RUHR-UNIVERSITÄT BOCHUM

RUB

RUHR-UNIVERSITÄT BOCHUM

Security of PDF Signatures

Karsten Meyer zu Selhausen

Master's Thesis – November 25, 2018.
Chair for Network and Data Security.

Supervisor: Prof. Dr. Jörg Schwenk
Advisor: M. Sc. Martin Grothe
Advisor: Dr.-Ing. Christian Mainka
Advisor: Dr.-Ing. Vladislav Mladenov

hgi Lehrstuhl für Netz- und Datensicherheit

Abstract

The *Portable Document Format* (PDF) is probably the most common file format for the exchange of digital documents of any type. Almost every company and governmental institution uses PDF files for communication and digital contracts due its universal compatibility. As these environments rely on the content of the documents it is necessary to secure them. The application of a digital signature is suitable for this task and allows to secure digital documents in terms of integrity, authenticity and non-repudiation. In recent years digital signatures gained the same legal status as handwritten signatures in many circumstances. There are even cases where digitally signed documents are mandatory. PDF files support the application of digital signatures natively. Due to the combination of PDF's advantages and the security of digital signatures, signed PDF files have become popular for the exchange of secure digital documents. Despite the fact that the PDF version introducing native digital signatures was published in 1999 there has been little to no research on the security of digital signatures embedded in PDF files in the past. Given the crucial environments signed PDF files are used in - including the judicial system, tax matters and all sorts of legally binding contracts - it is necessary to evaluate whether it is possible to bypass the protection of signatures in PDF files. This thesis contains the systematic and comprehensive evaluation of the security of PDF signatures. A total of 34 applications for different operating systems has been evaluated using different attacks from three novel attack classes. The evaluation results are alarming as vulnerabilities have been found in all but 4 applications. Details of the successful attacks have been given to the applications' vendors to enable them to fix the identified vulnerabilities.

Keywords: Portable Document Format, PDF, Adobe, Digital Signature, Signature Exclusion, Incremental Update Abuse, Signature Wrapping, Security Evaluation

Official Declaration

Hereby I declare, that I have not submitted this thesis in this or similar form to any other examination at the Ruhr-Universität Bochum or any other Institution of High School.

I officially ensure, that this paper has been written solely on my own. I herewith officially ensure, that I have not used any other sources but those stated by me. Any and every parts of the text which constitute quotes in original wording or in its essence have been explicitly referred by me by using official marking and proper quotation. This is also valid for used drafts, pictures and similar formats.

I also officially ensure, that the printed version as submitted by me fully confirms with my digital version. I agree that the digital version will be used to subject the paper to plagiarism examination.

Not this English translation, but only the official version in German is legally binding.

Eidesstattliche Erklärung

Ich erkläre, dass ich keine Arbeit in gleicher oder ähnlicher Fassung bereits für eine andere Prüfung an der Ruhr-Universität Bochum oder einer anderen Hochschule eingereicht habe.

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Die Stellen, die anderen Quellen dem Wortlaut oder dem Sinn nach entnommen sind, habe ich unter Angabe der Quellen kenntlich gemacht. Dies gilt sinngemäß auch für verwendete Zeichnungen, Skizzen, bildliche Darstellungen und dergleichen.

Ich versichere auch, dass die von mir eingereichte schriftliche Version mit der digitalen Version übereinstimmt. Ich erkläre mich damit einverstanden, dass die digitale Version dieser Arbeit zwecks Plagiatsprüfung verwendet wird.

Date	author

Erklärung

Ich erkläre mich damit einverstanden, dass meine Masterarbeit am Lehrstuhl NDS dauerhaft in elektronischer und gedruckter Form aufbewahrt wird und dass die Ergebnisse aus dieser Arbeit unter Einhaltung guter wissenschaftlicher Praxis in der Forschung weiter verwendet werden dürfen.

Date	author

Acknowledgments

First, I would like to thank my advisors M. Sc. Martin Grothe, Dr.-Ing. Christian Mainka, Dr.-Ing. Vladislav Mladenov and my supervisor Prof. Dr. Jörg Schwenk for supporting me so much during my master's thesis. I really appreciate the opportunity they provided me to complete my master's degree with an extremely interesting and - *prior to our efforts* - unexplored topic. I would like to thank them for providing me with insight and feedback whenever I needed it. Their enormous expertise helped me throughout the whole thesis and my results would not have been possible without their support. Additionally, I want to thank them for letting me participate in the process of creating a paper based on our results.

Furthermore, I would like to thank my superiors from Hackmanit for their understanding and flexibility allowing me to adjust my working hours so that I could complete the work for my thesis.

Despite the freedom and fun I enjoyed during my studies at the university, I am pleased that this chapter of my life is now complete. I look forward to the benefits and challenges of the next chapter when I finally enter the professional world and start a "real job" at a promising enterprise in 2019.

Contents

1 Introduction

In this chapter an introduction to the topic of this thesis is given. First, the motivation why it is important and worthwile to evaluate the security of digital signatures in PDF documents is pointed out. Afterwards, related work from earlier research and the contribution of this thesis is described. Finally, the methodology and structure of this thesis is explained.

1.1 Motivation

The *Portable Document Format* (PDF) is an universal file format initially developed by Adobe and later succeeded by the International Organization for Standardization as an ISO standard [1]. Since 1993 when its first version was introduced it has become one of the most-used file formats globally and the "de facto standard for electronic exchange of documents" of all kind [2, p. 23]. Today, PDF processing applications and libraries are available for almost any platform including desktop computer, mobile devices and servers. Due to its huge popularity and broad compatibility, PDF files are frequently used both for internal and external communication by almost every company as well as governmental institutions worldwide. The integrity, authenticity and non-repudiation of the documents' contents is a crucial factor especially in these environments. In order to ensure that documents cannot be manipulated unnoticed and that the document's author is known digital signatures can be applied. Authorities like the European Union empower digital signatures to have the same legal value as handwritten signatures.[1] Depending on the environment and specific use-case applying a signature to a digital document is even enforced by law for the document to be valid. One example is the process of registering a company in the german "Handelsregister". While the process is initially based on paper documents the notary certifying the certificate of incorporation has to send a digital copy to the responsible court to finish the process.[2] This digital copy is only accepted as a valid registration to the "Handelsregister" if it contains a valid digital signature from the notary[3] based on his "qualified certificate".[4]

[1] VERORDNUNG (EU) Nr. 910/2014 DES EUROPÄISCHEN PARLAMENTS UND DES RATES (eIDAS), Artikel 25 Rechtswirkung elektronischer Signaturen.
[2] Handelsgesetzbuch (HGB), § 12 Anmeldungen zur Eintragung und Einreichungen.
[3] Beurkundungsgesetz (BeurkG), § 39a Einfache elektronische Zeugnisse.
[4] Bundesnotarordnung (BNotO), § 33 Elektronische Signatur.

PDF supports the application of digital signatures to documents since version 1.3 published in 1999 [7, p. 13]. As mentioned earlier, PDF is widely used for documents of any kind. Therefore, using PDF for the distribution of signed digital documents is a logical choice. The European Telecommunications Standards Institute (ETSI) definded the "PDF Advanced Electronic Signature" (PAdES) [10] standard for law compliant digitally signed documents based on PDF, for example.

However, despite the great importance of PDF and in contrast to other file formats which support the digital signing of documents (e.g., XML [8]) the security of digital signatures in PDF documents has not been in the focus of research in the past. Research in the context of PDF was mostly focused on embedding different types of malicious content in PDF documents and detecting this embedded malware. There was only litte research focused on PDF signatures and no public evaluations which survey the security of PDF signatures in common PDF processing applications are available. The goal of the thesis is to fill this gap by providing a comprehensive evaluation of the security of PDF signatures.

1.2 Related Work

The research in the area of PDF was mostly focused on abusing PDF features maliciously for different attacks and detecting such embedded malware in PDF files. For example, Raynal et at. published multiple attacks including Denial-of-Service (DoS) and Server-Side-Request-Forgery (SSRF) based on PDF features in 2010 [20]. In 2012 another study abusing PDF feature for URL invocation attacks was presented by Hamon et al. [11]. Multiple tools to detect malicious content in PDF files were implemented in recent years to defend against the discovered attacks [9, 12, 14, 13, 21, 23]. Additionally to abusing PDF features, the exchange of a documents content by using a self-defined font was introduced by Markwood et al. in 2017 [15].

Research in the specific field of PDF signatures is barely available. Besides the proof-of-concept bypass for one specific signature presented by Popescu et al. in 2012 [19], there was only one attack published against the SHA-1 hash algorithm used during the signing process by Stevens et al. in 2017 [25] and a bachelor thesis written by Stefan containing a study of the signature verification process for digital signatures in PDF files in 2018 [24]. None of these publications makes use of general attack classes or contains a comprehensivce evaluation of the security of PDF signatures.

Attacks against digital signatures contained in documents based on other data structures were published in the past. For example, XML Signature Wrapping attacks were presented in 2005 by McIntosh and Austel [16]. In contrast to the wrapping attacks presented in this thesis XML Signature Wrapping makes use of object ids

instead of byte ranges to specify the signed data. Another example attack against signatures embedded in a data structure which has been applied to SAML [22] and JSON [17] is the deletion of a present signature. While the general idea is applicable to PDF as well the victim specified in the attacker model used in this thesis expects the document to contain a signature. The "Signature Exclusion" attack class extends the general idea of "deleting" a signature to make the victim's application state a valid signature is present although the verification could not be conducted successfully.

1.3 Contribution

This thesis provides the results of the first comprehensive evaluation of the security of digital signatures embedded in PDF documents. For all three attack classes presented in Chapter 3 various variants were practically evaluated and several vulnerabilities identified. During the evaluation described in Chapter 5 it was possible to execute successful attacks against 30 of 34 applications. These attacks allow an attacker to bypass the integrity and authenticity protection of PDF signatures completely and to change the displayed content of signed documents arbitrarily. A Python script which allows the automatic application of all manipulations used for the evaluation of the "Signature Exclusion" attack class to a signed PDF document was implemented. Additionally to the evaluated attack vectors, further manipulations and ideas for two attack classes were devised and are listed in Appendix A.3.1 and A.3.2 for future evaluations.

1.4 Methodology

The evaluation described in Chapter 5 was conducted as a black-box test. No source code analysis of any of the evaluated applications was part of this evaluation. Due to this the details how the applications actually process PDFs and validate present signatures is unknown. Therefore, the evaluation was based on the "trial-and-error" principle. Manipulations devised theoretically based on possible weak spots of PDF signatures were practically applied to signed PDF files. The manipulated files were opened in the different applications in the scope of this thesis. Afterwards, the results were compared to the application's behavior when the unaltered original document was opened. For the two attack classes "Incremental Update Abuse" and "Signature Wrapping" an adaptive process was used by adjusting the manipulations depending on the behavior of the applications until a successful attack was discovered or the application was considered to be not vulnerable to the attack class.

1.5 Organization of This Thesis

Chapter 2 provides the foundations needed to understand the later chapters in terms of PDF basic and digital signatures in PDF. Additionally, it introduces the attacker model used for the evaluation. In Chapter 3 the three attack classes evaluated in this thesis are described. Chapter 4 contains an overview of the attempts to develop a tool for the automatic execution of manipulations to given PDF documents. The main part of this thesis is Chapter 5 which contains the description and detailed results of the evaluation conducted during this thesis. The final Chapter 6 concludes this thesis with a summary and ideas for possible future evaluations. The Appendix consists of the original signed PDF document all manipulated documents were based on, details how to establish trust relationships between the evaluated applications and the signer's certificate, lists with ideas for future evaluations and tables with the complete results of the evaluation.

2 Foundations

In the following sections, foundations needed to understand the later chapters are introduced. First, details of the Portable Document Format including the general structure of PDF files and which parts are especially important for this thesis are described. Afterwards, it is explained how digital signatures are appended to PDF files and how signatures present in PDF files are verified by viewer applications. Note that all this information is based on the *"PDF Reference – Adobe Portable Document Format, Version 1.7"* [2] which defines PDF version 1.7. Applications processing PDF files might not implement this standard correctly and behave differently. Finally, the attacker model all attacks defined in Chapter 3 are based on is described.

2.1 The Portable Document Format

2.1.1 Overview

A digital document written in the Portable Document Format (PDF) is supposed to look identically independent from the application used to view or process it and the operation system the application runs on. Therefore, PDF files need to comply to a strict structure and contain specific information how the content of the document should exactly be presented to the user. An example file is shown in Listing 2.1. The structure of the PDF file is roughly divided into four parts: The "header" (line 1), the "body" (lines 3 - 35), the "XRef section" (lines 36 - 43) and the "trailer" (lines 44 - 52). Detailed descriptions of these parts are given in Section 2.1.2. When an application processes a PDF file it starts reading at the end of the file until it finds the trailer. The trailer's information is needed to access the XRef section which is used to access objects spread through the whole file. These objects are stored in the body of the file and contain the content of the document and the information needed to display this content correctly. Objects can be of several different types and either defined as "direct" or "indirect" (see Section 2.1.4). All objects of a document are organized in a hierarchy tree whose root is the "document catalog" (see Section 2.1.5). PDF files can be easily modified or extended by appending updates to the end of the original file. These so-called "incremental updates" allow to apply modifications to documents without altering the original file's content. This ensures signatures applied to the original document are still valid (see Section 2.1.3).

In general, PDF files are encoded in ASCII. However, they can contain plain binary data, as well. It is important for processing applications to recognize the mixture of ASCII encoded and binary data.

```
%PDF-1.7
%....
1 0 obj
<<
/Type /Catalog
/Version /1.4
/Pages 2 0 R
>>
endobj
2 0 obj
<<
/Type /Pages
/Kids [3 0 R]
/Count 1
>>
endobj
3 0 obj
<<
/Type /Page
/MediaBox [0.0 0.0 612.0 792.0]
/Parent 2 0 R
/Contents 4 0 R
/Resources 5 0 R
>>
endobj
4 0 obj
<< /Length 33 >>
stream
BT
/F1 12 Tf
(Hello World) Tj
ET
endstream
endobj
...
xref
0 8
0000000000 65535 f
0000000015 00000 n
0000000078 00000 n
0000000135 00000 n
0000000247 00000 n
...
trailer
<<
/Root 1 0 R
/ID [<0C7717C0B5D2D0C3BFE67DAFE7A3F997> <0C7717C0B5D2D0C3BFE67DAFE7A3F997>]
/Size 8
>>
startxref
491
%%EOF
```

Listing 2.1: Example PDF file (shortened).

2.1.2 PDF File Structure

Initially, every PDF file consists only of the four parts depicted in Figure 2.2:

Header The first part of a PDF file is the "header". The header consists of one single line[1]. This line is a comment and states the version of the PDF standard which the file is compliant to, e.g., `%PDF-1.7` states that the file is compliant to PDF version 1.7. Viewer applications need this information to be able to process and display the PDF file correctly [2, pp. 92-93].

Body The "body" of a PDF file is the part which contains most of the file's content. The body consists of a sequence of "objects" (see Section 2.1.4) [2, p. 93].

XRef section The "cross-reference section" *(short XRef section)* contains information needed to access "indirect objects" (see Section 2.1.4) without processing the whole file. The XRef section contains one line for each "indirect object". Files can contain multiple XRef sections which are combined by the processing application to the so-called "XRef table" [2, p. 93]. More details are provided in Section 2.1.6.

Trailer The last part of a PDF file is the "trailer". The trailer is used by viewer applications to quickly access the XRef section and special objects (e.g., the "document catalog", see Section 2.1.5). These objects are needed to start the processing or displaying of the file. Every trailer begins with the keyword `trailer` and ends with the last line of the file containing the end-of-file marker `%%EOF`. It contains a dictionary with references to the mentioned special objects. For example, the `/Root` entry references the "document catalog". The dictionary is followed by the keyword `startxref` and the byte offset of the beginning of the XRef section [2, pp. 97-98]. A complete trailer of an example file is shown in Listing 2.3.

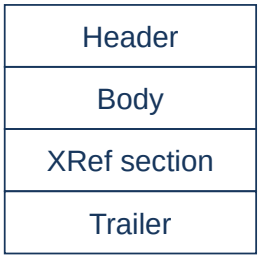

Figure 2.2: Initial structure of every PDF file.

[1] The header line is usually followed by another line containing four or more characters with codes greater than 128 to ensure that applications are able to determine whether the file contains text contents only or binary data, as well [2, pp. 92-93].

```
trailer
<<
/Root 1 0 R
/ID [<0C7717C0B5D2D0C3BFE67DAFE7A3F997> <79DE22583F7228E90C6B393465149058>]
/Size 13
/Prev 746
>>
startxref
39658
%%EOF
```

Listing 2.3: Trailer of an example PDF file.

If a PDF file is modified in the way compliant to the PDF reference further parts are appended to the initial file structure. These parts and the whole feature called "incremental update" is described in the following section.

2.1.3 Incremental Updates

Incremental updates allow applications to append new or updated information to the end of an original PDF file. This results in short saving times when small changes are applied to large PDF files. Additionally, this allows changes to PDF documents secured with a digital signature without invalidating the signature because the originally signed content is still available and not overwritten [2, pp. 98-99]. When a PDF file is extended using an incremental update the following steps are executed:

1. The new or updated objects are added to the end of the original file directly behind the old end-of-file marker.

2. The body updates are followed by a new XRef section. This new section only contains entries for objects which have been updated or added to the file or should be marked as deleted.

3. Lastly, a new trailer is appended. This trailer must contain all entries from the previous one, however they might be updated. The new trailer also references the XRef section from the previous trailer in its `/Prev` entry [2, p. 99].

This means a PDF file which has been updated n times contains n body updates, n new XRef sections and n updated trailer in addition to the original body, XRef section and trailer [2, p. 99]. The file structure of an example file appended using an incremental update is compared to its initial file structure in Figure 2.4.

Due to the fact that deleted objects or the old versions of updated objects are not removed from the file, a file which has been updated might contain several copies of one

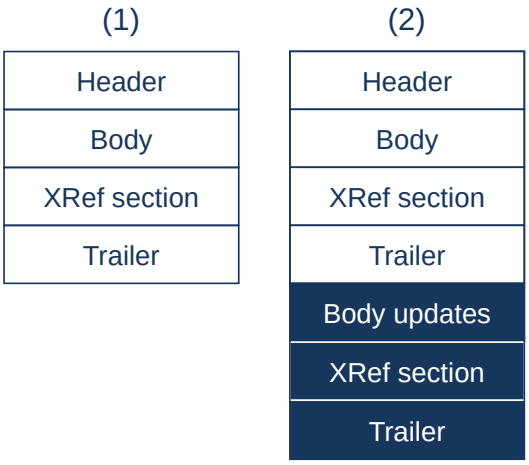

Figure 2.4: Comparison of the initial file structure of a PDF file (1) and its structure after it has been appended using an incremental update (2).

object. Applications processing a file that has been appended using incremental updates must read all XRef sections spread in the file and use them in a way that makes sure they use the latest version[2] of an object [2, p. 99].

It is also possible to update the version a PDF document is compliant to using an incremental update: The "document catalog" (see Section 2.1.5) can contain a `/Version` entry which overwrites the PDF version specified in the file's header [2, p. 99].

2.1.4 Objects

In general, almost every component (the pages, pictures and text on these pages, comments etc.) of a PDF file is an object. The only exceptions are the header, the XRef sections and their contents, and parts of the trailer. The PDF reference version 1.7 defines eight "basic types" of objects which can be used to build PDF files [2, p. 51]:

Boolean Objects of the type boolean can be either `true` or `false`. They can be stored in arrays and dictionaries or used in conditional expressions in so-called "Type 4 Functions" [2, p. 52].[3]

Strings The basic type string is either represented by characters enclosed in round brackets, for example `(Character String)`, or hexadecimal data enclosed in angle brackets, for example `<DEADBEEF>` [2, p. 53].

[2] This means the object with the unique object identifier written last to the file.

[3] Type 4 Functions allow to define and use functions inside of PDF files using a small subset of the PostScript language [2, p. 175]. These functions are out of the scope of this thesis.

Numbers PDF supports both positive and negative integers and real numbers. For real numbers there are different representations, for example `-0.1` or `-.1`. However, the representation using the exponential format (e.g., `1.8E48`) is not supported in PDF [2, p. 52].

Names Case-sensitive character units starting with a `/`, for example `/Type`, are so-called name objects. They need to be unique within a file and are used as keys or values in dictionaries usually, but can be used in arrays, as well [2, p. 58].

Arrays One-dimension collections of basic type objects can be grouped into arrays[4]. This includes arrays themselves which allows the creation of multi-dimension object collections. Arrays are enclosed in square brackets and written as a whitespace delimited sequence. The objects an array contains are not restricted to be of the same type; it is also possible to created mixed array. An example array could be `[1.8 (VfL) 48]` [2, p. 58].

Dictionaries A dictionary object is a collection of key-value-pair entries enclosed in double angle brackets. Every value can be accessed using the corresponding key. While the keys must be name objects, the values can be objects of any basic type including other dictionaries. According to Adobe dictionaries are the "main building block" of every PDF file [2, p. 59]. One shortened example dictionary is depicted in Listing 2.5. This dictionary is the "document catalog" (described in detail in Section 2.1.5) and contains three entries. For the first two entries both the keys and values are name objects, while for the third entry only the key is a name object and the value is a reference to an "indirect" object.

Streams Objects containing a sequence of bytes between the keywords `stream` and `endstream` are called streams. Additionally, the bytes can be encoded or compressed using a "filter" to decrease the file size. Every stream also includes a stream dictionary in front of the `stream` keyword which contains information about the length of the stream and - if necessary - about the filter needed to decode/decompress the stream data prior to its usage. Streams can contain different types of data like images or other objects. In contrast to strings streams can be read incrementally by processing applications and can contain an unlimited number of bytes theoretically [2, p. 60-62]. There are three special types of streams: content streams, object streams, and cross-reference streams. Content streams are used to define the appearances of pages [2, p. 151]. An object stream consists of a sequence of "indirect" objects and is used to compress a greater number of objects resulting in a decreased size of the PDF file [2, p. 100]. Cross-reference streams are needed to reference the objects stored in object streams. More information can be found in Section 2.1.6 [2, p. 106].

[4] Arrays could be used to wrap any kind of object which might be helpful for certain attacks.

Null The null object is denoted by the keyword `null` and can be used in different circumstances. For example, dictionary entries whose value is the null object or an "indirect" reference to the null object are ignored. In every document there can only be one object of the type null which needs to be referenced if it is needed in multiple places [2, p. 63].

```
<<
/Type /Catalog
/Version /1.7
/Pages 2 0 R
>>
```

Listing 2.5: Shortened "document catalog" as an example for a dictionary object.

All objects can be defined either as "direct" or "indirect" objects. While direct objects can be only used at the specific place they are defined in the document, indirect objects can be referenced from anywhere in the document using the keyword `R`. Direct objects are simply written directly at the place they are used. Indirect objects always start with a line containing an unique object identifier and the keyword `obj` and end with the keyword `endobj` independent of their object type. The unique object identifier contains two numbers: the object number and the generation number. The object number is incremented for every new object; the generation number is 0 for all objects in a file which has not been extended using incremental updates [2, pp. 63-64]. An example of an indirect string object with object number 31 and generation number 0 is shown in Listing 2.6. This object could be referenced from anywhere in the document using `31 0 R`.

```
31 0 obj
   (Character String)
endobj
```

Listing 2.6: Example definition of an indirect string object with object number 31 and generation number 0.

2.1.5 Special and Important Objects

Every PDF document contains several special objects which serve different important purposes. In the following, three of these special objects are described:

Document catalog The document catalog is a dictionary object and the root of the logical hierarchy tree all objects of a document are organized in. It is referenced by the `/Root` entry in the PDF file's trailer allowing applications to access it easily. Its entries define important information the processing application needs to display the document and references to other special objects [2, p. 137]. For example, its `/Version` entry states which PDF version the

document is compliant to and overwrites the version stated in the file's header. The `/Pages` entry references the root of the document's "page tree" ("Pages") and the `/AcroForm` entry its interactive forms dictionary ("AcroForm"). The entries `/PageMode` and `/OpenAction` can be used to specify how the document should be displayed and which actions should be executed when it is opened [2, pp. 139-141].

Pages Every page of a document is represented by a dictionary object called "page object". The pages are organized in a tree hierarchy called "page tree". The root of this tree is a dictionary object called "pages object" and the leaves are the page objects. This hierarchy enables processing applications to quickly access a certain page (for example to display it when the document is opened) without the need to process the whole document first. The pages object contains a `/Kids` entry which is an array containing either references to page objects directly or references to intermediate nodes of the tree. These intermediate nodes themselves can contain either references to other intermediate nodes or references to page objects [2, pp. 143-144].

AcroForm The "interactive forms dictionary" *("AcroForm" dictionary in the following)* is a dictionary object containing a `/Fields` entry. The value of this entry is an array with references to all interactive fields present in the document. Its other entries provide additional information regarding the interactive forms of the document. The `/SigFlags` entry specifies characteristics related to signature fields, for example [2, p. 672].

2.1.6 XRef Information

As described in Section 2.1.2 every PDF file contains an XRef section which is needed by applications to access indirect objects easily. Applications combine all XRef sections present in a file to the XRef table which contains information for all indirect objects. Initially a newly created PDF files contains only one XRef section which means this section alone is the whole XRef table. However, if a PDF file is extended or altered using an incremental update a new XRef section is added to the file. The new XRef table is the combination of all XRef sections present in the PDF file. All XRef sections have the same structure: They start with the keyword `xref` and contain one or more subsections. Every subsection contains XRef entries for a certain number of objects with continuous object numbers. An XRef subsection begins with a line containing two numbers. The first number is the object number of the first entry this subsection contains and the second number indicates how many entries this XRef subsection contains. Each XRef entry is a single line consisting of a 10 digit byte offset from the beginning of the file to the beginning of the object, a 5 digit generation number, and a keyword. The keyword at the end of the line states if the object number is "in-use" (`n`) or "free" (`f`). The object numbers marked as free are connected in a linked list whose head is the special

```
xref
0 8
0000000000 65535 f
0000000015 00000 n
0000000078 00000 n
0000000135 00000 n
0000000247 00000 n
0000000330 00000 n
0000000363 00000 n
0000000394 00000 n
```

Listing 2.7: XRef section of an example PDF file containing entries for the object numbers 0 to 8.

object number 0 which is always the first entry in the XRef table and has generation number 65535. If an indirect object is deleted its object number is marked as free, it is added to the linked list of free object numbers and the corresponding generation number is increased by one (until the maximum of 65535 is reached) to indicate the new generation number which should be used when the object number is reused [2, pp. 93-95]. Listing 2.7 shows an example XRef section containing one subsection with 8 entries starting with object number 0. All object numbers are in use except the special object number 0.

Beginning with PDF version 1.5 there is an alternative structure which can be used to store cross-reference information - cross-reference streams. These streams provide XRef information in a more compact way and are needed when objects are stored in object streams [2, p. 106].

2.2 Digital Signatures in PDF

2.2.1 Overview

Digital signatures are used to ensure the integrity of a document's contents, the authenticity of its author and the non-repudiation regarding its creation or approval. Since PDF version 1.3 it is possible to add digital signatures directly to PDF files using a special form field called "Signature Field".

Digests In the context of signatures in PDF there are two different digests used. The "ByteRange Digest" on the one hand is computed over a specific range of bytes. This byte range does not need to be one continuous range but can be divided into multiple smaller ranges with bytes excluded in between [2, p. 728]. The "Object Digest" on the other hand is computed over a subtree of the hierarchy tree the objects

are organized in. The subtree starts from the object referenced by the specific object digest [2, p. 725].

Signature fields The signature form field references the so-called "signature dictionary" which contains the signature value and other information needed, e.g., the encoding used for storing the signature value [2, p. 695]. Listing 2.8 shows the signature dictionary of an example file. While the signature value is a hexadecimal string stored in the `/Contents` entry, the used encoding is specified by the `/SubFilter` entry; version 1.7 of the PDF reference defines three standard values for this entry: `/adbe.x509.rsa_sha1`, `/adbe.pkcs7.detached` and `/adbe.pkcs7.sha1` [2, p. 727]. According to Adobe the `/adbe.pkcs7.detached` encoding is the most common one [2, p. 740]. The `/Filter` entry specifies the preferred signature handler which should be used to verify the signature. However, the PDF reference allows to use a different signature handler for verification if the substitute supports the encoding specified by the `/SubFilter` entry, as well. Depending on the used encoding the required certificate to verify the signature and certificate chain to verify this certificate are stored in the `/Cert` entry (`/SubFilter` = `/adbe.x509.rsa_sha1`) or part of the PKCS#7 data in `/Contents` (`/SubFilter` = `/adbe.pkcs7.detached` or `/adbe.pkcs7.sha1`) [2, p. 727]. For all signatures using a byte range digest the signature dictionary contains a `/ByteRange` entry specifying the byte range the digest is calculated of. This entry is an array containing integer pairs with the first integer specifying the starting byte offset and the second integer defining the length of the byte range. This format is needed to be able to include the signature dictionary in the digest calculation to secure its integrity but exclude the `/Contents` entry's value. The entries `/Name`, `/Location` and `/Reason` are character strings set by the author of the signature representing his name, his location and the reason for applying the signature. The `/M` entry gives information about the time of signing and can either be the computer time of the machine used to apply the signature or a timestamp from a secure time server. However, this value should only be used when there is no timestamp in the PKCS#7 data included [2, p.728].

```
10 0 obj
<<
/Type /Sig
/Filter /Adobe.PPKLite
/SubFilter /adbe.pkcs7.detached
/Name (Vladislav Mladenov)
/Location (Bochum)
/Reason (Security)
/M (D:20180809092110+02'00')
/Contents <308006092A864886F70D010702A0803080020101310F300D0609608648016503040201 05 ... 000>
/ByteRange [0 1633 20579 2437]
>>
endobj
```

Listing 2.8: Signature dictionary of an example PDF file (shortened).

Additional information about the type of the signature can be found in another object - the "Signature Reference Dictionary" referenced by the `/Reference` entry [2, p. 728]. This dictionary contains the transform method needed for the object digest computation or modification analysis (`/TransformMethod` entry), the digest algorithm used (`/DigestMethod` entry) and depending on the transform method either directly the value of the digest (`/DigestValue` entry) or the byte location of the digest value (`/DigestLocation` entry).

Signature types The PDF reference defines two different types of signatures: "certification" signatures and "approval" signatures. Each document can only contain one certification signature and this signature must be the first one in the document [6, p. 7]. A certification signature enables the author of a document to restrict which changes are allowed to be applied to the document after he initially created it (e.g., allowing the addition of other signatures or comments to the document) [5]. This is achieved using the "DocMDP" transform method. Approval signatures are used to approve the current contents of the document; a document can contain multiple approval signatures [6, p. 7]. Depending on the type of a signature applications display different information about the signature. A comparison of the signature panel of "Adobe Acrobat Reader DC" when a PDF file containing a certification signature and a PDF file containing an approval signatures is shown in Figure 2.9.

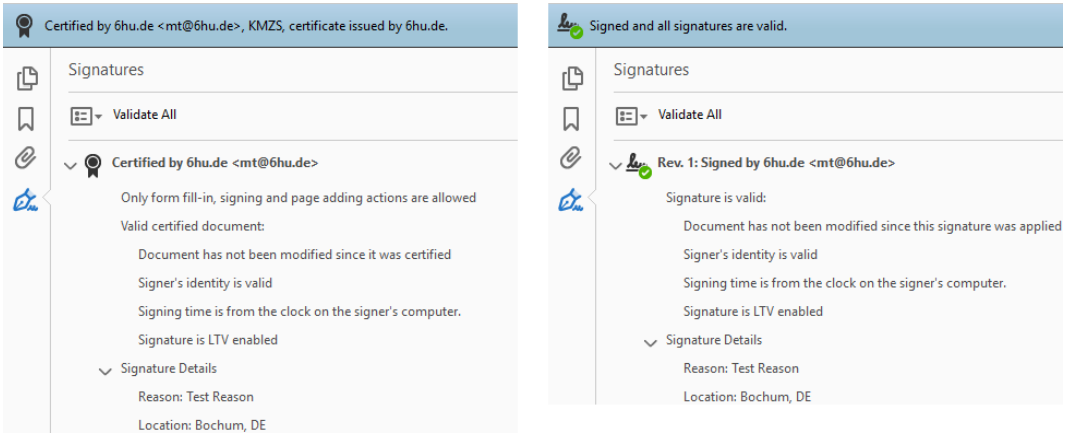

Figure 2.9: Comparison of the signature panel of "Adobe Acrobat Reader DC" when a certification signature (left) and an approval signature (right) is present in the opened document.

Transform methods Transform methods define which objects are included and which are excluded from the object digest computation needed to compare two revisions of a document with each other. There are four different transform methods specified in the PDF reference [2, p. 731]:

- DocMDP: The "DocMDP"[5] transform method is used to detect or prevent modifications of the document after it was signed by its author. It is used by the certification signature type; therefore, there must only be one signature using this transform method in a document and it must be the first one. Additional signatures must be approval signatures and use another transform method. The author of a document can use this transform method to specify which changes are allowed to be applied to the document without invalidating his signature. This is achieved by the `/P` entry in the transform parameter dictionary: If the value is set to 1 the document is intended to be final and any change invalidates the signature; the value 2 allows other users to fill in form fields and add (approval) signatures to the document; if the value is set to 3 users are additionally allowed to create, modify and delete annotations. The object digest is computed over a subset of objects including the ones whose modification is not permitted [2, pp. 731-733].
- FieldMDP: The "FieldMDP" transform method is used to detect changes to the values of a list of specified form fields. The object digest is calculated over this list of form field objects [2, pp. 736-737]. "FieldMDP" enables the author of a document to "lock" certain form fields and specify which form fields are allowed to be filled in without invalidating his signature [6, p. 9].
- UR: The "UR" ("Usage Rights") transform method is used to detect changes which would invalidate the "usage rights signature". This signature is used to enable "additional interactive features" in the document [2, p. 733].
- Identity: The "Identity" transform method is used when an object digest is computed without excluding any objects. This means the whole object hierarchy tree is walked and changes to any object invalidate the signature [2, p. 737].

Algorithms The algorithms available for the computation of the digest and the signature as well as the possible lengths of the keys used for signing depend on which encoding (resp. the value of the `/SubFilter` entry in the signature dictionary) is in use and to which PDF version the document is compliant to. In general, the SHA1 digest algorithm and RSA signing algorithm with keys with a length of 1024 bits are supported. Further algorithms, e.g., digest algorithms from the SHA2 family, RIPEMD160 or DSA signatures, and longer keys are supported by some encoding and some PDF versions higher than 1.3. RSA keys with a length of 2048 or 4096 bits

[5] "MDP" is short for "Modification Detection and Prevention" [2, p. 731].

are supported since PDF version 1.5, for example. The detailed listing of supported algorithms and key lengths is depicted in Figure 2.10.

	SubFilter value		
	adbe.pkcs7.detached	**adbe.pkcs7.sha1**	**adbe.x509.rsa.sha1**[a]
Message Digest	SHA1 (PDF 1.3) SHA256 (PDF 1.6) SHA384 (PDF 1.7) SHA512 (PDF 1.7) RIPEMD160 (PDF 1.7)	SHA1 (PDF 1.3)[b]	SHA1 (PDF 1.3) SHA256 (PDF 1.6) SHA384 (PDF 1.7) SHA512 (PDF 1.7) RIPEMD160 (PDF 1.7)
RSA Algorithm Support	Up to 1024-bit (PDF 1.3) Up to 2048-bit (PDF 1.5) Up to 4096-bit (PDF 1.5)	See **adbe.pkcs7.detached**	See **adbe.pkcs7.detached**
DSA Algorithm Support	Up to 4096-bits (PDF 1.6)	See **adbe.pkcs7.detached**	No

a. Despite the appearance of **sha1** in the name of this **SubFilter** value, supported encodings are not limited to the SHA1 algorithm. The PKCS#1 object contains an identifier that indicates which algorithm is used.

b. Other digest algorithms may be used to digest the signed-data field; however, SHA1 is always used to digest the data that is being signed.

Figure 2.10: Digest and signature algorithms and the maximum key lengths supported by different encodings and PDF versions.

Visible signatures Signatures in PDF files can be either "visible" or "invisible". Generally, every signature is associated with an "annotation". While the annotation has size 0 and the details (e.g., result of the verification and author) of invisible signatures can only be accessed using user interface (UI) elements of the processing application, visible signatures include a visible annotation on a page of the document. This annotation usually shows the result of the signature's verification and additional information, for example the author of the signature. The appearance of visible signatures contains multiple layers, each represented by an "XObject", and is customizable by its author [3, pp. 8-9].

2.2.2 Signing Process

When a user wants to add a digital signature to an existing PDF file he makes use of a PDF processing application and follows the instructions in the application's UI. He selects a present public-/private key pair or generates a new one to sign the PDF file and finally the signature is created and appended to the file using an incremental update. However, on the technical side there are multiple steps in the signing process. In the following, the steps of adding a visible approval signature which uses a byte range digest to a previously unsigned example document are given. Prior to the signing process the example file contains only 7 objects representing a document with one page displaying the string "Hello World!". The signed version of this document contains 14 objects and is later used as a basis for the evaluation described in Chapter 5. The signed document is depicted in Appendix A.1.

Singing process:

1. The document is read as a byte stream to perform the following actions.
2. New and updated objects are appended to the original document: First, an updated document catalog (object `1 0`) is added to the document. It includes a new AcroForm dictionary which references the new signature field (object `8 0`) and an object needed for its appearance (object `9 0`). Afterwards these two objects are added to the document. The signature field's value (`/V` entry) is a reference to the signature dictionary (object `10 0`) which is added to the document, as well. It contains an `/Contents` entry with a placeholder long enough for the signature value and a `/ByteRange` entry with worst-case placeholder values excluding only the value of the `/Contents` entry. Now the one page the document contains is updated (object `3 0`) to add a reference to the signature field. This is necessary because the visible signature should be displayed on this page. Afterwards the objects `11 0`, `12 0`, `13 0` and `14 0` are added to the document. These objects define the details of the visible appearance of the signature.[6] Finally, a new XRef section (containing entries for all new and updated objects) is calculated and appended to the file together with an updated trailer and the final `%%EOF` marker.
3. After all required objects are added, the actual byte offset of the `/Contents` entry can be calculated. In order to exclude its value from the signature verification the `/ByteRange` entry is updated correspondingly.
4. The hash over the bytes specified by the `/ByteRange` entry is calculated using one of the available algorithms (see Figure 2.10). The choice which specific algorithm is used depends on the used application and its configuration.

[6] These objects are not described in detail because the visible appearance is not in the scope of this thesis.

5. The calculated hash value is signed using the user's private key and encoded as specified by the value of the `/SubFilter` entry. Depending on the used encoding additional data (e.g., the signer's certificate, the certificate chain, revocation information or timestamps) is encoded along with the signature value.

6. Finally, the encoded data is placed in the value of the `/Contents` entry (as a hexadecimal string [2, p. 727]) overwriting the placeholder. If the placeholder was longer than the new encoded data the rest of the placeholder is overwritten with zeros.

[6, p. 5]

Note: A specific signing process might include additional steps depending on the application used to sign the document, its configuration and, the type of signature applied.

2.2.3 Verification Process

The verification of a digital signature in a PDF file is executed in the background when a user opens a signed PDF file with a processing application which supports digital signatures. While the details of the verification process depend on the specific processing application the general structure is similar for all processing applications. In the following, the steps of the verification process when using "Adobe Acrobat" are described as an example [4, p. 2]:

1. First the value of the `/ByteRange` entry is used to calculate the hash value over the document excluding the value of the `/Contents` entry. The calculated hash value is compared to the value stored in the signature. If the values differ the verification process is continued but the user is notified that the document was altered.

2. The certificate chain is built to ensure that the signer's certificate is trusted. There must be at least one path from the certificate to a trusted anchor.

3. The revocation information for the signer's certificate and the certificates in the certificate chain is checked to ensure that the certificates were valid at the time the document was signed. If both information to use the Online Certificate Status Protocol[7] (OCSP) and Certificate Revocation Lists[8] (CRL) is available, Acrobat tries to use OCSP first.

4. If the signature contains a timestamps from a trusted server the certificate for the timestamp is verified in the same way as the certificates described above including checking its revocation information.

[7] `https://tools.ietf.org/html/rfc6960`
[8] `https://tools.ietf.org/html/rfc5280`

2.3 Attacker Model

For the practical evaluation of the attack classes presented in Chapter 3 it is important to specify an attacker model. This allows to determine whether the result of an executed manipulation is considered to be a successful attack or not. The attacker model presented in the following defines two different UI-layers presenting information about the signature and its validity to the user: The first UI-layer is the information present when a signed document is opened using one of the viewer applications without any further actions executed by the user. The second UI-layer is the information accessible through different UI options available in the application. This includes both clicking on visible signature appearances and opening signature panels or executing certain program functionalities like "validating all signatures". All attack classes defined and evaluated in this thesis are based on the following attacker model:

The attacker is in possession of a valid and standard-compliant PDF document signed by a third-party using its private key. The document has not been altered after the application of the digital signature and the signature is valid from a cryptographic point-of-view. All viewer applications against which the manipulations are executed later trust the signer's certificate. When the document is opened they display it without error messages and successfully verify the validity of the signature. Depending on their normal behavior they display a message to the user stating that the document contains a valid digital signature and has not been modified since this signature was applied on the first, the second or on both UI-layers.

The attacker manipulates the document in different manners depending on the specific attack class he utilizes and exchanges the content displayed when the document is opened. He has no access to the third-party's private key. Therefore, he is not able to forge a cryptographically valid signature for the manipulated document. After he applied the manipulation, he makes the manipulated document available to the victim. The victim opens the manipulated document using one of the viewer applications. It does not know that the document was manipulated but expects the document to contain a digital signature. It further expects the viewer application to display any indication that a signature is present on any UI-layer and not display any warning or error messages regarding the verification or validity of signatures or certificates used in the context of the signatures on the first UI-layer. If the viewer application's behavior matches the victim's expectations the victim does not suspect that the document might be manipulated and uses it as it would use the original document.

An attack is considered successful if the manipulated content is displayed by the viewer application and no warnings or errors regarding a detected modification of the document after the signature was applied is displayed. The success of an attack can be classified depending on the two UI-layers: If the information presented on

the second UI-layer states that the signature is invalid or the document has been modified after the application of the signature the attack can still be classified as successful for the first UI-layer. On the other hand an attack resulting in information displayed which states that the signature is invalid or the document has been modified after the application of the signature on the first UI-layer always results in an unsuccessful attack. This classification is independent of the information presented on the second UI-layer because the victim expects no notice that the signature is invalid or the document has been altered on the first UI-layer. If the application does not show information regarding the signature on the first UI-layer in general and states that the signature is valid and the document has not been altered on the second UI-layer the attack is considered to be successful for the second UI-layer.

It is important to note that the victim does not perform any type of forensic analysis of the manipulated document. If a forensic analysis is performed all manipulations described in this thesis can be detected.

3 Attack Classes

This chapter introduces the three attack classes which have been evaluated in the terms of this thesis. First, the general idea each attack class is based on is presented. Afterwards, the details including different strategies or variants of each attack class are explained.

3.1 Signature Exclusion

3.1.1 Attack Idea

The basic idea of the attacks contained in the attack class "Signature Exclusion" is to *confuse* the signature verification logics and the information displaying logics of PDF processing applications. The goal of these attacks involves two aspects: The first one is to prevent the verification logic from being able to verify the digital signature present in a PDF document. Additionally, it is necessary to make the application display information stating that the present signature is valid.

3.1.2 Attack Details

Different objects of a signed document can be manipulated to achieve the goal defined above. These manipulations can be divided into two different strategies: The first strategy is to remove parts of the signature dictionary which are essential for the verification of the signature. Examples are the `/ByteRange` entry specifying which part of the document is signed, the `/Contents` entry whose value contains the signature value itself, or the `/SubFilter` entry stating which encoding was used to store the signature value. The second strategy is based on a different approach and summarizes the deletion of references to the signature. The `/V` entry which references the signature dictionary and `/P` entry which references the page the visible signature is displayed on from the form field dictionary could be altered or removed, for example. These two strategies result in a long list of possible manipulations. Due to the limited time of this thesis it was not possible to evaluate all manipulations which have been devised. Therefore, the evaluation described in Chapter 5 is based on the 24 manipulations depicted in Table 3.1. These 24 manipulations were choosen to cover both strategies with a systematic approach. For all entries the choosen manipulations include replacing its value with `null`, removing its value

and removing the entry itself, for example. Additionally, the choosen manipulations are the ones most likely to be successful according to the author's judgement. A complete list of all devised manipulations and ideas for further manipulations can be found in Appendix A.3.1 and could be used for a more comprehensive evaluation of the Signature Exclusion attack class in the future.

No.	Entry	Description	Result
1	`/Contents`	Replacing the entry's value with an empty hexadecimal string.	`/Contents <>`
2	`/Contents`	Replacing the entry's value with a hexadecimal string containing a null byte.	`/Contents <00>`
3	`/Contents`	Replacing the entry's value with the null object.	`/Contents null`
4	`/Contents`	Removing the entry's value.	`/Contents`
5	`/Contents`	Removing the entry.	-
6	`/ByteRange`	Replacing the entry's value with an empty array.	`/ByteRange []`
7	`/ByteRange`	Replacing the entry's value with an array containing zeros.	`/ByteRange [0 0 0 0]`
8	`/ByteRange`	Replacing the entry's value with an array containing a negative value.	`/ByteRange [1 -1500 2000 100]`
9	`/ByteRange`	Replacing the entry's value with an array containing overlapping byte ranges.	`/ByteRange [0 1500 1000 100]`
10	`/ByteRange`	Replacing the entry's value with an array containing a byte range outside of the document.	`/ByteRange [0 1500 2000000 100]`
11	`/ByteRange`	Replacing the entry's value with an array with a wrong size.	`/ByteRange [0]`
12	`/ByteRange`	Replacing the entry's value with the null object.	`/ByteRange null`
13	`/ByteRange`	Removing the entry's value.	`/ByteRange`
14	`/ByteRange`	Removing the entry.	-
15	`/SubFilter`	Replacing the entry's value with a non-existing name object.	`/SubFilter /None`
16	`/SubFilter`	Replacing the entry's value with the null object.	`/SubFilter null`
17	`/SubFilter`	Removing the entry's value.	`/SubFilter`
18	`/SubFilter`	Removing the entry.	-
19	`/V`	Replacing the entry's value with the null object.	`/V null`
20	`/V`	Removing the entry's value.	`/V`
21	`/V`	Removing the entry.	-
22	`/P`	Replacing the entry's value with the null object.	`/P null`
23	`/P`	Removing the entry's value.	`/P`
24	`/P`	Removing the entry.	-

Table 3.1: 24 different manipulations evaluated as part of the Signature Exclusion attack class.

3.2 Incremental Update Abuse

3.2.1 Attack Idea

"Incremental Update Abuse" attacks are based on the idea of applying malicious changes to a signed document using the incremental update feature specified in the PDF standard. As described in Section 2.1.3 the contents of the original file are kept unchanged and the signature is still valid after applying incremental updates to PDF files. Usually PDF processing applications display a note informing the user that the document has been modified after the signature was added when an incremental update was applied. However, instead of applying standard-compliant incremental updates the attacks use manipulated variants excluding certain parts of or adding additional parts to regular incremental updates. This means while updates to the original file's body are appended to the original file, the incremental update might not include a new XRef section or a new trailer (depending on the specific attack). The updates to the body consist of manipulated objects changing the visible content of the document. The goal of these manipulated incremental updates is to make the applications display the exchanged content of the document while still informing the user that the signature is valid and the document has not been modified after the signature was applied.

3.2.2 Attack Details

During this thesis differently structured incremental update variants were abused to manipulate the original document. The evaluation described in Chapter 5 is based on the four variants depicted in Figure 3.2 and described in the following:

Variant 1: without XRef section and trailer The first variant of manipulated incremental updates simply adds the manipulated objects which change the visible content of the document to the end of the file. Depending on the specific attack there might be additional content added to the file, as well. However, there is neither a new XRef section referencing the manipulated objects, nor a new trailer, appended to the file.

Variant 2: without XRef section, but with trailer Additionally to the manipulated objects a new trailer is appended to the file when applying an incremental update of this variant to a document. This addition is reasoned by the assumption that certain applications ignore content placed behind the last trailer of a file.

Variant 3: with new XRef section and trailer Similar to regular incremental updates the manipulated objects, a new XRef section and a new trailer are appended to the file. However, the new XRef section differs from regular

incremental updates in certain ways depending on the specific attack. In this thesis two different XRef sections were used: The first one is an empty XRef section. This means it does not contain a subsection or any entries but only consists of the keyword `xref`. The second one contains entries for all manipulated objects and all objects which were part of the incremental update added to the file when the signature was applied.

Variant 4: without XRef section and trailer, but with copy of signature This variant is based on variant 1 and does not include adding a new XRef section or trailer to the file. However, the manipulated objects appended to the end of the file include a copy of some objects related to the signature from the original file. These objects are the field dictionary of the signature form field and the signature dictionary.

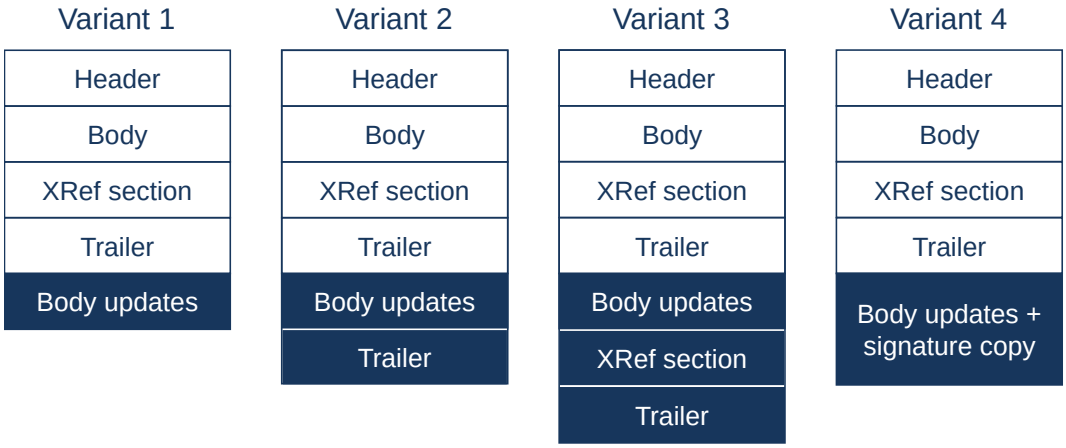

Figure 3.2: Different file structures after the four manipulated incremental update variants were applied to the original document.

3.3 Signature Wrapping

3.3.1 Attack Idea

The attack class called “Signature Wrapping” summarizes attacks which share the same basic idea: The signed data of the original document is (partly or completely) wrapped in another data structure (e.g. a stream) and placed at the end of the manipulated file. Manipulated objects which change the displayed content of the document and a new XRef section referencing these objects are added to the document. The resulting file structures allow an attacker to trick the applications into displaying manipulated content without the need of applying an incremental update. As the signed data remains unaltered the applications are able to successfully validate

the present signature. The absence of an incremental update appended to the end of the file could help to prevent applications from displaying notices that the document has been altered after the signature was applied.

3.3.2 Attack Details

The Signature Wrapping attacks can be divided into two different approaches: The first approach is based on placing the manipulated objects and XRef section in between the two signed byte ranges of the original document. This means the second signed byte range which starts with the `/ByteRange` entry of the signature dictionary is moved to the end of the file. A new `/ByteRange` entry is added to the signature dictionary referencing the new location of the moved second signed byte range and the signature dictionary object is closed. Afterwards, the manipulated objects and the new XRef section referencing these objects is added behind the signature dictionary. Some applications also require the addition of a trailer behind the new XRef section; a copy of the original file's last trailer can be used for this purpose. It is important for the attack to work that the `startxref` value of the file's last trailer is equal to the byte offset of the newly added XRef section. As the file's last trailer is part of the signed data and therefore not adjustable without invalidating the signature, it might be necessary to add padding in front of the newly added XRef section. The padding can be a fitting number of whitespaces. If the byte offset of the new XRef section is already too high it is possible to decrease it by deleting the zeros at the end of the hexadecimal value of the `/Contents` entry. These zeros are used as padding because the actual size of the signature value is unknown prior to its calculation. They are not part of the actual signature value (see Section 2.2.2 for details).

The second approach is to connect the two signed byte ranges to one single byte range and place it at the end of the file. In order to do this a copy of the first signed byte range needs to be placed between it and the second signed byte range. This copy needs to be modified by completely removing the signature value (the hexadecimal value of the `/Contents` entry) because it is dividing the two signed byte ranges in the original document. Afterwards, the `/ByteRange` entry of the original signature dictionary has to be modified to reference the new single signed byte range at the end of the file. The new value of the `/ByteRange` entry can either be an array containing one byte range with the size 0 and the single signed byte range at the end of the file or an array containing a short byte range at the beginning of the file[1] and a part of the signed byte range at the end of the file. This might lead to successful attacks for applications which do not accept a byte range with size 0. After the `/ByteRange` entry is adjusted to reference the signed data at the end of

[1] The beginning of every PDF file starts with `%PDF-` independently of the version the file is compliant to. Therefore, a byte range from bytes 0 to 5 can always be used as the first byte range referenced by the `/ByteRange` entry.

the file, manipulated objects can be added prior to the signed data. The advantage of this second approach is that it is also possible to directly manipulate the objects from the first original signed byte range instead of adding new manipulated objects to the file. Finally, an XRef section and a trailer have to be added in front of the signed data at the end of the file. While the trailer can be a copy of the original file's last trailer the XRef section must be a new one. It must contain XRef entries for all manipulated objects added to the file. As the signed data contains valid PDF objects processing applications might recognize these objects and detect the manipulation. In order to "hide" the signed data from the processing application it can be wrapped in different data structures the PDF standard defines. It can be placed inside of a dictionary or a stream, for example. In both cases the wrapping object can be either a direct or an indirect one. If a stream is used as the wrapping object it is possible to state that the content of the stream is deflated or a XML-based metadata stream, additionally. If a metadata stream is used the signed data can be placed inside of a "CDATA" section because it is not valid XML data.[2] Using these different wrappings the 41 manipulations listed in Table 3.4 were devised and evaluated in Chapter 5. Detailed descriptions for all manipulations can be found in Appendix A.3.2. All manipulations include the replacement of the displayed content of the original file.

For both approaches it is necessary to add XRef entries for all objects referenced in the XRef sections in the signed byte range moved to the end of the file to the newly added XRef section. Otherwise applications which rely on the XRef information are not able to locate the objects referenced in these XRef sections and are not able to process the PDF file correctly. These XRef entries must contain the correct byte offsets to the corresponding objects. It might be necessary to place a copy of the objects contained in the signed byte range to the manipulated objects and reference these copies instead of the original objects because some applications might ignore objects placed behind the XRef section.

The resulting file structures when one of the approaches is applied to a signed PDF document are compared to the original file structure in Figure 3.3. The yellow part is the first byte range specified by the `/ByteRange` entry of the signature dictionary, the green part the second byte range. The blue parts are the content added to the file to change the visible apprearance of the document.

[2] CDATA sections allow to include arbitrary strings in XML documents. The contents of CDATA sections are not interpreted by the XML interpreter which prevents errors with strings containing anything but valid XML data (see `https://www.w3.org/TR/xml/#sec-cdata-sect`).

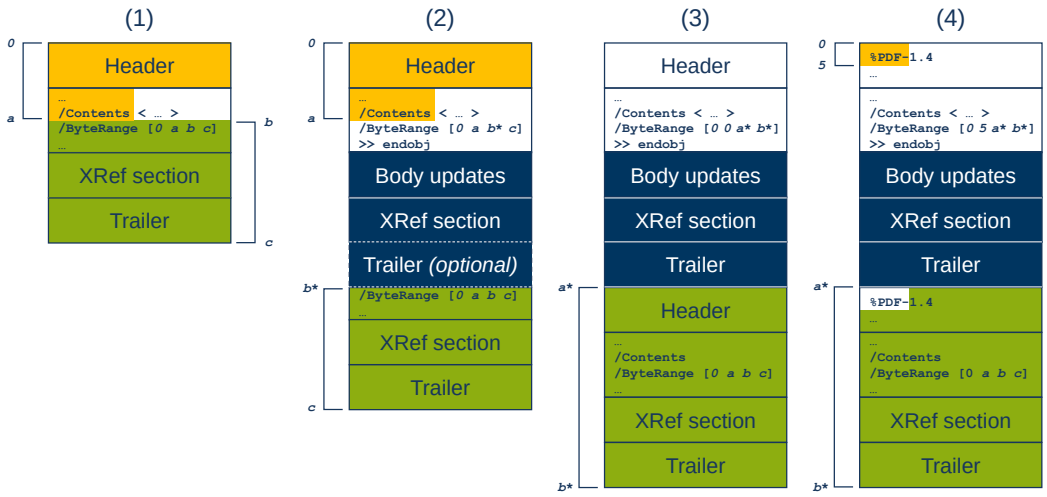

Figure 3.3: File structures of a signed PDF file before and after different Signature Wrapping attacks were applied: (1) Original file structure, (2) first Signature Wrapping approach, (3) second Signature Wrapping approach, (4) variant of the second Signature Wrapping approach. Yellow = first signed byte range, green = second signed byte range, blue = newly added content.

No.	Wrapping	First Byte Range
1	None	0 0
2 3	Dictionary	
4 5 6 7 8 9	Indirect dictionary object	
10 11	Stream	
12 13 14 15 16 17	Indirect stream object	
18 19 20 21 22 23	Deflated indirect stream object	
24 25 26 27 28 29	XML (indirect stream object)	
30 31 32 33 34 35	CDATA (indirect stream object)	
36	None	0 5
37 38 39 40	Indirect stream object	
41	None	First half of signed data

Table 3.4: 41 different manipulations evaluated as part of the second approach of the Signature Wrapping attack class.

4 Tool Development

The initial planning of this thesis contained the development of a tool which should allow to execute the specific attacks devised during this thesis automatically. The process of developing, the problems occured during this process and the final result are described in the following chapter. First, a short overview is given. Afterwards, the two approaches which have been pursued are explained in detail.

4.1 Overview

Initially, it was planed to develop a tool which allows the user to automatically apply the manipulations and attacks described in Chapter 3 to a given PDF file. This idea was based on the benefits a tool would be for both the evaluation described in Chapter 5 and future evaluations. During the first phase of this thesis information regarding both how PDF files are structured and which tools could be used to view, create or manipulate PDF files was collected. The first tool found which seemed to be able to fulfill all these tasks was the Java-based framework "Apache PDFBox"[1]. Therefore, the first approach for developing a tool to execute all the manipulations in the scope of this thesis was to use PDFBox as the basis. However, during the development based on PDFBox several problems occured. As these problems were critical to the functionality of the tool and could not be fixed easily it was concluded that PDFBox is not suitable for the tool development. Therefore, a second approach based on Python was started. In contrast to the first approach the second one does not make use of any framework or library to parse the PDF files but is completely based on modifying the PDF files like regular text files. This approach could be used to successfully create scripts for the first attack class - the Signature Exclusion attacks. During the evaluation of the manipulations contained in the attack classes Incremental Update Abuse and Signature Wrapping the conclusion was drawn that it is not meaningful to automate the manipulations. This is reasoned by the fact that the behavior of the applications differs strongly. Thus, it is necessary to conduct the evaluation of these two attack classes in an adaptive manner for every application independently and adjust the manipulated files accordingly to previous observations manually during the evaluation. Based on this conclusion the decision was made that no tool will be developed for the Incremental Update Abuse and Signature Wrapping attack classes.

[1] `https://pdfbox.apache.org/`

4.2 First Approach: Java Tool Based on PDFBox

As mentioned before, the first approach was based on the Java framework PDFBox. This framework allows to import existing PDF documents and parse their contents into a logical structure of Java objects. These Java objects can be modified in different ways depending on the API provided by the framework. While the capabilities of the framework seemed to be sufficient for the development of the planed tool at first, it turned out that this was not the case: First the problem arose that the PDFBox API makes it impossible to execute several kinds of manipulations described in Section 3.1. Some of these manipulations violate the PDF standard (e.g., deleting the value of a key in a dictionary) and are therefore not executable with PDFBox. Other manipulations (e.g., replacing the value of the `/V` entry of the form field dictionary with `null`) are compliant to the standard but still not executable with PDFBox because its API does not allow the access to all objects. For example, it was not accomplished to access the form field dictionary of the signature. The API does not provide an option to access objects which are referenced by other objects - in this example a form field dictionary referenced in the `/AcroForm` entry's value of the document catalog.

However, even for the manipulations which could be executed using PDFBox's API it was not possible to create working test files. This is reasoned by the fact that PDFBox does only allow to save the modified document in two ways: Either PDFBox performs a standard-compliant regular incremental update and appends the modified objects to the end of the original file or creates a completely new file. Both options result in PDF files which cannot be used for the evaluation of the devised attack classes. They either invalidate the signature present in the original document or result in a note that the document has been modified after the signature was applied.

After these problems occured the PDFBox documentation and source code was studied to find a solution. As there was no way found to solve or work around these problems in a reasonable amount of time the conclusion was made that PDFBox is not suitable for the development of the planed tool.

4.3 Second Approach: Tool Based on Python Scripts

The second approach is based on a different idea: Instead of parsing the PDF file into a logical structure and modifying the PDF objects as data structures, the PDF file is interpreted as a regular ASCII-encoded text file and modified by replacing existing text with manipulated one. This allows to execute all manipulations described in Section 3.1 *(and more)* regardless of whether they are compliant to the PDF standard or not. In order to apply one of the manipulations to an existing file regular expressions are used to locate the original version of the object which should

be manipulated. Afterwards, the original object is replaced with the manipulated one and the modified file is saved. For this approach multiple scripts based on Python were developed. Python was choosen because of its general simplicity and easy-to-use regular expressions.

The developed tool consists of the following seven Python scripts:

- executeManipulations.py
- manipulateByteRange.py
- manipulateContents.py
- manipulateDocumentCatalog.py
- manipulateFilterAndSubFilter.py
- manipulateSignatureReferenceDictionary.py
- pdfmanipulation.py

The script *executeManipulations.py* is used to start the tool and executes all "enabled" manipulations to an original PDF file given as a console parameter. It executes the manipulations one after the other and creates a new manipulated PDF file for each manipulation. The manipulations are divided into five collections depending on the object which is manipulated. The five collections are implemented as the scripts: *manipulateByteRange.py*, *manipulateContents.py*, *manipulateDocumentCatalog.py*, *manipulateFilterAndSubFilter.py* and *manipulateSignatureReferenceDictionary.py*. Each collection contains a function called `getManipulations` which returns a list of the manipulations it provides. As these lists are used by *executeManipulations.py* to execute the manipulations of the corresponding collection the `getManipulations` functions allow to "disable" certain manipulations by removing their names from the list object. Additionally, it is possible to disable collections completely by removing them from the `manipulation_classes` list in *executeManipulations.py*.

All manipulations make use of another script called *pdfmanipulation.py*. This script was originally developed by Christian Mainka and extended to allow the implemented manipulations during this thesis. It contains helper functions which allow to locate and replace objects and to update the XRef information of the manipulated file after the manipulations have been applied. Due to these helper functions the manipulations themselves are very easy to implement and often only contain of one single line of code. The replacement of the `/ByteRange` entry's value of the signature dictionary with an array which contains four zeros is shown in Listing 4.1 as an example.

While only the 24 manipulations described in Section 3.1 were used for the evaluation of the Signature Exclusion attack class presented in Chapter 5, a total of 73 manipulations is implemented in the Python scripts. The 56 other manipulations were

```
def replaceValueWithArrayZeros(inputPDF, updateXrefBoolean):
    return replaceValueInDictionary(inputPDF, "ByteRange", "[0 0 0 0]", updateXrefBoolean)
```

Listing 4.1: Manipulation which replaces the `/ByteRange` entry's value with an array containing four zeros.

disabled for the evaluation by removing them from the described lists. The implementation of the not used manipulations is reasoned by the fact that the implementation took place prior to the final decision which manipulations should be included in the evaluation. All implemented manipulations can be found in the complete list of manipulations devised for the Signature Exclusion attack class in Appendix A.3.1. Implemented manipulations are written in italic.

It is important to note that while the scripts worked flawlessly for several example PDF files during this thesis they are not developed to work for every signed PDF file and might result in errors or wrong and unsuccessful manipulations for certain PDF files. The following limitations are known:

- The functions used to locate objects in the document are based on regular expressions and do not interpret the document in a logical way. Therefore, it is possible that the functions might return a wrong object resulting in a wrong or unsuccessful manipulation.
- The localization of dictionaries is based on regular expressions searching for name objects present as keys in objects. It is possible that the searched name object in present in multiple dictionaries. The scripts always use the last object where the name object was found.
- The scripts are intended to be used with PDF files containing only one signature. If multiple signature dictionaries are present the manipulations will always be applied to the last one.

The scripts developed during this thesis and the helper script developed by Christian Mainka are available at: `https://pdf-insecurity.org/`

5 Evaluation

The theoretical attack classes illustrated in Chapter 3 need to be evaluated practically to determine whether they result in successful attacks in real-life scenarios. This chapter contains the details of this comprehensive evaluation. First, a short overview is given. Afterwards, the structure and details of the testing environment are pointed out and the evaluated PDF processing applications are listed. Finally, the results of the evaluation are described.

5.1 Overview

The evaluation can be roughly divided into three parts - one part for each of the attack classes explained in Chapter 3. The structure of the evaluation is depicted in Table 5.1. The first part is the evaluation of manipulations summarized in the Signature Exclusion attack class. The 24 manipulations listed in Table 3.1 could be executed automatically for a given original document using a Python script as described in Chapter 4. Afterwards, the resulting 24 test files were opened in all 34 processing applications named in Table 5.2. In case the processing application did not display a notice stating that the signature is invalid or the document has been altered after the application of the signature an advanced test file was created to verify the success of the manipulation. These advanced test files exchange the displayed content of the original document with a manipulated page. If the processing application displays the manipulated page and still states that the signature is valid and the document has not been altered on UI-layer 1 the attack is considered successful for UI-layer 1. Afterwards, UI-layer 2 is checked and the attack is considered successful for UI-layer 2 as well, if this layer does not show any information stating an invalid signature is present in the document or the document has been modified after the signature was applied. If the application does not show information regarding the signature on UI-layer 1 in general, UI-layer 2 is checked and the attack is considered to be successful for UI-layer 2 if it states that the signature is valid and the document has not been altered.

The second part of the evaluation was focused on the Incremental Update Abuse attack class and the third part on the Signature Wrapping attack class. As mentioned in Chapter 4, automation for these attack classes was not accomplished in this thesis. Therefore, these attack classes were evaluated manually. This manual

evaluation was conducted in the following way for every application: First, the application was evaluated in the terms of Incremental Update Abuse attacks in an adaptive manner. For this purpose the original document was modified by applying the different manipulated incremental update variants presented in Section 3.2. Afterwards, the application was evaluated in the terms of Signature Wrapping attacks. Again the original document was modified by adding new objects. However, these objects were not appended to the end of the original file but either placed in between the two byte ranges secured by the present signature or in front of them. The second approach of the Signature Wrapping attacks was evaluated by creating 41 test files with different wrappings around the signed data. These test files were opened in all applications to evaluate whether they are vulnerable to the manipulation. For both parts of the evaluation the goal of the manipulations was to change the visible content of the document without triggering any error or warning messages stating that the present signature is invalid or the document has been changed after the signature was applied. Depending on the tested application certain objects and manipulation steps were necessary to successfully exchange the displayed content.

Part	**Attack Class**	**Methodology**
1	Signature Exclusion	24 script generated test files based on selected manipulations (see Table 3.1) opened in all applications.
2	Incremental Update Abuse	Independent evaluation of all applications based on four different variants (see Section 3.2). Test files manually crafted in an adaptive manner depending on previous results.
3	Signature Wrapping	Independent evaluation of all applications based on the first approach (see Section 3.3). Test files manually crafted in an adaptive manner depending on previous results.
		41 manually crafted test files based on selected manipulations (see Table 3.4) opened in all applications.

Table 5.1: Overview of the structure of the evaluation.

It is important to note that the evaluation of some applications listed in Table 5.2 regarding their vulnerability against Incremental Update Abuse or Signature Wrapping attacks was not conducted by the author of this thesis but his advisors. However, their results are presented in this chapter for the sake of a complete and comprehensive evaluation. The corresponding evaluation results in Section 5.3 are marked with an asterisk.

OS	Application	Version
Windows	Adobe Acrobat Reader DC	2018.011.20058[1]
Windows	Adobe Reader XI	11.0.10
Windows	eXpert PDF 12 Ultimate	12.0.21.38686
Windows	Expert PDF Reader	9.0.180
Windows	Foxit Reader	9.2.0.9297
Windows	LibreOffice	6.0.6.2
Windows	Master PDF Editor	5.1.12
Windows	Nitro Pro	11.0.3.173
Windows	Nitro Reader	5.5.9.2
Windows	Nuance Power PDF Standard	3.0.0.17
Windows	PDF Architect 6	6.0.37.38653
Windows	PDF Editor 6 Pro	6.4.2.3521
Windows	PDFelement 6 Pro	6.8.0.3523
Windows	PDF Studio 12 Pro	12.0.7
Windows	PDF Studio Viewer 2018	2018.1.0
Windows	PDF-XChange Editor	7.0 (Build 326.1)
Windows	PDF-XChange Viewer	2.5 (Build 322.9)
Windows	Perfect PDF 10 Premium	10.0.0.1
Windows	Perfect PDF Reader	13.0.3 (2.0)
Windows	Soda PDF	9.3.17
Windows	Soda PDF Desktop	10.2.05.1128
Linux	Adobe Reader 9	9.5.5
Linux	LibreOffice	6.0.3.2
Linux	Master PDF Editor	5.1.12
Linux	PDF Studio 12 Pro	12.0.7
Linux	PDF Studio Viewer 2018	2018.1.0
macOS	Adobe Acrobat Reader DC	2018.011.20058[2]
macOS	Adobe Reader XI	11.0.10
macOS	LibreOffice	6.1.0.3
macOS	Master PDF Editor	5.1.24
macOS	PDF Editor 6 Pro	6.6.2.3315
macOS	PDFelement 6 Pro	6.7.1.3355
macOS	PDF Studio 12 Pro	12.0.7
macOS	PDF Studio Viewer 2018	2018.1.0

Table 5.2: List of all 34 PDF processing applications and their versions evaluated for different operating systems.

[3] An automatic update to version 2018.011.20063 was applied prior to the evaluation of the Incremental Update Abuse and Signature Wrapping attack classes by mistake.

[2] An automatic updates to version 2018.011.20063 was applied prior to the evaluation of the Incremental Update Abuse attack class by mistake. Another update to version 2019.008.20074 was applied prior to the second approach of the Signature Wrapping attack classes by mistake.

As described in Chapter 2, there are two different possibilities to store XRef information in PDF files: XRef sections and XRef streams. XRef streams have a much higher complexity than regular XRef sections. At the beginning of this thesis, the decision was made to work with PDF files which are compliant to PDF version 1.4 and do not involve XRef streams. Due to the limited processing period of this thesis it was not possible to survey files whose XRef information is based on XRef streams at a later time. Therefore, XRef streams are not in the scope of this thesis. All test files used for the evaluation are based on the original document depicted in Appendix A.1. It is compliant to PDF version 1.4 and uses XRef sections. However, the ideas and attacks described should be applicable to PDF files which are compliant to PDF versions 1.5, 1.6 and 1.7 and make use of XRef streams, as well.

"Nuance Power PDF Standard" and the three macOS applications "Master PDF Editor", "PDF Editor 6 Pro" and "PDFelement 6 Pro" state that the signature present in the unaltered original document is invalid. Another original document compliant to PDF version 1.4 whose signature is successfully verified by "Nuance Power PDF Standard" could be found and used for its evaluation. However, the three macOS applications were not able to verify that signature successfully and no other original document containing a valid signature and being compliant to PDF version 1.4 could be found. It was not possible to solve this problem during the evaluation. Therefore, these applications could not be evaluated successfully as it was not possible to determine whether a manipulation was detected.

For some applications updates were released during the evaluation. These updates were applied at the end to evaluate whether the latest versions are still vulnerable to the identified attacks. However, the unsuccessful attack vectors were not evaluated for the updated versions. The results of the re-evaluation are described in Section 5.3.4.

5.2 Testing Environment

The testing environment consists of two virtual machines (VM) and a MacBook: The first VM is based on the 64-bit EDU version of Windows 10 with the latest updates installed on 18.08.2018. The second VM uses the 64-bit version of Xubuntu 18.04.1 LTS as the operating system and was updates to the latest available software on 19.08.2018. The MacBook uses macOS High Sierra in version 10.13.6 updated to the latest available updates on 22.08.2018. In both VMs and on the MacBook different PDF processing applications which are in scope of this thesis and available for the corresponding operating system are installed in their latest versions available at the beginning of the evaluation[3].

[3]The Linux applications were installed from the official Ubuntu software repository if available; it is possible that newer versions are available directly at the vendor's websites.

The decision which applications are in the scope for this thesis was based on the following criteria:

- The application must be capable of processing PDF files.
- The application must support the validation of digital signatures and state the result of the validation to the user at least on one UI-layer.
- The application must be available at no charge - either as a free or a in terms of time-limited use or features constrained version.

As mentioned before, all applications which were found during the first phase of this thesis and fulfill the criteria defined above were evaluated in their latest version. Additionally, older versions of certain applications were added to the evaluation, as well: "Adobe Reader 9" for Linux was evaluated because it is the last version of "Adobe Reader" which was developed for Linux. "Adobe Reader XI" for Windows and macOS, "PDF Studio 12 Pro" for Windows, Linux and macOS and "Soda PDF" for Windows were added to the evaluation because they are the common predecessors of "Adobe Acrobat Reader DC", "PDF Studio Viewer 2018" and "Soda PDF Desktop". This results in a total of 34 applications - 21 applications for Windows, 5 applications for Linux and 8 applications for macOS. The list of all processing applications and the corresponding evaluated version is depicted in Table 5.2.

In the beginning the evaluation was conducted without establishing a trust relationship between the certificate used to sign the original document and the applications evaluated. After the first two parts of the evaluation were finished, this trust relationship was established to ensure that the identified attacks are also successful in a real-world environment[4] and to be compliant to the attacker model defined in Section 2.3. The steps needed to establish this trust relationship are described in Appendix A.2 for every application. However, during this thesis it was not accomplished to establish this trust relationship for all applications. For the following applications no trust relationship could be established:

- "Expert PDF Reader": The application seems to trust any certificate in general. It does not show any information regarding the validity of the signer's certificate.
- "LibreOffice" (Linux and macOS): While the Windows version seems to use the system's certificate store to validate the signer's certificate, the Linux and macOS versions seem to ignore certificates manually added to the system's certificate store. Other options to establish the trust relationship were not found.

[4] We assume that the signer's certificate should be trusted by the processing application in a real-life environment.

- "PDF-XChange Editor": The application seems to trust any certificate in general. It does not show any information regarding the validity of the signer's certificate.
- "PDF-XChange Viewer": The application seems to trust any certificate in general. It does not show any information regarding the validity of the signer's certificate.
- "Perfect PDF Reader": The application seems to trust any certificate in general. It does not show any information regarding the validity of the signer's certificate.

For the last part of the evaluation the trust relationship was not invalidated, except for "LibreOffice" (Windows), "Nitro Pro" and "Nitro Reader" because these applications showed different behaviors depending on the trust relationship during the second part of the evaluation.

During the evaluation some applications received updates on different operating systems. Attacks which have been successful before these updates were evaluated again with the new version of the corresponding application between 10.11.2018 and 12.11.2018. Unsuccessful manipulations and attacks were not evaluated again for the new versions due to the limited time of this thesis. The results of the re-evaluation are presented in Section 5.3. The affected applications are listed in Table 5.3.

OS	**Application**	**Updated Version**
Windows, macOS	Adobe Acrobat Reader DC	2019.008.20080
Windows, macOS	Adobe Reader XI	11.0.23[5]
Windows	Foxit Reader	9.3.0.10826
Windows, Linux, macOS	LibreOffice	6.1.3.2
Windows, Linux, macOS	Master PDF Editor	5.1.68
Windows	Nuance Power PDF Standard	3.0.0.30
Windows	PDF Architect 6	6.1.24.1862
macOS	PDF Editor 6 Pro	6.7.6.3400
Windows	PDFelement 6 Pro	6.8.4.3921
macOS	PDFelement 6 Pro	6.7.6.3399
Windows, Linux, macOS	PDF Studio Viewer 2018	2018.2.0
Windows	PDF-XChange Editor	7.0 (Build 327.1)
Windows	Perfect PDF Reader	13.1.5 (134.4)
Windows	Soda PDF Desktop	10.2.16.1217

Table 5.3: List of applications which received updates during the evaluation and their new versions for different operating systems.

[5]This update was released on 14.11.2017 as an update package and not as a regular installation package (see `https://www.adobe.com/devnet-docs/acrobatetk/tools/ReleaseNotes/`). Due to this it was overlooked during the initial installation of the testing environment.

5.3 Results

In the following sections, the results of the evaluation for the three attack classes are explained. A table containing all results for each attack class can be found in Appendix A.5.

5.3.1 Signature Exclusion

Based on the 24 manipulations described in Section 3.1 24 test files were created using Python scripts (see Chapter 4 for details). Every test file was opened using every application listed in Table 5.2. The information regarding the signature and its validity displayed on the different UI-layers was compared to the information displayed when the original document was opened.

It was possible to execute successful attacks against 4 applications for Windows and 2 applications for macOS. The other 28 applications are not vulnerable to any of the evaluated 24 manipulations. They either discover the manipulation and display a message stating that the signature is invalid, display an error message or crash when a manipulated document is opened.

In the following, the vulnerable applications and the successful manipulations are described:

Adobe Acrobat Reader DC (Windows, macOS): This application is vulnerable to two manipulations of the `/ByteRange` entry of the signature dictionary. Both replacing its value with the null object and removing the whole entry results in a successful attack. After applying one of these manipulations, it is possible to add new content or update the present content of the document using a regular incremental update. For the evaluation the string object (object `4 0`) displayed on the single page of the document was updated using the incremental update depicted in Listing 5.8. Although the signature should be invalid after applying the incremental update Adobe Acrobat Reader DC displays “Signed and all signatures are valid” in UI-layer 1. In UI-layer 2 the behavior differs depending on which UI options are used: If the user opens the signature panel the application again states that the “Signature is valid” and the “Document has not been modified sind this signature was applied” (see Figure 5.4). However, if the user clicks on the visible appearance of the signature placed at the top of the page an error message is displayed stating that there was an “Error during the signature verification”, the “Signature contains incorrect, unrecognized, corrupted or suspicious data” and that an “Unexpected byte range value defining scope of signed data” has been detected (see Figure 5.5). This means the attack is considered successful on UI-layer 1 and partly successful on UI-layer 2.

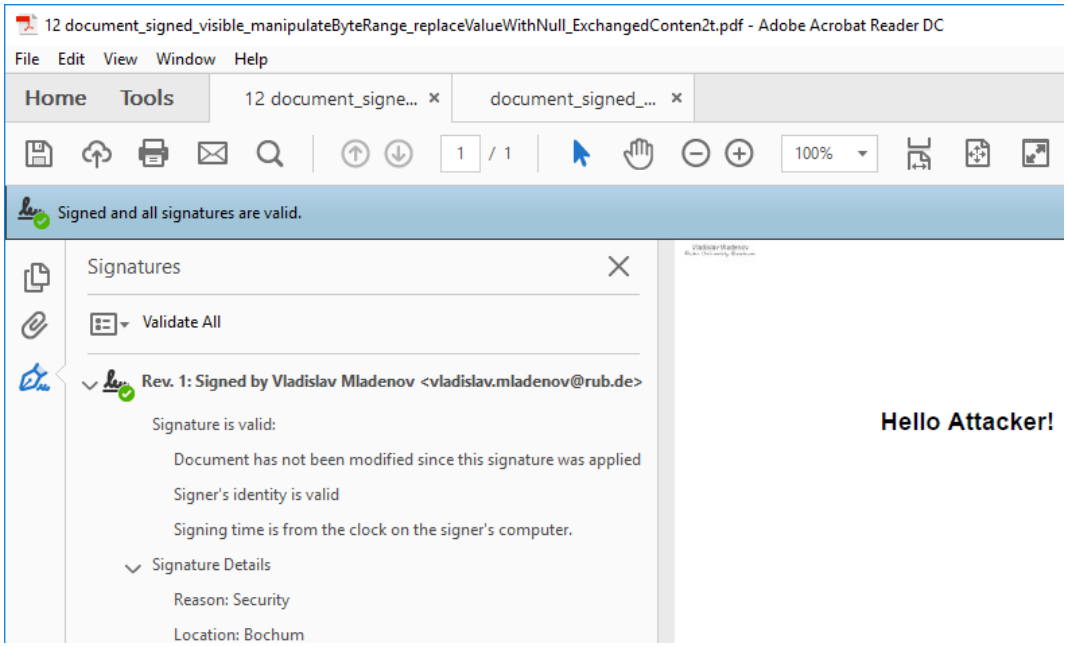

Figure 5.4: UI-layer 1 (blue banner) and UI-layer 2 (signature panel on the left) of Adobe Acrobat Reader DC when a manipulated document results in a successful attack.

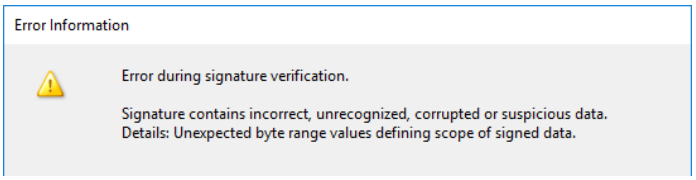

Figure 5.5: Error message displayed when the visible appearance of a signature in a manipulated document is clicked in Adobe Acrobat Reader DC.

Adobe Reader XI (Windows, macOS): The manipulations which can be used for successful attacks against this application are the same as for Adobe Acrobat Reader DC. Also the information displayed on UI-layers 1 and 2 is identical to the information displayed by Adobe Acrobat Reader DC.

PDF Editor 6 Pro (Windows): It was possible to bypass the signature verification for this application using all 5 evaluated manipulations regarding the `/Contents` entry of the signature dictionary. After the `/Contents` entry is manipulated in one of the presented ways, the string object displayed on the page of the document can be updated in a similar manner as depicted in Listing 5.8. However, the attacks were only successful on UI-layer 1: While the application displays a banner stating that the document is "Signed and all signatures are valid" when the manipulated file is opened (see Figure 5.6), it

informs the user that the “Signature is INVALID” and “The document has been altered or corrupted sind the Signature was applied” when the UI option “Validate All Signatures” is executed (see Figure 5.7). Additionally, a click on the visible appearance of the signature results in a dialog to sign the document instead of a windows stating information regarding the present signature. However, the attack could only be successfully evaluated for Windows. The macOS version of the application states that the “Signature is INVALID” and “contains incorrect, unrecognized, corrupted or supiscious data” even for the original document. Due to this behavior it was not possible to determine whether a manipulation was detected or not. It was not possible to solve this problem during the evaluation.

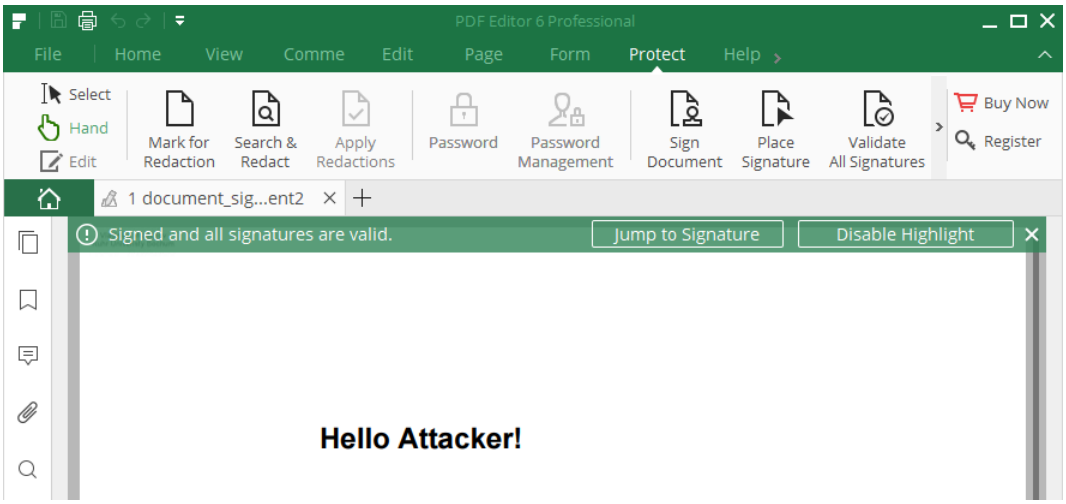

Figure 5.6: UI-layer 1 (green banner) of PDF Editor 6 Pro when a manipulated document results in a successful attack.

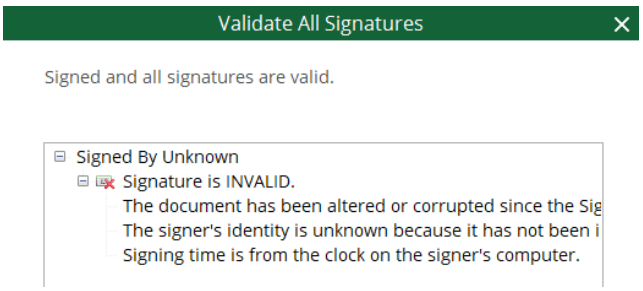

Figure 5.7: UI-layer 2 (“Validate All Signatures” option) of PDF Editor 6 Pro when a manipulated document which results in a successful attack on UI-layer 1 is opened.

PDFelement 6 Pro (Windows): The manipulations which can be used for successful attacks against this application are the same as for PDF Editor 6 Pro. Also the information displayed in UI-layers 1 and 2 is identical to the information displayed by PDF Editor 6 Pro.[6] Again, the attack is only successful for Windows as the macOS version of the application behaves in the same way as the macOS version of PDF Editor 6 Pro.

```
4 0 obj
<< /Length 48 >>
stream
BT
/F1 12 Tf
100 700 Td
(Hello Attacker!) Tj
ET
endstream
endobj
xref
0 1
0000000000 65535 f
4 1
0000022985 00000 n
trailer
<<
/Root 1 0 R
/ID [<1FDC264C24377F037DC7C2587F9C9AA8> <FD7ABFD8D7817F76AF29426E3A75B15A>]
/Size 15
/Prev 22624
>>
startxref
23083
%%EOF
```

Listing 5.8: Incremental update which updates the string object (`4 0`) to display "Hello Attacker!" instead of "Hello World!".

5.3.2 Incremental Update Abuse

In contrast to the evaluation of the Signature Exclusion attacks this attack class was evaluated completely manually without the help of any scripts. Each application listed in Table 5.2 was evaluated independently using the following adaptive procedure: The original file was appended with a manipulated incremental update based on one of the four variants described in Section 3.2. Afterwards the manipulated file was opened and depending on the behavior of the application and the information displayed in UI-layers 1 and 2 the manipulated file was adjusted. These steps were

[6] During the evaluation, the suspicion arose that the two applications share the same code base. This suspicion is based on the (except for the color) identical UI and identical behavior for all test cases. However, this suspicion could not be confirmed, which is why the evaluation was always conducted for both applications independently.

repeated for all four variants until either a successful attack was found or the conclusion was drawn that the application is not vulnerable to attacks based on the four variants. Later, every application was evaluated again using all manipulated files which led to a successful attack for at least one application. If no successful attack could be found for any of the four variants the application was considered to be not vulnerable against Incremental Update Abuse attacks.

Again, the classification if an attack is successful or not was conducted based on the comparison between the information displayed regarding the signature and its validity on the different UI-layers when the original and the manipulated document were opened.

During the evaluation it was possible to execute successful attacks against more than half of the 34 application in the scope of this thesis. 11 applications for Windows, 4 applications for Linux and 3 applications for macOS - a total of 19 applications - was vulnerable to at least one Incremental Update Abuse variant. However, for three of the vulnerable applications certain limitations apply. These limitations are described below.

Suprisingly, all incremental update variants could be used for successful attacks against multiple applications. Figure 5.9 shows the distribution of vulnerable applications among the four variants. Variant 3 was the most successful one. It was used to successfully attack all vulnerable applications except one. Variant 2 could be used for successful attacks against 14 applications, variant 4 for successful attacks against 11 applications and variant 1 for successful attacks against 8 applications.

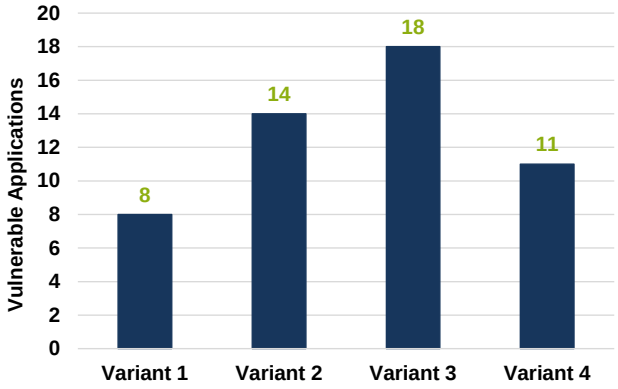

Figure 5.9: Distribution of applications vulnerable to attacks based on the different incremental update variants.

Note: The attacks for applications marked with an asterisk were not created by the author of this thesis but his advisors. Their results are presented here for the sake of a complete and comprehensive evaluation.

In the following, all successful attacks are presented in detail sorted according to the incremental update variant used:

Attacks based on variant 1:

Nuance Power PDF Standard: This application could be attacked using a PDF file manipulated in the same way as the one used to attack PDF Studio 12 Pro.[7] Before establishing the trust relationship between the signer's certificate and the application UI-layer 1 displayed a small question mark next to the visible signature and a banner stating that the document "is signed" but "At least one signature has problems". UI-layer 2 stated that the signature "validity is unknown" and the "revision of the document has not been modified since the signature was applied". Both UI-layers change after the trust relationship is established: UI-layer 1 displays a small purple check mark instead of the question mark and the banner states that the document "is signed" and "All signatures are valid". UI-layer 2 states that the signature "is valid" while still informing the user that the revision of the document has not been modified. This behavior is identical to the behavior when the original document is opened. Therefore, the attack is successful on both UI-layers.

***PDF Studio 12 Pro (Windows, Linux, macOS):** This attack is the simplest one of all successful Incremental Update Abuse attacks. Manipulated objects are added to the original file in order to change the displayed content of the document. Afterwards, a new comment line (starting with `%`) containing the keyword `startxref` is appended at the end. These two manipulations are sufficient to successfully attack this application. Before establishing the trust relationship between the signer's certificate and the application UI-layer 1 displayed a question mark next to the visible signature. UI-layer 2 stated that the signature "validity is UNKNOWN" and the "document has not been modified since the signature was applied". Both UI-layers change after the trust relationship is established: UI-layer 1 displays a green check mark instead of the question mark and UI-layer 2 states that the signature "is VALID" while still informing the user that the document has not been modified. This behavior is identical to the behavior when the original document is opened. Therefore, the attack is successful on both UI-layers.

***PDF Studio Viewer 2018 (Windows, Linux, macOS):** This application could be attacked using the same manipulated PDF file as PDF Studio 12 Pro and showed the exact same behavior on both UI-layers as PDF Studio 12 Pro. Therefore, the attack is successful on both UI-layers.

[7] As described in Section 5.1 Nuance Power PDF Standard states that the signature present in the unaltered original document which was used for the evaluation is invalid. Therefore, another original document had to be used as the base for manipulated files.

Perfect PDF 10 Premium: This application could be attacked using the same manipulated PDF file as PDF Studio 12 Pro. Its behavior is similar to the behavior described for the attack based on variant 2. Therefore, the attack is successful on both UI-layers.

Attacks based on variant 2:

Master PDF Editor (Windows, Linux): This application could be attacked using the same manipulated PDF files as PDF Editor 6 Pro and Perfect PDF 10 Premium. When the manipulated document is opened without establishing a trust relationship between the signer's certificate and the application the user is informed on UI-layer 2 that the "Signature validity is UNKNOWN" but the document "has not been changed since the signatures was applied". This information is similar to the information displayed when the original document is opened. After the trust relationship has been established UI-layer 2 states that the "Signature is VALID" and the document "has not been changed since the signatures was applied". UI-layer 1 does not contain any information related to the signature's validity independently of the opened file. It only contains a banner stating that the "document contains interactive form fields". Therefore, the attack is successful on UI-layer 2. However, the attack could only be successfully evaluated for Windows and Linux. The macOS version of the application states that the present signature is invalid and the document has been modified after the signature was added or is damaged even for the original document. Due to this behavior it was not possible to determine whether a manipulation was detected. It was not possible to solve this problem during the evaluation.

Nitro Pro: This application could be attacked using the same manipulated PDF files as PDF Editor 6 Pro and Perfect PDF 10 Premium. However the following limitations apply: When the manipulated document is opened before the trust relationship between the signer's certificate and the application is established UI-layer 1 shows a question mark next to the visible signature. UI-layer 2 informs the user that the signature "validity is Unknown" and the "document has not changed since it was signed, or only contains changes that are permitted by a previous signer". This information is identical to the information displayed when the original file is opened. After the trust relationship is established the manipulation is detected. On UI-layer 1 the question mark is replaced by a green check mark and a warning symbol (yellow triangle with an exclamation mark). UI-layer 2 now states that the "Signature is Valid", but also contains a note that the document was changed after the signature was applied: "The revision of the document that was covered by this signature has not been altered; however there have been subsequent changes to the document". Additionally, in both cases the button "View Signed Version..." on UI-layer 2 opens a new tab in the application which shows the original

content of the document. Despite the attacker model specifying that viewer applications trust the signer's certificate the attack prior to establishing this trust relationship is still classified as limited successful.

Nitro Reader: This application could be attacked using the same manipulated PDF files as PDF Editor 6 Pro and Perfect PDF 10 Premium. It showed the exact same behavior on both UI-layers with the same limitations as Nitro Pro. Therefore, the attack is considered to be only limitedly successful.

Nuance Power PDF Standard: This application could be attacked using PDF files manipulated in the same way as the ones used to attack PDF Editor 6 Pro and Perfect PDF 10 Premium.[8] Its behavior is similar to the behavior described for the attack based on variant 1. Therefore, the attack is successful on both UI-layers.

PDF Editor 6 Pro (Windows): The attack created for PDF Editor 6 Pro is based on variant 2 which means adding the manipulated objects and a trailer to the original document is sufficient. The added trailer can be a modified copy of the last trailer of the original document. The only modification needed is to replace the value of the `startxref` entry with any value higher than the original one which references the last XRef section of the original file. When the manipulated file is opened without establishing the trust relationship between the application and the signer's certificate UI-layer 1 states that "At least one signature is invalid" while UI-layer 2 names the validity of the signature "UNKNOWN" and clearly states that the document "has not been modified since the signature was applied". The application shows the same behavior when the original document is opened. After the trust relationship has been established the application shows coherent information on both UI-layers: UI-layer 1 states that the document is "Signed and all signatures are valid" and UI-layer 2 states that the "Signature is VALID" and the document "has not been modified since the signature was applied". Therefore, the attack is successful on both UI-layers. However, the attack could only be successfully evaluated for Windows. The macOS version of the application states that the "Signature is INVALID" and "contains incorrect, unrecognized, corrupted or supiscious data" even for the original document. Due to this behavior it was not possible to determine whether a manipulation was detected. It was not possible to solve this problem during the evaluation. *Note: This application could be attacked using the same manipulated PDF file as Perfect PDF 10 Premium, as well.*

PDFelement 6 Pro (Windows): This application could be attacked using the same manipulated PDF file as PDF Editor 6 Pro and showed the exact same behavior on both UI-layers as PDF Editor 6 Pro. Therefore, the attack is successful

[8] As described in Section 5.1 Nuance Power PDF Standard states that the signature present in the unaltered original document which was used for the evaluation is invalid. Therefore, another original document had to be used as the base for manipulated files.

on both UI-layers. Again, the attack is only successful for Windows as the macOS version of the application behaves in the same way as the macOS version of PDF Editor 6 Pro.

PDF Studio 12 Pro (Windows, Linux, macOS): This application could be attacked using the same manipulated PDF files as PDF Editor 6 Pro and Perfect PDF 10 Premium. Its behavior is similar to the behavior described for the attack based on variant 1. Therefore, the attack is successful on both UI-layers.

PDF Studio Viewer 2018 (Windows, Linux, macOS): This application could be attacked using the same manipulated PDF files as PDF Editor 6 Pro and Perfect PDF 10 Premium. It showed the exact same behavior on both UI-layers as PDF Studio 12 Pro. Therefore, the attack is successful on both UI-layers.

Perfect PDF 10 Premium: The attack against Perfect PDF 10 Premium is based on variant 2 of the manipulated incremental updates. In addition to the manipulated objects changing the visible content of the document, a new trailer must be appended to the end of the original file. This trailer can be a modified copy of the last trailer of the original file. However, it must contain a `startxref` entry whose value must not be the actual byte offset of the last XRef section but a value higher by at least 7. During this thesis no explanation could be found why a smaller increase of the value is not sufficient. When the manipulated document is opened before the trust relationship between the signer's certificate and the application is established UI-layer 1 only shows a question mark next to the visible signature. UI-layer 2 states that the signature status is "Valid, signer identity is unknown" and the document "has not been modified since this signature was applied". After establishing the trust relationship, UI-layer 1 shows a green check mark instead of the question mark and on UI-layer 2 the signature status has changed to "Valid, trusted signer identity or issued by CA" while the note that the document has not been modified is still present. Therefore, the attack is successful on both UI-layers.

Attacks based on variant 3:

LibreOffice (Windows, Linux, macOS): The attack for this application is based on incremental update variant 3. This means the incremental update consists of the manipulated objects, a new empty XRef section and a trailer. The empty XRef section contains only the keyword `xref`. The `startxref` value of the trailer must be the correct byte offset of this new XRef section and the trailer dictionary does not need to contain anything except the `/Root` entry referencing the document catalog. The following limitation applies to this attack: When the manipulated document is opened before the trust relationship between the signer's certificate and the application is established UI-layer 1 shows a yellow banner stating that the "signature is OK, but the certificate could not be validated" and UI-layer 2 states that the certificate "could not

be validated". This information is identical to the information displayed when the original document is opened. After establishing the trust relationship[9] the application's behavior is different for the manipulated document than for the original one: Opening the original document results in a new blue banner on UI-layer 1 stating that "This document is digitally signed and the signature is valid" while opening the manipulated one results in a new yellow banner stating that the "signature is OK, but the document is only partially signed". This means after establishing the trust relationship the application detects that the document has been updated after the signature was applied. Despite the attacker model specifying that viewer applications trust the signer's certificate the attack prior to establishing this trust relationship is still classified as limited successful. *Note: This application could be attacked using the same manipulated PDF file as Perfect PDF Reader, as well. However, the same limitation applies.*

Master PDF Editor (Windows, Linux): This application could be attacked using the same manipulated PDF files as LibreOffice and Perfect PDF Reader. Its behavior is similar to the behavior described for the attack based on variant 2. Therefore, the attack is successful on UI-layer 2. The attack is only successful for Windows and Linux; the macOS version of the application shows the same behavior as described for the attack based on variant 2.

Nitro Pro: This application could be attacked using the same manipulated PDF files as LibreOffice and Perfect PDF Reader. Its behavior is similar to the behavior described for the attack based on variant 2. This means the same limitations apply for this attack, too. When the trust relationship between the application and the signer's certificate is established the application detects that the document has been updated after the signature was applied. Due to these limitations the attack is considered to be only limitedly successful.

Nitro Reader: This application could be attacked using the same manipulated PDF files as LibreOffice and Perfect PDF Reader. It showed the exact same behavior on both UI-layers with the same limitations as Nitro Pro. Therefore, the attack is considered to be only limitedly successful.

Nuance Power PDF Standard: This application could be attacked using PDF files manipulated in the same way as the ones used to attack LibreOffice and Perfect PDF Reader.[10] Its behavior is similar to the behavior described for the attack based on variant 1. Therefore, the attack is successful on both UI-layers.

[9] As described in Section 5.2 it was not achieved to establish the trust relationship between the signer's certificate and the Linux or macOS version of LibreOffice during this thesis. Therefore, no statement can be made as to whether these versions show the same behavior as the Windows version when the trust relationship is established.

[10] As described in Section 5.1 Nuance Power PDF Standard states that the signature present in the unaltered original document which was used for the evaluation is invalid. Therefore, another original document had to be used as the base for manipulated files.

PDF Editor 6 Pro (Windows): This application could be attacked using the same manipulated PDF files as LibreOffice and Perfect PDF Reader. Its behavior is similar to the behavior described for the attack based on variant 2. Therefore, the attack is successful on both UI-layers. The attack is only successful for Windows; the macOS version of the application shows the same behavior as described for the attack based on variant 2.

PDFelement 6 Pro (Windows): This application could be attacked using the same manipulated PDF files as LibreOffice and Perfect PDF Reader. It showed the exact same behavior on both UI-layers as PDF Editor 6 Pro. Therefore, the attack is successful on both UI-layers. Again, the attack is only successful for Windows as the macOS version of the application behaves in the same way as the macOS version of PDF Editor 6 Pro.

PDF Studio 12 Pro (Windows, Linux, macOS): This application could be attacked using the same manipulated PDF files as LibreOffice and Perfect PDF Reader. Its behavior is similar to the behavior described for the attack based on variant 1. Therefore, the attack is successful on both UI-layers.

PDF Studio Viewer 2018 (Windows, Linux, macOS): This application could be attacked using the same manipulated PDF files as LibreOffice and Perfect PDF Reader. It showed the exact same behavior on both UI-layers as PDF Studio 12 Pro. Therefore, the attack is successful on both UI-layers.

Perfect PDF 10 Premium: This application could be attacked using the same manipulated PDF file as Perfect PDF Reader. Its behavior is similar to the behavior described for the attack based on variant 2. Therefore, the attack is successful on both UI-layers.

Perfect PDF Reader: While this attack was also based on variant 3 it differs strongly from the one against LibreOffice: The incremental update also includes adding the manipulated objects, a new XRef section and a trailer to the original file. However, the XRef section must not be empty but contain entries for all manipulated objects and all objects added to the file as a part of the signature. When the original document is opened UI-layer 1 contains a green badge containing a checkmark next to the visible signature. UI-layer 2 states that the signature is “Gültig” and shows the same badge. Opening the manipulated document results in different information presented to the user: A tiny yellow warning symbol is added to the green badge on both UI-layers. The information on UI-layer 2 states that the signature is “Gültig für unterschriebene Version”. The comparison of UI-layer 2 for the original and manipulated document is shown in Figure 5.10. Despite this difference in behavior the attack is classified as successful on both UI-layers. This classification is justified on the missing option to view the signed version of the document and the minor difference between the displayed badges. The comparison of the two badges

(see Figure 5.11) shows that the tiny warning symbol can easily be overlooked by the user.

Note: As described in Section 5.2 it was not necessary to establish a trust relationship between the signer's certificate and the application because the application seems to trust any certificate in general.

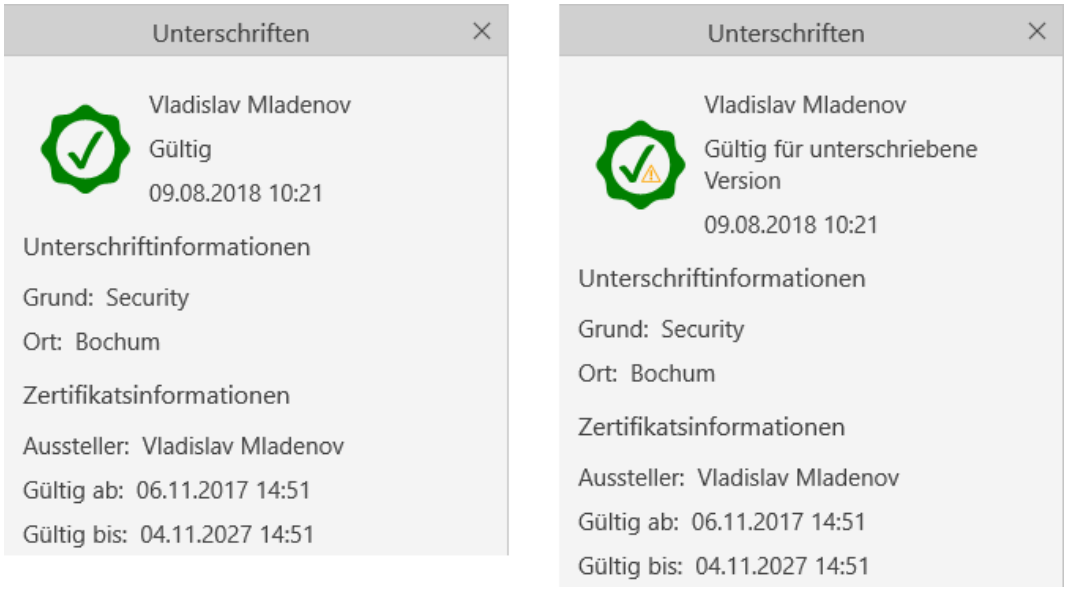

Figure 5.10: Comparison of UI-layer 2 of Perfect PDF Reader when the unaltered original document (left) and the manipulated document (right) are opened.

Figure 5.11: Comparison of the badges displayed on both UI-layers of Perfect PDF Reader when the original (left) and the manipulated document (right) are opened.

Attacks based on variant 4:

***Foxit Reader:** This application could be successfully attacked by a manipulated file appended using incremental update variant 4. This means the manipulated objects added to the end of the original file include a copy of some signature related objects from the original document (field dictionary of the signature form field and signature dictionary). UI-layer 1 does not show any

information regarding the signature or its validity when the original or manipulated document is opened. UI-layer 2 however, states that the signature is "unknown" and the document "has not been modified since this signature was applied" before the trust relationship between the signer's certificate and the application is established. After establishing the trust relationship, UI-layer 2 states that signature is "valid" and still informs the user that the document has not been modified. Again, this behavior is identical for the manipulated and the original document. Therefore, the attack is successful on UI-layer 2.

Master PDF Editor (Windows, Linux): This application could be attacked using the same manipulated PDF file as Foxit Reader. Its behavior is similar to the behavior described for the attack based on variant 2. Therefore, the attack is successful on UI-layer 2. The attack is only successful for Windows and Linux; the macOS version of the application shows the same behavior as described for the attack based on variant 2.

PDF Editor 6 Pro (Windows): This application could be attacked using the same manipulated PDF file as Foxit Reader. Its behavior is similar to the behavior described for the attack based on variant 2. Therefore, the attack is successful on both UI-layers. The attack is only successful for Windows; the macOS version of the application shows the same behavior as described for the attack based on variant 2.

PDFelement 6 Pro (Windows): This application could be attacked using the same manipulated PDF file as Foxit Reader. It showed the exact same behavior on both UI-layers as PDF Editor 6 Pro. Therefore, the attack is successful on both UI-layers. Again, the attack is only successful for Windows as the macOS version of the application behaves in the same way as the macOS version of PDF Editor 6 Pro.

PDF Studio 12 Pro (Windows, Linux, macOS): This application could be attacked using the same manipulated PDF file as Foxit Reader. Its behavior is similar to the behavior described for the attack based on variant 1. Therefore, the attack is successful on both UI-layers.

PDF Studio Viewer 2018 (Windows, Linux, macOS): This application could be attacked using the same manipulated PDF file as Foxit Reader. It showed the exact same behavior on both UI-layers as PDF Studio 12 Pro. Therefore, the attack is successful on both UI-layers.

5.3.3 Signature Wrapping

The evaluation of the Signature Wrapping attack class was divided into two parts because this attack class is based on two different approaches. The first approach which involves placing manipulated objects in between the two signed byte ranges

of the original document was evaluated using an adaptive procedure similar to the Incremental Update Abuse attack class. The original file was manipulated by adding objects and a new XRef section in between the two signed byte ranges, adjusting the `/ByteRange` entry of the signature dictionary accordingly and adding padding if necessary.[11] Afterwards, the file was opened in the currently tested application. If the attack was not successful further changes (e.g., adding a trailer behind the newly added XRef section) were made adaptively. This process was repeated for each application until a successful attack was found or the application was considered to be not vulnerable to this Signature Wrapping approach.

The second Signature Wrapping approach was evaluated differently. Similar to the Signature Exclusion attack class general test files were created using different manipulations and different wrappings around the signed data. The signed data consists of the two signed byte ranges which were connected and placed at the end of the file. The 41 test files based on the manipulations described in Table 3.4 were opened in all applications to check whether they are vulnerable to the specific manipulation.

The classification if an attack is successful or not was again conducted based on the comparison between the information displayed regarding the signature and its validity on the different UI-layers when the original and the manipulated document were opened.

Using the first approach it was possible to successfully attack 21 applications. One single manipulated file initially created during the evaluation of Nitro Pro led to successful attacks for 18 applications without any adjustments. The second approach could successfully be used for attacks against 18 applications. Three test files turned out to be the most universal ones: Test file 1 which includes no wrapping around the signed data placed at the end of the file could be used for attacks against 16 applications. The two files 36 and 41 which are both variants of test file 1 and do not include any wrapping either could even be used to successfully attack 18 and 17 applications. In contrast to test file 1 they do not contain a `/ByteRange` entry specifying a first byte range which has the size 0. The two applications PDF-XChange Editor and Perfect PDF Reader are vulnerable to test files 36 and 41 but not to test file 1 which means they do not accept byte ranges whose size is 0. Figure 5.12 shows the distribution of vulnerable applications among the different test files. However, for all applications which are vulnerable to this Signature Wrapping approach the first approach also led to successful attacks.

Note: The attacks for applications marked with an asterisk were not created by the author of this thesis but his advisors. Their results are presented here for the sake of a complete and comprehensive evaluation.

[11] Padding might by necessary to ensure that the `startxref` value of the last trailer points to the new XRef section.

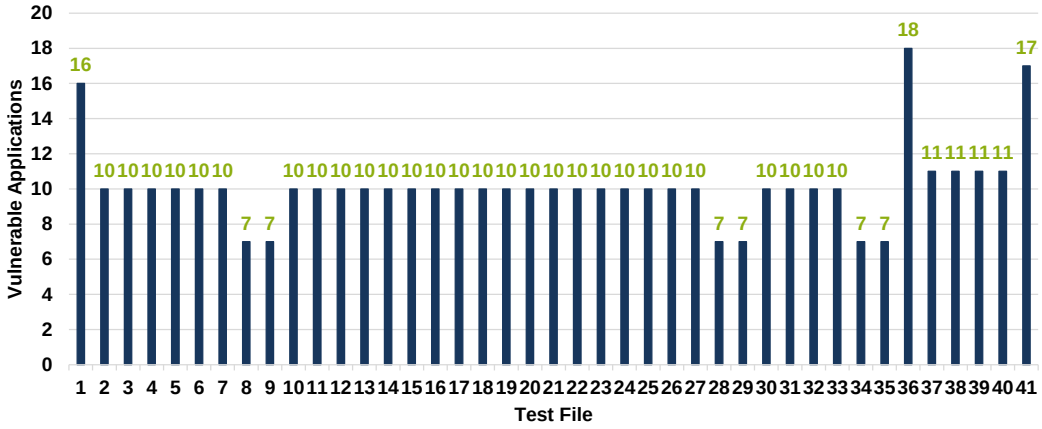

Figure 5.12: Distribution of applications vulnerable to attacks based on the test files from the second Signature Wrapping approach.

In the following, all successful attacks achieved with the first Signature Wrapping approach are presented in detail:

eXpert PDF 12 Ultimate: This application could be attacked using the same manipulated PDF files as Nitro Pro and Soda PDF Desktop. When one of the manipulated files is opened the application displays the manipulated content. While UI-layer 1 does not contain any information regarding the signature UI-layer 2 states that the "Signature is Valid" and "After adding the signature, the document has not been modified". This behavior is similar to the behavior when the original document is opened. Therefore, the attack is successful on UI-layer 2.

Expert PDF Reader: This application could be attacked using the same manipulated PDF file as Nitro Pro. When the manipulated file is opened the application displays the manipulated content. While UI-layer 1 does not contain any information regarding the signature's validity UI-layer 2 states that the "Signature is VALID" and "The revision of the document that was covered by this signature has not been altered". This behavior is similar to the behavior when the original document is opened. Therefore, the attack is successful on UI-layer 2.

***Foxit Reader:** The following manipulations were applied to the original document to successfully attack Foxit Reader: First, a copy of the `/ByteRange` entry was added behind the `/Contents` entry of the signature dictionary. Afterwards, the signature dictionary was closed by adding `» endobj`. An updated version of the stream displayed on the single page of the document was added behind the closed signature dictionary to manipulate the visible content of the document. Then, a new XRef section was placed behind this object. The XRef section

must contain correct XRef entries for the newly added object as well as the objects contained in the second signed byte range. Otherwise the application is not able to locate these objects and fails to validate the present signature. To ensure that the new XRef section starts at the byte offset specified by the `startxref` value of the last trailer (which is part of the signed data and cannot be adjusted without invalidating the signature) whitespaces are used as padding in front of the XRef section. Finally, the third value of the `/ByteRange` entry's array is replaced with the new starting byte offset of the second signed byte range. When the manipulated file is opened the application displays the manipulated content. While UI-layer 1 does not contain any information regarding the signature UI-layer 2 states that the "Signature is VALID" and "The document has not been modified since the signature was applied". This behavior is similar to the behavior when the original document is opened. Therefore, the attack is successful on UI-layer 2.

Nitro Pro: The following manipulations were applied to the original document to successfully attack Nitro Pro: First, a copy of the `/ByteRange` entry was added behind the `/Contents` entry of the signature dictionary. Afterwards, the signature dictionary was closed by adding `» endobj`. An updated document catalog, a new pages object, a new page object and a new stream were added behind the closed signature dictionary to manipulate the visible content of the document. Then, a new XRef section and a copy of the original file's last trailer were placed behind these objects. The copy of the trailer does not need to contain more than the trailer dictionary. The XRef section must contain correct XRef entries for the newly added objects as well as the objects contained in the second signed byte range. Otherwise the application is not able to locate these objects and fails to validate the present signature. To ensure that the new XRef section starts at the byte offset specified by the `startxref` value of the last trailer (which is part of the signed data and cannot be adjusted without invalidating the signature) whitespaces are used as padding in front of the XRef section. Finally, the third value of the `/ByteRange` entry's array is replaced with the new starting byte offset of the second signed byte range. When the manipulated file is opened the application displays the manipulated content. A green checkmark is displayed next to the visible signature on UI-layer 1 and the information on UI-layer 2 states that the "Signature is Valid" and "The document has not changed since it was signed, or only contains changes that are permitted by a previous signer". This behavior is similar to the behavior when the original document is opened. Therefore, the attack is successful on both UI-layers.

Nitro Reader: This application could be attacked using the same manipulated PDF file as Nitro Pro. It showed the exact same behavior on both UI-layers as Nitro Pro. Therefore, the attack is successful on both UI-layers.

Nuance Power PDF Standard: As described in Section 5.1 this application states that the signature present in the unaltered original document which was used for the evaluation is invalid. Therefore, another original document had to be used as the base for manipulated files. Using this original document and applying the following manipulations it was possible to attack the application: The `/ByteRange` entry is in front of the `/Contents` entry in this original document. Therefore, it is part of the first signed byte range and cannot be manipulated using the first Signature Wrapping approach. However, adding a second `/ByteRange` entry behind the `/Contents` entry's value can be used to overwrite the original `/ByteRange` entry. The second signed byte range of the original document which starts with the `/M` entry behind the `/Contents` entry's value must be placed at the end of the file and the manipulations must be added in between the two signed byte ranges. First, a copy of the `/M` entry and the closing `» endobj` of the signature dictionary must be placed behind the newly added `/ByteRange` entry. Afterwards, an updated version of the original page object and a new stream object referenced by this page object are added to the file. These two objects manipulate the visible content of the document. Finally, the third value of the newly added `/ByteRange` entry's array is replaced with the new starting byte offset of the second signed byte range. When the manipulated file is opened the application displays the manipulated content. A purple checkmark is displayed next to the visible signature on UI-layer 1 and the information on UI-layer 2 states that the "Signature is valid" and the "Document has not been modified since this signature was applied". This behavior is similar to the behavior when the original document is opened. Therefore, the attack is successful on both UI-layers.

PDF Architect 6: This application could be attacked using the same manipulated PDF files as Nitro Pro and Soda PDF Desktop. It showed the exact same behavior on both UI-layers as eXpert PDF 12 Ultimate. Therefore, the attack is successful on UI-layer 2.

PDF Editor 6 Pro (Windows): This application could be attacked using the same manipulated PDF file as Nitro Pro. When the manipulated file is opened the application displays the manipulated content. A banner stating that the document is "Signed and all signatures are valid" is displayed on UI-layer 1 and the information on UI-layer 2 states that the "Signature is VALID" and "The document has not been modified since the signature was applied". This behavior is similar to the behavior when the original document is opened. Therefore, the attack is successful on both UI-layers. However, the attack could only be successfully evaluated for Windows. The macOS version of the application states that the "Signature is INVALID" and "contains incorrect, unrecognized, corrupted or supiscious data" even for the original document. Therefore, it was not possible to determine whether a manipulation was detected. It was not possible to solve this problem during the evaluation.

PDFelement 6 Pro (Windows): This application could be attacked using the same manipulated PDF file as Nitro Pro. It showed the exact same behavior on both UI-layers as PDF Editor 6 Pro. Therefore, the attack is successful on both UI-layers. Again, the attack is only successful for Windows as the macOS version of the application behaves in the same way as the macOS version of PDF Editor 6 Pro.

PDF Studio 12 Pro (Windows, Linux, macOS): This application could be attacked using the same manipulated PDF file as Nitro Pro. When the manipulated file is opened the application displays the manipulated content. A green checkmark is displayed next to the visible signature on UI-layer 1 and the information on UI-layer 2 states that the "Signature is VALID" and "The document has not been modified since the signature was applied". This behavior is similar to the behavior when the original document is opened. Therefore, the attack is successful on both UI-layers.

PDF Studio Viewer 2018 (Windows, Linux, macOS): This application could be attacked using the same manipulated PDF file as Nitro Pro. It showed the exact same behavior on both UI-layers as PDF Studio 12 Pro. Therefore, the attack is successful on both UI-layers.

PDF-XChange Editor: This application could be attacked using the same manipulated PDF file as Nitro Pro. When the manipulated file is opened the application displays the manipulated content. While UI-layer 1 does not contain any information regarding the signature's validity UI-layer 2 states that the "Signature is VALID" and "The document was not modified since the signature was applied". This behavior is similar to the behavior when the original document is opened. Therefore, the attack is successful on UI-layer 2.

PDF-XChange Viewer: This application could be attacked using the same manipulated PDF file as Nitro Pro. When the manipulated file is opened the application displays the manipulated content. While UI-layer 1 does not contain any information regarding the signature UI-layer 2 states that the "Signature is VALID" and "The document was not modified since the signature was applied". This behavior is similar to the behavior when the original document is opened. Therefore, the attack is successful on UI-layer 2.

Perfect PDF 10 Premium: This application could be attacked using the same manipulated PDF file as Nitro Pro. When the manipulated file is opened the application displays the manipulated content. A green checkmark is displayed next to the visible signature on UI-layer 1 and the information on UI-layer 2 states that "The signature is valid" and "This document has not been modified since this signature was applied". This behavior is similar to the behavior when the original document is opened. Therefore, the attack is successful on both UI-layers.

Perfect PDF Reader: This application could be attacked using the same manipulated PDF file as Nitro Pro. When the manipulated file is opened the application displays the manipulated content. A green badge containing a checkmark is displayed next to the visible signature on UI-layer 1 and the information on UI-layer 2 states that the signature is "Gültig". This behavior is similar to the behavior when the original document is opened. Therefore, the attack is successful on both UI-layers.

Soda PDF: This application could be attacked using the same manipulated PDF files as Nitro Pro and Soda PDF Desktop. It showed the exact same behavior on both UI-layers as eXpert PDF 12 Ultimate. Therefore, the attack is successful on UI-layer 2.

***Soda PDF Desktop:** The following manipulations were applied to the original document to successfully attack Soda PDF Desktop: First, a copy of the `/ByteRange` entry was added behind the `/Contents` entry of the signature dictionary. Afterwards, the signature dictionary was closed by adding `» endobj`. A copy of all objects contained in the second signed byte range, an updated document catalog, an updated pages object, an updated page object and an updated stream were added behind the closed signature dictionary to manipulate the visible content of the document. Then, a new XRef section and a copy of the original file's last trailer were placed behind these objects. The copy of the trailer does not need to contain more than the trailer dictionary. The XRef section must contain correct XRef entries for the newly added objects. Otherwise the application is not able to locate these objects and fails to validate the present signature. To ensure that the new XRef section starts at the byte offset specified by the `startxref` value of the last trailer (which is part of the signed data and cannot be adjusted without invalidating the signature) whitespaces are used as padding in front of the XRef section. Finally, the third value of the `/ByteRange` entry's array is replaced with the new starting byte offset of the second signed byte range. When the manipulated file is opened the application displays the manipulated content. While UI-layer 1 does not contain any information regarding the signature UI-layer 2 states that the "Signature is Valid" and "After adding the signature, the document has not been modified". This behavior is similar to the behavior when the original document is opened. Therefore, the attack is successful on UI-layer 2.

In the following, all successful attacks achieved with the second Signature Wrapping approach are presented in detail:

eXpert PDF 12 Ultimate: This application could be successfully attacked using the test files 1, 36 and 41. These files have in common that no wrapping around the signed data at the end of the file is used. The only difference between the files is the `/ByteRange` entry. File 1 uses a `/ByteRange` entry specifying one byte range with size 0 and a second byte range at the end of the file containing all

signed data. File 36 makes use of a `/ByteRange` entry specifying a byte range with size 5 at the beginning of the file and a second byte range at the end of the file. In file 41 the `/ByteRange` entry specifies two byte ranges close to the end of the file which means the first byte range does not start at byte offset 0. When one of the three files is opened the application displays the manipulated content. While UI-layer 1 does not contain any information regarding the signature UI-layer 2 states that the "Signature is Valid" and "After adding the signature, the document has not been modified". This behavior is similar to the behavior when the original document is opened. Therefore, the attack is successful on UI-layer 2. In addition to the successful attacks, an interesting observation was made for some test files. The signature panel (part of UI-layer 2) classifies the opened document as "Revision 1" for some test files (8, 9, 16, 17, 22, 23, 28, 29, 34, 35); for others the document is called "Revision 2" (2-7, 10-15, 18-21, 24-27, 30-33, 37-40). The signature panel contains an option called "View Signed Version". This button opens a new tab which should present the originally signed version of the document to the user.[12] For all test files called "Revision 1" this button fulfills its task - the content of the original document is displayed instead of the manipulated one. However, for all test files called "Revision 2" the button opens a new tab but still displays the manipulated content. The signature panel now states that the "Signature is Valid" and "After adding the signature, the document has not been modified"; the button "View Signed Version" is disabled and not clickable. This implies to the user that the opened document has been altered after the signature was applied but the content displayed in the new tab is the original file content. These attacks are not classified as successful because the attacker model (see Section 2.3) specifies that both UI-layers must not state that the document was modified after the application of the signature when the manipulated document is opened.

Expert PDF Reader: This application could be successfully attacked using the same test files as eXpert PDF 12 Ultimate. When one of the three files is opened the application displays the manipulated content. While UI-layer 1 does not contain any information regarding the signature's validity UI-layer 2 states that the "Signature is VALID" and "The revision of the document that was covered by this signature has not been altered". This behavior is similar to the behavior when the original document is opened. Therefore, the attack is successful on UI-layer 2.

Foxit Reader: This application could be successfully attacked using 34 of the 41 test files. All test files which did not lead to successful attacks (8, 9, 28, 29, 34, 35), except one (41), have in common that the added XRef section and trailer were placed behind the wrapped signed data at the end of the file. When one

[12] The test files which led to successful attacks (1, 36 and 41) are also called "Revision 2", but the "View Signed Version" button is disabled and not clickable.

of these test files is opened the application does not show any indication that a signature is present on any UI-layer. Test file 41 contains a `/ByteRange` entry which specifies two byte ranges close to the end of the file instead of one starting at the beginning of the file. The application detects this and states that there war an "Error during signature verification" and that "Unexpected byte range values defining scope of signed data" were found on UI-layer 2. When one of the successful test files (1-7, 10-27, 30-33, 36-40) is opened the application displays the manipulated content. It states that the "Signature is VALID" and "The document has not been modified since the signature was applied" on UI-layer 2 while UI-layer 1 does not contain any information regarding the signature. This behavior is similar to the behavior when the original document is opened. Therefore, the attack is successful on UI-layer 2.

PDF Architect 6: This application could be successfully attacked using the same test files as eXpert PDF 12 Ultimate. It showed the exact same behavior for all test files on both UI-layers as eXpert PDF 12 Ultimate (including the described behavior in terms of "Revision 1" or "Revision 2"). Therefore, the attack is successful on UI-layer 2.

PDF Editor 6 Pro (Windows): This application could be successfully attacked using 35 of the 41 test files. All test files which did not lead to successful attacks (8, 9, 28, 29, 34, 35) have in common that the added XRef section and trailer were placed behind the wrapped signed data at the end of the file. When one of these test files is opened the application displays a banner stating that the document is "Signed and all signatures are valid" on UI-layer 1, but displays the original content of the document instead of the manipulated one. Additionally, UI-layer 2 states that the "Signature is INVALID" and "The document has been altered or corrupted since the Signature was applied". When one of the successful test files (1-7, 10-27, 30-33, 36-41) is opened the application displays the manipulated content. It shows a banner stating that the document is "Signed and all signatures are valid" on UI-layer 1 and the information on UI-layer 2 states that the "Signature is VALID" and "The document has not been modified since the signature was applied". This behavior is similar to the behavior when the original document is opened. Therefore, the attack is successful on both UI-layers. However, the attack could only be successfully evaluated for Windows. The macOS version of the application states that the "Signature is INVALID" and "contains incorrect, unrecognized, corrupted or supiscious data" even for the original document. Due to this behavior it was not possible to determine whether a manipulation was detected. It was not possible to solve this problem during the evaluation.

PDFelement 6 Pro (Windows): This application could be successfully attacked using the same test files as PDF Editor 6 Pro. It showed the exact same behavior for all test files on both UI-layers as PDF Editor 6 Pro. Therefore, the attack

is successful on both UI-layers. Again, the attack is only successful for Windows as the macOS version of the application behaves in the same way as the macOS version of PDF Editor 6 Pro.

PDF Studio 12 Pro (Windows, Linux, macOS): This application could be successfully attacked using all 41 test files. Regardless of which test file is opened the application displays the manipulated content. A green checkmark is displayed next to the visible signature on UI-layer 1 and the information on UI-layer 2 states that the "Signature is VALID" and "The document has not been modified since the signature was applied". This behavior is similar to the behavior when the original document is opened. Therefore, the attack is successful on both UI-layers.

PDF Studio Viewer 2018 (Windows, Linux, macOS): This application could be successfully attacked using the same test files as PDF Studio 12 Pro. It showed the exact same behavior for all test files on both UI-layers as PDF Studio 12 Pro. Therefore, the attack is successful on both UI-layers.

PDF-XChange Editor: This application could be successfully attacked using the test files 36 and 41. These files have in common that no wrapping around the signed byte range at the end of the file is used and the size of the first byte range specified by the `/ByteRange` entry is not 0. File 36 makes use of a `/ByteRange` entry specifying a byte range with size 5 at the beginning of the file and a second byte range at the end of the file. In file 41 the `/ByteRange` entry specifies two byte ranges close to the end of the file. When one of the two files is opened the application displays the manipulated content. While UI-layer 1 does not contain any information regarding the signature's validity UI-layer 2 states that the "Signature is VALID" and "The document was not modified since the signature was applied". This behavior is similar to the behavior when the original document is opened. Therefore, the attack is successful on UI-layer 2.

PDF-XChange Viewer: This application could be successfully attacked using the same test files as eXpert PDF 12 Ultimate. When one of the three files is opened the application displays the manipulated content. While UI-layer 1 does not contain any information regarding the signature UI-layer 2 states that the "Signature is VALID" and "The document was not modified since the signature was applied". This behavior is similar to the behavior when the original document is opened. Therefore, the attack is successful on UI-layer 2.

Perfect PDF 10 Premium: This application could be successfully attacked using all 41 test files. Regardless of which test file is opened the application displays the manipulated content. A green checkmark is displayed next to the visible signature on UI-layer 1 and the information on UI-layer 2 states that "The

signature is valid" and "This document has not been modified since this signature was applied". This behavior is similar to the behavior when the original document is opened. Therefore, the attack is successful on both UI-layers.

Perfect PDF Reader: This application could be successfully attacked using the test files 36-41. In contrast to all others, these test files use `/ByteRange` entries which specify a first byte range whose size is not 0. When one of these six test files is opened the application displays the manipulated content. For test files 36 and 41 a green badge containing a checkmark is displayed next to the visible signature on UI-layer 1. UI-layer 2 states that the signature is "Gültig" and contains the same badge. This behavior is similar to the behavior when the original document is opened. Therefore, the attack is successful on both UI-layers. For test files 37-40 a tiny yellow warning symbol is added to the green badge on both UI-layers and the information on UI-layer 2 states that the signature is "Gültig für unterschriebene Version". Similar to the results of the Incremental Update Abuse attack class the attacks based on these test files are classified as successful despite this difference in behavior. This classification is justified on the missing option to view the signed version of the document and the minor difference between the displayed badges.

Soda PDF: This application could be successfully attacked using the same test files as eXpert PDF 12 Ultimate. It showed the exact same behavior for all test files on both UI-layers as eXpert PDF 12 Ultimate (including the described behavior in terms of "Revision 1" or "Revision 2"). Therefore, the attack is successful on UI-layer 2.

Soda PDF Desktop: This application could be successfully attacked using the same test files as eXpert PDF 12 Ultimate. It showed the exact same behavior for all test files on both UI-layers as eXpert PDF 12 Ultimate (including the described behavior in terms of "Revision 1" or "Revision 2"). Therefore, the attack is successful on UI-layer 2.

5.3.4 Re-evaluation after Application Updates

For several applications new versions were released during the time of the evaluation. To ensure the identified attacks are still successful for the latest versions of the applications when the thesis is submitted all applications were updated to their latest versions on 10.11.2018 (Windows and Linux) resp. 12.11.2018 (macOS). Afterwards, all attacks which have been successful for the previous versions were evaluated again.

All attacks - except one attack of the first Signature Wrapping approach - were still successful for the updated applications. The behavior of the updated applications was similar to the behavior of their initially evaluated versions in all cases. The only exception is the evaluation of the first Signature Wrapping approach for PDF Architect 6. While the old version could be successfully attacked on UI-layer 2 using the same manipulated file initially created for Nitro Pro and showed the exact same behavior as eXpert PDF 12 Ultimate (see Section 5.3.3), the new version was not vulnerable to this attack. Instead of displaying the manipulated content and informing the user that the "Signature is Valid" on UI-layer 2, the new version showed an error message that it "Failed to open" the file. This behavior is similar to the behavior of Soda PDF Desktop (both the old and updated version). However, the Signature Wrapping attacks based on the manipulated file initially created for Soda PDF Desktop and test files 1, 36 and 41 of the second Signature Wrapping approach are still successful.

5.3.5 Summary

During the comprehensive evaluation of digital signatures embedded in PDF documents described in this chapter various vulnerabilities were identified. They allow an attacker to bypass the signature protection in 30 of the 34 evaluated applications completely and to change the displayed content of signed PDF documents arbitrarily. Table 5.14 contains an overview of the results of this evaluation and tables containing the complete results can be found in Appendix A.5.

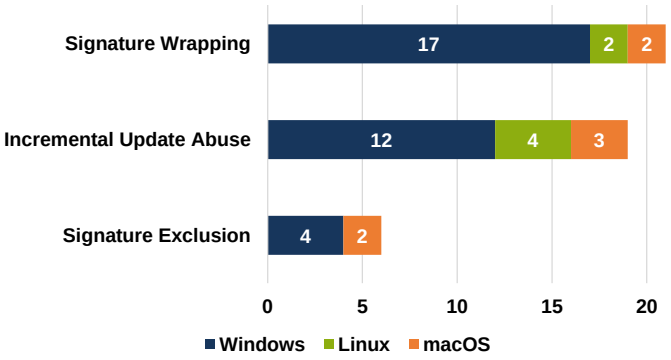

Figure 5.13: Number of applications vulnerable to the three attack classes for different operating systems.

During this thesis 21 applications for Windows, 5 applications for Linux and 8 applications for macOS were evaluated. In Figure 5.13 an overview of the vulnerability to different attack classes and the vulnerable applications for different operating systems is given. All three attack classes resulted in successful attacks for multiple applications. While the Signature Exclusion attack class could be used to

successfully attack 6 applications, using the Incremental Update Abuse attack class vulnerabilities in 19 applications were identified. The Signature Wrapping attack class was the most successful one leading to attacks against 21 applications. While vulnerabilities for at least one attack class were identified in all 21 Windows applications, 4 of the 5 Linux and 5 of the 8 macOS applications could be successfully attacked using at least one attack vector. However, the 3 macOS applications for which no vulnerabilities were identified could not be evaluated successfully. They stated that the signature present in the unaltered original document is invalid. Due to this behavior it was not possible to determine whether the applied manipulations were successful. Therefore, the evaluation result for these applications is unknown. It is likely that these applications can be attacked using on of the described attacks if another original document is used for the manipulations because the Windows and Linux versions of the corresponding applications are vulnerable to different attacks. Despite its age the only application which could not be successfully attacked with attack vectors based on any attack class is the last version of Adobe Reader developed for Linux which was released in 2013. Surprisingly, the last versions of Adobe Reader for Windows and macOS and the latest versions for its successor Adobe Acrobat Reader DC for Windows and macOS are vulnerable to a very simple Signature Exclusion attack - deleting the `/ByteRange` entry or replacing its value with `null`.[13]

During the evaluation three manipulated documents were the most successful ones. Using them it was possible to successfully attack 18 applications. These three documents are the Incremental Update Abuse attack based on incremental update variant 3 created for Perfect PDF Reader, the Signature Wrapping attack based on the first approach created for Nitro Pro and test file 36 created for the second approach of Signature Wrapping.

At the end of the evaluation all applications were updated to their latest version and all successful attacks were executed again to determine if the updated applications are still vulnerable to the specific attack. All but one attacks could still be executed successfully. This means the latest versions of 30 of the 34 evaluated applications available at the time this thesis is submitted are vulnerable to critical attacks completely bypassing the protection of digital signatures applied to PDF documents.

The results of the evaluation and all identified vulnerabilities were responsibly disclosed to the applications' vendors in cooperation with the computer emergency response team ("CERT-Bund")[14] of the german "Bundesamt für Sicherheit in der Informationstechnik" (BSI) to enable them to implement updates which fix the identified vulnerabilities.

[13] According to the PDF reference v1.7 a dictionary entry whose value is `null` should be treated similar to a non present entry [2, p. 63].

[14] `https://www.bsi.bund.de/DE/Themen/Cyber-Sicherheit/Aktivitaeten/CERT-Bund/certbund_node.html`

Application	OS	Attack Class			Comments
		Signature Exclusion	Incremental Update Abuse	Signature Wrapping	
Adobe Acrobat Reader DC	Windows	●	✓	✓	Error message displayed when the visible signature is clicked.
Adobe Acrobat Reader DC	macOS	●	✓	✓	Error message displayed when the visible signature is clicked.
Adobe Reader 9	Linux	✓	✓	✓	
Adobe Reader XI	Windows	●	✓	✓	Error message displayed when the visible signature is clicked.
Adobe Reader XI	macOS	●	✓	✓	Error message displayed when the visible signature is clicked.
eXpert PDF 12 Ultimate	Windows	✓	✓	②	
Expert PDF Reader	Windows	✓	✓	②	
Foxit Reader	Windows	✓	②	②	
LibreOffice	Windows	✓	◑	✓	Incremental Update Abuse detected when certificate is trusted.
LibreOffice	Linux	✓	◑	✓	Not accomplished to trust certificate.
LibreOffice	macOS	✓	◑	✓	Not accomplished to trust certificate.
Master PDF Editor	Windows	✓	②	✓	
Master PDF Editor	Linux	✓	②	✓	
Master PDF Editor	macOS	-	-	-	Not possible to determine whether a manipulation was detected or not.
Nitro Pro	Windows	✓	◑	●	Incremental Update Abuse detected when certificate is trusted.
Nitro Reader	Windows	✓	◑	●	Incremental Update Abuse detected when certificate is trusted.
Nuance Power PDF Standard	Windows	✓	●	●	
PDF Architect 6	Windows	✓	✓	②	
PDF Editor 6 Pro	Windows	①	●	●	
PDF Editor 6 Pro	macOS	-	-	-	Not possible to determine whether a manipulation was detected or not.
PDFelement 6 Pro	Windows	①	●	●	
PDFelement 6 Pro	macOS	-	-	-	Not possible to determine whether a manipulation was detected or not.
PDF Studio 12 Pro	Windows	✓	●	●	
PDF Studio 12 Pro	Linux	✓	●	●	
PDF Studio 12 Pro	macOS	✓	●	●	
PDF Studio Viewer 2018	Windows	✓	●	●	
PDF Studio Viewer 2018	Linux	✓	●	●	
PDF Studio Viewer 2018	macOS	✓	●	●	
PDF-XChange Editor	Windows	✓	✓	②	
PDF-XChange Viewer	Windows	✓	✓	②	
Perfect PDF 10 Premium	Windows	✓	●	●	
Perfect PDF Reader	Windows	✓	●	●	
Soda PDF	Windows	✓	✓	②	
Soda PDF Desktop	Windows	✓	✓	②	
Successful Attacks		**6/34**	**19/34**	**21/34**	
		Total Signature Vulnerabilities: 30/34			

✓ Attack not successful
● Attack successful on both UI-layers
◑ Attack with limited success
- Evaluation not possible
① Attack successful on UI-layer 1
② Attack successful on UI-layer 2

Table 5.14: Evaluation results of 34 applications showing critical vulnerabilities in 30 of them.

6 Conclusion and Future Work

The final chapter summarizes and concludes the contents and results of this thesis. Additionally, an outlook to possible further evaluations in the terms of the security of PDF signatures is given.

6.1 Conclusion

The integrity, authenticity and non-repudiation protection digital signatures provide to digital documents is especially important in critical environments such as the judicial system or tax matters. Additionally, PDF is a common choice for the exchange of digital documents. Nevertheless, the security of digital signatures in PDF documents has not been in the focus of research in the past. The goal of the thesis was to fill this gap by providing the first comprehensive evaluation of PDF processing applications regarding the security of digital signatures embedded in documents.

During this thesis 34 applications for Windows, Linux and macOS were evaluated using various manipulations and attacks based on the three different attack classes Signature Exclusion, Incremental Update Abuse and Signature Wrapping. The evaluation resulted in the identification of an alarming number of critical vulnerabilities. The identified vulnerabilities allow an attacker to completely bypass the protection provided by a signature in a PDF document and change its contents arbitrarly in 30 of the 34 evaluated applications. While the vendors of the vulnerable applications have been informed during the evaluation no updated versions fixing the vulnerabilities have been released at the time this thesis is submitted.

The results of the evaluation show that it was - *and still is* - necessary to survey the security of PDF signatures to ensure they fulfill their purpose of protecting the document's contents. The mere amount of vulnerable applications - 30 of 34 - emphasizes that the applications' vendors need to pay more attention to security aspects during the signature validation and probably when processing PDF documents in general. The overall weak security of PDF signatures in common PDF processing applications is especially alarming as governmental institutions as well as private companies rely on the protection provided by digital signatures in PDF documents in various environments including the judicial system, tax matters and all sorts of legally binding

contracts. However, a first general approach to fix the identified vulnerabilities has been developed by Mladenov et al. [18]. They propose a secure signature validation algorithm which prevents the attacks presented in this thesis. Parts of this thesis contributed to the publication.

Although this thesis contains a comprehensive evaluation for three very diverse attack classes it can only be the starting point for research on the security of PDF signatures. Further attack classes and ideas need to be devised and evaluated in the future. Additionally, not only the security of signatures in PDF documents but the overall security of PDF documents should be in the focus of future research due to the high popularity of PDF documents and the criticality of the mentioned environments PDF documents are used in. This includes the confidentiality provided by encrypted PDF documents and additional features provided by PDF such as the inclusion of external resources.

The following section gives a more detailed outlook of ideas for future research on the security of PDF signatures.

6.2 Future Work

As mentioned before further research on the security of PDF signatures is necessary to ensure that digital signatures in PDF documents properly protect the document's contents. The following ideas and approaches are based on the knowledge and experiences gained during the processing of this thesis and may help to identify further vulnerabilities regarding PDF signatures in the future.

Extension of the Signature Exclusion attack class: In addition to the specific manipulations surveyed in the evaluation described in Chapter 5, other manipulations could lead to successful attacks and should be evaluated. All manipulations and manipulation ideas which have been devised during this thesis are listed in Appendix A.3.1 and can be helpful for future evaluations.

Extension of the Signature Wrapping attack class: The second Signature Wrapping approach was evaluated using 41 test files. In addition to the ideas and manipulations these files are based on, ideas for the creation of further test files are listed in Appendix A.3.2.

Using another original document to evaluate macOS applications: As described in Section 5.1 three macOS applications could not be successfully evaluated because they stated that the signature of the unaltered original document is invalid. Another original document with a signature which is successfully verified by these applications needs to be found to be able to evaluate these applications.

Hybrid attacks: The evaluation of the three attack classes Signature Exlcusion, Incremental Update Abuse and Signature Wrapping showed very different results for different applications. While some attacks could be used to attack more than half of the evaluated applications no attack was successful for all vulnerable applications. In a real-world attack an attacker might not know which application his victim uses to view PDF files. This means he cannot be sure that the attack will be successful if he makes use of only one of the manipulations evaluated in this thesis. To increase the chances of a successful attack *(as long as the victim uses any of the vulnerable applications)* a manipulated file which contains manipulations from multiple attack classes could be used. For example, a combination of the most successful test file of the second Signature Wrapping approach and the Incremental Update Abuse attack for Perfect PDF Reader could led to a successful attack against 25 of the 34 applications. Another example would be the combination of the first approach Signature Wrapping attack for Nitro Pro and the Signature Exclusion attack which removes the `/ByteRange` entry. If these attacks could be successfully combined into one manipulated document 22 applications including Adobe Acrobat Reader DC and Adobe Reader could be attacked with one file. A first attempt to create such a hybrid attack was created by combining the Incremental Update Abuse attack for Perfect PDF Reader and Signature Exclusion attack for Adobe Acrobat Reader DC. The details of this attempt can be found in Appendix A.4.

Adapt attacks to different PDF versions: For the evaluation conducted during this thesis all manipulations were applied to the same original document depicted in Appendix A.1.[1] This document is compliant to PDF version 1.4 to prevent the usage of object and XRef streams. The manipulations evaluated should be applicable to an original document compliant to newer PDF versions, like 1.7, as well. However, an evaluation must be conducted to verify that this is the case and that the identified attacks are also successful with an original document using object and XRef streams.

Adapt attacks to different types of signatures: As mentioned before, the evaluation in this thesis was based on one single document. The signature present in this document is a visible approval signature (see Section 2.2.1). Future evaluations could adapt the manipulations to original documents which contain other types of signatures, for example a certification signature, invisible signatures or multiple signatures. Especially the evaluation of an original document containing a signature compliant to PAdES would provide important results because of the relevance of PAdES in certain environments.

Attacks specific to the signature type: Certification signatures allow the author to specify which changes to a document are permitted without invalidating

[1] As described in Section 5.1 another original document had to be used to evaluate Nuance Power PDF Standard. However, this original document is compliant to PDF version 1.4, as well.

the certification signature. It might be possible to bypass this protection by transforming a certification signature into an approval signature. As the permissions are specified using a regular PDF object it might be possible to change them using an attack from the Incremental Update Abuse or Signature Wrapping attack class. According to the PDF reference approval signatures cannot be used to specify permissions. However, it might also be possible to transform an approval signature into a certification signature to enable the specification of permissions.

Attacks against object digests: The PDF reference defines two possible digests in the context of signatures: ByteRange digests and object digests. During the processing time of this thesis no PDF document which makes use of an object digest could be found. Therefore, all manipulations and attacks in this thesis were applied to signatures which use ByteRange digests. However, object digests might allow other interesting attack vectors similar to XML Signature Wrapping attacks [16] because they make use of references to the signed objects which are similar to object ids used by XML Signatures.

Exchange embedded content after signing: This idea is based on the PDF features which allow to embed external multimedia data (see "WebCapture" [2, pp. 946-961] and "File Specification" [2, p. 178-191]) or Javascript in PDF documents. Depending on the way the digest is calulated (including the external content or only the reference to it) it might be possible to modify the document's content after the signature was applied by exchanging the content of the referenced external file. The victim would not be able to detect the manipulation as the signature would be valid. Using embedded Javascript it might be possible to change the visible content of a document depending on different circumstances (e.g., the time and date when the document is opened or details of the used application or device). This could be used to manipulate a document prior to the application of a signature: A victim opens the manipulated document, reviews its contents and signs it when the content is as expected. Afterwards, the signed document is opened by another party. The Javascript displays other content than the one displayed prior to the application of the signature, but as the signed data is not changed the signature is still valid. Another idea makes use of the PDF feature to include Postscript code in PDF. Embedded Postscript might be able to change the visible content of the document depending on the interpreter used to process the code. This might allow to change the content when the document is viewed in a specific application or printed, for example.

Custom Signature Appearance: This attack class, which was initially planed to be part of this thesis, is based on the idea to manipulate the information regarding the signature visible to the victim. The idea is based on the assumption that this information is (partly) customizable and that it might be possible for an attacker to pretend the signature verification was successful. One possibility

could be to place an overlay over the visible part of the signature. However, the possibilities are limited depending on the viewer application's behavior and UI.

Attacks bases on the signer's certificate: All attacks evaluated in this thesis and described above focus on the signature details or its validation. However, the initial plan for this thesis contained an attack class called "Certificate Issues". This attack class should deal with the security in terms of the certificate and certificate chain used to verify a signature. While initial effort in this area has been part of this thesis (Appendix A.2 contains information on how to establish a trust relationship between the evaluated applications and the signer's certificate) a comprehensive evaluation was not conducted for this attack class. There might be applications which accept self-signed certificates or accept a certificate packed with a certificate chain ending in a self-signed root certificate as trusted certificates. Different details during the certificate generation might lead to different behavior in some applications (e.g., setting the "CA" or "keyUsage" extensions to different values). This would allow an attacker to simply change a document arbitrarly and sign the manipulated version using his own certificate. This certificate could contain the same information except the keys used as the original signer's certificate making it difficult for the victim to detect the attack.

Special applications not available to the public: For this thesis all PDF processing applications which were found and support digital signatures as well as provide a free version were evaluated. However, there are applications which do not provide a free version or are not available to the public at all. These non-public applications are used in critical environments. For example, the courts in "North Rhine-Westphalia" (NRW) are in the process of changing from paper records to the "elektronische Akte" (eAkte).[2] The eAkte is a collection of digital documents for a lawsuit. As PDF is a common format for digital documents it is used in the eAkte, as well.[3] The documents contained in an eAkte are important and - depending on the specific document - secured by digital signatures. The application used to manage the eAkte which is called "ergonomisch elektronischer Arbeitsplatz" (e^2A) is specifically developed for this purpose and not available to the public.[1] However, the correct validation of the signatures present in documents contained in an eAkte is crucial as the judicial system relies on the integrity and authenticity of these documents for the flawless processing of lawsuits. Therefore, these applications must be evaluated to ensure they are not vulnerable to the attacks presented in this thesis or other attacks.

[2] `https://justiz.de/elektronischer_rechtsverkehr/nordrhein-westfalen/index.php`
[3] Elektronischer-Rechtsverkehr-Verordnung vom 24. November 2017, § 5 Absatz 1 Nummer 1.

A Appendix

A.1 Example PDF Document Containing a Visible Signature

This document contains a single page with a visible signature placed on it. It is used as the original document for all manipulations and attacks evaluated in Chapter 5 except for Nuance Power PDF Standard. This application stated that the signature in the unaltered original document is invalid. Therefore, another signed document whose signature was successfully verified by the application was used for its evaluation.

Note: The second line and the contents of the streams contained in this document had to be removed because LaTeX is not able to display the binary data correctly. Additionally, the signature value (value of the `/Contents` entry in object `10 0`) had to be shortened because LaTeX does not support lines long enough.

```
%PDF-1.4
%...
1 0 obj
<<
/Type /Catalog
/Version /1.4
/Pages 2 0 R
>>
endobj
2 0 obj
<<
/Type /Pages
/Kids [3 0 R]
/Count 1
>>
endobj
3 0 obj
<<
/Type /Page
/MediaBox [0.0 0.0 612.0 792.0]
/Parent 2 0 R
/Contents 4 0 R
/Resources 5 0 R
>>
endobj
4 0 obj
<<
/Length 51
/Filter /FlateDecode
>>
```

```
stream
...
endstream
endobj
5 0 obj
<<
/Font 6 0 R
>>
endobj
6 0 obj
<<
/F1 7 0 R
>>
endobj
7 0 obj
<<
/Type /Font
/Subtype /Type1
/BaseFont /Helvetica-Bold
/Encoding /WinAnsiEncoding
>>
endobj
xref
0 8
0000000000 65535 f
0000000015 00000 n
0000000078 00000 n
0000000135 00000 n
0000000247 00000 n
0000000371 00000 n
0000000404 00000 n
0000000435 00000 n
trailer
<<
/Root 1 0 R
/ID [<1FDC264C24377F037DC7C2587F9C9AA8> <1FDC264C24377F037DC7C2587F9C9AA8>]
/Size 8
>>
startxref
537
%%EOF

1 0 obj
<<
/Type /Catalog
/Version /1.4
/Pages 2 0 R
/AcroForm <<
/Fields [8 0 R]
/SigFlags 3
/DR <<
/XObject <<
/FRM 9 0 R
>>
/ProcSet [/PDF /Text /ImageB /ImageC /ImageI]
>>
>>
>>
endobj
8 0 obj
<<
/FT /Sig
```

```
/Type /Annot
/Subtype /Widget
/F 132
/T (Signature1)
/V 10 0 R
/P 3 0 R
/Rect [0.0 777.0 45.0 792.0]
/AP <<
/N 11 0 R
>>
>>
endobj
9 0 obj
<<
/Length 52
/Type /XObject
/Subtype /Form
/Resources <<
/XObject <<
/n2 12 0 R
/n0 13 0 R
>>
/ProcSet [/PDF /Text /ImageB /ImageC /ImageI]
>>
/BBox [100.0 50.0 0.0 0.0]
/FormType 1
>>
stream
q 1 0 0 1 0 0 cm /n0 Do Q q 1 0 0 1 0 0 cm /n2 Do Q

endstream
endobj
10 0 obj
<<
/Type /Sig
/Filter /Adobe.PPKLite
/SubFilter /adbe.pkcs7.detached
/Name (Vladislav Mladenov)
/Location (Bochum)
/Reason (Security)
/M (D:20180809092110+02'00')
/Contents <308006092A864886F70D010702A0803080020101310F300D06096086480165030402010 5 ... 000>
/ByteRange [0 1633 20579 2437]
>>
endobj
3 0 obj
<<
/Type /Page
/MediaBox [0.0 0.0 612.0 792.0]
/Parent 2 0 R
/Contents 4 0 R
/Resources 5 0 R
/Annots [8 0 R]
>>
endobj
11 0 obj
<<
/Length 27
/Type /XObject
/Subtype /Form
/Resources <<
/XObject <<
```

```
/FRM 9 0 R
>>
/ProcSet [/PDF /Text /ImageB /ImageC /ImageI]
>>
/BBox [100.0 50.0 0.0 0.0]
/FormType 1
>>
stream
q 1 0 0 1 0 0 cm /FRM Do Q

endstream
endobj
12 0 obj
<<
/Length 31
/Type /XObject
/Subtype /Form
/BBox [100.0 50.0 0.0 0.0]
/Matrix [1.0 0.0 0.0 1.0 0.0 0.0]
/Resources <<
/XObject <<
/img1 14 0 R
>>
/ProcSet [/PDF /Text /ImageB /ImageC /ImageI]
>>
/FormType 1
>>
stream
q 100 0 0 50 0 0 cm /img1 Do Q

endstream
endobj
13 0 obj
<<
/Length 0
/Type /XObject
/Subtype /Form
/BBox [100.0 50.0 0.0 0.0]
/Resources <<
>>
/FormType 1
>>
stream

endstream
endobj
14 0 obj
<<
/Length 1064
/Type /XObject
/Subtype /Image
/Filter /FlateDecode
/BitsPerComponent 8
/Width 90
/Height 30
/ColorSpace /DeviceGray
>>
stream
...
endstream
endobj
xref
```

```
0 2
0000000000 65535 f
0000000838 00000 n
3 1
0000020638 00000 n
8 7
0000001027 00000 n
0000001178 00000 n
0000001451 00000 n
0000020766 00000 n
0000021004 00000 n
0000021282 00000 n
0000021420 00000 n
trailer
<<
/Root 1 0 R
/ID [<1FDC264C24377F037DC7C2587F9C9AA8> <FD7ABFD8D7817F76AF29426E3A75B15A>]
/Size 15
/Prev 537
>>
startxref
22656
%%EOF
```

A.2 Steps Needed to Establish a Trust Relationship between the Signer's Certificate and the Applications

In the following, the steps needed to establish a trust relationship between the signer's certificate and each application are described. While 11 applications use the certificate store of the operating system, 7 maintain their own trust management and 4 applications trust every certificate in general. During this thesis it was not accomplished to establish the trust relationship for the Linux and macOS version of one application and three further macOS applications.

Adobe Acrobat Reader DC: The application does not use the trusted certificate authories of the operating system but includes an own trust management in its UI. After opening a signed document it is possible to trust the signer's certificate by opening the "Signature Panel" and the "Certificate Details". Afterwards the tab "Trust" allows to establish the trust relationship using the "Add to Trusted Certificates..." button.

Adobe Reader 9: The application does not use the trusted certificate authories of the operating system but includes an own trust management in its UI. The steps to be executed are the same as for Adobe Acrobat Reader DC.

Adobe Reader XI: The application does not use the trusted certificate authories of the operating system but includes an own trust management in its UI. The steps to be executed are the same as for Adobe Acrobat Reader DC.

eXpert PDF 12 Ultimate: The application makes use of the trusted certificate authorities of the operating system. For the Windows operating system this means the signer's certificate must be imported to the "Trusted Root Certification Authories" certificate store of the local machine. If the certificate has one of the supported formats[1] it can be imported by simply double-clicking it and following the UI instructions for "Install Certificate ...".

Expert PDF Reader: The application seems to trust every certificate in general and no UI option could be found to inspect the validity of the signer's certificate or to modify the trust relationship.

Foxit Reader: The application makes use of the trusted certificate authorities of the operating system. Therefore, the steps to be executed are the same as for eXpert PDF 12 Ultimate.

LibreOffice: The application makes use of the trusted certificate authorities of the operating system. Therefore, the steps to be executed are the same as for eXpert PDF 12 Ultimate for the Windows version. For the Linux and macOS version of the application it was not accomplished to establish the trust relationship during this thesis.

Master PDF Editor: The application makes use of the trusted certificate authorities of the operating system. Therefore, the steps to be executed are the same as for eXpert PDF 12 Ultimate for the Windows version. The steps to add the certificate to the trusted certificate authorities for Linux depend on the specific distribution. Additionally, it is possible to establish the trust relationship using the application's own trust management. After opening a signed document it is possible to trust the signer's certificate by opening the "Signature Panel" and executing a double click on the signature. Afterwards the button "Info" opens the details of the certificate; the button "Add to Trusted Identities" establishes the trust relationship. The macOS version of the application states that the signature present in the original document is invalid. It was not possible to determine whether the trust relationship between the signer's certificate and the application was established successfully.

Nitro Pro: The application does not use the trusted certificate authories of the operating system but includes an own trust management in its UI. After opening a signed document it is possible to trust the signer's certificate by opening the properties of the signature (e.g., by clicking on the visible signature or opening the signature panel and right-clicking the first line of the information) and clicking on "Add to Trusted Contacts".

[1] Windows supports DER-/PEM-encoded certificates with file ending ".cer" or ".crt", PKCS#12-encoded certificates with file ending ".p12" or PFX files with file ending ".pfx".

Nitro Reader: The application does not use the trusted certificate authories of the operating system but includes an own trust management in its UI. The steps to be executed are the same as for Nitro Pro.

Nuance Power PDF Standard: The application does not use the trusted certificate authories of the operating system but includes an own trust management in its UI. After opening a signed document it is possible to trust the signer's certificate by opening the properties of the signature (e.g., by clicking on the visible signature or opening the signature panel and right-clicking the first line of the information), clicking on "Verify Identity" and clicking on "Add as a Trusted Root".

PDF Architect 6: The application makes use of the trusted certificate authorities of the operating system. Therefore, the steps to be executed are the same as for eXpert PDF 12 Ultimate.

PDF Editor 6 Pro: The application makes use of the trusted certificate authorities of the operating system. Therefore, the steps to be executed are the same as for eXpert PDF 12 Ultimate for the Windows version. However, it is also sufficient to import the certificate to the "Intermediate Certificate Authorities" certificate store instead of the "Trusted Root Certificate Authories" certificate store. The macOS version of the application states that the signature present in the original document is invalid. It was not possible to determine whether the trust relationship between the signer's certificate and the application was established successfully.

PDFelement 6 Pro: The application makes use of the trusted certificate authorities of the operating system. Therefore, the steps to be executed are the same as for eXpert PDF 12 Ultimate for the Windows version. However, it is also sufficient to import the certificate to the "Intermediate Certificate Authorities" certificate store instead of the "Trusted Root Certification Authories" certificate store. The macOS version of the application states that the signature present in the original document is invalid. It was not possible to determine whether the trust relationship between the signer's certificate and the application was established successfully.

PDF Studio 12 Pro: The application makes use of the trusted certificate authorities of the operating system. Therefore, the steps to be executed are the same as for eXpert PDF 12 Ultimate for the Windows version. For macOS executing a double click on the certificate and confirming that it should be added to the "Keychain" is sufficient to add the certificate to the trusted system certificates. The steps to add the certificate to the trusted certificate authorities for Linux depend on the specific distribution. Additionally, it is possible to establish the trust relationship using the application's own trust management. After opening a signed document it is possible to trust the signer's certificate by opening the "Signature Panel" and the "Details" of the signature. Afterwards

the button “Details” opens the “Certificate Details” which allow to establish the trust relationship using the “Trust Certificate” button.

PDF Studio Viewer 2018: The application makes use of the trusted certificate authorities of the operating system. Therefore, the steps to be executed are the same as for eXpert PDF 12 Ultimate for the Windows version. For macOS executing a double click on the certificate and confirming that it should be added to the “Keychain” is sufficient to add the certificate to the trusted system certificates. The steps to add the certificate to the trusted certificate authorities for Linux depend on the specific distribution. Additionally, it is possible to establish the trust relationship using the application’s own trust management. After opening a signed document it is possible to trust the signer’s certificate by opening the “Signature Panel” and the “Details” of the signature. Afterwards the button “Details” opens the “Certificate Details” which allow to establish the trust relationship using the “Trust Certificate” button.

PDF-XChange Editor: The application seems to trust every certificate in general and no UI option could be found to inspect the validity of the signer’s certificate or to modify the trust relationship.

PDF-XChange Viewer: The application seems to trust every certificate in general and no UI option could be found to inspect the validity of the signer’s certificate or to modify the trust relationship.

Perfect PDF 10 Premium: The application does not use the trusted certificate authories of the operating system but includes an own trust management in its UI. After opening a signed document it is possible to trust the signer’s certificate by clicking on the visible signature and clicking on “Add Signer to Trusted Identities”.

Perfect PDF Reader: The application seems to trust every certificate in general and no UI option could be found to inspect the validity of the signer’s certificate or to modify the trust relationship.

Soda PDF: The application makes use of the trusted certificate authorities of the operating system. Therefore, the steps to be executed are the same as for eXpert PDF 12 Ultimate.

Soda PDF Desktop: The application makes use of the trusted certificate authorities of the operating system. Therefore, the steps to be executed are the same as for eXpert PDF 12 Ultimate.

A.3 Complete Lists of Manipulations and Manipulation Ideas Devised for Different Attack Classes

A.3.1 Signature Exclusion

The following two lists contain all specific manipulations and further ideas which have been devised in the context of the Signature Exclusion attack class during this thesis. The Python scripts developed during this thesis (see Chapter 4) implement 73 of the described manipulations. These manipulation are written in italic. The 24 manipulations evaluated during this thesis (see Chapter 5) are written in bold, additionally.

Strategy 1: Remove parts which are essential for the verification of the signature itself.

- Manipulate the whole signature dictionary:
  - Replace it with an empty dictionary.
  - Replace it with a dictionary which only contains a `/Type` entry.
  - Remove the signature dictionary.
- Manipulate the `/Contents` entry of the signature dictionary:
  - ***Replace its value with an empty hexadecimal string.***
  - ***Replace its value with a hexadecimal string containing a null byte.***
  - *Replace its value with an invalid hexadecimal string, for example* `<XX>`*.*
  - ***Replace its value with the null object.***
  - *Replace its value with a reference to a non-existing indirect object.*
  - *Replace its value with a reference to the object with object number 0.*
  - ***Remove its value.***
  - ***Remove the entry.***
- Manipulate the `/ByteRange` entry of the signature dictionary:
  - ***Replace its value with an empty array.***
  - ***Replace its value with an array containing only zeros.***
  - *Replace its value with an array containing low numbers, for example* `[0 5 6 10]`*.*

  - *Replace its value with an array containing two identical integer pairs, for example `[0 1500 0 1500]`.*
  - ***Replace its value with an array containing a negative value at position 2.***
  - *Replace its value with an array containing a negative value at position 1, 3 or 4.*
  - *Replace its value with an array containing four negative values.*
  - ***Replace its value with an array containing overlapping byte ranges.***
  - ***Replace its value with an array containing a byte range outside of the document.***
  - ***Replace its value with an array containing only one integer.***
  - *Replace its value with an array containing two, three or five integers.*
  - ***Replace its value with the null object.***
  - *Replace its value with a reference to a non-existing indirect object.*
  - *Replace its value with a reference to the object with object number 0.*
  - ***Remove its value.***
  - ***Remove the entry.***
- Manipulate the `/Contents` and `/ByteRange` entries of the signature dictionary:
  - Replace both their values with invalid values mentioned above.
  - Replace both their values with the null object.
  - Replace both their values with non-existing name objects.
  - Remove both their values.
  - Remove both entries.
- Manipulate the `/Filter` entry of the signature dictionary:
  - *Replace its value with another existing name object stating a valid filter.*
  - Replace its value with another existing name object stating anything but a filter or subfilter.
  - *Replace its value with a non-existing name object.*
  - *Replace its value with an arbitrary string.*
  - *Replace its value with the null object.*

  - *Replace its value with a reference to a non-existing indirect object.*
  - *Replace its value with a reference to the object with object number 0.*
  - *Remove its value.*
  - *Remove the entry.*

- Manipulate the `/SubFilter` entry of the signature dictionary:
  - *Replace its value with another existing name object stating a valid subfilter.*
  - Replace its value with another existing name object stating anything but a filter or subfilter.
  - ***Replace its value with a non-existing name object.***
  - *Replace its value with an arbitrary string.*
  - ***Replace its value with the null object.***
  - *Replace its value with a reference to a non-existing indirect object.*
  - *Replace its value with a reference to the object with object number 0.*
  - ***Remove its value.***
  - ***Remove the entry.***

- Manipulate the `/Filter` and `/SubFilter` entries of the signature dictionary:
  - *Exchange their values.*
  - *Replace both their values with the null object.*
  - *Replace both their values with non-existing name objects.*
  - *Remove both their values.*
  - *Remove both entries.*

- Manipulate the `/Reference` entry[2] of the signature dictionary:
  - Add the entry if it is not present and set its value to an array containing a reference to a non-existing indirect object or replace its value with it.
  - Add the entry if it is not present and set its value to an array containing a reference to the object with object number 0 or replace its value with it.
  - Add the entry if it is not present and set its value to anything other than an array containing object references or replace its value with it.

[2]This entry is not mandatory. It contains an array with references to "Signature Reference Dictionaries" if present [2, pp. 725-728].

  - Add the entry if it is not present and set its value to the null object or *replace its value with it.*
  - *Remove its value if the entry is present.*
  - *Remove the entry if it is present.*
- Manipulate other entries of the signature dictionary.
- Manipulate the whole signature reference dictionary[3]:
  - Replace the signature reference dictionary with an empty dictionary.
  - Replace the signature reference dictionary with a dictionary which only contains a `/Type` entry.
  - Remove the signature reference dictionary.
- Manipulate the `/DigestMethod` entry of the signature reference dictionary:
  - Replace its value with an existing name object stating another algorithm.
  - Replace its value with an existing name object stating anything but an algorithm.
  - Replace its value with a non-existing name object.
  - *Replace its value with the null object.*
  - *Remove its value.*
  - *Remove the entry.*
- Manipulate the `/TransformMethod` entry of the signature reference dictionary:
  - Replace its value with an existing name object stating another transform method.
  - Replace its value with an existing name object stating anything but a transform method.
  - Replace its value with a non-existing name object.
  - *Replace its value with the null object.*
  - *Remove its value.*
  - *Remove the entry.*
- Manipulate the `/TransformParams` entry of the signature reference dictionary:
  - Replace its value with a reference to a non-existing indirect object.

[3]Signature reference dictionaries are not mandatory. If they are present in a document they are reference in the `/Reference` entry in the signature dictionary [2, pp. 725-728].

  - Replace its value with a reference to the object with object number 0.
  - Replace its value with the null object.
  - Remove its value.
  - Remove the entry.
- Manipulate the whole transform parameters dictionary[4]:
  - Replace the transform parameters dictionary with an empty dictionary.
  - Replace the transform parameters dictionary with a dictionary which only contains a `/Type` entry.
  - Remove the transform parameters dictionary.
- Manipulate the `/V` entry of the transform parameters dictionary:
  - Replace its value with the null object.
  - Remove its value.
  - Remove the entry.
- Manipulate the `/P` entry of the transform parameters dictionary:
  - Replace its value with the null object.
  - Remove its value.
  - Remove the entry.
- Test different combinations of the manipulations mentioned above.

Strategy 2: Remove references to the signature.

- Manipulate the `/Perms` entry[5] of the document catalog:
  - Replace its value with a reference to a non-existing indirect object.
  - Replace its value with a reference to the object with object number 0.
  - *Replace its value with the null object.*
  - *Remove its value.*
  - *Remove the entry.*

[4] A transform parameter dictionary is not mandatory. If it is present in a document it is the value of the `/TransformParams` entry in the signature reference dictionary [2, p. 730].

[5] The `/Perms` entry is only present in the updated document catalog if the document contains a certification signature [2, p. 726]. It references the permissions dictionary [2, p. 142].

- Manipulate the whole permissions dictionary[6]:
  - Replace it with an empty dictionary.
  - Replace it with a dictionary containing only the null object.
  - Remove the permissions dictionary.
- Manipulate the `/DocMDP` entry of the permissions dictionary:
  - *Replace its value with the null object.*
  - *Remove its value.*
  - *Remove the entry.*
- Manipulate the `/AcroForm` entry of the document catalog:
  - Replace its value with an empty dictionary.
  - Replace it with a dictionary containing only the null object.
  - Replace its value with the null object.
  - Remove its value
  - Remove the entry.
- Manipulate the `/Fields` entry inside of the dictionary which is the value of the `/AcroForm` entry:
  - Replace its value with an array containing a reference to a non-existing indirect object.
  - Replace its value with an array containing a reference to the object with object number 0.
  - Replace its value with an empty array.
  - *Replace its value with the null object.*
  - Remove the reference to the form field dictionary of the signature field from the entry's value.
  - *Remove its value.*
  - *Remove the entry.*
- Manipulate the whole form field dictionary of the signature field referenced in the `/Fields` array:
  - Replace it with an empty dictionary.

[6] A permissions dictionary is only used when the document contains a certification signature which sets the modification permissions for other users [2, p. 726]. If it is present in a document it is referenced by the `/Perms` entry of the document catalog [2, p. 142].

  - Replace it with a dictionary containing the null object.
- Manipulate the `/V` entry[7] of the form field dictionary of the signature field:
  - Replace its value with a reference to a non-existing indirect object.
  - Replace its value with a reference to the object with object number 0.
  - ***Replace its value with the null object.***
  - ***Remove its value.***
  - ***Remove the entry.***
- Manipulate the `/P` entry[8] of the form field dictionary of the signature field:
  - Replace its value with a reference to a non-existing indirect object.
  - Replace its value with a reference to the object with object number 0.
  - ***Replace its value with the null object.***
  - ***Remove its value.***
  - ***Remove the entry.***
- Test different combinations of the manipulations mentioned above.

A.3.2 Signature Wrapping

This section contains the details of the 41 manipulations devised and evaluated as the second approach of the Signature Wrapping attack class. Additionally, a list of further ideas for manipulations is given.

Details of the 41 different manipulations evaluated as part of the second approach of the Signature Wrapping attack class:

No.	Wrapping	Description
1	None	Connected two signed byte ranges at the end of the file, replaced visible content and adjusted `/ByteRange`: The first byte range starts at byte 0 and has size 0. The second byte range contains the whole signed data.

[7] The `/V` entry references the value of the form field [2, p. 676]. In the case of a signature form field it references the signature dictionary [2, p. 695].

[8] The `/P` entry references the page object the annotation of the form field is associated with [2, p. 606].

2	Dictionary	Based on 1. Wrapped signed data in direct dictionary object.
3	Dictionary	Based on 2. Added `%%EOF` behind wrapping object.
4	Indirect dictionary object	Based on 1. Wrapped signed data in indirect dictionary object.
5	Indirect dictionary object	Based on 4. Added `%%EOF` behind wrapping object.
6	Indirect dictionary object	Based on 4. Added XRef entry for wrapping object and adjusted `/ByteRange`.
7	Indirect dictionary object	Based on 6. Added `%%EOF` behind wrapping object.
8	Indirect dictionary object	Based on 6. Moved wrapping object in front of last XRef section and trailer and adjusted `startxref` value of last trailer, XRef entry of wrapping object and `/ByteRange`.
9	Indirect dictionary object	Based on 6. Moved wrapping object in front of last XRef section and trailer, deleted zeros in `/Contents` to make sure `startxref` is correct and adjusted all XRef entries and `/ByteRange`.
10	Stream	Based on 1. Wrapped signed data in direct stream object.
11	Stream	Based on 10. Added `%%EOF` behind wrapping object.
12	Indirect stream object	Based on 1. Wrapped signed data in indirect stream object.
13	Indirect stream object	Based on 12. Added `%%EOF` behind wrapping object.
14	Indirect stream object	Based on 12. Added XRef entry for wrapping object and adjusted `/ByteRange`.
15	Indirect stream object	Based on 14. Added `%%EOF` behind wrapping object.
16	Indirect stream object	Based on 14. Moved wrapping object in front of last XRef section and trailer and adjusted `startxref` value of last trailer, XRef entry of wrapping object and `/ByteRange`.

17	Indirect stream object	Based on 14. Moved wrapping object in front of last XRef section and trailer, deleted zeros in `/Contents` to make sure `startxref` is correct and adjusted all XRef entries and `/ByteRange`.
18	Deflated indirect stream object	Based on 1. Wrapped signed data in indirect stream object and added `/Filter /FlateDecode` to the stream dictionary.
19	Deflated indirect stream object	Based on 18. Added `%%EOF` behind wrapping object.
20	Deflated indirect stream object	Based on 18. Added XRef entry for wrapping object and adjusted `/ByteRange`.
21	Deflated indirect stream object	Based on 20. Added `%%EOF` behind wrapping object.
22	Deflated indirect stream object	Based on 20. Moved wrapping object in front of last XRef section and trailer and adjusted `startxref` value of last trailer, XRef entry of wrapping object and `/ByteRange`.
23	Deflated indirect stream object	Based on 20. Moved wrapping object in front of last XRef section and trailer, deleted zeros in `/Contents` to make sure `startxref` is correct and adjusted all XRef entries and `/ByteRange`.
24	XML (indirect stream object)	Based on 1. Wrapped signed data in indirect stream object and added `/Type Metadata` and `/Subtype XML` to the stream dictionary.
25	XML (indirect stream object)	Based on 24. Added `%%EOF` behind wrapping object.
26	XML (indirect stream object)	Based on 24. Added XRef entry for wrapping object and adjusted `/ByteRange`.
27	XML (indirect stream object)	Based on 26. Added `%%EOF` behind wrapping object.
28	XML (indirect stream object)	Based on 26. Moved wrapping object in front of last XRef section and trailer and adjusted `startxref` value of last trailer, XRef entry of wrapping object and `/ByteRange`.

29	XML (indirect stream object)	Based on 26. Moved wrapping object in front of last XRef section and trailer, deleted zeros in `/Contents` to make sure `startxref` is correct and adjusted all XRef entries and `/ByteRange`.
30	CDATA (indirect stream object)	Based on 24. Wrapped signed data in CDATA section inside of stream, additionally, and adjusted `/ByteRange`.
31	CDATA (indirect stream object)	Based on 30. Added `%%EOF` behind wrapping object.
32	CDATA (indirect stream object)	Based on 30. Added XRef entry for wrapping object and adjusted `/ByteRange`.
33	CDATA (indirect stream object)	Based on 32. Added `%%EOF` behind wrapping object.
34	CDATA (indirect stream object)	Based on 32. Moved wrapping object in front of last XRef section and trailer and adjusted `startxref` value of last trailer, XRef entry of wrapping object and `/ByteRange`.
35	CDATA (indirect stream object)	Based on 32. Moved wrapping object in front of last XRef section and trailer, deleted zeros in `/Contents` to make sure `startxref` is correct and adjusted all XRef entries and `/ByteRange`.
36	None	Based on 1. Adjusted `/ByteRange`: The first byte range is the first 5 bytes of the file and the second byte range starts 5 bytes later and is 5 bytes shorter.
37	Indirect stream object	Based on 12. Adjusted `/ByteRange`: The first byte range is the first 5 bytes of the file and the second byte range starts 5 bytes later and is 5 bytes shorter.
38	Indirect stream object	Based on 13. Adjusted `/ByteRange`: The first byte range is the first 5 bytes of the file and the second byte range starts 5 bytes later and is 5 bytes shorter.
39	Indirect stream object	Based on 14. Adjusted `/ByteRange`: The first byte range is the first 5 bytes of the file and the second byte range starts 5 bytes later and is 5 bytes shorter.

40	Indirect stream object	Based on 15. Adjusted `/ByteRange`: The first byte range is the first 5 bytes of the file and the second byte range starts 5 bytes later and is 5 bytes shorter.
41	None	Based on 1. Adjusted `/ByteRange`: The first byte range starts at the beginning of the signed data and ends behind the `/Contents` entry. The second byte range starts right after the first one and ends at the end of the signed data.

Further manipulation ideas:

- As arrays can contain any kinds of objects and can have a mixed structure it is possible to use a direct or indirect array object as the wrapping for the signed data. This allows to create 8 further test files using the same structures as test files 2-9.
- The `/ByteRange` entry specifying the signed data can be set to `[0 0 X -Y]` or `[0 5 X -(Y-5)]` with X being the ending byte offset of the signed data and Y being its length. Due to the negative length the signed data is referenced correctly and this might be usable for attacks against applications which do not check if the `/ByteRange` entry contains a negative value.
- Some applications might display an error or recognize the manipulation because the wrapped signed data is present in the document but never used. An initial approach to prevent this was to add an XRef entry for the wrapping object to an XRef section. This idea could be extended by actually "using" the wrapping object somewhere in the document. For example, if the wrapping is a metadata stream it could be referenced in an updated document catalog. However, this might result in other errors because the signed data is not a valid metadata stream.
- In order to "hide" the signed data from the application's content processing logic it might be possible to use a transformation which compresses the signed data. The transformation information needed to decompress the signed data prior to the signature validation could be added to the signature dictionary.
- The following idea is based on the observations made for eXpert PDF 12 Ultimate, PDF Architect 6, Soda PDF and Soda PDF Desktop regarding the different "Revisions" (see Section 5.3.3): If it is not possible to apply manipulations undetected it might be possible to "confuse" applications to display the original content when the document is opened and display the manipulated content when the "View Signed Version" option is selected. This would allow attacks when another attacker model is defined as the one used

in this thesis. The used attacker model classifies attacks as not successful if the UI-layer which contains the first information regarding the signature states that the document has been modified after the signature was applied.

A.4 First Attempt to Create a Hybrid Attack Based on Manipulations from Two Attack Classes

The two applications Adobe Acrobat Reader DC and its predecessor Adobe Reader XI are only vulnerable to two attacks from the Signature Exclusion attack class. These attacks are not successful for any other application. As Adobe Acrobat Reader DC is probably the most common PDF processing application surveyed in this thesis it would be desirable to combine one of the successful attacks with an attack from another attack class. This could led to a manipulated document usable for attacks against multiple applications including the most common one.

As a first attempt the Signature Wrapping manipulation #12 (replacing the value of the `/ByteRange` entry with `null`) was combined with the Incremental Update Abuse attack created for Perfect PDF Reader. This attack is based on incremental update variant 3 and could be used to successfully attack 18 applications during the evaluation. This attack is a reasonable choice because incremental update variant 3 is almost similar to a regular incremental update. Regular incremental updates are used to manipulate the visible content of a document after the application of the Signature Wrapping manipulation during the attack against Adobe Acrobat Reader DC.

In order to combine these two attacks the following steps need to be applied to the original document:

1. Append the manipulated objects changing the visible content of the document to the end of the file.
2. Append a copy of the signature dictionary of the original document to the end of the file.
3. Replace the value of the `/ByteRange` entry in the signature dictionary copy with `null`. This is the Signature Exclusion part of the hybrid attack.
4. Append a copy of the last XRef section to the end of the file. This is the Incremental Update Abuse part of the hybrid attack.
5. Add XRef entries for all manipulated objects to the copied XRef section. However, the XRef entry referencing the signature dictionary must not be updated and still reference the unaltered original signature dictionary instead of the manipulated copy. Otherwise all applications relying on XRef information will use the manipulated copy and will not be able to verify the signature correctly.

6. Append a copy of the last trailer of the file to the end of the file.
7. Update the `/Prev` entry of the copied trailer to reference the last XRef section of the original file and update `startxref` to reference the newly added XRef section.

The resulting file was opened in all applications which are vulnerable to the Incremental Update Abuse attack created for Perfect PDF Reader, Adobe Acrobat Reader DC and Adobe Reader XI. While the hybrid attack was successful for both Adobe Acrobat Reader DC and Adobe Reader XI, it was not successful for all applications vulnerable to the original Perfect PDF Reader attack. Nevertheless, it was still successful for PDF Studio 12 Pro, PDF Studio Viewer 2018, Perfect PDF 10 and Perfect PDF Reader and still limited successful for Nitro Pro and Nitro Reader.

The hybrid attack is successful for two types of applications: First, all applications which use the last version of the signature dictionary independently of the XRef information and are vulnerable to the Signature Exclusion manipulation. Second, all applications which use the version of the signature dictionary referenced by the XRef section and are vulnerable to the Incremental Update Abuse manipulation.

Although it was not possible to increase the number of total applications vulnerable to a single manipulated file this hybrid attack was still a successful first attempt. It proves that it is possible to combine two attack classes for a successful attack against applications which cannot be attacked using the same attack class.

A.5 Tables Containing Complete Results of the Evaluation

The following three tables contain the results of the evaluation described in Chapter 5. Each table is focused on one of the three attack classes explained in Chapter 3 and uses the following symbols:

- ✓ Attack not successful
- ● Attack successful on both UI-layers
- ① Attack successful on UI-layer 1
- ② Attack successful on UI-layer 2
- ◑ Attack with limited success
- \- Evaluation not possible

Note: The results marked with an asterisk were not produced by the author of this thesis but his advisors. Their results are presented in this thesis for the sake of a complete and comprehensive evaluation.

Signature Exclusion

OS	Application	Entry No.	Manipulation																						
			/Contents					/ByteRange								/SubFilter				/V			/P		
			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Windows	Adobe Acrobat Reader DC		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	●	✓	●	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	Adobe Reader XI		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	●	✓	●	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	eXpert PDF 12 Ultimate		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	Expert PDF Reader		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	Foxit Reader		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	LibreOffice		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	Master PDF Editor		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	Nitro Pro		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	Nitro Reader		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	Nuance Power PDF Standard		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	PDF Architect 6		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	PDF Editor 6 Pro		①	①	①	①	①	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	PDFelement 6 Pro		①	①	①	①	①	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	PDF Studio 12 Pro		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	PDF Studio Viewer 2018		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	PDF-XChange Editor		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	PDF-XChange Viewer		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	Perfect PDF 10 Premium		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	Perfect PDF Reader		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	Soda PDF		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	Soda PDF Desktop		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Linux	Adobe Reader 9		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Linux	LibreOffice		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Linux	Master PDF Editor		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Linux	PDF Studio 12 Pro		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Linux	PDF Studio Viewer 2018		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
macOS	Adobe Acrobat Reader DC		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	●	✓	●	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
macOS	Adobe Reader XI		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	●	✓	●	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
macOS	LibreOffice		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
macOS	Master PDF Editor		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
macOS	PDF Editor 6 Pro		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
macOS	PDFelement 6 Pro		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
macOS	PDF Studio 12 Pro		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
macOS	PDF Studio Viewer 2018		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Successful Attacks		**2**	**2**	**2**	**2**	**2**	**0**	**0**	**0**	**0**	**0**	**0**	**4**	**0**	**4**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
			Total: 6/34																						

Incremental Update Abuse

		Variant			
OS	**Application**	**1**	**2**	**3**	**4**
Windows	Adobe Acrobat Reader DC	✓	✓	✓	✓
Windows	Adobe Reader XI	✓	✓	✓	✓
Windows	eXpert PDF 12 Ultimate	✓	✓	✓	✓
Windows	Expert PDF Reader	✓	✓	✓	✓
Windows	Foxit Reader	✓	✓	✓	②*
Windows	LibreOffice	✓	✓	◑	✓
Windows	Master PDF Editor	✓	②	②	②
Windows	Nitro Pro	✓	◑	◑	✓
Windows	Nitro Reader	✓	◑	◑	✓
Windows	Nuance Power PDF Standard	●	●	●	✓
Windows	PDF Architect 6	✓	✓	✓	✓
Windows	PDF Editor 6 Pro	✓	●	●	●
Windows	PDFelement 6 Pro	✓	●	●	●
Windows	PDF Studio 12 Pro	●*	●	●	●
Windows	PDF Studio Viewer 2018	●*	●	●	●
Windows	PDF-XChange Editor	✓	✓	✓	✓
Windows	PDF-XChange Viewer	✓	✓	✓	✓
Windows	Perfect PDF 10 Premium	●	●	●	✓
Windows	Perfect PDF Reader	✓	✓	●	✓
Windows	Soda PDF	✓	✓	✓	✓
Windows	Soda PDF Desktop	✓	✓	✓	✓
Linux	Adobe Reader 9	✓	✓	✓	✓
Linux	LibreOffice	✓	✓	◑	✓
Linux	Master PDF Editor	✓	②	②	②
Linux	PDF Studio 12 Pro	●*	●	●	●
Linux	PDF Studio Viewer 2018	●*	●	●	●
macOS	Adobe Acrobat Reader DC	✓	✓	✓	✓
macOS	Adobe Reader XI	✓	✓	✓	✓
macOS	LibreOffice	✓	✓	◑	✓
macOS	Master PDF Editor	-	-	-	-
macOS	PDF Editor 6 Pro	-	-	-	-
macOS	PDFelement 6 Pro	-	-	-	-
macOS	PDF Studio 12 Pro	●*	●	●	●
macOS	PDF Studio Viewer 2018	●*	●	●	●
	Successful Attacks	**8**	**14**	**18**	**11**
		Total: 18/34			

Signature Wrapping (Part 1)

		First Approach	Second Approach															
OS	**Application**		**Wrapping**	**None**	**Dictionary**		**Indirect Dictionary Object**						**Stream**		**Indirect Stream Object**					
			No.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
Windows	Adobe Acrobat Reader DC	✓*		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	Adobe Reader XI	✓*		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	eXpert PDF 12 Ultimate	②		②	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	Expert PDF Reader	②		②	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	Foxit Reader	②*		②	②	②	②	②	②	②	✓	✓	②	②	②	②	②	②	②	②
Windows	LibreOffice	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	Master PDF Editor	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	Nitro Pro	●		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	Nitro Reader	●		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	Nuance Power PDF Standard	●		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	PDF Architect 6	②		②	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	PDF Editor 6 Pro	●		●	●	●	●	●	●	●	✓	✓	●	●	●	●	●	●	●	●
Windows	PDFelement 6 Pro	●		●	●	●	●	●	●	●	✓	✓	●	●	●	●	●	●	●	●
Windows	PDF Studio 12 Pro	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Windows	PDF Studio Viewer 2018	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Windows	PDF-XChange Editor	②		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	PDF-XChange Viewer	②		②	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	Perfect PDF 10 Premium	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Windows	Perfect PDF Reader	●		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	Soda PDF	②		②	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	Soda PDF Desktop	②*		②	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Linux	Adobe Reader 9	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Linux	LibreOffice	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Linux	Master PDF Editor	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Linux	PDF Studio 12 Pro	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Linux	PDF Studio Viewer 2018	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
macOS	Adobe Acrobat Reader DC	✓*		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
macOS	Adobe Reader XI	✓*		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
macOS	LibreOffice	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
macOS	Master PDF Editor	-		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
macOS	PDF Editor 6 Pro	-		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
macOS	PDFelement 6 Pro	-		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
macOS	PDF Studio 12 Pro	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
macOS	PDF Studio Viewer 2018	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
	Successful Attacks	**Total: 21/34**		**16**	**10**	**10**	**10**	**10**	**10**	**10**	**7**	**7**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**
				Total: 16/34																

Signature Wrapping (Part 2)

OS	Application	Second Approach																					
	Wrapping	Deflated Stream Object						XML (Stream Object)						CDATA (Stream Object)						None					None
	Manipulation No.	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
Windows	Adobe Acrobat Reader DC	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	Adobe Reader XI	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	eXpert PDF 12 Ultimate	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	②	✓	✓	✓	✓	②
Windows	Expert PDF Reader	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	②	✓	✓	✓	✓	②
Windows	Foxit Reader	②	②	②	②	②	②	②	②	②	②	✓	✓	②	②	②	②	✓	✓	②	②	②	②	②	✓
Windows	LibreOffice	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	Master PDF Editor	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	Nitro Pro	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	Nitro Reader	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	Nuance Power PDF Standard	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	PDF Architect 6	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	②	✓	✓	✓	✓	②
Windows	PDF Editor 6 Pro	●	●	●	●	●	●	●	●	●	●	✓	✓	●	●	●	●	✓	✓	●	●	●	●	●	●
Windows	PDFelement 6 Pro	●	●	●	●	●	●	●	●	●	●	✓	✓	●	●	●	●	✓	✓	●	●	●	●	●	●
Windows	PDF Studio 12 Pro	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Windows	PDF Studio Viewer 2018	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Windows	PDF-XChange Editor	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	②	✓	✓	✓	✓	②
Windows	PDF-XChange Viewer	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	②	✓	✓	✓	✓	②
Windows	Perfect PDF 10 Premium	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Windows	Perfect PDF Reader	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	●	●	●	●	●	●
Windows	Soda PDF	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	②	✓	✓	✓	✓	②
Windows	Soda PDF Desktop	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	②	✓	✓	✓	✓	②
Linux	Adobe Reader 9	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Linux	LibreOffice	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Linux	Master PDF Editor	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Linux	PDF Studio 12 Pro	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Linux	PDF Studio Viewer 2018	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
macOS	Adobe Acrobat Reader DC	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
macOS	Adobe Reader XI	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
macOS	LibreOffice	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
macOS	Master PDF Editor	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
macOS	PDF Editor 6 Pro	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
macOS	PDFelement 6 Pro	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
macOS	PDF Studio 12 Pro	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
macOS	PDF Studio Viewer 2018	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
	Successful Attacks	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**7**	**7**	**10**	**10**	**10**	**10**	**7**	**7**	**18**	**11**	**11**	**11**	**11**	**17**
		Total: 18/34																							

List of Figures

List of Tables

List of Listings

Bibliography

[1] PDF Reference and Adobe Extensions to the PDF Specification, 2018. URL `https://www.adobe.com/devnet/pdf/pdf_reference.html`. Accessed on 15.08.2018.

[2] Adobe Systems Incorporated. PDF Reference — Adobe Portable Document Format, Version 1.7, November 2006.

[3] Adobe Systems Incorporated. Digital Signature Appearances, October 2006. URL `https://www.adobe.com/content/dam/acom/en/devnet/acrobat/pdfs/PPKAppearances.pdf`. Accessed on 15.08.2018.

[4] Adobe Systems Incorporated. Digital Signatures in Acrobat, May 2007. URL `https://www.adobe.com/content/dam/acom/en/devnet/acrobat/pdfs/digisig_in_acrobat.pdf`. Accessed on 15.08.2018.

[5] Adobe Systems Incorporated. Signature Validation Guide, 2009. URL `https://www.adobe.com/content/dam/acom/en/devnet/reader/pdfs/acrobat_sig_validation_cheat_sheet9_1.pdf`. Accessed on 15.08.2018.

[6] Adobe Systems Incorporated. Digital Signatures in a PDF, 2012. URL `https://www.adobe.com/devnet-docs/acrobatetk/tools/DigSig/Acrobat_DigitalSignatures_in_PDF.pdf`. Accessed on 15.08.2018.

[7] Tim Bienz, Richard Cohn, and Jim Meehan. Portable Document Format Reference Manual, Version 1.3, March 1999.

[8] Tim Bray, Jean Paoli, Christopher Michael Sperberg-McQueen, Eve Maler, and Francois Yergeau. Extensible Markup Language (XML) 1.0 (Fifth Edition). *W3C Standard*, November 2008. URL `https://www.w3.org/TR/2008/REC-xml-20081126`.

[9] Igino Corona, Davide Maiorca, Davide Ariu, and Giorgio Giacinto. Lux0r: Detection of malicious pdf-embedded javascript code through discriminant analysis of api references. In *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*, pages 47–57. ACM, 2014.

[10] European Telecommunications Standards Institute. Electronic Signatures and Infrastructures (ESI); PDF Advanced Electronic Signature Profiles; Part 1: PAdES Overview - a framework document for PAdES, April 2016.

URL https://www.etsi.org/deliver/etsi_en/319100_319199/31914201/01.01.01_60/en_31914201v010101p.pdf.

[11] Valentin Hamon. Malicious URI resolving in PDF Documents. *Blackhat Abu Dhabi*, 2012. URL https://media.blackhat.com/ad-12/Hamon/bh-ad-12-malicious%20URI-Hamon-Slides.pdf.

[12] Pavel Laskov and Nedim Šrndić. Static detection of malicious JavaScript-bearing PDF documents. In *Proceedings of the 27th annual computer security applications conference*, pages 373–382. ACM, 2011.

[13] Davide Maiorca, Giorgio Giacinto, and Igino Corona. A pattern recognition system for malicious pdf files detection. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 510–524. Springer, 2012.

[14] Davide Maiorca, Davide Ariu, Igino Corona, and Giorgio Giacinto. A structural and content-based approach for a precise and robust detection of malicious pdf files. In *2015 International Conference on Information Systems Security and Privacy (ICISSP)*, pages 27–36. IEEE, 2015.

[15] 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.

[16] Michael McIntosh and Paula Austel. XML signature element wrapping attacks and countermeasures. In *SWS '05: Proceedings of the 2005 Workshop on Secure Web Services*, pages 20–27, New York, NY, USA, 2005. ACM Press.

[17] Tim McLean. Blog post: Critical vulnerabilities in JSON Web Token libraries, March 2015. URL https://www.chosenplaintext.ca/2015/03/31/jwt-algorithm-confusion.html. Accessed on 10.11.2018.

[18] Vladislav Mladenov, Christian Mainka, Karsten Meyer zu Selhausen, Martin Grothe, and Jörg Schwenk. How To Break PDF Signatures. *Unpublished*, 2018.

[19] Dan-Sabin Popescu. Hiding Malicious Content in PDF Documents. *CoRR*, 2012.

[20] Frédéric Raynal, Guillaume Delugré, and Damien Aumaitre. Malicious Origami in PDF. *Journal in Computer Virology*, 6(4):289–315, 2010. URL http://esec-lab.sogeti.com/static/publications/08-pacsec-maliciouspdf.pdf.

[21] Charles Smutz and Angelos Stavrou. Malicious PDF detection using metadata and structural features. In *Proceedings of the 28th annual computer security applications conference*, pages 239–248. ACM, 2012.

[22] 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, 2012.

[23] Nedim Šrndić and Pavel Laskov. Hidost: a static machine-learning-based detector of malicious files. *EURASIP Journal on Information Security*, (1):22, 2016.

[24] Tomáš Stefan. Digital Signature Verification in PDF, 2018. URL `https://dspace.cvut.cz/bitstream/handle/10467/76810/F8-BP-2018-Stefan-Tomas-thesis.pdf`. Accessed on 10.11.2018.

[25] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. The first collision for full SHA-1. In *Annual International Cryptology Conference*, pages 570–596. Springer, 2017.