RUHR-UNIVERSITÄT BOCHUM

RUB

Vulnerability Report

Attacks bypassing the signature validation in PDF

Christian Mainka, Vladislav Mladenov, Simon Rohlmann, Jörg Schwenk

02. March, 2020
Chair for Network and Data Security

hgi Lehrstuhl für Netz- und Datensicherheit

1 The scope of the vulnerability report

Research Results

Digitally signed PDFs are used in contracts, bills, and agreements to guarantee the authenticity and integrity of their content. A typical user would assume that digitally signed PDF files are final and cannot be further modified. However, various changes like adding annotations to a signed PDF or filling out form fields are allowed and do not invalidate PDF signatures.

In this report, we show that this flexibility allows attackers to completely change a document's content while keeping the original signature validation status untouched. Our attacks work in a novel attacker model, which allows attackers hiding content in a PDF. After signing this PDF by a benign entity, the attackers reveal the hidden content by using permitted manipulations. Our results reveal that out of 27 tested PDF viewers, 15 of them.

About us

The Chair of Network and Data Security (NDS) has been working since 2003 under the direction of Prof. Dr.-Ing. Jörg Schwenk on the security analysis of cryptographic protocols (especially in connection with browser-based protocols based on TLS) and the XML format for signature generation and encryption. One focus of the department is to comprehensively explore the multitude of cryptographic techniques and standards used in these fields.

2 Overview

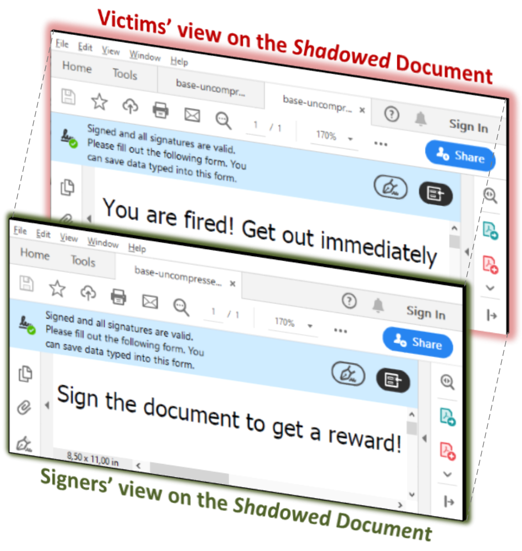
Portable Document Format (PDF) documents are a major office format. According to Adobe, more than 250 billion PDFs were opened in Adobe products in 2019 [1].

Since 1999 PDFs can be protected against manipulations with digital signatures enabling use-cases such as signing contracts, agreements, payments, and bills. This way, the necessity to send printouts via mail all over the world is eliminated. Regulations like the eSign Act in the USA [6] or the eIDAS regulation in Europe [8] facilitate the acceptance of digitally signed documents by companies and governments. Asian and South American countries also accept digitally signed documents as an equivalent to manually signed paper documents [9]. Adobe Cloud, a leading online service for signing PDF documents, provided 8 billion electronic and digital signature transactions in 2019 [1]. The same year, DocuSign processed in the same year 15 million documents each day [2].

The process of signing a PDF typically looks as follows: The PDF document is prepared by the collaborators and agreed to be signed in the final step. Then, the document is sent electronically to each project partner who signs it. In the end, one PDF contains all digital signatures from each partner.

Security of PDF Signatures. In 2019, a comprehensive analysis of the security of digitally signed PDFs revealed severe flaws in multiple applications and found almost all of them vulnerable [5]. However, the authors limited the attackers' capabilities who only possess a digitally signed PDF and manipulate it afterwards. In this paper, we extend this attacker model and assume that the attackers can have write access to the PDF file before it is signed. This assumption is motivated by real-world scenarios and usage of PDFs. For instance, the attackers may be employees or partners preparing a contract before all partners review and sign it. This new attacker model rises the question: *Can the visible content of a digitally signed PDF be altered without invalidating a signature if attackers craft the PDF before it is signed?*

Shadow Attacks. In the analog world, a handwritten signature is typically added at the end of the document. This has one major downside – it is possible to exchange all pages before the signed page with arbitrary content. This exchange is impossible when using digital signatures because this type of signature protects the entire content of a document. So it is assumed that such an attack from the analog world cannot be transferred to digital signatures. In this paper, we show that this assumption is incorrect by introducing a new attack class: *Shadow Attacks*. The idea of *Shadow Attacks* is that the attackers create a PDF document with two different contents: content expected by the authority reviewing


Victims' view on the Shadowed Document
File Edit View Window Help
Home Tools
Sign In
Share
Signers' view on the Shadowed Document

Signed and all signatures are valid.
Please fill out the following form. You can save data typed into this form.

You are fired! Get out immediately

Sign the document to get a reward!

Figure 2.1: A signed PDF document remains valid after manipulations. A *Shadowed* PDF document presents a trustworthy content to the *Signers*. After signing the document, the attackers modify the document and enforce another view of the document on victims' side without invalidating the signature.

and signing the PDF and hidden content that will be displayed after the PDF is signed. In Figure 2.1, an overview of the attack is shown. The *Signers* of the PDF receive the document, review it, and sign it. The attackers use the signed document, modify it slightly, and send it to the victims.

After opening the signed PDF the victims check whether the digital signature was successfully verified. However, the victims see different content than the *Signers*.

In this paper, we introduce three different variants of the *Shadow Attacks* allowing attackers to *hide*, *replace*, and *hide and replace* content in digitally signed PDFs. The *Shadow Attacks* do not rely on dynamic content replacement, for example, by using JavaScript, or content loaded from external resources that can be modified after signing the PDF. We consider such attacks trivial, and according to our observations, all viewers already prevent such attacks by warning the user.

We propose a generic countermeasure that is standard compliant and does not affect the usability of digitally signed PDFs in contrast to previous work [5].

3 Basics

This section provides a brief overview of the PDF file structure and explains the two most important features for this paper: Incremental Saving (IS) and PDF Signatures.

3.1 PDF File Structure

The Portable Document Format (PDF) is a platform-independent document format. It consists of three main parts, as depicted in Figure 3.1.

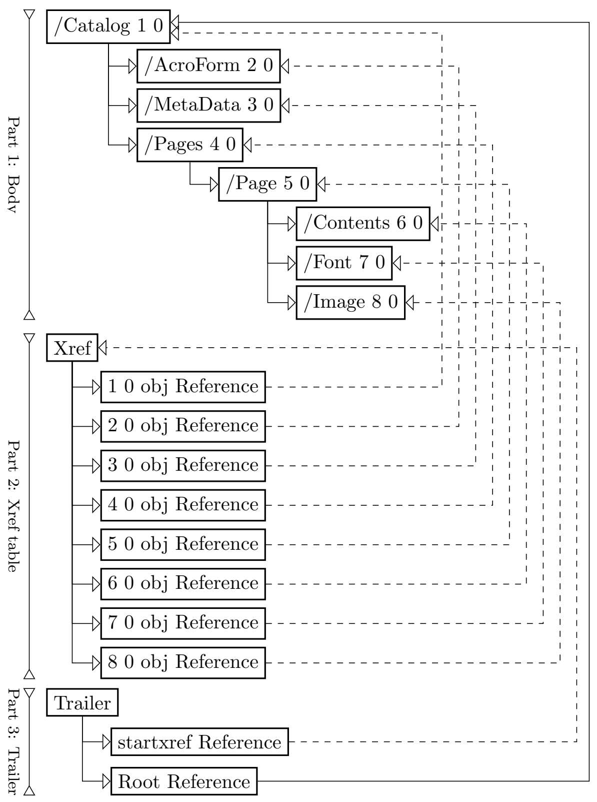


Figure 3.1: A PDF consists of three parts: body, *Xref table*, and *Trailer*. Objects in the body and the root reference in the trailer link to other object (solid-lined arrow). The *Xref table* part and the startxref entry contain the byte position within the file of the corresponding entry (dashed-lined arrows).

The first part defines the *PDF body*. It contains different objects, which are identified by its object number. The most important object is the root object, which is called the *Catalog*. In Figure 3.1, the Catalog has the object identifier *1 0*. The Catalog defines the whole PDF structure by linking to other objects in the body. In the example given, the Catalog links to form object *AcroFrom*, to some PDF *MetaData*, and to actual PDF *Pages*. The latter can reference multiple *Page* objects, which in turn reference, for example, the actual *Content*, *Font*, and *Images*.

The second part of the PDF is the *Xref table*. It contains references to the byte positions of all objects used in the PDF body.

The third part is the *Trailer*. It consists of two further references: one to the byte position at which the *Xref table* starts, and another link to the identifier of the root object (*1 0*).[1]

3.2 Incremental Saving

The content of a PDF may be updated for different reasons, for example, by adding review comments or by filling out PDF forms. From a technical perspective, it is possible to add this new content directly into the existing PDF body and add new references in the *Xref table*. However, this is not the case according to the PDF specification. Updates are implemented using Incremental Saving (IS).

An IS adds new objects into a new PDF body, which is directly appended after the previous *Trailer*. To adequately address the new objects, a new *Xref table* and *Trailer* are appended as well for each IS. Summarized, a PDF can have multiple bodies, *Xref table*s, and *Trailer*s, if IS is applied.

3.3 PDF Signature

For protecting the integrity and the authenticity of a PDF, digital signatures can be applied. For this purpose, a *Signature* object is created and appended to the PDF by using IS. It is also possible to sign a PDF multiple times (e.g., a contract), resulting in multiple ISs. The Signature object contains all relevant information for validating the signature, such as used algorithms and the signing certificate. It also defines which bytes of the PDF are protected by the Signature. A typical signature starts at the first byte and ends at the last byte of the trailer.[2] For the attacks presented in this paper, more technical details on PDF Signatures are not necessary, but interested readers find them in the specification [3, Section 12.8]. Once a PDF that contains a PDF Signature is opened, the viewer application automatically validates the signature and provides a warning if the content has been modified.

[1]The root element does not need to have the identifier *1 0*.

[2]For technical reasons, there is a gap inside this range, which is unprotected. It contains a PKCS#7 blob of the signature itself.

4 Attacker Model

The attacker model is based on the real-world use-case in which a document, for example a contract, will be signed. We distillate this process and identify possible situations in which the attackers can manipulate a Portable Document Format (PDF) before signing it.

Generally spoken, the idea is to let the attackers chose a PDF, and use the *Signers* as a signing oracle. After the signing, the attackers manipulate the signed PDF again in order to enforce a change in its content without invalidating the signature.

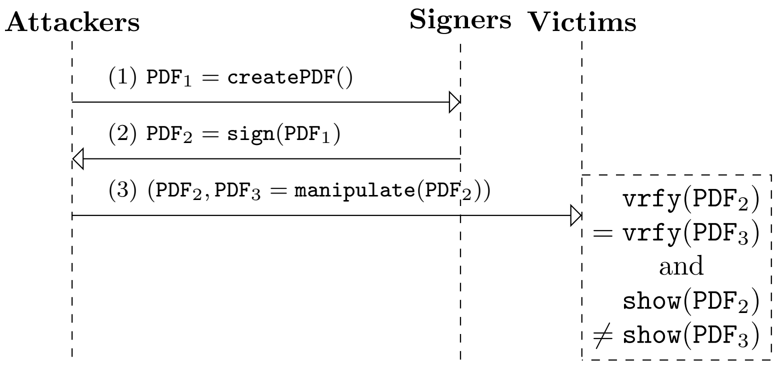

Attackers
Signers
Victims
(1) $\mathtt{PDF}_1 = \mathtt{createPDF}()$
(2) $\mathtt{PDF}_2 = \mathtt{sign}(\mathtt{PDF}_1)$
(3) $(\mathtt{PDF}_2, \mathtt{PDF}_3 = \mathtt{manipulate}(\mathtt{PDF}_2))$
$\mathtt{vrfy}(\mathtt{PDF}_2)$
$= \mathtt{vrfy}(\mathtt{PDF}_3)$
and
$\mathtt{show}(\mathtt{PDF}_2)$
$\neq \mathtt{show}(\mathtt{PDF}_3)$

Figure 4.1: Attacker Model: the attackers prepare the *Shadowed Document* ($\mathtt{PDF}_1$) which the *Signers* sign ($\mathtt{PDF}_2$). The attackers afterward modify the content of the signed PDF ($\mathtt{PDF}_3$) and sends it to the *Victims*.

Attacker Capabilities. As shown in Figure 4.1, the attacker capabilities can be divided into three phases. The output of each phase is a PDF file.

1. The attackers create the shadowed PDF document $\mathtt{PDF}_1 = \mathtt{createPDF}()$. They can embed arbitrary content into this file. Arbitrary in this context means that the attackers can embed invisible content into the PDF document. The content can be either invisible due to an overlaying content (e.g., an image), because the corresponding PDF object is not referenced in the *Xref table*, or due to any other masking attack techniques.
2. The signers create a new document $\mathtt{PDF}_2$ by signing $\mathtt{PDF}_1$, i.e. $\mathtt{PDF}_2 = \mathtt{sign}(\mathtt{PDF}_1)$. The signers can be a human, for example, receiving $\mathtt{PDF}_1$ via email, or an online sign-

ing service, such as DocuSign[1] or Adobe Document Cloud [2] to which the attackers upload the file.

3. In the end, the attackers receive $\mathtt{PDF}_2$. They can modify the file again, for instance, the attackers create $\mathtt{PDF}_3 = \mathtt{manipulate}(\mathtt{PDF}_2)$. The attackers send $\mathtt{PDF}_2$ and $\mathtt{PDF}_3$ to the victims. The victims verify both files according to the winning conditions.

The main difference to previous work [5] is that the attackers are allowed to embed malicious content *before* the PDF is signed instead of solely modifying it after the signature has been applied.

Winning Conditions. The attackers are successful if the victims accept the following conditions on receiving $\mathtt{PDF}_2$ and $\mathtt{PDF}_3$:

1. The signature verification of $\mathtt{PDF}_3$ returns the exact same result as the signature verification of $\mathtt{PDF}_2$: both are *valid*.
2. $\mathtt{PDF}_2$ and $\mathtt{PDF}_3$ show different content, for example, a different text on the same page.
3. Opening $\mathtt{PDF}_3$ does not show any kind of errors or warnings, for example, due to a malformed file format.

On an abstract level, the attack is successful if the victims are unable to decide if either the content of $\mathtt{PDF}_2$ or $\mathtt{PDF}_3$ was initially signed.

Out-of-Scope. The attackers do not embed any dynamic content loaded from external sources. This attack idea was already published by Popescu [7]. Moreover, the PDF viewers either prevent the loading of external content or throw a warning in such cases. We consider both events as an unsuccessful attack because the Signers can detect dangerous content.

Summarized, we consider that the Signer of the document reacts on warnings thrown by opening the document with refusing to sign the document.

[1]`https://www.docusign.com/`
[2]`https://acrobat.adobe.com/us/en/`

5 Shadow Attacks

In this section, we present a new attack class, which we call *Shadow Attacks*. The attack class bypasses the PDF's signature integrity protection, allowing the modification of the content without the victim noticing.

The main idea of the attack is that the attackers prepare a PDF document containing invisible content. Afterward, the document is sent to a signing entity like a person or a service which reviews the document, signs it, and sends it back to the attackers. Despite the integrity protection provided by the digital signature, the attackers can make modifications to the document and change the visibility of the hidden content. Nevertheless, the manipulation is not detected and the digital signature remains valid. Finally, the attackers send the modified signed document to the victim. Although the document is altered, the signature validation is successful, but the victims see different content than the signing entity.

5.1 *Shadow Documents* in the Real World

Considering the applicability of *shadow* documents we focus on the following two questions: (1) How can the attackers force the signing of a *shadow document*? (2) Why are the attackers capable of modifying a signed *shadow document*?

Signing a *Shadow Document*. In companies and authorities, relevant documents like contracts or agreements are often prepared by the employees taking care of all details and technicalities. Then, the document is signed by an authorized person after a careful review. Another scenario is the signing process of a document within a consortium. Usually, one participant creates the final version of the document, which is then signed by all consortium members. Considering the given examples, an employee or consortium member acting maliciously can hide invisible *shadow* content during the editing. Consequentially, this content will be signed later.

Additionally, multiple cloud signing services like Adobe Cloud, DocuSign, or Digital Signature Service exist. Among other functionalities, such services receive a document and sign it. For instance, USENIX uses the DocuSign service for the camera-ready version of the accepted papers. Such services can also be used to sign *shadow* documents.

Manipulating a *shadow* document. One can assume that a signed PDF document cannot be changed and that it is final. This is not the case due the desired features like multiple signatures or annotations. For example, a PDF document can be signed multiple times.

This process is essential in many use-cases: it allows stakeholders within a consortium to have one single document containing the signatures from all partners. From a technical perspective, each new signature appends new information to the already signed document (see section 3.2). Nevertheless, the document should still be successfully verified for each signature. Additionally, the PDF specification defines interactive features like annotations (e.g., sticky notes and text highlighting). Since annotations do not change the content, but only put remarks on it, the PDF specification allows the insertion of annotations in a signed file without invalidating the signature. In summary, further information can be appended to a digitally signed PDF file without invalidating the signature, as long as these changes are considered harmless.

5.2 Analysis of Document Modifications

Currently, the PDF applications analyze the changes made after signing and try to estimate if these changes are legit. For instance, overwriting content on a page of the document is not allowed and thus leads to invalid signature verification. Such attacks were evaluated in 2019 by Mladenov et al. [5].

In this paper, we first analyzed which changes are considered harmless by the PDF applications and abused these to exchange the entire content within a PDF document. The allowed changes can be summarized as follows.

Appending new *Xref table* and *Trailer*. Appending a new *Xref table* and *Trailer* occurs on each change on PDF documents. For instance, for each signing process among the signature information new *Xref table* and *Trailer* are generated. Thus, appending these at the end of the file is considered harmless.

Overwriting harmless objects. In their paper, Mladenov et al. [5] were able to append new objects beyond the signed document overwriting existing objects and thus replacing the content. The attack was called Incremental Saving Attack (ISA). Nevertheless, the authors considered only object types: `Catalog`, `Pages`, `Page`, and `Contents`. This is reasonable since these objects directly influence the content shown by opening the document. As a reaction to their findings, the applications fixed the vulnerabilities by detecting the definition of such objects after the signature was applied. Inspired by the work presented by Markwood et al. [4], we considered the definition of further objects like fonts or metadata which also influence the presented content.

Changing interactive forms. During our research, we observed an unexpected feature applied on interactive forms, which overlays the content of a text field. By clicking on the text field, its content is shown, and the overlay disappears. Ignoring the usefulness of this feature, we observed that changes on the overlay are considered harmless and do not invalidate the signature.

Summary. The PDF specification defines a compromise between usability and security by softening the rules regarding the integrity protection of digitally signed documents. By defining exceptions regarding allowed and forbidden changes, the responsibility of the developer teams regarding the detection and classification of manipulations raises and leads to vulnerabilities. In the following, we show how the allowed changes which are considered harmless, can be used to exchange content within the document without invaliding the signature.

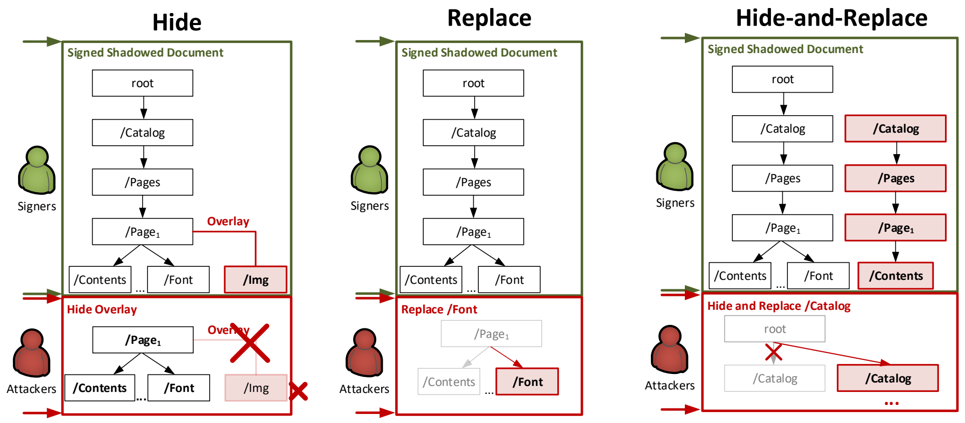


Figure 5.1: We show three variants of manipulating a *Shadowed* PDF documents without being detected: *Hide*, *Replace*, and *Hide-and-Replace*.

5.3 Shadow Attack: *Hide*

The concept of *Hide* Shadow Attacks is to hide the content relevant for the victims behind a visible layer. For example, the attackers can hide the text “You are fired!” behind a full-page picture showing “Sign me to get the reward!”. Once the attackers receive the signed document, they manipulate the document in such a way, that the picture is no longer rendered by the viewer application.

Hide attacks have two advantages from the attackers’ perspective:

1. Many viewers show warnings, if new visible content is added using IS. However, they do not warn in most cases if content is removed.
2. The objects are still accessible within the PDF. In the example above, the text “You are fired!” can still be detected by a search function. This might be important if an online signing service is used and it reviews the document by searching for specific keywords.

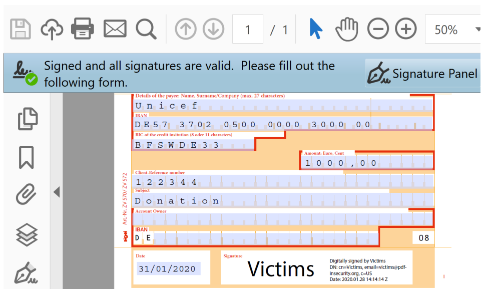
We identified two variants of the *Hide* attacks.

Hiding Content via Page. This attack variant uses an IS to create a new `Page` object. It contains all previously used objects except for the overlay, for example the image. This attack variant is depicted on the left side in Figure 5.1.

Hiding Content via Xref. If the viewer application does not accept changes to PDF structuring objects, such as `Page`, `Pages`, or `Contents`, the second attack variant can be applied. This variant directly affects the overlay object. The simplest method for this is to create an IS, which only updates the *Xref table* table by setting the overlay object to *free*. However, making this change is interpreted as a dangerous in many viewers and an error or a warning is thrown. For this reason, we use another approach: We use the same object ID within the IS, but we define it as a different object type. For example, we change the overlay type *image* to *XML/Metadata*. Additionally, we added an *Xref table* update pointing to the metadata object, but keeping the object ID of the overlay.

When opening this manipulated document, the overlay is hidden, because Metadata cannot be shown. Since adding Metadata to a signed PDF using IS is considered harmless, the signature remains valid.

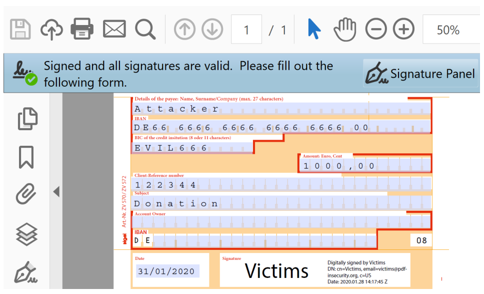
5.4 Shadow Attack: *Replace*



Signed and all signatures are valid. Please fill out the following form.

Details of the payee: Name, Surname/Company (max. 27 characters): U n i c e f
IBAN: D E 5 7 3 7 0 2 0 5 0 0 0 0 0 0 3 0 0 0 0 0
BIC of the credit insitution (8 oder 11 characters): B F S W D E 3 3
Amount: Euro, Cent: 1 0 0 0 , 0 0
Client-Reference number: 1 2 2 3 4 4
Subject: D o n a t i o n
Account Owner:
IBAN: D E 08
Date: 31/01/2020
Signature: Victims
Digitally signed by Victims
DN: cn=Victims, email=victims@pdf-insecurity.org, c=US
Date: 2020.01.28 14:14:14 Z

(a) A *shadow* PDF document digitally signed by the victims containing a donation amount.



Signed and all signatures are valid. Please fill out the following form.

Details of the payee: Name, Surname/Company (max. 27 characters): A t t a c k e r
IBAN: D E 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 0 0
BIC of the credit insitution (8 oder 11 characters): E V I L 6 6 6
Amount: Euro, Cent: 1 0 0 0 , 0 0
Client-Reference number: 1 2 2 3 4 4
Subject: D o n a t i o n
Account Owner:
IBAN: D E 08
Date: 31/01/2020
Signature: Victims
Digitally signed by Victims
DN: cn=Victims, email=victims@pdf-insecurity.org, c=US
Date: 2020.01.28 14:17:45 Z

(b) Manipulated PDF document after signing which contains attackers' account data.

Figure 5.2: Form-Based Attack. The document on the left side is signed by the victims donating to a non-profit organization. The attackers manipulate the document which displays different account information than the signed one. The validity status of digital signature remains untouched. Apart from the account information, both documents are indistinguishable.

The main idea of this variant is to append new objects to the signed document which are considered harmless but influence directly the presentation of the signed content, see Figure 5.1. For instance, the (re)-definition of fonts does not change the content directly. However, it influences the view of the displayed content making number or character swapping possible.

Replace via Overlay

This attack targets an interactive feature in PDFs – interactive forms. Forms support different input masks (e.g., text fields, text areas, radio/selection buttons) where users dynamically enter new content and store it in the PDF document. Forms can also have default values which can be changed if needed. An example of a transfer slip as a PDF document containing forms is depicted in Figure 5.2.

The main idea of the attack is to create a form, which shows one value before ($\texttt{PDF}_1$) and after signing ($\texttt{PDF}_2$) as shown on the left side in Figure 5.2. After the attackers manipulate the PDF and create $\texttt{PDF}_3$, different values are shown in the form (right side in Figure 5.2). The attack abuses a special property of PDF text fields: a text field can show two different *values*: the real field value and an overlay value which disappears as soon as the text field is selected. The real value of a form field is contained in an object key named `/V`. The content of the overlay element is defined within a `/BBox` object. The `/BBox` object is comparable to the hint labels known from HTML forms, for example the hint *username* to indicate that the username should be entered into a specific login field. *In contrast* to HTML, in PDF there is no visual difference between the *hint* and the *actual* value.

Description.

We explain the attack on an example depicted in Figure 5.2. The attackers create a transfer slip ($\texttt{PDF}_1$) containing an interactive form which the signers fill-out before signing the document. The attackers initialize some of the form elements with default values.

The preparation steps of the *shadow* document ($\texttt{PDF}_1$) look as follows:

1. In the example provided in Figure 5.2, the attackers set the values `/V` of the first three form fields to `Attacker` and the attackers' `IBAN` and `BIC`.
2. Second, the attackers set the overlay values (`/BBox`) to `unicef` and the corresponding`IBAN` and `BIC`. As long as the signers do not focus on the prepared values they believe that the correct values are already pre-filled.

The signers signs the PDF without changing the pre-filled forms. Once the attacker receives $\texttt{PDF}_2$, they update the text fields by replacing the overlay stored in `/BBox` with different values. The values stored in `/V` remain unchanged. This replacement is considered harmless since the original text field value is not changed but only the overlay.

Once the victims open $\texttt{PDF}_3$, the viewer proceeds as follows:

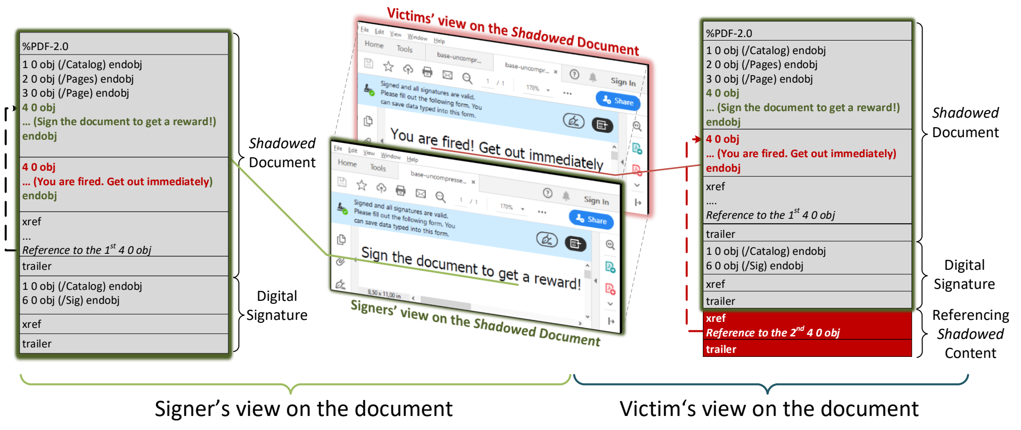
1. The viewer verifies if the values stored in `/V` within each text field have been changed and differ from the signed values? If true, the signature validation fails. Since the attackers do not change any values stored in `/V` the signature remains *valid*.
2. The viewer processes each text field object and shows the `/BBox` value if it maps to the signed one. Otherwise the value stored in `/V` is presented. Since the attackers change the `/BBox` value, the value `/V` is shown, which is `Attacker` and the corresponding malicious transaction slips.

As a result, the signers and the victims have different views on the same document which should be prevented by the digital signature.

Replace via Overwrite

As shown in Figure 5.1, the attackers prepare a *shadow* document which defines a font used for the presentation of a specific content. After the document is being signed, the attackers append a new font description and overwrite the previous description. Since, the definition of new fonts is considered harmless, the applications verifying the signature, does not throw any warning regarding the made changes.

5.5 Shadow Attack: *Hide-and-Replace*



%PDF-2.0
1 0 obj (/Catalog) endobj
2 0 obj (/Pages) endobj
3 0 obj (/Page) endobj
4 0 obj
... (Sign the document to get a reward!)
endobj
4 0 obj
... (You are fired. Get out immediately)
endobj
xref
...
Reference to the 1st 4 0 obj
trailer
1 0 obj (/Catalog) endobj
6 0 obj (/Sig) endobj
xref
trailer
Shadowed Document
Digital Signature
Victims' view on the Shadowed Document
You are fired! Get out immediately
Sign the document to get a reward!
Signers' view on the Shadowed Document
xref
Reference to the 2nd 4 0 obj
trailer
Referencing Shadowed Content
Signer's view on the document
Victim's view on the document

Figure 5.3: The attackers successfully manipulate a signed document and force different views on the *signers* and the *victims* by using the *Hide-and-Replace* attack variant.

In this attack variant, the attackers create a *shadow* PDF document, which is sent to the signers. The PDF document contains a hidden description of another document with different content. Since the signers cannot detect the hidden (malicious) content, they sign the document. After signing, the attackers receive the document and append only a new *Xref table* table and *Trailer*. Within the *Xref table* table, only one change takes place - the reference to the document `/Catalog` (or any other hidden object) which now points to the *shadow* document.

Description. In Figure 5.3, an example of the attack is depicted and will be explained further.

- The attackers create a PDF file containing two objects with the same object ID (e.g., `4 0 obj`) but different content: *Sign the document to get a reward!* and *You are fired. Get out immediately*.
- As shown on the left side in Figure 5.3, within the *Xref table* section the seemingly harmless content is referenced.

- The signers only see this content and sign the PDF file.
- After receiving the signed PDF, the attackers append a new *Xref table* table and exchange the reference to the `4 0` object with the malicious content – *You are fired. Get out immediately*. A new *Trailer* is also appended.
- Since the inclusion of an *Xref table* table pointing to an already defined object within the signed area is considered harmless, no warning regarding the made changes occur, and the signature verification is successful. Nevertheless, the victims see another content than the signers.

Advantages and Disadvantages. This attack variant is the most powerful one since the content of the entire document can be exchanged. The attacker can build a complete *shadow* document influencing the presentation of each page and each object.

A possible disadvantage could occur in case that during the signing process, unused objects are removed. Thus, the *shadow* elements could be deleted, making the second step of the attack obsolete. A security scanner could also detect the unused objects within the PDF and throw a warning. Currently none of these disadvantages occur.

6 Evaluation

In this section we present the results of our evaluation. The manipulated PDF documents created during the research, were tested as black box procedures, under all viewing applications listed in Table 6.1.

			Shadow Attack Category			Summary
Application	Version		Hide	Replace	Hide-and-Replace	
Adobe Acrobat Reader DC	2019.021.20061	Windows	●	●	●	●
Adobe Acrobat Pro 2017	2017.011.30156	Windows	●	●	●	●
Expert PDF 14	14.0.25.3456 64-bit	Windows	◐	◐	◐	◐
Foxit Reader	9.7.0.29455	Windows	○	●	●	●
Foxit PhantomPDF	9.7.0.29478	Windows	○	●	●	●
LibreOffice Draw	6.2.5.2 (x64)	Windows	○	◐	◐	◐
Master PDF Editor	5.4.38, 64 bit	Windows	○	●	●	●
Nitro Pro	12.16.3.574	Windows	◐	◐	◐	◐
Nitro Reader	5.5.9.2	Windows	◐	◐	◐	◐
PDF Architect 7	7.0.26.3193 64-bit	Windows	◐	◐	◐	◐
PDF Editor 6 Pro	6.5.0.3929	Windows	●	●	●	●
PDFelement	7.4.0.4670	Windows	●	●	●	●
PDF-XChange Editor	8.0 (Build 331.0)	Windows	◐	◐	◐	◐
Perfect PDF Reader	V14.0.9 (29.0)	Windows	◐	◐	◐	◐
Perfect PDF 8 Reader	8.0.3.5	Windows	●	●	●	●
Perfect PDF 10 Premium	10.0.0.1	Windows	●	●	●	●
Power PDF Standard	3.0 (Patch-19154.100)	Windows	●	●	●	●
Soda PDF Desktop	11.1.09.4184 64-bit	Windows	○	◐	◐	◐
Adobe Acrobat Reader DC	2019.021.20061	macOS	●	●	●	●
Adobe Acrobat Pro 2017	2017.011.30156	macOS	●	●	●	●
Foxit Reader	3.4.0.1012	macOS	●	●	●	●
Foxit PhantomPDF	3.4.0.1012	macOS	●	●	●	●
Master PDF Editor	5.4.38, 64 bit	macOS	○	○	○	○
PDF Editor 6 Pro	6.8.1.3450	macOS	○	○	○	○
PDFelement	7.5.7.2895	macOS	○	○	○	○
Master PDF Editor	5.4.38, 64 bit	Linux	○	●	●	●
LibreOffice Draw	6.0.7.3	Linux	○	◐	◐	◐
		∑ 27	11● 6◐	15● 9◐	15● 9◐	15● 9◐

● Application vulnerable ◐ Vulnerability limited ○ Not vulnerable

Table 6.1: **Evaluation results.** 15 out of 27 applications are vulnerable to at least one attack. In 9 cases, the *vulnerability is limited*, i.e., the same warning is raised in case of an *allowed* modification (e.g., commenting) as well as in case of *unallowed* modifications (attacks). Victims are unable to distinguish between both cases.

7 Countermeasures

Mladenov et al. [5] already proposed a countermeasure to prevent their identified attacks. The downside of their approach is that the algorithm only accepts a PDF signature if the entire PDF document is signed. Consequentially, their algorithm would also detect our attacks.

However, it is not applicable in the real world. Speaking of contracts signed by multiple persons would cause problems since a multiple-signed PDF is invalid according to the algorithm by Mladenov et al. [5]. This is reasoned by the fact that the first PDF signature does not cover the last byte of the PDF.

For this reason, we extend the validation algorithm as follows:

1. Take the input PDF $\texttt{PDF}_0$ and split it into its revisions $\mathbb{P} = \{\texttt{PDF}_{rev_1}, \ldots, \texttt{PDF}_{rev_n}\}$ according to its Incremental Savings.
2. Find the first signed revision $\texttt{PDF}_{rev_i} \in \mathbb{P}$ with $i \geq 0$.
    a) If no signature is found, *return false*
3. For $j = \{i, \ldots, n\}$
    a) If $\texttt{PDF}_{rev_j}$ has no signature, *return false*
    b) Verify $\texttt{PDF}_{rev_j}$, i.e., $true \stackrel{?}{=} \texttt{vrfy}_{\text{single}}\left(\texttt{PDF}_{rev_j}\right)$, or *return false*
4. *return true*

Our algorithm is a composition. It uses an algorithm $\texttt{vrfy}_{\text{single}}\,()$, which can verify a PDF that contains precisely one signature, for example, the algorithm proposed by Mladenov et al. [5].

The major problem that we identified during our research is the combination of PDF Signature and IS. This combination is addressed by our composition algorithm. Due to our evaluation results, most viewers allow *some* changes in an IS without invalidating the signature. This limited number of allowed changes can be abused in too many cases. For this reason, we argue, that once a PDF is signed (Step 2), all further revision must be signed (Step 3), without a single exception (Step 3.1). This behavior has one downside: it does not allow any kind of change to the document without signing this change. For loosening this restriction, Step 3.1 might raise a warning ("Document has been updated.") instead of returning *false*. The interested user could then manually view and inspect this particular revision. Currently, lots of viewers already show a similar warning, but it is not precisely defined in which cases they show it. In contrast, our results show that this behavior leads to security issues.

Bibliography

[1] Adobe. Adobe fast facts, November 2018. URL https://www.adobe.com/about-adobe/fast-facts.html.

[2] DocuSign. Docusign 2019 annual report. Technical report, 2019. URL https://s22.q4cdn.com/408980645/files/doc_financials/2019/Annual/DocuSign-FY2019-Annual-Report.pdf.

[3] Adobe Systems Incorporated. *PDF Reference, version 1.7*, sixth edition edition, November 2006.

[4] 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.

[5] Vladislav Mladenov, Christian Mainka, Karsten Meyer zu Selhausen, Martin Grothe, and Jörg Schwenk. 1 trillion dollar refund – how to spoof pdf signatures. In *ACM Conference on Computer and Communications Security*, November 2019.

[6] United States Government Printing Office. Electronic signatures in global and national commerce act, 2000. URL https://www.govinfo.gov/content/pkg/PLAW-106publ229/pdf/PLAW-106publ229.pdf.

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

[8] European Union. Regulation (eu) no 910/2014 of the european parliament and of the council on electronic identification and trust services for electronic transactions in the internal market and repealing directive 1999/93/ec, 2014. URL https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32014R0910.

[9] Wikipedia. Electronic signatures and law, 2019. URL https://en.wikipedia.org/wiki/Electronic_signatures_and_law.