, Objekte direkt in ihrem nativen Format im Container des Dokumentes als Datei abzulegen. Das allgegenwärtigste Objekt in Dokumenten sind Grafiken. Im Dokument enthaltene Grafiken werden zunächst unverändert im Container abgelegt und dann vom Dokument referenziert.

In DOCX beispielsweise liegt ein eingefügtes Beispielbild als `/doc/media/image1.jpeg` im Container. Im Container wurde sein ursprünglicher Dateiname von MS Office ersetzt. MS Office erkennt auch, wenn das gleiche Bild mit unterschiedlichen Dateinamen mehrfach eingebunden wird und nimmt es dennoch nur einmal in den Container auf. In einer zusätzlichen Datei `/doc/media/_rels/document.xml.rels` wird dann für jedes mit dem Dokument verbundenen Objekt ein Identifier vergeben, z.B. `rId4`. Das ist grundsätzlich gut, es entsteht jedoch trotzdem ein Informationsleck, da der ursprüngliche Dateiname in `document.xml` als Attribut der Referenz zu `rId4` als `name="BlaueBerge.jpg"` und dann nochmals als sog. `descr="BlaueBerge.jpg"` wieder auftaucht. Diese Zusatzinformation über den ursprünglichen Dateinamen (ohne Pfad) ist nur direkt aus der XML-Datei ersichtlich; eine Funktion, die diese Information auch über die GUI einsehbar macht, ist nicht vorhanden. Diese ungewollte Zusatzinformation bleibt also einem normalen Nutzer verborgen und stellt ein potentielles Sicherheitsproblem dar.

In OpenOffice.org erhalten Bilder einen neuen Dateinamen. Das gleiche Beispielbild wird in `/Pictures/10000000000003200000002582C99D0FF.jpg` im Container abgelegt. Dieser Bezeichner wird, wie Tests ergeben haben, auf der Basis der Dateiinhalte bestimmt, d.h. gleiche Objekte erhalten gleiche interne Bezeichner, ungeachtet der ursprünglichen Dateinamen. Doppelte Bilder werden damit auch nur einmal in den Container aufgenommen. In der Datei `/META-INF/manifest.xml` und im Dokument, also unter `content.xml`,

wird es dann entsprechend direkt über diesen Bezeichner referenziert. Zusätzliche Informationen über den ursprünglichen Dateinamen werden also entfernt.

2.1.3 Rohdaten eingebetteter Objekte

Wie bereits erwähnt befinden sich die eingebetteten Objekte direkt als Dateien im Container. Dies hat bei Abbildungen zur Folge, dass nach Bearbeitungsschritten wie „Zuschneiden“ das Originalbild weiterhin im Container gespeichert bleibt. Beim Zuschneiden wird der sichtbare Ausschnitt eines Bildes verändert. Dies wird jedoch nicht auf der Bilddatei selber vorgenommen, sondern nur als zusätzliche Information im Dokument gespeichert. Somit kommt es zu Bildbereichen, die nicht im angezeigten oder gedruckten Dokument, wohl aber im Container sichtbar sind. Hiervon sind beide Dokumentenformate betroffen.

In MS Office verhindert man dies durch einen Aufruf der Kompressionsfunktion für Grafiken. Dieses Verhalten deckt sich mit dem vorheriger Versionen von Office [21] und es dürfte dem Nutzer daher vielleicht als Vorgehen zum Entfernen ungewünschter Informationen bekannt sein (vgl. Abb. 3). Irreführend könnte dagegen die neuere Funktionalität sein, ein Dokument als „endgültig“ zu markieren, diese Dokumente enthalten allerdings immer noch die kompletten Rohdaten eingebetteter Objekte.

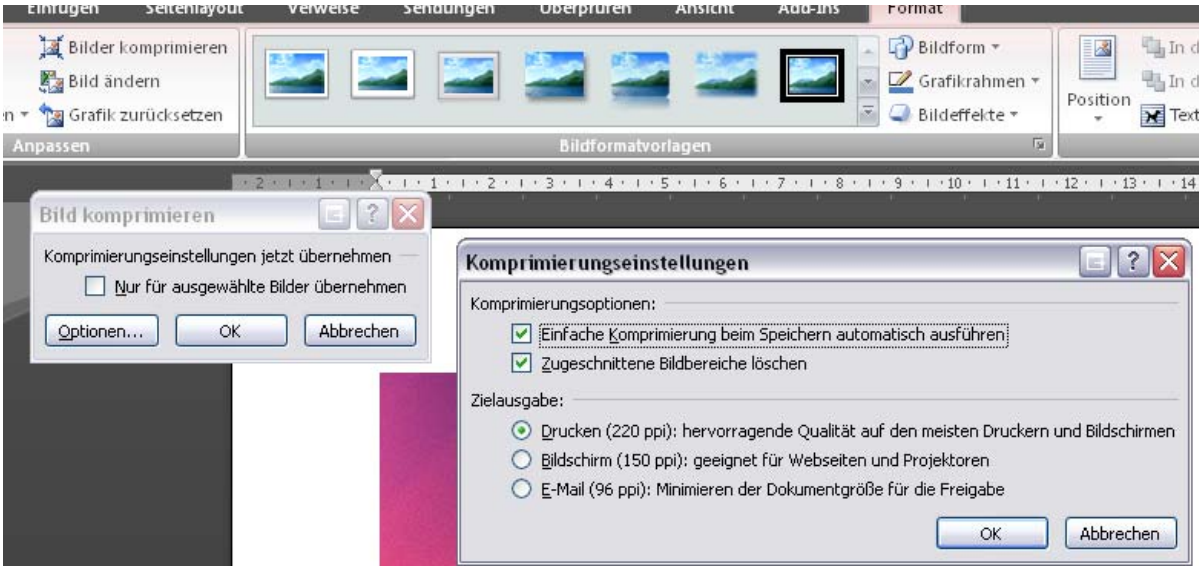

Abbildung 3: Word - Nur Kompression entfernt die beschnittenen Bereiche von Bildern auch aus dem Container

In OpenOffice ist eine solche Funktion nicht vorhanden, man sollte die Bilder daher vor dem Einbetten mit einer externen Anwendung entsprechend beschneiden. Es gibt jedoch von SUN eine Erweiterung für Präsentationen [33], welche zur Platzoptimierung genau dies tut. Eine solche Extension wäre auch für die OpenOffice-Textverarbeitung möglich und sinnvoll.

Bilder sind hier nur als Beispiel genannt. Es gilt generell für alle eingebetteten Objekte, deren Daten nur zum Teil in der Darstellung sichtbar werden. Beide Formate erlauben

es Daten auch in XML-Objekten im Container zu speichern und diese dann in Tabellen oder Dokumenten zu referenzieren. Dies ist für dynamisch generierte Dokumente sehr praktisch [15]. Generell sollten eingebettete Datenobjekte nur solche Daten enthalten, die auch für den Empfänger bestimmt sind, damit keinerlei vertrauliche Informationen weitergegeben werden (vgl. [17, Folie 67]).

2.2 Mangelnder Integritätsschutz trotz digitaler Signatur

Digitale Signaturen werden eingesetzt, um die Integrität des Dokuments zu bestätigen und den Autor zu authentifizieren. Beide Dokumentenformate, bzw. die entsprechenden Office-Anwendungen, bieten die Möglichkeit, Dokumente digital zu signieren. Hier haben wir besonders schwerwiegende Defizite gefunden. Generell setzten beide Formate auf den Standard für digitale Signaturen von XML-Dokumenten [6]. Es gibt einige Kritik an der generellen Benutzbarkeit von Digitalen Signaturen im XML-Umfeld, insbesondere die Möglichkeiten einer falschen Implementation oder fehlerhafter Transformationen werden erwähnt [9] [11]. Die hier vorgestellten Probleme sind auch der hohen Modularität der XML-basierten Formate zuzuschreiben, dadurch entsteht eine hohe Komplexität, die offensichtlich die untersuchten Anwendungen überfordert.

Der Aufbau bzw. die Berechnung einer digitalen Signatur in XML erfolgt, vereinfacht dargestellt, in folgenden Schritten:

1. Auswahl der zu signierenden Elemente. Dies können ganze XML-Dateien sein, aber auch einzelne Strukturen innerhalb eines Dokumentes.

2. Anwendung der Transformationsregel. Dies ist im Allgemeinen die Überführung in eine kanonische Form [1], wobei z.B. überschüssige Leerzeichen und Zeilenumbrüche entfernt werden. Es können aber auch kompliziertere Transformationen vorgeschrieben werden, diese können prinzipiell beliebige Änderungen des Dokumentes ermöglichen.

3. Hashing der so transformierten Daten.

4. Aufbau der XML-Strukturen zum Ablegen der digitalen Signatur. Hier werden nun die in Schritt 1 ausgewählten Elemente referenziert, deren Transformationsvorschrift und der berechnete Hash abgelegt. Ferner werden die für die Berechnung der Hashes und der Signatur relevanten Informationen (Zertifikate, Algorithmen etc.) hier aufgeführt.

5. Digitale Signatur für die transformierte (meist kanonische) Form dieser Struktur berechnen und anfügen.

2.2.1 Manipulation von Links und Metainformation trotz Signatur

In digital signierten DOCX-Dokumenten wird die Signatur in der Datei `/_xmlsignatures/sig1.xml` abgelegt. Folgende Dateien sind in einem normalen

DOCX-Testdokument als zu signierend aufgelistet (Reference URIs): `/_rels/.rels`, `/word/_rels/document.xml.rels`, `/word/document.xml`, `/word/fontTable.xml`, `/word/settings.xml`, `/word/styles.xml`, `/word/theme/theme1.xml`, `/word/webSettings.xml` sowie eventuell eingebundene Objekte wie z.B. Grafiken.

Diese (und nur diese Dateien) werden von der digitalen Signatur vor unauthorisierten Änderungen geschützt. Dabei wurden einige wichtige Bestandteile eines DOCX-Dokumentes übergangen: In `/docProps/core.xml` lassen sich weiterhin Änderungen an den Metainformationen (z.B. Autor) ändern vornehmen [29]. Des Weiteren lassen sich die Ziele für bereits eingebettete Hyperlinks nachträglich ändern [30]. Die über Hyperlinks verlinkten Ziele liegen bei DOCX in der Datei `/word/_rels/document.xml.rels`. Obgleich diese als Reference URIs gelistet ist und damit im Rahmen der digitalen Signatur eigentlich geschützt sein sollte, wird eine Manipulation nicht erkannt. Dies könnte an der Transformationsvorschrift liegen, die exakten Gründe herauszuarbeiten würde jedoch den Rahmen des vorliegenden Papiers sprengen.

Bei OpenOffice umfasst der Schutzbereich der Signatur auch die Metainformationen in der `meta.xml`. Die Liste der Referenzen aus der Datei `/META-INF/documentsignatures.xml` umfasst: `content.xml`, `styles.xml`, `meta.xml`, `settings.xml` sowie eventuell eingebundene Objekte wie etwa Grafiken. Dadurch lassen sich die Metainformationen des Dokumentes nicht verändern. Des Weiteren werden die Ziele für Hyperlinks bei ODT-Dokumenten direkt in `content.xml` gespeichert. Damit ist die Manipulation in OpenOffice-Dokumenten nicht möglich.

OpenOffice hat allerdings auch Schwächen, wenn es um digitale Signaturen geht. So lassen sich beispielsweise in ODT die zunächst angezeigten Informationen des zur Unterschrift benutzten Zertifikates ändern [31]. In `/META-INF/documentsignatures.xml` dupliziert OpenOffice Informationen über die Zertifizierungsstelle des zur Unterschrift benutzen Zertifikates und speichert diese in XML-Elementen in der Datei. Diese Informationen sind auch in dem X.509 kodierten Zertifikat enthalten, welches auch in `/META-INF/documentsignatures.xml` innerhalb eines XML-Elementes liegt. Die im Zertifikat enthaltenen Informationen sind Teil des Zertifikates und damit vor Manipulationen geschützt. Die duplizierten Zertifikats-Informationen sind allerdings nicht digital signiert. Aber genau diese manipulierbaren Information werden dem Nutzer von der Applikation OpenOffice Writer zunächst präsentiert, wie Abbildung 4 zeigt.

Dies erlaubt es dem Angreifer beispielsweise, den Namen der Zertifizierungsstelle frei zu verändern. Erst in tieferen GUI-Dialogen, nach zusätzlichen Mausklicks, erfährt der Nutzer die Daten des tatsächlich genutzten und zur Verifikation benutzen Zertifikates.

2.2.2 Benutzerschnittstelle für digital signierte Dokumente

Wir werden noch kurz allgemein auf die GUI eingehen. Beide Anwendungen, sowohl MS Office als auch OpenOffice, weisen den Nutzer durch ein Symbol in der Statusleiste auf die Existenz von digitalen Unterschriften hin.

Abbildung 5 zeigt hier den deutlichen Unterschied: Während OpenOffice dem Benutzer den tatsächlichen Status der enthaltenen digitalen Signatur über das Symbol und den

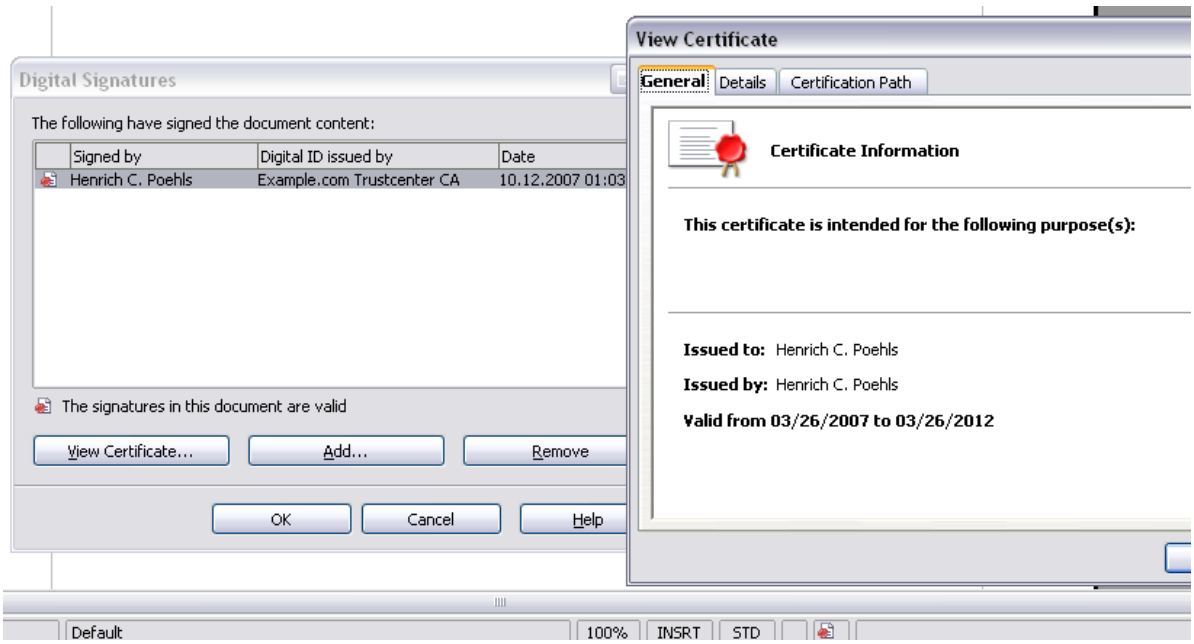

Abbildung 4: Writer - links: Zuerst angezeigte Zertifikatsinformationen (manipuliert), rechts: Erst ein Klick auf „View Certificate“ zeigt den tatsächlichen, manipulationsgeschützten Aussteller

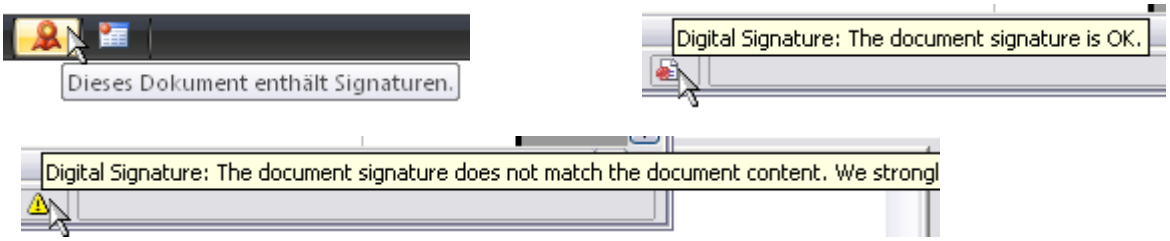

Abbildung 5: Hinweis auf digitale Signaturen in der GUI - Oben links: MS Office, oben rechts und unten: OpenOffice

Mouse-Over-Text mitteilt, hält sich MS Office zurück und weist nur darauf hin, dass das Dokument Signaturen enthält. Das hat zur Folge, dass im Falle einer ungültigen Signatur sich die Darstellung des Statusfeldes bei MS Office nicht ändert, bei OpenOffice hingegen ändern sich Symbol und Text und zeigt deutlich das die Signatur ungültig ist. Es gibt in MS Office einen zusätzlichen Warnhinweis, der im oberen Bereich beim Öffnen eines Dokumentes mit ungültiger Signatur, allerdings kam dieser Warnhinweis bei unseren Tests nicht immer. In MS Office muss der Nutzer also immer zusätzlich ein Side-Panel öffnen um den tatsächlichen Status der enthaltenen Signaturen einzusehen. Das sich das Status-Symbol nicht ändert ist kein Bug, sondern von Microsoft absichtlich so gewollt, wie uns auf Anfrage mitgeteilt wurde. Insgesamt sollte aber der Status einer Signatur für den Nutzer leicht zu erkennen sein.

2.3 Container für ungewollte Inhalte

Dokumente können als Transport-Container missbraucht werden, um Inhalte an Filtern vorbei zu schleusen. Es lassen sich generell beliebige Dateien im ZIP-Container ablegen. Um jedoch bei Microsoft Office die Validität des Dokumentes zu erhalten, muss die Datei im richtigen Ordner residieren und einem im Dokument spezifizierten Dateityp angehören [23]. Der dort spezifizierte Dateityp wird über die Endung festgelegt und zumindest bei DOCX wird auch nur diese Endung geprüft. Es ist egal, was die Datei tatsächlich enthält oder ob diese Datei innerhalb des Dokumentes referenziert wird oder nicht.

In einem OpenOffice-Dokument spielt es keine Rolle, ob eine eingefügte Datei an anderer Stelle referenziert wird. Auch der Speicherort innerhalb des Containers ist nicht wichtig. Das Dokument kann trotzdem ohne Fehler geöffnet werden. Solche Dateien werden beim Speichern des Dokuments allerdings nicht erneut in den Container geschrieben, fehlen dann also. Auch Dateien, die in der Datei `META-INF\manifest.xml` aufgeführt, aber im Dokument nicht eingebettet sind, führen nicht zu einem Fehler.

Es ist zudem relativ einfach, zusätzliche Informationen (oder auch vollständige Dateien) an den Container anzuhängen, nachdem die Datei gespeichert wurde. Dazu reichen Bordmittel des jeweiligen Betriebssystems aus. Unter Windows lautet der Befehl:

```
copy /b host_data.odf + hidden_data.txt new_document.odf
```

Und unter Linux, Unix und MacOS X:

```
cat host_data.odf hidden_data.txt >> new_document.odf
```

Das funktioniert, weil es sich bei dem Container um eine standardkonforme ZIP-Kompression handelt. Dadurch können aber Gateway Scanner, z.B. Virenscanner, auch ohne weiteres die einzelnen Dateien innerhalb des Containers untersuchen. Es ist daher wichtig, in den Gateways die neuen Dokumentenformate, falls noch nicht geschehen, zumindest als ZIP-Archive inspizieren zulassen. Dies geht nicht mehr, wenn beispielsweise Dokumente verschlüsselt bzw. mit Kennwort geschützt sind, da die Inhalte nicht mehr zugänglich sind.

2.4 Versteckte Inhalte in existierenden Dateien

Inhalte sind im Sinne dieser Untersuchungen versteckt, wenn sie mit steganographischen Methoden [16] im Dokument untergebracht wurden, so dass sie vom Betrachter nicht ohne zusätzliches Wissen und Hilfsmittel bemerkt werden können. Dies geht über das bloße Verstecken von zusätzlichen Dateien, wie oben geschildert, hinaus, weil sie direkt im Dokument verborgen werden. Da es sich bei beiden untersuchten Dokumentenformaten im Kern um XML-Dateien handelt, beschränken wir uns auf Methoden, die geeignet sind, um Informationen in XML verbergen. Dies bedeutet allerdings nicht, dass nicht auch Methoden anwendbar sind, die für unformatierten Text entwickelt wurden.

Beide Anwendungen schreiben die Dokumente bei jedem Speichern vollständig neu. Versteckte Inhalte lassen sich also einfach entfernen.

2.4.1 Benutzung von Tags zum Verbergen von Informationen

In XML [3] werden die Textdateien mit Auszeichnungen (Tags) versehen. Dabei gibt es nach der Spezifikation mehrere gültige (valide) Varianten. Dies kann ausgenutzt werden, um Informationen im Dokument zu verstecken. Inoue et al. haben bereits Möglichkeiten untersucht, um Wasserzeichen in XML zu verbergen [13]. Für Office-Dateien können zwei dieser Methoden benutzt werden: Zusätzliche Leerzeichen vor schließender Klammer in Tags und Varianten von Single-Element-Tags. Bei ersterem wird z.B. ein kein Leerzeichen benutzt (`<tag>`) um 0 und ein Leerzeichen (`<tag >`) um 1 abzubilden. Im zweiten Fall wird `<tag />` als 0 und `<tag></tag>` als 1 interpretiert. Natürlich könnte man in beiden Fällen die Werte auch umgekehrt belegen. Diese Leerzeichen werden laut OpenDocument-Spezifikation [24, Abschnitt 1.6] im Einklang mit der XML-1.0-Spezifikation [2] ignoriert, das Dokument bleibt also gültig.

Als Beispiel möchten wir eine beliebige Bitfolge in einem beliebigen Dokument übertragen. Wir wählen in diesem Beispiel eine kleine Zahl: `0101`. Dieser Wert kann nun wie oben beschrieben in einem XML-Office-Dokument verborgen werden. Abbildung 6 zeigt einen Teil einer `content.xml`-Datei (Open Office), mit bereits eingefügten Leerzeichen in den Zeilen drei und sechs[2]:

```
1  <text:h text:style-name="Heading_20_1" text:outline-level="1">
2  Lorem ipsum
3  </text:h >
4  <text:p text:style-name="Standard">
5  Lorem ipsum dolor sit amet, consetetur sadipscing elitr.
6  </text:p >
```

Abbildung 6: Auszug aus der Datei content.xml

Wie man an Abbildung 7 sieht, hat die Änderung keinen Einfluss auf die Darstellung der Seite im Programm. Das gilt auch für Ausdrucke oder exportierte Dateien. Für Microsoft Office verzichten wir auf eine Abbildung, da das Prinzip identisch ist und es hinsichtlich der Behandlung von zusätzlichen Leerzeichen im XML-Code keinen Unterschied zwischen den beiden Anwendungen gibt.

Informationen lassen sich wie hier beschrieben zum Teil auch dann verstecken, wenn das Dokument signiert wird. Wie bereits in Abschnitt 2.2 beschrieben, kann vor dem eigentlichen Signieren (und auch beim Verifizieren) eine Transformation beispielsweise eine Kanonisierung der signierten XML-Dateien durchgeführt werden. MS Office tut dies für `document.xml` nicht, aber es gibt andere XML-Dateien, welche kanonisiert werden. OpenOffice kanonisiert `content.xml`, da hierbei auch optionale Leerzeichen entfernt werden, führen solche Varianten zum gleichen Hash-Wert und die Signatur bleibt gültig.

Damit ist gezeigt, dass beliebige Bitfolgen eingefügt werden können. Allerdings ist die Anzahl der Bits beschränkt auf die Anzahl der Tags im Dokument. Wir zeigen nun, dass auch eine Bitfolge beliebiger Länge eingefügt werden kann.

[2] Es gibt in diesem Beispiel keine Single-Element-Tags.

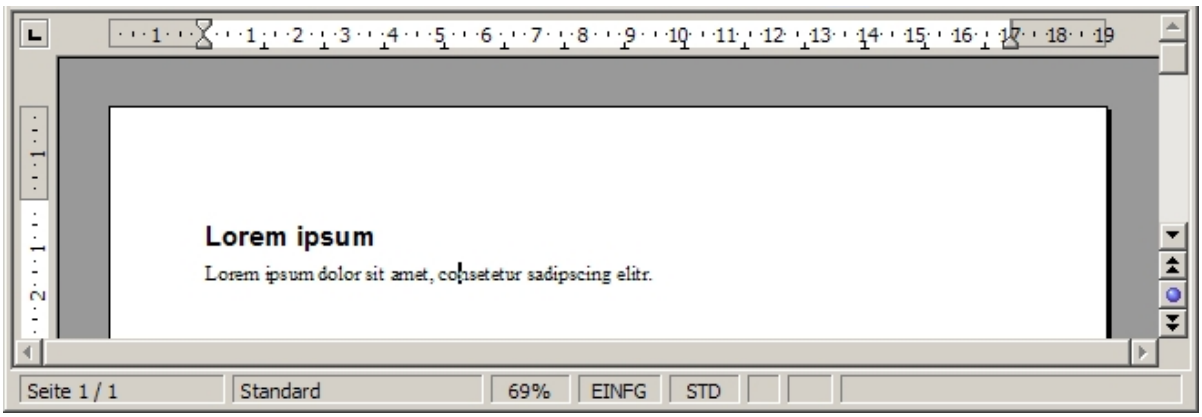

Abbildung 7: Im Dokument ist keine Änderung zu erkennen.

2.4.2 Einfügen von Informationen beliebiger Länge

In einem Dokument werden Absätze auch dann angezeigt, wenn sie leer sind, nämlich als Zeilenumbrüche. Allerdings ist nicht erkennbar, ob ein Dokument leere Formatierungstags enthält. Die Anwendung selber speichert keine redundanten Tags, aber ist bei der Anzeige tolerant gegenüber nachträglichen Änderungen. Für OpenOffice bietet es sich beispielsweise an, einen Stil wie Fettschrift zu verwenden: `<text:span><text:style-name="T1"></text:span>`. Dieser kann nun an beliebiger Stelle im Textkörper beliebig oft eingefügt werden, ohne die Darstellung zu verändern. Bei MS Office kann dasselbe mit so genannten *Runs* erreicht werden. Ein Run ist ein Textabschnitt mit identischen Eigenschaften [23]. Wird also die Schriftauszeichnung gewechselt oder beginnt ein neuer Absatz, ist dies auch immer ein anderer Run. Zusätzliche Runs, wie `<w:r></w:r>` werden im Dokument nicht sichtbar. Diese Tags können nun wie oben markiert werden. Offensichtlich wird dadurch die Dateigröße beeinflusst; allerdings ist davon auszugehen, dass die ZIP-Komprimierung des Containers sehr gute Komprimierungsergebnisse bei den immer gleichen Tags erzielt, präparierte Dokumente werden also nicht extrem viel größer.

2.5 Verschlüsselte Dokumente

Wir wollen an dieser Stelle kurz anmerken, dass es natürlich auch in beiden Anwendungen die Möglichkeit gibt Dokumente zu verschlüsseln. Es gibt hier einen entscheidenden Unterschied zwischen beiden Formaten, den wir nicht unerwähnt lassen wollen. Während bei ODT der Container und damit zum Teil auch einige der Metainformationen auch bei verschlüsselten Dokumenten zugänglich bleiben [24, Abschnitt 17.3], wird in OOXML der komplette Container verschlüsselt und ist nicht mehr als ZIP zu öffnen. Dies hat keine Auswirkungen auf die Verschlüsselung der Inhalte, diese werden bei beiden Dokumentenformaten verschlüsselt. Es zeigt aber die unterschiedliche Herangehensweise beider Formate.

3 Themenverwandte Arbeiten

Information Hiding ist ein weiter Bereich, in dem vor allem im Zusammenhang mit Urheberschutz für digitale Dokumente geforscht wird [27]. In diesem Kontext haben auch Yoshioka et al. [34] untersucht, inwieweit Informationen in ZIP-Archiven versteckt werden können. Sion hat verschiedene Arbeiten (u. a. [32]) zu Wasserzeichen von Non-Media-Content veröffentlicht.

Natürlich gibt es auch verschiedenste Betrachtungen der Dokumenten Formate, gerade seit Microsoft versucht OOXML als Standard zu etablieren, haben sich viele mit den Standards auseinander gesetzt. Unter sicherheitsrelevanten Aspekten untersucht Lagadec in [18] die Risiken, die durch das Einbetten von Objekten via Object Linking and Embedding (OLE) oder Skripten entstehen. Sein Hauptaugenmerk gilt dabei der Frage, wie man maliziöse, aktive Inhalte in Dokumente einfügen kann und ob die Schutzfunktionen der Applikationen eine Ausführung der aktiven Inhalte verhindern. Es werden hier also die Möglichkeiten für das Einbringen von Viren oder Trojanischen Pferden in DOCX und ODF näher untersucht. Lagadec führt in [17] auch aus, dass ungewollte vertrauliche Informationen über OLE-Objekte in Dokumente eingebettet werden.

Desweiteren gibt es natürlich ein großes Interesse, generelle Softwareschwachstellen, wie z.B. Buffer Overflows, in Standardapplikationen wie OpenOffice.org oder Microsoft Office zu finden, da diese weit verbreitet sind und dadurch potentiellen Angreifern eine große Angriffsfläche bieten. Da es sich um große Softwaresysteme mit hoher Komplexität handelt und wir von sicherer Softwareerstellung noch immer ein weites Stück entfernt sind, gab es solche Lücken [4] [5] in beiden Anwendungen. Es handelt sich hierbei aber um eine ganz andere Klasse von Fehlern.

4 Schlussbetrachtung

Dies ist keine vollständige Sicherheitsanalyse der XML-basierten Dokumentenformate OOXML und ODT. Dies war jedoch auch nicht das Ziel der vorliegenden Arbeit. Diese sollte vielmehr eine Auswahl kritischer Punkte aufzeigen, die beim Umgang mit Dokumenten in diesen Formaten beachtet werden sollten. Die von uns angeführten Punkte sind keine Verwundbarkeiten im klassischen Sinne, wir wollten gerade keine Buffer Overflows demonstrieren oder zeigen, wie man mit einem trojanisierten Dokument einen Rechner angreift. Dennoch offenbaren sich die erkundeten „Untiefen“ bei näherer Betrachtung allesamt als Ausprägungen bereits bekannter sicherheitsrelevanter Probleme:

- Zu hohe Komplexität: Nicht immer entspricht der Inhalt, den der Nutzer in einem Dokument zu speichern glaubt, auch dem Inhalt, der tatsächlich in der Datei gespeichert wird. Wir haben mit konkreten Beispielen gezeigt, welche zusätzlichen Informationen in den Dokumenten stecken bzw. welche man in ihnen verstecken kann.
- Transparenz: Der Nutzer muss nicht verstehen wie seine Dokumente repräsentiert werden, dies wird durch die Applikation verborgen. Dadurch ist es möglich auf

unterer Ebene Informationen einzubringen, welche nicht dargestellt werden (Covert Channels [19]).

- Falsche Anwendung von Sicherheitsfunktionen: Die unterschiedlichen „Reichweiten“ bei Verschlüsselung und digitaler Signatur zeigen, dass auch XML-basierte Standards kein Garant für die richtige Verwendung von kryptografischen Maßnahmen sind. Diese Schwächen sind zum Teil schon deutlich schwerwiegender.
- Schlechtes GUI-Design: Es wird dem Nutzer unnötig kompliziert gemacht, Entscheidungen über die Vertrauenswürdigkeit von Dokumenten zu treffen.

Wir hoffen, wir haben mit dieser Arbeit diese allgemeinen Unzulänglichkeiten am Beispiel der Dokumentenformate „erlebbar“ gemacht und ins Gedächtnis zurückgeholt (Awareness). Die neuen Dokumentenformate erlauben einen einfacheren Zugriff auf ihre Inhalte, bergen trotz ihrer Offenheit aber auch einige sicherheitsrelevante Probleme, so dass man die elektronisch ausgetauschten Dokumente wachsam betrachten sollte.

Literatur

[1] John Boyer. Canonical XML V 1.0. `www.w3.org/TR/xml-c14n`, Mar. 2001. [Online; accessed 9-december-2007].

[2] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible Markup Language (XML) 1.0 (Fourth Edition). `www.w3.org/TR/2006/REC-xml-20060816/`, Aug. 2006. [Online; accessed 9-december-2007].

[3] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, and John Cowan. Extensible Markup Language (XML) 1.1 (Second Edition). `www.w3.org/TR/2006/REC-xml11-20060816/`, Aug. 2006. [Online; accessed 9-december-2007].

[4] Common Vulnerabilities and Exposures. CVE-2007-1747. `www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1747`, May 2007. [Online; accessed 9-december-2007].

[5] Common Vulnerabilities and Exposures. CVE-2007-2834. `cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-2834`, Sep. 2007. [Online; accessed 9-december-2007].

[6] Eastlake, Reagle, and Solo. XML-signature syntax and processing. W3C recommendation. `www.w3.org/TR/xmldsig-core/`, Feb. 2002.

[7] ECMA. *Standard ECMA-376 Office Open XML File Formats*. ECMA International, Dec. 2006. [Online; accessed 11-October-2007].

[8] Erika Ehrli. Walkthrough: Word 2007 XML Format. `msdn2.microsoft.com/en-us/library/ms771890.aspx`, June 2006. [Online; accessed 9-december-2007].

[9] Johannes Ernst. So what about really simple XML Signatures? `netmesh.info/jernst/Technical/really-simple-xml-signatures.html`, Feb. 2006. [Online; accessed 9-december-2007].

[10] FFII. NO OOXML campaign website. `www.noooxml.org`, Sept. 2007.

[11] Peter Gutman. Why XML Security is Broken. `www.cs.auckland.ac.nz/~pgut001/pubs/xmlsec.txt`, Oct. 2004. [Online; accessed 9-december-2007].

[12] Dublin Core Metadata Initiative. DCMI Metadata Terms. `dublincore.org/documents/dcmi-terms`, Dec. 2006. [Online; accessed 9-december-2007].

[13] Shingo Inoue, Kyoko Makino, Ichiro Murase, Osamu Takizawa, Tsutomu Matsumoto, and Hiroshi Nakagawa. A Proposal on Information Hiding Methods using XML. In the First NLP and XML Workshop.

[14] ISO. Information technology – Open Document Format for Office Applications (OpenDocument) v1.0. `www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=43485`, Nov. 2006. ISO/IEC 26300:2006.

[15] Brian Jones. Integrating with business data: Store custom XML in the Office XML formats. `blogs.msdn.com/brian_jones/archive/2005/11/04/integrating-with-business-data-store-custom-xml\-in-the-office-xml-formats.aspx`, Nov. 2005. [Online; accessed 9-december-2007].

[16] Stefan Katzenbeisser and Fabien A. Petitcolas, editors. *Information Hiding Techniques for Steganography and Digital Watermarking*. Artech House, Inc., Norwood, MA, USA, 2000.

[17] Philippe Lagadec. OpenOffice/OpenDocument and MS OpenXML security. In *PacSec*, 2006.

[18] Philippe Lagadec. OpenDocument and Open XML security (OpenOffice.org and MS Office 2007). *Journal in Computer Virology*, pages –, 2007.

[19] Butler W. Lampson. A note on the confinement problem. In *Communications of the ACM*, volume 16, pages 613–615, New York, NY, USA, 1973. ACM Press.

[20] Microsoft. Privacy supplement for microsoft office word 2007, Oct. 2006.

[21] Microsoft. About resizing or cropping a picture. `office.microsoft.com/en-us/help/HP051951021033.aspx`, Sept. 2007. [Online; accessed 9-december-2007].

[22] Microsoft Knowledge Base. How to Minimize Metadata in Microsoft Word 2003. `support.microsoft.com/kb/825576/EN-US/`, Aug. 2006. [Online; accessed 9-december-2007].

[23] Tom Ngo. Übersicht über Office Open XML. `www.ecma-international.org/news/TC45_current_work/OpenXML%20White%20Paper_German.pdf`, Oct. 2006.

[24] OASIS. Open document format for office applications (opendocument) specification v1.1. docs.oasis-open.org/office/v1.1/OS/OpenDocument-v1.1-html/OpenDocument-v1.1.html, Feb. 2007. [Online; accessed 9-december-2007].

[25] OpenXML4J Project. Office Open XML File Format library for Java. www.openxml4j.org/, 2007. [Online; accessed 21-october-2007].

[26] Christian Persson and Peter Siering. Big brother bill. *c't*, 6:16, 1999.

[27] Fabien A. P. Petitcolas, Ross J. Anderson, and Markus G. Kuhn. Information hiding—a survey. In Robert J. Trew and James Calder, editors, *Information Hiding—A Survey*, pages 1062–1078, Jul. 1999.

[28] PKWARE. .zip file format specification. www.pkware.com/documents/casestudies/APPNOTE.TXT, Sep. 2007. [Online; accessed 9-december-2007].

[29] Henrich C. Poehls, Dong Tran, Finn Petersen, and Frederic Pscheid. MS Office 2007: Digital Signature does not protect Meta-Data. www.securityfocus.com/archive/1/484919/30/0/, Dec. 2007. [Online; accessed 9-december-2007].

[30] Henrich C. Poehls, Dong Tran, Finn Petersen, and Frederic Pscheid. MS Office 2007: Target of Hyperlinks not covered by Digital Signatures. www.securityfocus.com/archive/1/485031/30/0/, Dec. 2007. [Online; accessed 9-december-2007].

[31] Henrich C. Poehls, Dong Tran, Finn Petersen, and Frederic Pscheid. OpenOffice: Duplicated, Unprotected Certificate Information shown in Signed ODF Documents. www.securityfocus.com/archive/1/485034/30/0/, Dec. 2007. [Online; accessed 9-december-2007].

[32] Radu Sion. Digital rights protection outside multimedia. Thesis Outline, 2004.

[33] Sun Microsystems, Inc. Sun Presentation Minimizer. http://extensions.services.openoffice.org/project/PresentationMinimizer, 2007. [Online; accessed 9-december-2007].

[34] K. Yoshioka, K. Sonoda, O. Takizawa, and T. Matsumoto. Information hiding on lossless data compression. In *Intelligent Information Hiding and Multimedia Signal Processing, 2006. IIH-MSP '06. International Conference on*, pages 15–18, 2006.