Breaking the Specification: PDF Certification

Simon Rohlmann
Ruhr University Bochum
simon.rohlmann@rub.de

Vladislav Mladenov
Ruhr University Bochum
vladislav.mladenov@rub.de

Christian Mainka
Ruhr University Bochum
christian.mainka@rub.de

Jörg Schwenk
Ruhr University Bochum
joerg.schwenk@rub.de

***Abstract*— The Portable Document Format (PDF) is the de-facto standard for document exchange. The PDF specification defines two different types of digital signatures to guarantee the authenticity and integrity of documents: approval signatures and certification signatures. *Approval signatures* testify one specific state of the PDF document. Their security has been investigated at CCS'19. *Certification signatures* are more powerful and flexible. They cover more complex workflows, such as signing contracts by multiple parties. To achieve this goal, users can make specific changes to a signed document without invalidating the signature.**

This paper presents the first comprehensive security evaluation on *certification signatures* in PDFs. We describe two novel attack classes – *Evil Annotation* and *Sneaky Signature* attacks which abuse flaws in the current PDF specification. Both attack classes allow an attacker to significantly alter a certified document's visible content without raising any warnings. Our practical evaluation shows that an attacker could change the visible content in 15 of 26 viewer applications by using Evil Annotation attacks and in 8 applications using Sneaky Signature by using PDF *specification compliant exploits*. We improved both attacks' stealthiness with applications' *implementation issues* and found only two applications secure to all attacks. On top, we show how to gain high privileged JavaScript execution in Adobe.

We responsibly disclosed these issues and supported the vendors to fix the vulnerabilities. We also propose concrete countermeasures and improvements to the current specification to fix the issues.

I. Introduction

PDF signatures are a well-established protection mechanism to guarantee the integrity, authenticity, and non-repudiation of a PDF document. Introduced in 1999, PDF signatures are used to protect important documents such as certification documents, contracts, and invoices. According to Adobe, 250 billion PDF documents were opened by their applications in 2018. Among them, 8 billion were signed [1]. The legal basis for digitally signed documents is provided in the European Union (EU) by the eIDAS Regulation [2] and in the United States of America (USA) for the Electronic Signatures in Global and National Commerce Act (ESIGN) [3] and the Uniform Electronic Transactions Act (UETA) [4].

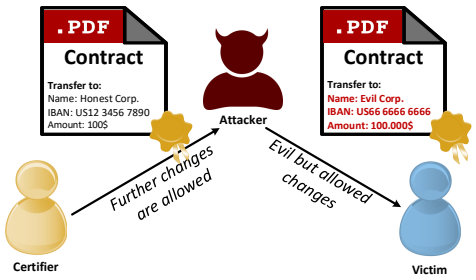


Figure 1. In an exemplary attack scenario, the certifier creates a certified contract with sensitive information which cannot be exchanged. The certifier allows specific changes to the PDF contract, for example, further signatures. Using these permitted changes, the attacker can change the amount from $100 to $100,000 and display the IBAN of his own account. Based on the attacks presented in this paper, the victim cannot detect the manipulation and thus accepts the modified contract.

Different Types of PDF Signatures. The PDF specification defines two types of digital signatures.[1]

1) Approval signatures testify a specific document state. The specification allows the usage of multiple signatures on the same document. Any other change on a signed document leads to an invalidation of the approval signature or warnings in most PDF viewers. Hereafter, we use the terms "signature" and "signed document" for approval signatures.

2) Certification signatures provide a more powerful and flexible mechanism to handle digitally signed documents. During the document's certification, the owner defines a list of allowed modifications that do not invalidate the document's certification signature. These allowed modifications can be a subset of the following actions: writing text to specific form fields (even without signing the document), providing annotations to the document, or adding approval signatures. Since a certification signature sets permissions on the entire document, only one certification signature is allowed within a PDF document. This certification signature must also be the first signature in the PDF. Hereafter, we use the terms "certification" and "certified document" for certification signatures.

Certification signatures in the wild. Companies and organizations can use certification signatures to protect ready-made forms such as contracts, non-disclosure agreements, or

[1]Digital scans of handwritten signatures, if embedded as an image in a PDF document, are called 'electronic signatures'. Since they do not protect the integrity of the document, they are out of scope here.

access control documents against changes and, at the same time, allow signatures in the shape of approval signatures [5, 6, 7, 8]. For example, the United States Government Publishing Office (GPO), a US federal legislative authority, and the Legislative Assembly of British Columbia use certified documents for official publications [9, 10, 11, 12]. The European Telecommunications Standards Institute (ETSI) also specifies the support of certified documents within the EU [13]. Beside the PDF applications, there exist multiple commercial and governmental online services capable of signing and certifying PDF documents [14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24].

Use Case: Certified Document. Suppose that two companies have agreed on a contract but cannot meet in-person to sign it. As shown in Figure 2, the text of the contract is converted to PDF. Both companies want to guarantee that this text is displayed unaltered to any party (e.g., a CEO, lawyer, or judge), even outside the two companies. The CEOs of both companies sign the PDF contract to make it legally binding, but the sales departments of both companies should be allowed to add some parameters (e.g., payment dates) and provide explanations to their CEOs via annotations to the contract.

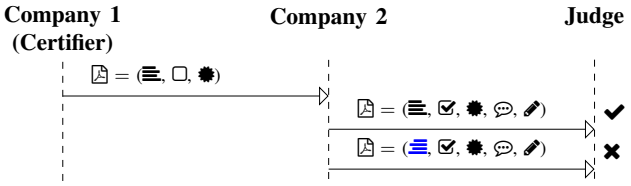


Figure 2. PDF certification use case. The PDF consist of content (e.g., text, images, etc.), and forms. The PDF is protected by a certification signature that prohibits text modifications (e.g., →). Company 2 can add annotations, fill-out forms, and apply a signature. An independent party (e.g., a judge) can verify whether the PDF is valid ✔ or invalid ✖.

In the complete scenario, the CEO of company 1 uses a *certification* on the PDF document. This certification covers the entries of their own sales department and allows for some alterations after certifying. The sales department of company 2 should be able to enter data into some specified form fields which are displayed by the certified document. They should also be allowed to make annotations and to add the signature of the CEO of company 2. Company 2 then fills in the form fields, adds some annotations and signs the slightly modified document. From this scenario, it should be clear that company 2 must not be able to modify the original text of the contract before or when signing, for example, by changing the negotiated payment (→). At the very least, all changes made to the contract by company 2 should be visible to a judge using any PDF viewer in a legal trial.

Unfortunately, this is not the case. In this paper, we present attacks where the content of the PDF document can be altered by company 2 in such a way that the changes are undetectable, either in all PDF applications or in a subset of them.

Security of PDF Certification. We investigate the following question: "*How dangerous are permitted changes in certified documents?*". To answer this question we systematically analyze the allowed modifications in certified documents and reveal two new vulnerabilities abusing flaws in the PDF specification: Evil Annotation Attack (EAA) and Sneaky Signature Attack (SSA). These vulnerabilities allow an attacker to change the visible content of a PDF document by displaying malicious content over the certified content. Nevertheless, the certification remains valid and the application shows no warnings.

PDF Applications are Vulnerable. We evaluated 26 PDF applications and were able to break the security of certified documents in 15 applications by using the EAA. For the attack class SSA, 8 applications proved to be vulnerable. In total, 24 applications were vulnerable to at least one specification flaw. Additionally, we analyzed 26 applications to determine whether the permissions for adding annotations and signatures, as defined in the PDF specification, were implemented correctly. We show that for 11 of 26 applications, a permission mismatch exists.

Code Injection Attack on Adobe. Only certified documents may execute high privileged JavaScript code in Adobe products. For example, a high-level JavaScript can call an arbitrary URL without user confirmation to deanonymize a user. Our research reveals that such code is also executed if it is added as an allowed incremental update. We are the first to reveal that this behavior allows attackers to directly embed malicious code into a certified document.

Responsible Disclosure. We started a coordinated vulnerability disclosure and reported all issues to the respecting vendors. We cooperated with CERT-Bund (BSI) and provided a dedicated vulnerability report including all exploits to them. Adobe, Foxit, and LibreOffice responded quickly and provided patches for late 2020 (CVE-2020-35931) or early 2021 (CVE-2021-28545, CVE-2021-28546) see Appendix B. Adobe fixed the code injection vulnerability in early Nov. 2020 within a patch outside the regular update cycle (CVE-2020-24432). Currently, we participate in the standardization process via the German National Organization for Standardization (DIN) and the International Organization for Standardization (ISO) to address the reported attacks in the next PDF specification.

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

- We define an attacker model based on real-world usage of certified documents. We are the first to consider the attack success on three different PDF application UI Layers (section III).
- We are the first to provide an in-depth security analysis of certified documents by analyzing the 994-pages PDF 2.0 specification [25] (section IV).
- We discovered the EAA and SSA, two vulnerabilities based on security gaps in the PDF specification. Both allow modifying the displayed content arbitrarily in a certified document (section V).
- We implemented two tools – PDF Tester and PDF Detector. PDF Tester is a tool to automatically evaluate

the security of multiple PDF applications for multiple PDF files. PDF Detector is an online service evaluating certified documents and recognizing the attacks described in this paper. Both tools are described in section VI.

- We evaluate the security of certified documents in 26 popular PDF applications on Windows, macOS, and Linux. The results are alarming. In only 2 cases, we could not find a vulnerability; 15 viewers were vulnerable to EAA, 8 to SSA, including Adobe, Foxit, and LibreOffice. We additionally analyzed the standard-compliant implementation of PDF certification applications and found issues in 11 of them. Tools and exploits are available on https://pdf-insecurity.org/.
- Since EAA and SSA abuse flaws in the current PDF specification, we propose fixes for them (section VIII).
- We extend the impact of EAA and SSA by showing how to use them for executing high privileged JavaScript (section IX).

II. Basics

A. PDF Structure

Figure 3 shows the file structure of a certified document. The first four building blocks are: *header*, *body*, *xref table*, and *trailer*. The *header* defines the version of the document, for example `%PDF-2.0` for version 2.0. The *body* defines the content shown to the user after opening the file. The *body* contains different objects with different types. Common types are text, font, or image. There are dedicated objects that control the presentation of the PDF, such as *Catalog*, *Pages*, and *Page*. An example of an object is depicted in Listing 1.

```
1 0 obj                                % Object with ID "1"
/Type /Page          % Definition of one page of the document
/Contents 2 0 R          % Ref. to 2 0 obj defining the text
/Resources 3 0 R         % Ref. to 3 0 obj defining the font
endobj                                   % End of the object
```

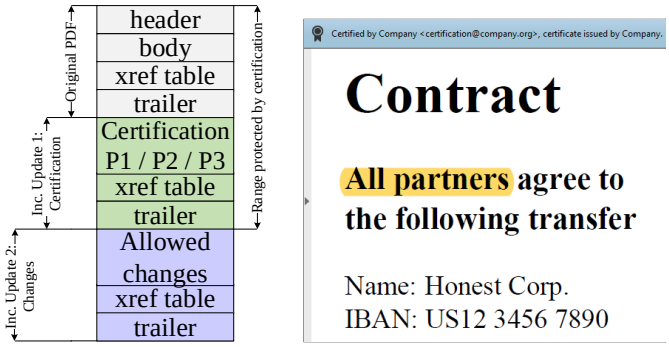
Listing 1. Part of a PDF document depicting the definition of one objects – the *Page* `1 0 obj`.

The *xref table* contains the byte position of each object in the PDF. It allows PDF viewers to efficiently find all objects for processing. The *trailer* defines the byte position of the *xref table* and the root object of the PDF document's object tree. The root object is named *Catalog* and it is the first object to be processed, because it contains all relevant information about the document's structure.

B. Interactive Elements

The PDF specification additionally defines *interactive* elements that allow user input into the document. Such elements are separated in two categories: forms and annotations.

Forms. PDF forms allow user input in a predefined mask, such as a text field, a radio button, or a selection box. Facilities, such as the administration, usually use forms to create PDF documents with predefined areas which are intended to be filled out by users. The user input is, however, limited to the defined form fields and cannot change other content within the PDF.



(a) Structural view of a certified document.

(b) Certified document within an application.

Figure 3. An example of a *certified* document with allowed changes is shown here by highlighting the text "All partners" after certification. The figure is divided into the structure (a) and actual view (b). *Original PDF* depicts the PDF document before it is certified. *Inc. Update 1* presents the PDF document after applying a certification. *Inc. Update 2* shows changes on the document made after its signing and appended at the end of the file.

Annotations. Annotations introduce a different method for a user input by allowing a user to put remarks in a PDF document like text highlighting, strikeouts, or sticky notes. Annotations are not limited to predefined places within the PDF and can be applied everywhere within the document.

C. Incremental Update

An Incremental Update introduces a possibility to extend a PDF by appending new information at the end of the file, see *Inc. Update 1* in Figure 3(a). In this way, the original document stays unmodified and a revision history of all document changes is kept. Each Incremental Update defines new objects, a new *xref table*, and a new *trailer*. An example of an Incremental Update is the inclusion of an certification, signature, annotation, or the filling out forms within a PDF.

D. Integrity Protection of PDFs

Signed Documents. By signing a PDF document, a Signature object is created. This object contains the trusted public keys to verify the document, the signature value, the range of bytes that are protected by the signature, and a user-friendly information regarding the signer of the document. The Signature object is usually added to the PDF document by using an Incremental Update.

Certified Documents. Certifications have two main differences to signatures. First, each PDF can have only one certification and must be the first in the document. Second, certifications define permissions that allow certain changes to the certified document. Signatures have been investigated in previous work (cf., section XI). This paper focuses on certified documents, which have not yet been analyzed. As depicted in Table I, certifications define a more flexible way to handle Incremental Updates, and *allowed Incremental Update do not lead to a warning*. The certifier chooses between three different permission levels (*P*) to allow different modifications.

`P1`: No modifications on the document are allowed.

Incremental Update	Signature *Prev. work [26]*	Certification *This paper*		
		P1	P2	P3
Add/change visible content	⊟	⊟	⊟	⊟
Fill out form inputs	⊞	⊟	⊞	⊞
Multiple signatures	⊞	⊟	⊞	⊞
Add/change annotations	⊖	⊟	⊟	⊞

⊟ Modification not allowed
⊞ Modification allowed
⊞ Only allowed when adding a signature at the same time
⊖ Leads to warnings in most PDF applications

Table I
COMPARISON OF SIGNATURES AND CERTIFICATIONS IN A PDF.

`P2`: Filling out forms, and digitally signing the document are allowed.

`P3`: In addition to `P2`, annotations are also allowed.

The allowed modifications are defined within the *DocMDP Transformation* parameter contained in the certification object. With respect to the integrity protection of the PDF, the PDF application must execute the following steps. First, it must verify if an Incremental Update was applied after the PDF was certified. Second, it must verify if the defined changes are legitimate according to the given permissions.

III. ATTACKER MODEL

The goal of our attacks is to change the *view* on the certified document, and to block warnings on these changes. Therefore, successful attacks must be defined in the context of the PDF viewer's User Interface (UI).

A. UI Layer

The UI in many PDF viewing applications can be divided into three layers that are important for the verification of the certification.

UI-Layer 1: Top Bar Validation Status. UI-Layer 1 is usually displayed immediately after opening. Typical applications use a clearly visible bar on top of the PDF content. The status of the certification and signatures validation is provided as a text (e.g., *valid/invalid*), often combined with green, blue or red background colors, cf. Figures 3, 5, and 6.

UI-Layer 2: Detailed Validation and Information. UI-Layer 2 provides detailed information about the certification and the signatures applied to the PDF. It can be implemented by the viewer in numerous ways, but viewers typically do not show this information automatically once the PDF file is opened. Instead, it must be opened manually by clicking a certain button. For example, this button can be placed on the top-bar (UI-Layer 1). Some viewers use sidebars which provide detailed information regarding the certified document, other use pop-up windows.

UI-Layer 3: PDF Annotations. UI-Layer 3 is another UI element that shows all PDF annotations. Typically, a sidebar is used for this purpose. This layer is of particular importance for certified documents, since with level `P3`, adding and changing PDF annotations is allowed. Without this layer, some annotations (e.g., text blocks) would be indistinguishable from regular PDF text content.

B. Entities

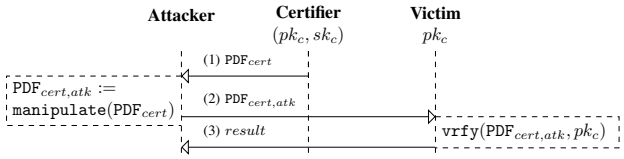


Figure 4. Attacker Model. The attacker is allowed to manipulate the certified document (i) after its certification. and the victim then verifies it.

The attacker model defines multiple entities that are involved during the process of creating a certified document (cf. section III). We assume that private keys remain private, and that public keys are known to all other involved parties.

Certifier. The *certifier* is the entity who initially protects the content of the PDF. The *certifier* sets the level of permissions (`i.e., P1,P2,P3`) and certifies the PDF document.

Victim. The victim can be any person, group, or service that trusts the public keys used by the certifier. The victim uses a PDF viewer application to display the PDF document.

Attacker. The attacker manipulates a given PDF document in order to change its visible content. The attacker is allowed to modify arbitrary parts of the PDF. Arbitrary in this context means that the attacker is not bound to the allowed modifications defined in the certified document. For example, the attacker can technically add annotations to a P1 certified document by manually editing the file without a viewer application. The goal of the attacker is to prevent the victim's detection of these manipulations.

C. Success Conditions

The first success condition is that the PDF application displays the manipulated content. Second, we differentiate the success of the attack in dependence of the UI Layers. On each UI Layer, the application can be *vulnerable* ●, *limited vulnerability* ◐, or *secure* ○. In the following paragraph, we summarize the success conditions for each UI Layer.

UI-Layer 1.

- ● *Vulnerable*: To be classified as vulnerable, UI-Layer 1 must display that the signature *valid*.
- ◐ *Limited Vulnerability*: If in addition to the *valid* signature status, generic information regarding PDF changes is show, we classify the attack as *partially vulnerable*.
- ○ *Secure*: If UI-Layer 1 shows that the signature is invalid, we classify the application as *secure*.

UI-Layer 2.

- ● *Vulnerable*: All signatures shown in UI-Layer 2 must be *valid* to evaluate the PDF application *vulnerable*.
- ◐ *Limited Vulnerability*: If UI-Layer 2 displays *hints* about allowed modifications (e.g., "An annotation has been added, but this is allowed."), then the PDF application is *partially vulnerable*.

○ *Secure*: If UI-Layer 2 shows a *warning* or an *error* with respect to the signature validation, the PDF viewer is *secure* ○.

UI-Layer 3.

● *Vulnerable*: The attacker's annotations that change the visible content must not be shown in UI-Layer 3 to evaluate the PDF application as *vulnerable*.

◐ *Limited Vulnerability*: If the application does not provide any possibility of listing annotations in a dedicated panel, that is UI-Layer 3, we classify the application as *partially vulnerable*.

○ *Secure*: If the attacker's annotations are visible in UI-Layer 3, we evaluate the PDF application as *secure*.

A perfect attack would be successful on all three layers. We argue that if a victim does not validate all UI Layers, an attack on UI-Layer 1 or on UI Layers 1+2 might be sufficient. This assumption especially holds, because only UI-Layer 1 is automatically shown on opening the certified document. The victim must open and inspect other layers manually. Note that the victim can open UI-Layer 2 and UI-Layer 3 independently. With respect to the used application, this opening can involve multiple clicks in nested sub-menus.

Comparison to Previous Work. We used the attacker model introduced by Mladenov et al. [26] for *approval signatures* as a foundation. For *certified documents*, we extended the *success conditions* to consider *PDF annotations* in two ways. First, they can be recognized in UI-Layer 2 as indicated by the status *limited vulnerability* ◐. Second, a PDF viewing application displays PDF annotations in a dedicated user interface. Previous work [26] did not consider UI-Layer 3.

IV. Methodology

A. Manual Specification Analysis

During our research, we carefully analyzed which changes are allowed and which are permitted. We then concentrated on the allowed changes and investigated their capabilities and how these can be used for attacks. The results are depicted in Table II and Table III.

Allowed and Prohibited Changes. At the beginning of our analysis, we investigated which modifications can be included or removed within certified documents with respect to the defined modification permissions (i.e., `P1`, `P2`, and `P3`, cf., section VII). Without any surprises, annotations are allowed only in certified documents with `P3` and form modifications – in `P2` and `P3`. We determined three different categories with respect to possible changes on the document.

- *Changing Static Content*: Independent of the defined permissions (`P1`, `P2`, `P3`), the static text content cannot be changed. This restriction includes changes such as adding/removing/switching pages, replacing fonts, and replacing the text or images within a page.
- *Changing Forms*: We wondered which changes on forms are allowed if the permission is set to `P2` or `P3`. According to the specification, it is only allowed to modify the value of the form field. The modification of its appearance, for example, its position, color, and font, must not be allowed. Forms can also be used to insert digital signatures.
- *Adding/Removing/Modifying Annotations*: Annotations can be added if the permission is set to `P3`. According to the specification, there is no restriction on the type of annotation. In contrast to the specification, our analysis reveals that not all annotations are allowed (cf. Table II).

Capabilities of Allowed Changes. As a next step, we analyzed the capabilities of the allowed modifications. This process was the most time-consuming part of the evaluation since all features for each annotation and each form were analyzed. We identified four different categories:

- *Add/Hide Text*: These modifications are able to change the visible text content. Configurations such as font style, font size, position within the PDF document can be also specified. We argue that such modifications can insert new text content or overlay existing one.
- *Add/Hide Graphic*: Modifications belonging to this category are able to change the visualization of content by overlaying original graphics. These kind of modifications allow to either use images for displaying text or even to change graphics in the document.
- *Field Value*: With respect to forms, only modifications within the form in a predefined input mask are allowed.

Danger Level. We estimated the danger level for each modification and define four levels: *High*, *Medium*, *Low*, and *None*.

- *High*: The highest level results from the fact that the modification allows the insertion of text indistinguishable from the original one. Thus, a user opening a document cannot detect the inserted annotation and interprets the newly added content as part of the certified document.
- *Medium*: The level *Medium* covers modifications capable of hiding content. The user is then unable to detect that the modifications overlay some part of the document, for example, an important point of a contract.
- *Low*: Annotations with the level *Low* are potentially able to hide content of the original PDF, but the modification is visible for the user. For such modifications we abuse features like annotation icons shown in the PDF. Since the icon can be exchanged, one can define an icon overlaying content. Nevertheless, we could not find a way to change the icon without invalidating the certification. A residual risk remains, but we consider this to be low.
- *None*: All modifications which are not allowed to be used in certified documents have the level *None*.

B. Manual vs. Automated Approach

At the beginning of our research, we raised the question if the analysis of the PDF specification, which has more than 990 pages, can be automated. One possibility to extract security requirements from the documentation is to apply a lexicographic analysis based on natural language processing as described in [27, 28, 29, 30]. This approach can be applied

on API descriptions and similar guidelines since they define concrete validation steps. Thus, requirements can be extracted and evaluated. Unfortunately, this approach is not suitable to analyze a specification and determine if an introduced feature is potentially dangerous or not.

Error Guessing. We decided to apply an analyzing technique known from software testing called "error guessing" [31]. This approach is dependent on the experience of the security experts to guess a problematic part of the specification or the application. The result of such an analysis is a list of possible errors or dangerous features which need to be further analyzed. Although this analysis highly depends on the expertise of the analysts, it is often used to understand complex technologies where security considerations are missing. Thus, novel security issues can be discovered manually and in a second step can be executed automatically. This is an established approach in the security community [32, 33, 34, 35, 36, 37, 38, 26, 39, 40].

Semi-automated Test Case Generation. Once we created a list of possible errors and test cases, we concentrated on the creation process of the malicious PDFs. We decided to apply a semi-automated approach by using different scripts to create, sign, and modify a PDF document. A fully automated approach is not necessary since the number of test cases is manageable. Moreover, no variances of the test cases need to be created. Thus, fuzzing techniques are also considered out of scope.

Fully-automated Evaluation and Detection. In summary, we created 45 test cases which needed to be evaluated on 26 applications. This process was automated since it is inefficient to manually open $45 \cdot 26 = 1170$ PDF documents and evaluate them. Following this approach, we are also able to test newer or fixed versions of the applications. We also can detect maliciously crafted PDFs with EAA and SSA automatically, see subsection VIII-B.

V. Breaking PDF Certification

In this section, we present different attack techniques to break the integrity protection of certified documents. We found two specification flaws, which lead to security vulnerabilities in most PDF applications that are compliant to the PDF specification. The first one is the Evil Annotation Attack (EAA) and it breaks the `P3` permission (subsection V-A). The second one is the Sneaky Signature Attack (SSA), breaking the `P2` permission (subsection V-A). In addition, we apply obfuscation techniques through further implementation flaws, which allow us to hide the attacks based on specification flaws even better.

A. Evil Annotation Attack (Specification Flaw: Breaking P3)

The idea of the Evil Annotation Attack (EAA) is to show arbitrary content in a certified document by abusing annotations for this purpose. Since `P3` certified document allow to add annotations, EAA breaks the integrity of the certification.

Evaluating Permission P3. According to the specification, the following changes in a certified document with `P3` are allowed: 1) adding/removing/modifying annotations, 2) filling-out forms, 3) and signing the document. We started with an in-depth analysis of all annotations and their features. We evaluated 28 different annotations and classified these with respect to their capabilities and danger level. The results are depicted in Table II and will be further explained.

Annotation	Capabilities: Text Add	Text Hide	Image Add	Image Hide	Allowed in: P1	P2	P3	Danger Level
FreeText	✓	✓	✗	✓	−	−	⊞	High
Redact	✓	✓	✗	✗	−	−	⊞	High
Stamp	✗	✓	✓	✓	−	−	⊞	High
Caret	✗	✓	✗	✓	−	−	⊞	Medium
Circle	✗	✓	✗	✓	−	−	⊞	Medium
Highlight	✗	✓	✗	✓	−	−	⊞	Medium
Ink	✗	✓	✗	✓	−	−	⊞	Medium
Line	✗	✓	✗	✓	−	−	⊞	Medium
Polygon	✗	✓	✗	✓	−	−	⊞	Medium
PolyLine	✗	✓	✗	✓	−	−	⊞	Medium
Square	✗	✓	✗	✓	−	−	⊞	Medium
Squiggly	✗	✓	✗	✓	−	−	⊞	Medium
StrikeOut	✗	✓	✗	✓	−	−	⊞	Medium
Underline	✗	✓	✗	✓	−	−	⊞	Medium
FileAttachment	✗	✓	✗	✓	−	−	⊞	Low
Sound	✗	✓	✗	✓	−	−	⊞	Low
Text(Sticky Note)	✗	✓	✗	✓	−	−	⊞	Low
3D	✗	✓	✗	✓	−	−	−	None
Link	✗	✓	✗	✓	−	−	−	None
Movie	✗	✓	✗	✓	−	−	−	None
Popup	✗	✗	✗	✗	−	−	⊞	None
PrinterMark	✗	✗	✗	✗	−	−	−	None
Projection	✗	✗	✗	✗	−	−	−	None
RichMedia	✗	✓	✗	✓	−	−	−	None
Screen	✗	✗	✗	✗	−	−	−	None
TrapNet	✗	✗	✗	✗	−	−	−	None
Watermark	✓	✓	✓	✓	−	−	−	None
Widget	✗	✗	✗	✗	−	−	−	None

⊞ Usage allowed − Usage not allowed

Table II

List of all specified PDFs annotations, categorized according to: 1) their capabilities, 2) their permission in certified documents, and 3) the danger level with respect to their permission.

Danger Level of Annotations. We determined three annotations with a danger level *high* capable to hide and add text and images: `FreeText`, `Redact`, and `Stamp`. All three can be used to stealthily modify a certified document and inject malicious content. In addition, 11 out of 28 annotations are classified as *medium* since an attacker can hide content within the certified document. The danger level of the remaining annotations is classified as *low* or *none* since such annotations are either quite limited or not allowed in certified documents.

Attacking with Annotations. According to our attacker model, the attacker possesses a validly certified document allowing the insertion of annotations. To execute the attack, the attacker modifies a certified document by including the annotation with the malicious content at a position of attacker's choice. Then, the attacker sends the modified file to the victim who verifies the digital signature. The victim could detect the attack if it manually opens UI-Layer 3 or clicks on the

Figure 5. A certified document. The *Price per share* was manipulated by a `FreeText` annotation to show the value $100,000,000. The PDF viewer displays this annotation in UI-Layer 3. By deleting the `/Subtype` value the PDF object, it can be removed.

annotation. However, none of the tested PDF applications opened UI-Layer 3 automatically. Additionally, the attacker can lock an annotation to disable clicking on it.

Improving the stealthiness of EAA. To improve the attack, we elaborated techniques to prevent the annotation's visualization, so that it does not appear in UI-Layer 3. Surprisingly, we found a generic and simple bypass that can be applied to all annotations. PDF viewers identify annotations by their specified `/Subtype`. This `/Subtype` is also used by the viewer to assign the various editing tools, such as a text editor for `FreeText` comments. If the value of `/Subtype` is either missing or set to an unspecified value, whereby both cases are not prohibited according to the specification, the PDF viewer is unable to assign the annotation. As depicted in Figure 5, the annotation is not listed in UI-Layer 3. In summary, the annotation is indistinguishable from the original content.

Special Modifications. For some annotations, such as `FreeText` or `Stamp`, the editing tools of appropriate PDF applications can be easily used to completely design the visible content of a certified document. This is not the case for other annotations, which are classified as suitable for hiding text and images. The `Underline` annotation, for example, only creates a small line below the selected text. For hiding the text that is located below this line, the PDF object must be manually edited. By using a text editor, the thickness of the line can be adjusted within the annotation's appearance (parameter: `/N`) to hide the whole text. It is also possible to define the coordinates of an annotation to hide a particular area on a page. A special feature among the annotations is `Redact`. It allows new text to be placed over existing text. If the user moves the mouse over the text, the new text is displayed and hides the original text. To display this new text permanently, it is sufficient to redirect the object number (parameter: `/N`) to the object with the new text. Summarized, the specification does not restrict the size, color or characteristics of annotations and offers arbitrary possibilities to change the displayed content.

B. Sneaky Signature Attack (Specification Flaw: Breaking P2)

The idea of the Sneaky Signature Attack is to manipulate the appearance of arbitrary content within the PDF by adding overlaying signature elements to a PDF document that is certified at level `P2`.

Evaluating Permission P2. According to the specification, the following changes in a certified document with `P2` are allowed: filling-out forms, and signing the document. We started the analysis of forms as depicted in Table III and evaluated their capabilities.

Form	1) Capabilities					2) Allowed in			3) Danger Level
	Text		Graphic		Form	P1	P2	P3	
	Add	Hide	Add	Hide					
Signature	✔	✔	✔	✔	✖	−	⊞	⊞	High
Text Field	✖	✖	✖	✖	✔	−	⊞	⊞	None
Button Field	✖	✖	✖	✖	✔	−	⊞	⊞	None
Choice Field	✖	✖	✖	✖	✔	−	⊞	⊞	None

⊞ Usage allowed − Usage not allowed

Table III
A LIST OF ALL SPECIFIED PDFS FORMS. WE CATEGORIZED THEM BY 1) THEIR CAPABILITIES, 2) THEIR PERMISSION IN CERTIFIED DOCUMENTS, AND 3) THE DANGER LEVEL WITH RESPECT TO THEIR PERMISSION. ONE FORM IS CLASSIFIED AS HIGHLY DANGEROUS SINCE TEXT AND GRAPHICS CAN BE HIDDEN OR ADDED VIA IT.

Danger Level of Forms. According to our analysis, the danger level was *none* because the insertion of new form elements, customizing the font size and appearance, and removing form elements is prohibited. The only permitted change is on the value stored in the field. Thus, an attacker is not able to create forms which hide arbitrary content within the PDF document. Surprisingly, these restrictions are not valid for the signature field. By inserting a signature field, the signer can define the exact position of the field, and additionally its appearance and content. This flexibility is necessary since each new signature could contain the signer's information. The information can be a graphic, a text, or a combination of both. Nevertheless, the attacker can misuse the flexibility to stealthy manipulate the document and insert new content.

Attacking with Forms: SSA. The attacker modifies a certified document by including a signature field with the malicious content at a position of attacker's choice. The attacker then needs to sign the document, but he does not need to possess a trusted key. A self-signed certificate for SSA is sufficient. The only restriction is that the attacker needs to sign the document to insert the malicious signature field. This signing information can be seen by opening the PDF document and showing detailed information of the signature validation. In

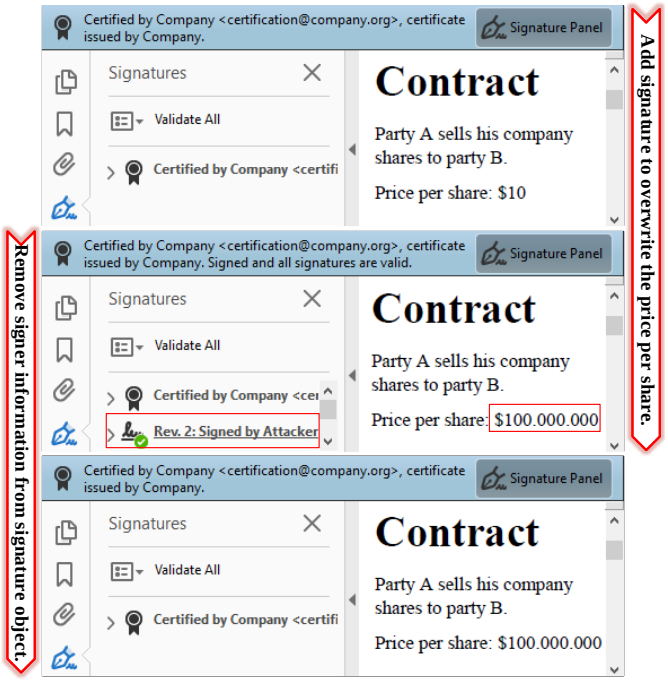


Figure 6. A certified document. The *Price per share* was manipulated using a sneaky signature which overwrites the price with $100,000,000. The PDF viewer displays this signature in UI-Layer 2. By manipulating the signature object, the signer information can be removed.

this case, the victim opening the file can get suspicious and refuse to accept the document, even though the certification is valid.

Improving the Stealthiness of SSA. To circumvent this limitation, we found a bypass to hide this information in UI-Layer 2. Thus, the victim is not able to determine the attacker's manipulations (see Figure 6). Basically, we have three tasks to improve the attack execution: 1) hide the signature information in the signature panel on UI-Layer 2 , 2) skip the validation of attacker's signature, and 3) make the signature field read-only to make it indistinguishable from the text content. To solve all tasks, we need to adjust one object – the one responsible for the appearance of the signature. It contains three relevant parameters: `/P`, `/V`, and `/Ff`. The `/P` is a reference to the page where signature should be displayed. We found out that if this reference is not valid, the signature disappears from the signature panel on UI-Layer 2, but the malicious content is still shown on the page. A signature added to a PDF document is usually verified by processing its referenced signature data. If the stored cryptographic values are correct and the document is not manipulated within the signed area, the signature is technically valid. The `/V` parameter references the signature value which needs to be validated. We found out that if this reference is also invalid, the signature validation is skipped. Finally, we set the parameter `/Ff` to `1` which means that the content is read-only. If a certified document is opened in a common PDF application, signatures can only be added to free signature fields provided by the certifier. Adding empty signature fields is normally no longer possible within the application. However, the specification does not prohibit adding empty signature fields to a certified document. By using frameworks like Apache PDFBox[2], empty signature fields can be placed anywhere in the document and filled with arbitrary content.

C. Limitations of EAA and SSA

Both attacks can be detected by searching for a specific text which is hidden behind the annotation or the signature. The editor signals that a searched term is found but the user is unable to see it. Another limitation could occur dependent of the UI Layer. In the default configuration, most PDF applications do not show the applied annotations on UI-Layer 1. The evil annotations are also not shown on UI-Layer 2. Nevertheless, it should be mentioned that the UI Layer of some PDF applications can be configured to show all UI Layers after opening a PDF document.

D. It's Not a Bug, It's a Feature

We classified EAA and SSA as vulnerabilities in the PDF specification. Considering the fact that the person certifying the document could know that additional signatures and annotations might be added to the document, the risks caused by these attacks should be known and accepted by all involved entities. However, our attacks reveal that signatures and annotations can 1) be customized to appear as a normal text/images above the signed content, 2) they can be indistinguishable from the original content, and 3) their indications can be hidden from UI Layers. Only 3) requires application implementation issues. Studying the PDF specification and guidelines regarding the validation of certified documents, we did not find any security considerations mentioning the potential risks and summarizing the best practices. This leads to the assumption that the risks mentioned in this paper have been overlooked and need to be addressed on specification and implementation level.

E. Permission Mismatch

Besides the specification, PDF applications can also implement the basic verification of the permissions of certified documents wrongly. These issues enable prohibited changes. We determine two permission mismatches according to the allowed changes described in Table I:

- The adding of annotations and signatures is allowed regardless of the permission level `P1` / `P2`.
- Annotations are allowed to be added starting at permission level `P2`.

Faulty Permission Verification. As already described, the EAA and SSA attack classes require certain permission levels with regard to document certification. However, this restriction requires the correct implementation of the permission levels within the individual PDF implementations. If an application does not check the set permissions `P1` and `P2` at all, or not

[2]https://pdfbox.apache.org/

completely, the attack classes can be successfully executed even at lower permission levels. Editing functions within the PDF applications can be easily outsmarted, for example, to add annotations to PDFs with permission levels lower than `P3`. For this purpose, it is sufficient to manually adjust the permission level `P1` or `P2` of a certified document to `P3` using a text editor. Of course, this initially breaks the certification, since this corresponds to a change in the signed area. However, the invalid certification state is, in practice, no reason for the PDF application to prevent functions such as adding annotations or signatures. Now that an annotation has been added to the document, the permission level can be manually reset to the original value `P1` or `P2`. The signed area now corresponds to the initial state again and the certification is valid from a cryptographic point of view. The annotation is now outside the signed area within an Incremental Update. If a PDF application does not check when opening the PDF whether the attached Incremental Updates are allowed within the initial permission level, the execution of the attack classes EAA and SSA on a lower permission level is possible.

VI. Methodology: Automatic Viewer Analysis with PDF Tester

During our research, we created 45 attack vectors in certified documents. Each vector must be tested on each of the 26 viewer applications using the black box analysis method. The sheer number of resulting test cases clearly indicated, that a fully automated evaluation system is inevitable. To automate the creation and evaluation process, we developed the analysis tool *PDF Tester*. PDF Tester's workflow is depicted in Appendix C. PDF Tester's functionality can be divided into two tasks: (1) the creation of the application's screenshot, including the certification validation status and (2) the evaluation of the attack vector according to the screenshot's validation status. We implemented two approaches for this purpose: a pixel-based and a text recognition approach.

Screenshot Creation. The screenshot creation is used to document manipulation effects in the individual PDF application. The user enters the paths to the PDF files and PDF applications in PDF Tester. PDF Tester automates the opening of PDF documents in different PDF applications and automatically takes a screenshot that includes UI-Layer 1.

Screenshot Evaluation: Pixel-based. The screenshot evaluation compares all created screenshots on a pixel level. It needs a reference image (i.e., the ground truth) of the unmanipulated PDF document. The PDF Tester estimates the difference between the exploit screenshots and the reference. If the difference is close to zero, the tested PDF has a valid certification and thus, the manipulations did not invalidate the certification. The pixel-base approach is very fast, but it requires a uniform image template and offers no flexibility in the form and design of the content. All screenshots must have the exact same resolution. Offsets, for example, due to a window movement, or opening of sub-menus, are causing issues.

Screenshot Evaluation: Text-based. To analyze screenshots that do not have the properties for a direct pixel comparison, we implemented text recognition. For this purpose, we use the Tesseract Optical Character Recognition (OCR) engine. The screenshots to be analyzed and the terms to be searched for in them serve as input. PDF Tester extracts the entire text from the image files and compares it with provided search terms. This approach is slower than the pixel-based one, but it can be used to circumvent its limitations.

VII. Evaluation

In this section, we describe the results of our analysis. We created 45 certified documents during our research and tested 26 applications. The results are shown in Table IV.

A. Test Environment

To create and evaluate the certified documents, we used a three-stage test environment, divided into systems for certification, manipulation, and validation. The certifier's system is based on Windows 10 and uses Adobe Acrobat to create and certify the PDF documents. Based on their respective market shares [41, 42], this selection makes the best combination regarding a real-world scenario. The attacker's system uses the same software combination as the certifier's system. The victim's system splits up into systems with Windows 10, macOS Catalina, and Ubuntu 18.04.4 as a Linux derivative. The private keys used for certification are only available on the certification system.

B. Tested Applications

To analyze the handling of different PDF applications on regularly certified documents, we developed four sample documents. We found out that not all tested applications could handle certified documents correctly. The Master PDF Editor application did not show a single certified document as valid under macOS. PDF Studio 2019 in the Standard and Pro variants (i.e., Windows, macOS, and Linux) changed the certification status to unknown if any subsequent changes were added. Since this was also the case for permitted changes, such as the addition of annotations in `P3` or further signatures in `P2`, we were unable to make a statement about the certification status. Since an evaluation for Master PDF Editor (macOS) and PDF Studio 2019 was not possible due to the fuzzy implementation concerning certified documents, this application was excluded from further consideration. We additionally observed limited support for certified documents in PDF Editor 6 Pro and PDFelement Pro under macOS; a valid verification of the certification was only possible for documents without additional signatures.

C. Results

We evaluated all 26 PDF applications on each of the three UI Layers against EAA and SSA attacks. We used two different types of exploits for this purpose: 1) exploits that are compliant to the PDF specification and 2) exploits that

Application	Version	OS	PDF Specification Flaws *All exploits are compliant to the PDF specification* UI-Layer 1 EAA	UI-Layer 1 SSA	UI-Layer 2 EAA	UI-Layer 2 SSA	UI-Layer 3 EAA	Applications' Implementation Flaws *Attacks improving the stealthiness of EAA and SSA* UI-Layer 1 EAA	UI-Layer 1 SSA	UI-Layer 2 EAA	UI-Layer 2 SSA	UI-Layer 3 EAA
Adobe Acrobat Reader DC	2020.009.20074	Windows	●	○	●	○	○	●	●	●	●	●
Adobe Acrobat Pro 2017	2017.011.30171		●	○	●	○	○	●	●	●	●	●
Expert PDF 14	14.0.28.3456		◐	●	◐	○	○	●	●	◐	◐	●
Foxit PhantomPDF	9.7.1.29511		●	○	○	○	○	●	○	●	○	●
Foxit Reader	9.7.1.29511		●	○	○	○	○	●	○	●	○	●
LibreOffice Draw	6.4.2.2		◐	◐	◐	◐	◐[1]	◐	◐	◐	◐	◐[1]
Master PDF Editor	5.4.38		●	●	●	○	○	●	●	●	●	○
Nitro Pro	13.13.2.242		●	○	◐	○	○	●	○	◐	○	●
Nitro Reader	5.5.9.2		●	○	●	○	○	●	○	●	○	●
PDF Architect	7.1.14.4969		◐	●	◐	○	○	●	●	◐	◐	●
PDF Editor 6 Pro	6.5.0.3929		○[2]	●	○[2]	○	○[2]	○[2]	●	○[2]	◐	○[2]
PDFelement Pro	7.5.1.4782		○[2]	●	○[2]	○	○[2]	○[2]	●	○[2]	◐	○[2]
PDF-XChange Editor	8.0 (Build 336.0)		●	●	◐	○	○	●	●	◐	◐	●
Perfect PDF 8 Reader	8.0.3.5		●	○	●	○	○	●	●	○	●	○
Perfect PDF 10 Premium	10.0.0.1		●	○	●	○	○	●	●	○	●	○
Power PDF Standard	3.10.6687		◐	○	◐	○	○	◐	●	◐	●	●
Soda PDF Desktop	11.2.46.6035		◐	●	◐	○	○	●	●	◐	◐	●
Adobe Acrobat Reader DC	2020.009.20074	macOS	●	○	●	○	○	●	●	●	●	●
Adobe Acrobat Pro 2017	2017.011.30171		●	○	●	○	○	●	●	●	●	●
Foxit PhantomPDF	3.4.0.1012		●	○	●	○	○	●	○	●	○	●
Foxit Reader	3.4.0.1012		●	○	●	○	○	●	○	●	○	●
PDF Editor 6 Pro	6.5.0.3929		○[2]	○	○[2]	○	○[2]	○[2]	○	○[2]	○	○[2]
PDFelement Pro	7.5.9.2925.5460		○[2]	○	○[2]	○	○[2]	○[2]	○	○[2]	○	○[2]
LibreOffice Draw	6.4.2.2		◐	◐	◐	◐	◐[1]	◐	◐	◐	◐	◐[1]
LibreOffice Draw	6.4.2.2	Linux	◐	◐	◐	◐	◐[1]	◐	◐	◐	◐	◐[1]
Master PDF Editor	5.4.38		●	●	●	○	○	●	●	●	●	○
∑ Applications that are *vulnerable* ●, max 26			15	8	11	0	0	18	15	11	9	15
∑ Applications that are *limited vulnerability* ◐, max 26			7	3	9	3	3	4	3	9	9	3

● Vulnerable: Attack is undetectable on the UI Layer.
◐ Limited Vulnerability: Attack is undetectable on the UI Layer but a general notification is shown.
○ Secure: Attack is clearly detectable on the UI Layer.

[1]LibreOffice does not provide a UI-Layer 3 and attacks can, henceforce, not be detected.
[2]Every kind of annotation, whether it is allowed or not, leads to an invalid certification.

Table IV
WE EVALUATED 26 DIFFERENT PDF APPLICATIONS AGAINST EAA AND SSA. THE APPLICATION IS VULNERABLE ● IF THE ATTACK IS UNDETECTABLE, THAT IS, IF NO ERROR OR SIGNATURE WARNING IS SHOWN. IF THE APPLICATION SHOWS A GENERIC INFORMATION MESSAGE, WE CALL IT A LIMITED VULNERABILITY ◐. WE EVALUATED THE ATTACK SUCCESS ON EACH DIFFERENT UI LAYER. ATTACK DETECTION ON DEEPER UI LAYERS MEANS THAT THE ATTACK IS HARDER TO DETECT, BECAUSE THE VICTIM HAS TO INSPECT MULTIPLE APPLICATION PANELS.

improved the stealthiness of the attacks by abusing implementation flaws, for example, by parsing errors. The results are depicted in Table IV.

1) Abusing PDF Specification Flaws: The middle part of Table IV shows the results for all 26 PDF applications when using exploits that abuse PDF specification flaws.

UI-Layer 1. The most critical UI Layer from the attacker's perspective is UI-Layer 1, because it is the only layer that automatically displays the signature status by opening the PDF. On this layer, 15 applications are vulnerable ● to EAA and 7 have limited vulnerabilities ◐. The SSA attack is less successful: 8 applications are vulnerable ● and 3 have limited vulnerabilities ◐. PDF Editor 6 Pro and PDFelement Pro revealed a notable behavior: whenever an annotation is added to a certified document, the signature validation status is invalid. Although this behavior is not compliant with the PDF specification, it prevents all our attacks.

UI-Layer 2. One could guess that the more profound the UI Layer is, the more attacks could be detected. Our evaluation confirms this assumption since most applications detected the SSA attack on UI-Layer 2, only LibreOffice have limited vulnerabilities ◐. This results from a bug in LibreOffice that causes no signatures to be displayed in UI-Layer 2.

UI-Layer 3. UI-Layer 3 is only relevant for EAA. The SSA attack could not be detected on UI-Layer 3, because SSA adds a signature which does not appear in the UI element showing PDF annotations. For UI-Layer 3, the EAA attack could be detected in all cases. The only exception is LibreOffice Draw, because it does not provide a dedicated panel that lists all PDF annotations.

2) Abusing Applications' Implementation Flaws: The right part of Table IV depicts the results for all 26 PDF application when using exploits that improve the attacks' stealthiness by abusing implementation flaws. In the following section, we

compare UI-Layer 1 for specification (i.e., the middle part) with implementation flaws (i.e., the right part).

UI-Layer 1. We could find 3 further vulnerable ● applications for EAA: Expert PDF 14, PDF Architect, and SodaPDF. For SSA, we could find vulnerabilities ● in 7 further applications: Adobe Acrobat Reader DC and Pro 2017 (Windows and macOS), Perfect PDF 8 Reader and 10 Premium, and Power PDF Standard.

UI-Layer 2. The attack that leverages implementation flaws the most is SSA. While the specification compliant attacks had only a few successes, the improved attacks lead to 9 vulnerable ● and 9 limited vulnerable ◐ applications. For EAA, two further applications are vulnerable ●: Foxit PhantomPDF and Foxit Reader.

UI-Layer 3. Similarly to SSA on UI-Layer 2, the EAA attack could be drastically improved on UI-Layer 3 when using additional implementation flaws. In total, 15 applications were vulnerable ● and 3 had limited vulnerabilities ◐.

Permission Implementation Analysis. For our evaluation in Table IV, we used `P2` certified documents for SSA and `P3` certified documents for EAA. This restriction raises the question of how permissions are validated in general. Firstly, the SSA attacks that work for an application on `P2` work in the same way for `P3`. Secondly, when considering attacks for *lower* permission levels, that is, EAA for `P2` or `P1`, respective SSA for `P1`, it depends on the application's implementation of those permissions. According to the PDF specification, these kinds of attacks should be impossible in those cases. However, we conducted an analysis of the permission behavior of these applications. We revealed that 11 of 26 applications revealed incorrectly implemented permissions, see Appendix A. In order to analyze how the applications reacted to manipulations prohibited by the permission levels `P1` or `P2`, annotations, like stamps (image files) and free text comments, were placed within a `P2` certified document. In addition to annotations, existing forms were filled out in a `P1` certified document. For PDF Architect and Soda PDF we have seen the partial implementation of the permissions. For example, level `P1` is implemented and any subsequent change is penalized with an invalid certification, while no distinction is made between `P2` and `P3`, and annotations are classified as permitted from `P2` onwards. From an attacker's perspective, this means that for these 11 applications, the attack classes EAA and SSA can be executed at lower permission levels.

Additional Findings. For Foxit Reader and Foxit PhantomPDF (Windows and macOS), the implementation of the individual permissions conformed to the specification. However, we discovered a serious bug that completely overrides signature and certification validation for signed and certified documents in `P2` and `P3`. If the order of the incremental update of *body*, *xref*, *trailer* to *xref*, *trailer*, *body* is swapped and the *xref* table is adjusted according to the new byte values, the PDF document can be completely changed without invalidating the certification or signature.

VIII. Countermeasures

We elaborated short-term and long-term countermeasures, which we explain further in this section.

A. Long-Term Countermeasures: Fixing the PDF Specification

Preventing Evil Annotations. With the availability of many permitted annotations at permission level `P3`, there is a large arsenal to manipulate the appearance of the content of a certified document. A particular risk is posed by the `FreeText`, `Stamp` and `Redact` annotations, as they allow new content such as text or images to be inserted into a certified document. Even without using the EAA techniques for hiding inserted annotations, they pose a great risk of tricking normal users. Therefore, these three annotation types should be classified as prohibited within the PDF specification for use within certified documents. The remaining annotations can be used to hide existing text or images and should be limited in their attributes. For example, a line of type `Underline` or `StrikeOut` should never be larger than the underlying text part. This could be achieved by calculating the amount of collision between two rectangles using the `/BBox` coordinates, when taking into account the line thickness. In case of overlap, the integrated editing tool should reject the drawing with a corresponding message. To capture manually created incremental updates, collision calculations should also be performed during certification validation. An empty or undefined value for the `/Subtype` element must also be penalized with an invalid certification status.

Preventing Sneaky Signatures. In practice, annotations within a certified document can often be omitted. Therefore, a lower permission level can be chosen as a precaution. Unfortunately, this does not apply to signatures to the same extent. In many situations, it may be useful and necessary to allow the addition of signatures after certification. For example, the certified document can be signed by multiple contract partners. However, to prevent attacks of the SSA class, signature fields must be set up at defined locations in the PDF document before the document is certified. A subsequent addition of signature fields must be penalized with an invalid certification status. Otherwise, it can always be used to add text or images included in the signature at any position. Within our analysis, the contained fieldtype `/FT` with the value `/Sig` was decisive for whether an object was identified as a signature and thus classified as a permitted change. Nevertheless, it was possible to redirect or omit the reference to the signature data `/V`, and resulted in the signature not being validated and thus not being listed in UI-Layer 2. Therefore the specification should show the parameter `/V` as mandatory and not optional. Suppose the signature cannot be validated due to missing or incomplete signature data. In that case, it should be listed as an invalid signature in UI-Layer 1 and UI-Layer 2.

B. Short-term Countermeasures

PDF-Detector. We analyzed the possibilities to provide a short-term countermeasure which is standard compliant. The

main cause for the vulnerabilities described in this paper is the overlay over the original content by using annotations or signatures. We determined that we can detect such an overlay by analyzing the position of annotations and signatures within the document and estimating if these intersect with some content. If such an intersection is found, a warning can be thrown. We implemented a tool called *PDF-Detector* which is capable of detecting EAAs and SSAs. PDF-Detector is available as an online service at http://pdf-demos.de and as an open-source library. The PDF-Detector is a python based tool which takes certified documents as an input and produces a report whether dangerous elements were found in the PDF document, see Listing 2. As a first step, PDF-Detector analyzes if the submitted PDF is digitally signed and if the signature is a certification or *approval signature*. The PDF-Detector evaluates the document's permission level and estimates if any Incremental Updates are applied. If true, PDF-Detector determines if the appended elements within the Incremental Update are allowed according to the permission level. If they are denied, an error is thrown, and the report's status is set to `error`. Otherwise, PDF-Detector determines the type of the appended elements, for example, a `FreeText` annotation or a signature. Independent of the element's type, the `changes-danger-level` is defined. The values corresponds to the values depicted in Table II and Table III. Finally, PDF-Detector analyzes each annotation or signature position and estimates the intersection with the content of the page. If such an intersection is found, the `changes-danger-level` is raised to `very high`. PDF-Detector does not provide any cryptographic signature validation. The reason for this decision was that the management of trusted or revoked certificates and the support of standards like PAdES [43] and CAdES [44] are considered out of scope and irrelevant for the attacks described in this paper.

Visible Panel for Annotation and Signatures. To reduce the attacks' stealthiness, we also recommend making annotations and additional signatures visible on UI-Layer 1. Currently, none of the tested PDF applications do this. Thus, the attacks can only be detected if the user pro-actively looks into the PDF application's corresponding panels.

IX. High Privileged JavaScript Code Execution

A dedicated feature of certified documents is executing high privileged JavaScript code within Adobe Acrobat Pro and Reader (Windows and macOS). This section describes how the attack classes EAA and SSA can be used to inject and execute JavaScript code into certified documents.

Overview. JavaScript code can be embedded in PDF documents to provide various interactive features. The code execution is started by several triggers, such as opening a page within the PDF document. In the past, researchers used the JavaScript functionality to execute malicious code [45, 46, 47, 48]. Adobe requires that potentially dangerous functions can only be executed as high privileged JavaScript code to address these issues. An example of such a function is calling a website without asking or warning the user. Suppose a certified document contains JavaScript code and the certificate used is fully trusted. In that case, the execution of high privileged JavaScript code occurs without asking the user [49, 50].

Using EAA and SSA to inject JavaScript. For annotations and signature fields, it is possible to pass a reference to an object containing JavaScript. For this purpose, a trigger is chosen within an additional-actions dictionary `/AA`. To execute the JavaScript code directly upon opening the document, the value `/PO` is suitable and which triggers the code execution when opening the page. Suppose the EAA or SSA is placed on the first page of the PDF document. In that case, the code execution starts immediately after the document is opened. The object referenced by `/PO` contains the JavaScript code. The following example opens the website https://www.malicious.org/ using the system's default browser. The victim is unable to prevent this call. The attack is not limited to calling up a website but can execute any high privileged JavaScript code. The only requirement is that the victim fully trusts the certificate used to certify the PDF document. We were able to identify a total of 117 JavaScript methods that are protected by special security restrictions and, for this reason, can be executed only in a privileged context [49].

X. Future Research Directions

Page Templates. In addition to the permitted modifications described in this paper, certified documents of permission levels `P2` and `P3` offer the possibility to instantiate page templates. However, the page templates must already be included in the document before certification. This restriction results in a completely different attacker model which was introduced in 2021 by Mainka et al. [51]. In this case the attacker must manipulate the document before it is certified. For this paper, an analysis of page templates is, thus, out of scope, but offers an interesting research aspect for future work.

Design of the update mechanism. Incremental Updates introduce a desired feature to flexibly extend PDF documents. From a security perspective, however, Incremental Updates enable a great attack surface allowing the stealthy manipulation of digitally signed documents. The current countermeasures are only a patchwork and do not introduce a systematic approach to address the existing security issues. Future research should concentrate on new design concepts which allow the extendability of PDF documents without weakening the integrity protection.

Editing Restrictions. Microsoft Office provides restrictions such as: only allowing to fill-in forms, add comments, track changes, or open a document as read-only [52]. This restriction is similar to the `P2` restrictions in PDFs. Microsoft Office also allows applying multiple signatures with different permission levels. The question if similar attacks like EAA and SSA can be applied is obvious. Future research projects should analyze the Open Document Format (ODF), Office Open XML

(OOXML) format, and all previously used but still supported versions like *ECMA-376 Office Open XML*.

Privilege Escalation. Similar to PDFs, other documents like `*.docx` or `*.odf` can contain macros. Due to previously discovered security issues leading to remote code execution, the processing of macros is restricted. It leads to warnings by opening the document. Such a warning can be skipped if the document is digitally signed and the validation is successful. If attacks can bypass the signature validation, they can inject arbitrary macro code, which is stealthily processed after the victim opens the document. We adapted the attack described in section IX on LibreOffice as a proof-of-concept and reported it. The responsible disclosure process is not yet finalized.

Usability of Validation Warnings. During our evaluation, we estimated a large number of confusing warnings during the validation process. Such warnings include a wide range of different messages and symbols independent of the used application. Even for experts who know how PDFs work and what behavior is expected from the application, it was challenging to recognize the validation result and classify the attack's success. We strongly recommend a usability study that addresses the following questions: 1) How users handle signed and certified documents and raised warnings? 2) How the usability problems with respect to warnings can be systematically analyzed? 3) What are best current practices that can be used by developers to improve the validation results presentation? Such a study could be broaden by extending the scope to different formats like MS Office, Open Document Format, AutoCAD, and DWFx Drawings.

XI. Related Work

PDF Signatures. In 2008 and 2012, I. Grigg [54] and [53] described attacks on electronic signatures. Their attacks highlighted the abuse of missing cryptographic protection. In 2010, Raynal et al. [55] concentrated on the security of the applied certificate chain and criticized the design of the trust establishment. In 2015, Lax et al. [56] considered signed documents containing dynamic content like macros or JavaScript. The authors highlighted the possibility of changing the dynamic content without invalidating the signature. 2017 presented the first attacks bypassing the cryptographic protection in PDFs [57]. They successfully created two different PDF files resulting in the same SHA-1 digest value. The scope of their research was the collision resistance of SHA-1 and the PDF files were used as a proof-of-concept. In 2018, Domingues and Frade [58] studied the occurrence of digitally signed and permission-protected PDFs. The authors found multiple problems related to digitally signed PDFs like expired or revoked certificates. No attacks bypassing the integrity protection of PDFs or the evaluation of previously discovered attack vectors was made. In 2019, Mladenov et al. [26] published a comprehensive study regarding the security of PDF signatures and they discovered three novel attacks and revealed all current applications vulnerable. In 2021, Mainka et al. [51] revealed *Shadow Attacks* on signed PDFs. Their attack embedded hidden content into a PDF. Once a victim signs that PDF, they could uncover the hidden content while keeping the signature valid. In both works [26, 51], the authors concentrated only on approval signatures and left certified documents out of scope.

Polyglot Attacks. In 2005, Buccafurri [59] described a novel file format attack where the attacker forces two different views of the same signed document by switching the file format between `BMP` and HTML [59]. PDF files are mentioned as a possible target for such an attacker, but no concrete ideas are described. Other research combines file formats of PDF and image format [60, 61, 62]. Depending on the viewer in use, different content is shown. He combined a PDF and a JPEG into a single polyglot file.

Security Apart from PDF Signatures. In 2005, McIntosh and Austel [63] described issues in partially signed documents with the XML rewriting attack. Somorovsky et al. [64] adapted the attack in 2012 to SAML-based single sign-on.

XII. Conclusion

Certified documents enable complex and highly desired use-cases. In contrast to approval signatures on PDFs, certified documents allow certain changes to the PDF after its certification, such as filling out forms or adding annotations and further signatures. However, the devil is in the specification details. This 994-page specification grants great flexibility in using these changes. In this paper, we shed light on the abuse of annotation, forms, and signatures. We misused their specified features to change the visible content of a certified document and, thereby, introduced two specification flaws: the Evil Annotation Attack (EAA) and the Sneaky Signature Attack (SSA). Although neither EAA nor SSA can change the content itself – it always remains in the PDF – annotations and signature fields can be used as an overlay to add new content. Victims opening the PDF are unable to distinguish these additions from regular content. And even worse: annotations can embed high privileged JavaScript code that is allowed to be added to certain certified documents. We proposed countermeasures to prevent the vulnerabilities. However, the underlying problem remains: the flexibility to protect the integrity of parts of documents, while allowing to change other parts, is manifold. The research community has struggled with similar problems on other data formats, such as XML or Email, without finding a satisfying solution so far. In the case of PDF, the specification must be updated to address these issues.

Acknowledgment

Simon Rohlmann was supported by the German Federal Ministry of Economics and Technology (BMWi) project "Industrie 4.0 Recht-Testbed" (13I40V002C). Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972. We acknowledge the University of Konstanz for supporting Vladislav Mladenov.

References

[1] Adobe Inc. (2020, Jun.) Adobe Fast Facts. [Online]. Available: https://www.adobe.com/content/dam/cc/en/fast-facts/pdfs/fast-facts.pdf

[2] European Parliament and Council of the European Union. (2014, Jul.) Regulation (EU) No 910/2014 of the European Parliament and of the Council of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC. [Online]. Available: https://eur-lex.europa.eu/legal-content/DE/TXT/?uri=CELEX:32014R0910

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

[4] Uniform Law Commission. Electronic Transactions Act. [Online]. Available: https://www.uniformlaws.org/committees/community-home/librarydocuments?communitykey=2c04b76c-2b7d-4399-977e-d5876ba7e034&tab=librarydocuments

[5] Adobe Inc. What is a Certified Document and when should you use it? [Online]. Available: https://blogs.adobe.com/security/2012/03/what-is-a-certified-document-and-when-should-you-use-it.html

[6] Lakehead University. Electronic Approval Standards. [Online]. Available: https://www.lakeheadu.ca/sites/default/files/profile-data/dcataldo/ElectronicApprovalStandards.pdf

[7] Bank of Italy (Banca d'Italia). USER'S MANUAL SOFTWARE TO SIGN AND ENCRYPT DOCUMENTS. [Online]. Available: https://www.bancaditalia.it/footer/firmadigitale/Software_manual.pdf?language_id=1

[8] Certipost. Definitions and Acronyms. [Online]. Available: http://www.certipost.org/wp-content/uploads/2015/06/DaA_CTP_TSP_V1_0.pdf

[9] United States Government Publishing Office (GPO). Authentication. [Online]. Available: https://www.govinfo.gov/about/authentication

[10] ——. Congressional Bills. [Online]. Available: https://www.govinfo.gov/app/collection/bills/

[11] ——. Collection of Certified Documents by the United States Government Publishing Office (GPO). [Online]. Available: https://www.govinfo.gov/app/

[12] Legislative Assembly of British Columbia. Digitally Signed PDFs. [Online]. Available: https://www.leg.bc.ca/content-hansard/Pages/Digital-Signatures.aspx

[13] European Telecommunications Standards Institute (ETSI), "Electronic signatures and infrastructures (esi); pdf advanced electronic signature profiles; part 4," Tech. Rep., 2009. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/102700_102799/10277804/01.01.01_60/ts_10277804v010101p.pdf

[14] I. DocuSign. (2018, oct) Docusign validation service. [Online]. Available: https://validator.docusign.com/

[15] Adobe Inc. Adobe Acrobat Online Service. [Online]. Available: https://www.adobe.com/acrobat/online.html?promoid=85665T9B&mv=other

[16] E. Commission. (2018, oct) Dss demonstration webapp v5.3.1. [Online]. Available: https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/DSS

[17] R. U. T. REGULIERUNGS-GMBH. (2018, oct) Rtr - signatur-prüfung. [Online]. Available: https://www.signatur.rtr.at/de/vd/Pruefung.html

[18] A. Group. (2018, Oct.) Ellis digital signature. [Online]. Available: https://ellis.arhs-spikeseed.com/

[19] E. doo. (2018, Oct.) Vep e-obrazci. [Online]. Available: https://www.vep.si/validator/forms/document-verify

[20] A. for Digital Italy. (2018, oct) Dss demonstration webapp v5.2. [Online]. Available: https://dss.agid.gov.it/validation

[21] eesti. (2018, oct) Siva demo application. [Online]. Available: https://siva-arendus.eesti.ee/

[22] Evrotrust. (2018, Oct.) Validate a signature. [Online]. Available: https://www.evrotrust.com/landing/en/a/validation

[23] iText PDF. iText Online PDF Service. [Online]. Available: https://itextpdf.com/en/

[24] intarsys. intarsys Online PDF Service. [Online]. Available: https://www.intarsys.de/

[25] ISO, *ISO 32000-2:2017 - Document management – Portable Document Format – Part 2: PDF 2.0*, 1st ed., Sep. 2017.

[26] V. Mladenov, C. Mainka, K. Meyer zu Selhausen, M. Grothe, and J. Schwenk, "1 trillion dollar refund – how to spoof pdf signatures," in *ACM Conference on Computer and Communications Security*, Nov. 2019.

[27] Y. Chen, L. Xing, Y. Qin, X. Liao, X. Wang, K. Chen, and W. Zou, "Devils in the guidance: Predicting logic vulnerabilities in payment syndication services through automated documentation analysis," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 747–764. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/chen-yi

[28] T. Lv, R. Li, Y. Yang, K. Chen, X. Liao, X. Wang, P. Hu, and L. Xing, "Rtfm! automatic assumption discovery and verification derivation from library document for api misuse detection," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1837–1852. [Online]. Available: https://doi.org/10.1145/3372297.3423360

[29] M. Acharya and T. Xie, "Mining api error-handling specifications from source code," in *International Conference on Fundamental Approaches to Software Engineering*. Springer, 2009, pp. 370–384.

[30] H. A. Nguyen, R. Dyer, T. N. Nguyen, and H. Rajan, "Mining preconditions of apis in large-scale code

corpus," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 166–177.

[31] B. Homès, *Fundamentals of Software Testing*, ser. ISTE. Wiley, 2013. [Online]. Available: https://books.google.de/books?id=z1XDNlSceFkC

[32] K. G. Paterson, T. Ristenpart, and T. Shrimpton, "Tag size does matter: Attacks and proofs for the tls record protocol," in *Proceedings of the 17th International Conference on The Theory and Application of Cryptology and Information Security*, ser. ASIACRYPT'11. Berlin, Heidelberg: Springer-Verlag, 2011, p. 372–389. [Online]. Available: https://doi.org/10.1007/978-3-642-25385-0_20

[33] H. Böck, J. Somorovsky, and C. Young, "Return of bleichenbacher's oracle threat (ROBOT)," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, pp. 817–849. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/bock

[34] K. Bhargavan, A. D. Lavaud, C. Fournet, A. Pironti, and P. Y. Strub, "Triple handshakes and cookie cutters: Breaking and fixing authentication over tls," in *2014 IEEE Symposium on Security and Privacy*, 2014, pp. 98–113.

[35] M. Heiderich, J. Schwenk, T. Frosch, J. Magazinius, and E. Z. Yang, "Mxss attacks: Attacking well-secured web-applications by using innerhtml mutations," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 777–788. [Online]. Available: https://doi.org/10.1145/2508859.2516723

[36] S. Lekies, K. Kotowicz, S. Groß, E. A. Vela Nava, and M. Johns, "Code-reuse attacks for the web: Breaking cross-site scripting mitigations via script gadgets," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1709–1723. [Online]. Available: https://doi.org/10.1145/3133956.3134091

[37] Y. Zhang, X. Zheng, Z. Wang, G. Ai, and Q. Huang, "Implementation of a parallel gpu-based space-time kriging framework," *ISPRS Int. J. Geo Inf.*, vol. 7, no. 5, p. 193, 2018. [Online]. Available: https://doi.org/10.3390/ijgi7050193

[38] J. Müller, F. Ising, V. Mladenov, C. Mainka, S. Schinzel, and J. Schwenk, "Practical decryption exfiltration: Breaking pdf encryption," in *ACM Conference on Computer and Communications Security*, Nov. 2019.

[39] M. Grothe, C. Mainka, P. Rösler, and J. Schwenk, "How to break microsoft rights management services," in *10th USENIX Workshop on Offensive Technologies (WOOT 16)*. Austin, TX: USENIX Association, Aug. 2016. [Online]. Available: https://www.usenix.org/conference/woot16/workshop-program/presentation/grothe

[40] J. Müller, F. Ising, C. Mainka, V. Mladenov, S. Schinzel, and J. Schwenk, "Office document security and privacy," in *14th USENIX Workshop on Offensive Technologies (WOOT 20)*. USENIX Association, Aug. 2020. [Online]. Available: https://www.usenix.org/conference/woot20/presentation/muller

[41] Statista, Inc. (2019, Sep.) Operating Systems - Statistics & Facts. [Online]. Available: https://www.statista.com/topics/1003/operating-systems/

[42] Datanyze. Adobe Acrobat DC Market Share and Competitor Report. [Online]. Available: https://www.datanyze.com/market-share/other-sales-software--408/adobe-acrobat-dc-market-share

[43] European Telecommunications Standards Institute (ETSI), "Electronic signatures and infrastructures (esi); pdf advanced electronic signature profiles; part 1," Tech. Rep., 2009. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/102700_102799/10277801/01.01.01_60/ts_10277801v010101p.pdf

[44] ——, "Electronic signatures and infrastructures (esi); cades baseline profile," Tech. Rep., 2012. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/103100_103199/103173/02.01.01_60/ts_103173v020101p.pdf

[45] J. M. Esparza. (2014, Oct.) Pdf attack - a journey from the exploit kit to the shellcode. [Online]. Available: https://www.blackhat.com/docs/eu-14/materials/eu-14-Esparza-PDF-Attack-A-Journey-From-The-/Exploit-Kit-To-The-Shellcode.pdf

[46] V. Hamon, "Portable Document Format (PDF) Security Analysis and Malware Threats," *Black Hat Abu Dhabi*, 2012.

[47] P. Stokes. (2019, Mar.) Malicious pdfs — revealing the techniques behind the attacks. [Online]. Available: https://www.sentinelone.com/blog/malicious-pdfs-revealing-techniques-behind-attacks/

[48] R. Brandis and L. Steller, "Threat Modelling Adobe PDF," *Command, Control, Communications and Intelligence Division, DSTO, Australien*, 2012.

[49] Adobe Systems Incorporated, *JavaScript for Acrobat API Reference - Adobe Acrobat SDK 8.1*, Apr. 2007.

[50] Adobe Inc. (2020, Jul.) Javascript controls - acrobat application security guide. [Online]. Available: https://www.adobe.com/devnet-docs/acrobatetk/tools/AppSec/javascript.html

[51] C.-t.-a. Mainka, V.-d.-l. Mlade nov, and S. Rohlmann, "Shadow Attacks: Hiding and Replacing Content in Signed PDFs," in *Proceedings 2019 Network and Distributed System Security Symposium*. Internet Society, Feb. 2021.

[52] Microsoft. (2020) Allow changes to parts of a protected document. [Online]. Available: https://support.microsoft.com/en-us/office/allow-changes-to-parts-of-a-protected-document-187ed01c-8795-43e1-9fd0-c9fca419dadf

[53] I. Grigg. (2008) Technologists on signatures: looking in the wrong place. [Online]. Available: http:

//financialcryptography.com/mt/archives/001056.html
[54] I. Grigg. (2012) Signatures on fax & email - if you did not intend to be bound, why did you bother to write it? [Online]. Available: http://financialcryptography.com/mt/archives/001364.html
[55] F. Raynal, G. Delugré, and D. Aumaitre, "Malicious Origami in PDF," *Journal in Computer Virology*, vol. 6, no. 4, pp. 289–315, 2010. [Online]. Available: http://esec-lab.sogeti.com/static/publications/08-pacsec-maliciouspdf.pdf
[56] G. Lax, F. Buccafurri, and G. Caminiti, "Digital document signing: Vulnerabilities and solutions," *Information Security Journal: A Global Perspective*, vol. 24, no. 1-3, pp. 1–14, 2015.
[57] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, "The first collision for full sha-1," in *Annual International Cryptology Conference*. Springer, 2017, pp. 570–596.
[58] P. Domingues and M. Frade, "Digitally signed and permission restricted pdf files: A case study on digital forensics," in *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ser. ARES 2018. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3230833.3232811
[59] F. Buccafurri, "Digital signature trust vulnerability: A new attack on digital signatures," *Information Management & Computer Security*, vol. 4, pp. 28–6, 2005. [Online]. Available: http://www.unirc.it/firma/en/Buccafurri_ISSA_1008.pdf
[60] F. Buccafurri, G. Caminiti, and G. Lax, "Fortifying the dalì attack on digital signature," in *Proceedings of the 2nd International Conference on Security of Information and Networks*, ser. SIN '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 278–287. [Online]. Available: https://doi.org/10.1145/1626195.1626262
[61] D. Popescu, "Hiding malicious content in PDF documents," *CoRR*, vol. abs/1201.0397, 2012. [Online]. Available: http://arxiv.org/abs/1201.0397
[62] A. Albertini, "This PDF is a JPEG; or, This Proof of Concept is a Picture of Cats," *PoC 11 GTFO 0x03*, 2014. [Online]. Available: https://www.alchemistowl.org/pocorgtfo/pocorgtfo03.pdf
[63] M. McIntosh and P. Austel, "XML signature element wrapping attacks and countermeasures," in *SWS '05: Proceedings of the 2005 Workshop on Secure Web Services*. New York, NY, USA: ACM Press, 2005, pp. 20–27.
[64] J. Somorovsky, A. Mayer, J. Schwenk, M. Kampmann, and M. Jensen, "On breaking saml: Be whoever you want to be," in *21st USENIX Security Symposium*, Bellevue, WA, Aug. 2012.

APPENDIX

A. List of Permission-Incompliant PDF Applications

The following applications do not correctly implement permission-level checks. This implementation issue enables the adaption of SSA to `P1` certified documents and EAA to `P1` and `P2` certified documents.

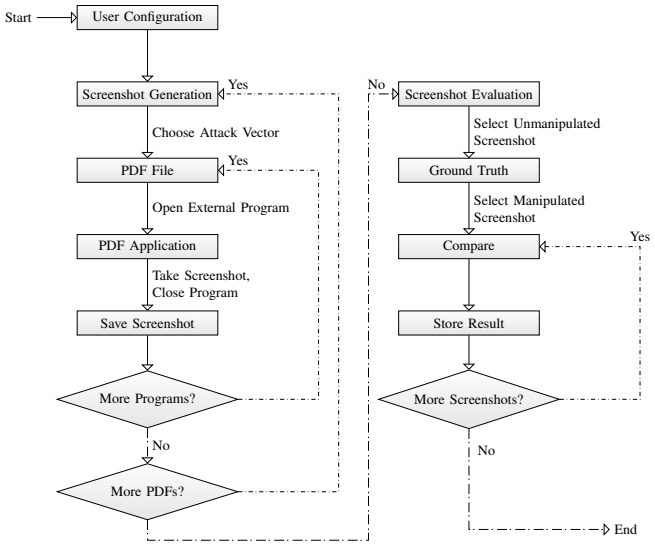
- Expert PDF 14, 14.0.28.3456, Windows
- LibreOffice Draw, 6.4.2.2, Windows
- Master PDF Editor, 5.4.38, Windows
- PDF Architect 7, 7.1.14.4969, Windows
- PDF-XChange Editor, 8.0 (Build 336.0), Windows
- Perfect PDF 8 Reader, 8.0.3.5, Windows
- Perfect PDF 10 Premium, 10.0.0.1, Windows
- Soda PDF Desktop, 11.2.46.6035, Windows
- LibreOffice Draw, 6.4.2.2, macOS
- Master PDF Editor, 5.4.38, Linux
- LibreOffice Draw, 6.4.2.2, Linux

B. Fixed Applications

The following applications have been reported to us by the vendors as fixed.

- Adobe Acrobat DC, 2021.001.20315, Windows
- Adobe Acrobat 2020, 2020.001.30020, Windows
- Adobe Acrobat 2017, 2017.011.30190, Windows
- Foxit PhantomPDF, 10.1.1, Windows
- Foxit Reader, 10.1.1, Windows
- LibreOffice, 7.0.4, Windows
- Adobe Acrobat DC, 2021.001.20315, macOS
- Adobe Acrobat 2020, 2020.001.30020, macOS
- Adobe Acrobat 2017, 2017.011.30190, macOS
- Foxit PhantomPDF, 4.1.1, macOS
- Foxit Reader, 4.1.1, macOS
- LibreOffice, 7.0.4, macOS
- LibreOffice, 7.0.4, Linux

C. PDF-Tester Workflow



PDF Tester Concept. The left part shows PDF Tester's screenshot generation. In the right part, PDF Tester's screenshot evaluation is depicted.

D. PDF-Detector Report

```
{
   "status": "OK/warning/error",
   "type": "approval/certified/none",
   "permission": "1/2/3/none",
   "incremental-update-changes": "annotation/signature/
       annotation+signature/exists/none",
   "changes-danger-level": "very high/high/medium/low/
       none",
   "message": ""
}
```

Listing 2. *PDF-Detector* returns a report as a JSON message. The message contains information if dangerous elements intersecting with original content occur in the document.

E. Priviliged JavaScript Execution in PDF

```
<<
  /JS (app.launchURL(
    "https://www.malicious.org/",
    true
  );)
  /S /JavaScript
>>
```

Listing 3. Privileged JavaScript execution automatically invoking a URL.