1 Trillion Dollar Refund – How To Spoof PDF Signatures

Aman Sharma — Universität Paderborn
Supervisor: Prof. Dr.-Ing. Juraj Somorovsky — System Security

Contents

1 Abstract

Since the release of PDF in the early 1990s by Adobe, this file format has gained tremendous popularity ranging from Private sectors such as E-commerce and Banking to Government sectors like Defense. PDF works independently of hardware and OS and is the main reason for its gain in popularity. It has also become a standard for writing an invoice and sharing critical information because of its security feature called Digital signature. Digital signatures ensure the authenticity and the integrity of the document. They are used to digitally sign the content of the PDF and work as a trademark of security. But due to out-of-date versions of PDF readers and lack of standards to verify signed PDFs, the content of these files can be easily manipulated. This paper contains such attack strategies and possible ways to protect PDF document from them. These attacks have been performed on popular PDF viewers available online and offline. The results during the evaluation phase showed that 95% of the major offline PDF viewers and 75% of online PDF validators were prone to these attacks. The work in this paper is a summary of the paper named '*Trillion Dollar Refund – How To Spoof PDF Signatures*'.

Keywords: PDF(Portable Document Format), Digital Signature

2 Motivation

The idea behind this research lies in the outburst of the number of PDF files created in the year 2015, raising the number to 2.5 trillion from 1.6 billion[10]. Along with the increase in the files, we have seen an increase in the number of malware attacks as well. In countries like the USA, in the government sector and in e-commerce, digital signatures are legally used in place of hand-written ones. Moreover, the lack of specific standards for validation of signed PDFs and coding errors make attacks more feasible.

3 Technical Background and basics of PDF

Before understanding the vulnerabilities in the signed PDF, it is necessary to know its structure. The following sub-sections explain the basic structure of the PDF and the PDF signature.

3.1 PDF Structure

PDF documents not only include text but can contain images, javascript, links to a web page. But every PDF has basically 4 components (refer figure 1):

1. **Header**: The header is the first line of the PDF file and it specifies the interpreter version to be used. It is written as %PDF-1.3 where a '%' sign represents a comment and 1.3 specifies the interpreter version.

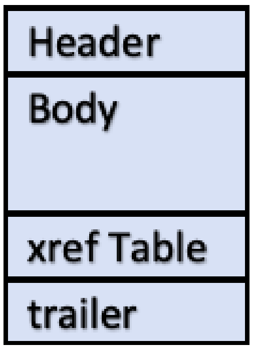
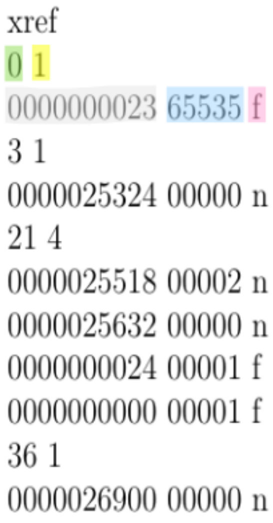


Figure 1: PDF Components

2. **Body**: The body is a list of objects such as Catalog, Pages, Page and Stream. The Catalog holds the reference to the Pages and the Pages object further refers to each Page object. Every object is described by an Object number, Generation number and a string constant "obj". The first object will have the Object number as 1 and every following object will have the Object number incremented by 1. The Generation number is incremented every time the object is modified. Document structure and permissions are defined in the Catalog object which is also the root object of the PDF whereas the Pages object contains information about the number of Page objects in the body component. The Page contains the document's data and information about a single page.

3. **Xref Table**: This is a cross-reference table that contains a reference (one entry is 20 bytes long) to all the objects in the PDF (see figure 2). This helps in random access of any object within the document.



```
xref
0 1
0000000023 65535 f
3 1
0000025324 00000 n
21 4
0000025518 00002 n
0000025632 00000 n
0000000024 00001 f
0000000000 00001 f
36 1
0000026900 00000 n
```

Figure 2: Xref Table Structure [6]

Each object in the Xref table has two parts ’a’ (in green colour) and ’b’ (in yellow colour) separated by a space. The parameter ’a’ describes the ID of the object and ’b’ describes the number of entries for the object. Here, the first object having ID 0 has 1 entry. Each object entry has further three parts namely ’x’ describing byte offset(in grey colour), ’y’ generation number(in blue colour), ’z’(in pink colour) separated by spaces. The ’z’ describes whether the object is free(denoted by ’f’) or in use(denoted by ’n’).

4. **Trailer**: PDF documents read PDF from the end of the file. The trailer refers to the Xref Table and the Catalog object and the last line of the trailer contains End of File string ‘*%%EOF*’. It starts with the string ’trailer’ and the contents are enclosed between ’«’ and ’»’. The /size tag specifies the number of entries in the Xref table. The /root tag describes a reference to the Catalog object. Before *%%EOF* there is a *startxref* that specifies the offset from the beginning of the file to the Xref table.

   For example[6]:
   trailer
   «
   /Size 22
   /Root 2 0 R
   /Info 1 0 R
   »
   startxref
   24212
   %%EOF.

3.2 PDF Signature

The PDF signature depends up on *incremental saving* that involves appending a new body (a Catalog and a signature object), a new Xref Table and a new trailer. The signature object present inside the body contains */Contents* and */ByteRange* parameters. The */Contents* stores the certificates and the calculated signature value. The signature object also holds the information about the cryptographic algorithms used for signing the document. The /ByteRange defines the parts of the PDF document to be used as the arguments to the algorithm that calculates the signature. Figure 3 shows the structure of a PDF after applying the incremental updates. The application of incremental saving does not modify the original content of the document. Moreover, it saves time if the changes applied are small on a large PDF file.

The signatures are also of two types. "Certification Signature" which put a restriction on the addition of the other signatures/comments to the file. Only one certificate signature is present for each document. The other type is the "Approval Signature" which can be more than one for a single document. Its basic task is to validate the content inside the document[11].

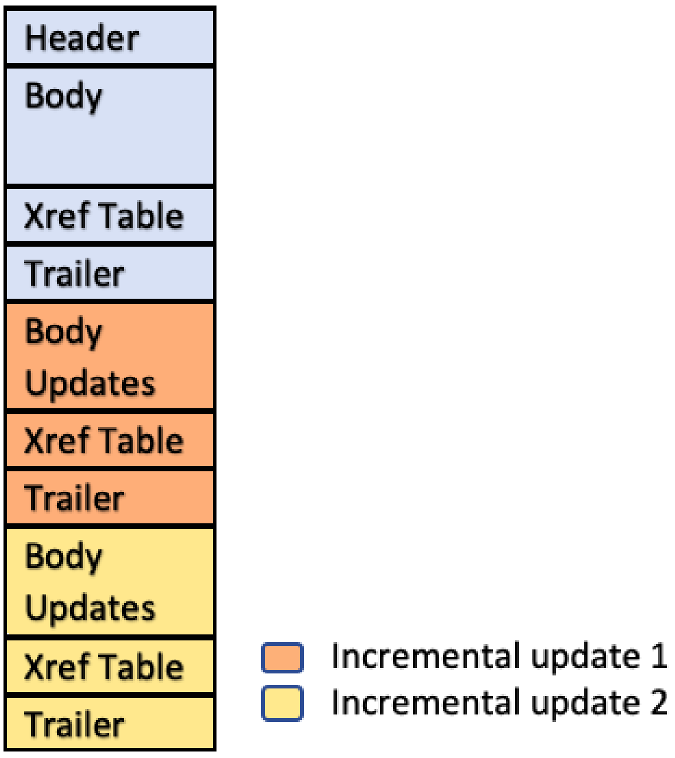


Figure 3: PDF structure after applying the Incremental Updates

The signed PDF is verified in the background when a user opens the PDF document with an application that supports signatures. It compares the calculated hash value using */ByteRange* parameters in the document (except the values from the */Contents*) against the value stored in the signature. If both the values do not match, the user is warned that the content of the document was altered. Figure 4 shows how Adobe Acrobat Reader DC depicts that the signature is invalid.

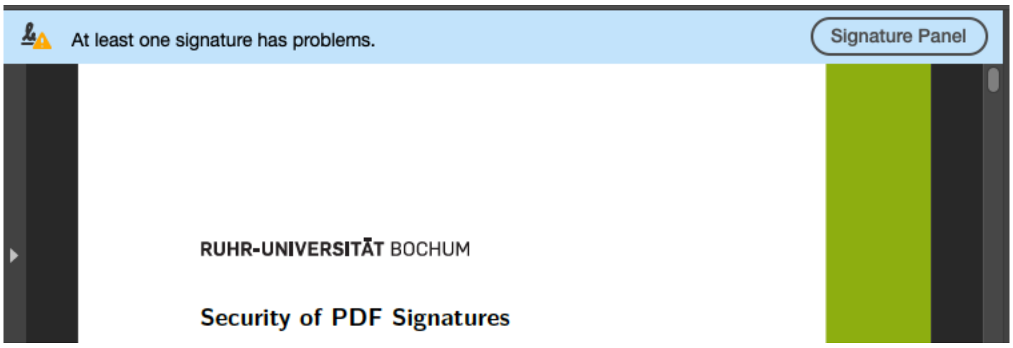

At least one signature has problems.

Figure 4: PDF with an Invalid Signature [11]

4 Problem Description

The signed PDFs are good enough to hold the security of the document, but the classes of attacks mentioned in the next section are still feasible just by having access to the document. Such attacks either break the reference between the signature of the document and the signed part or change the meta-data and parameters related to the signature. After these attacks, the victim is still shown that the content is not modified since the application of the last signature. This incompetence of the PDF viewers raises a question of security and data breach of every computer user as PDF has become a known standard for document exchange worldwide. In the next section, we discuss which parts of the document are modified to carry out these attacks successfully. We also discuss the success rate of these attacks on 22 native PDF applications on macOS, Windows and Linux and 8 online PDF viewers. To carry out the following attacks the assumption is that an attacker has the access to the signed PDF file but does not have the private key used while signing the file. hence, the attacker can change the content of the file. An attack is considered successful if the manipulated content is displayed by the PDF viewer without showing any warning that the signature has problems(depicted in figure 4).

5 Possible Attacks

5.1 Universal Signature Forgery (USF)

USF is a class of attack inspired by XML signature wrapping attack where the attacker adds the malicious content into signed XML and changes the document structure in such a way that the signature remains valid[4]. Another similar kind of attack is performed on JSON wherein the argument in the header field called ‘alg’ (meaning algorithm) is set to none. After setting the header, the signature is also set to an empty string removing the original signature from the file and making it prone to modification [5]. The same approach is followed for PDFs where attacker corrupts the signature body by changing the cryptographic algorithm to an asymmetric one where the signing is done by a private key but the verification is done using a public key. The attacker also changes the signature value present in the PKCS7 Blob getting access to public key present in the certificate. This leaves a loophole that makes this attack successful.

This attack is implemented by defining the attack vector described in figure 5

- Variant 1: removes either the Contents or Byte Range part to completely eradicate the signature value present in them or the information about the part which is encrypted by the algorithm.
- Variant 2: does not remove the complete section but make them empty.
- Variant 3: sets the value of both the parameters as null.
- Variant 4: also works similarly by setting invalid values to both the parameters.

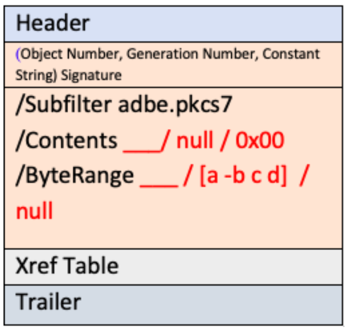

Header
(Object Number, Generation Number, Constant String) Signature
/Subfilter adbe.pkcs7
/Contents ___/ null / 0x00
/ByteRange ___/ [a -b c d] / null
Xref Table
Trailer

Figure 5: PDF structure in USF attack

Evaluation: USF attack was done on 22 PDF reader applications out of which 4 applications including Adobe's own PDF readers were prone to this attack class[10].

5.2 Incremental Saving Attack (ISA)

To understand ISA, we need to know how an update is done on a PDF file because ISA relies heavily on it. Whenever an update is performed to change the content of a PDF, the updated portion is always added at the end of the file along with a new Xref table. The new Xref table modifies the part of the document to be encrypted leaving the newly added body part out of the scope of encryption (see figure 6). This attack can be done in 4 variants described below:

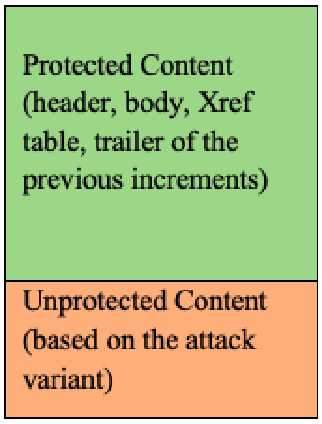

Protected Content (header, body, Xref table, trailer of the previous increments)
Unprotected Content (based on the attack variant)

Figure 6: PDF structure in ISA attack

- Variant 1: ISA with Xref table and Trailer Variant 1 of ISA attack includes Xref table and Trailer along with the body updates. Signature validation can be easily bypassed by adding either an Xref table with nothing inside it or a faulty Xref table and wrong body content. Update body contains a wrong byte offset and as a result, the updated content in the PDF is displayed without any signs of warning to the end-user.

- Variant 2: ISA without Xref table and Trailer. Another similar addition to PDF also makes this attack successful by simply removing both the Xref table and the trailer. As a consequence of this addition, the PDF viewer does not interpret the new addition and displays the new content without any warnings.
- Variant 3: ISA with a Trailer. ISA attack is also possible with a trailer if it points to a Byte offset other than Xref table. The modern PDF viewers check for a trailer to be present and is the main reason for the discovery of this variant of ISA.
- Variant 4: ISA with a copied signature and without an Xref table and Trailer PDF viewers prone to variant 4 of ISA attack does not validate the content of the signature but only check its presence. Thus, by adding a signature body while adding update body makes this attack successful in such viewers.

Evaluation: ISA was successful on 11 PDF viewers out of 22. Applications like *PDF Studio Viewer 2018* and *Perfect PDF 10 Premium* are prone to ISA by adding 'startxref' as a comment and removing the Xref Trailer during incremental saving. This class of attack was successful on 5 out of 8 online PDF validation services [10].

5.3 Signature Wrapping Attack (SWA)

SWA is a unique technique and a bit different than the last two attacks. The signature object has a parameter called ByteRange which has four arguments namely [a, b, c, d]. These four arguments define the part which is signed within the document. a and b is the first argument to the hash calculation and c and d serve as the second argument. The region between b and c is the unprotected area and is mainly targeted in this attack. The basic idea is to increase this region to add harmful content in this part and modify the values of other parameters accordingly. SWA is performed in two different variants described below:

- Variant 1: Relocating second hashed part
- Variant 2: Relocating both hashed parts.

The attack starts with the creation of a new Xref table beginning at position c and the second signed part now starts at c* which is a different position than c. Contents part in the Signature object contains zero bytes as padding. The next step is to remove this padding to make space for malicious content still keeping the new Xref starting at the old value of c. This is done to preserve the valid signature. The last step is the addition of the trailer after the Xref table and moving the old signed content after the trailer (at position c*) wrapped in a stream object. This step is purposely done not to end the document with %%EOF as some PDF viewers give a warning of modification if found. The trailer also now automatically points to the location defined by the old position c (refer figure 7). Instead of changing c to c*, the 2 variant changes all the 4 parameters a, b to 0 and c, d to b+d, and move the first hashed part (a+b) along with

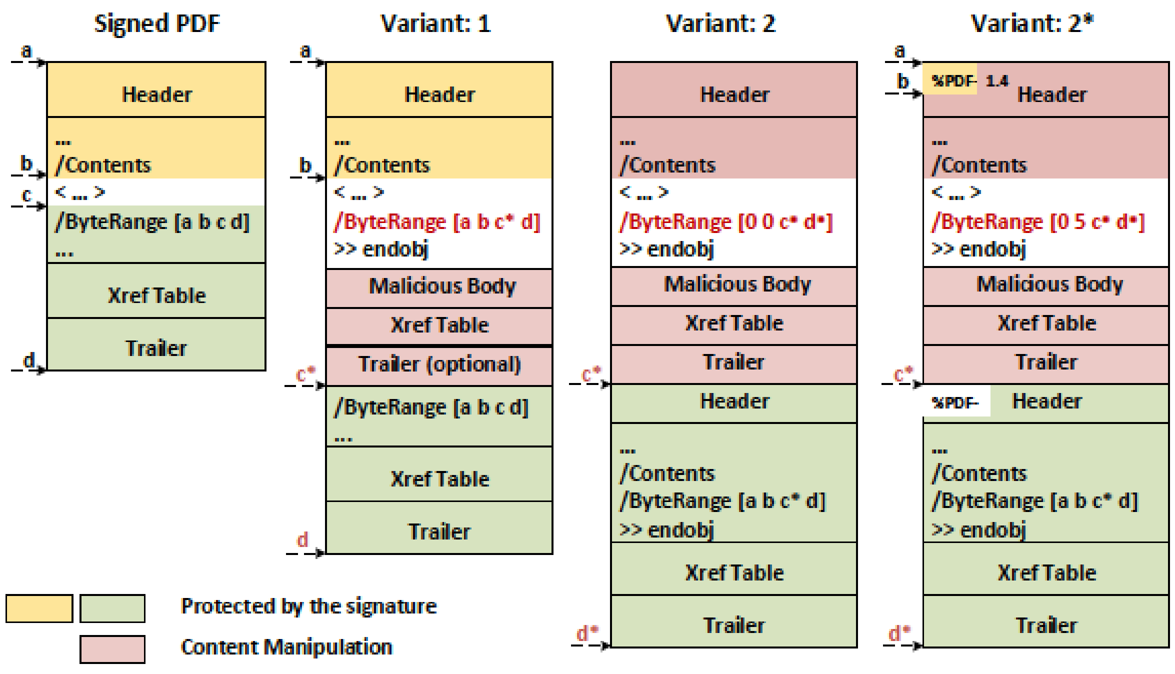


Figure 7: PDF structure before and after variants of SWA attack [10]

second hashed part (c+d) to c*. This variant completely removes the first hashed part from the document, and still makes this attack successful. But to be on the safer side instead of setting the value of b to 0, bigger value is selected to incorporate just the PDF interpreter version. By this modification, both the hashed parts will be in place making this attack even more robust.

Evaluation: This was successful on 17 PDF viewers out of 22 making it the most prolific attack. It was also successful on 2 out of 8 online PDF validation services [10].

6 Combined Evaluation

This section gives a visualization of the success of USF, ISA and SWA altogether on 22 PDF Readers and 8 Online Validation Services (see figure 8). These bar-charts have PDF viewer services (in number) on the y-axis and attack classes on the x-axis. The data used in this evaluation is taken from the paper *'Trillion Dollar Refund – How To Spoof PDF Signatures'*[10]. These diagrams give us the insight that the ISA was most successful with Online PDF Validation Services whereas the Desktop PDF Applications were more prone to the SWA.

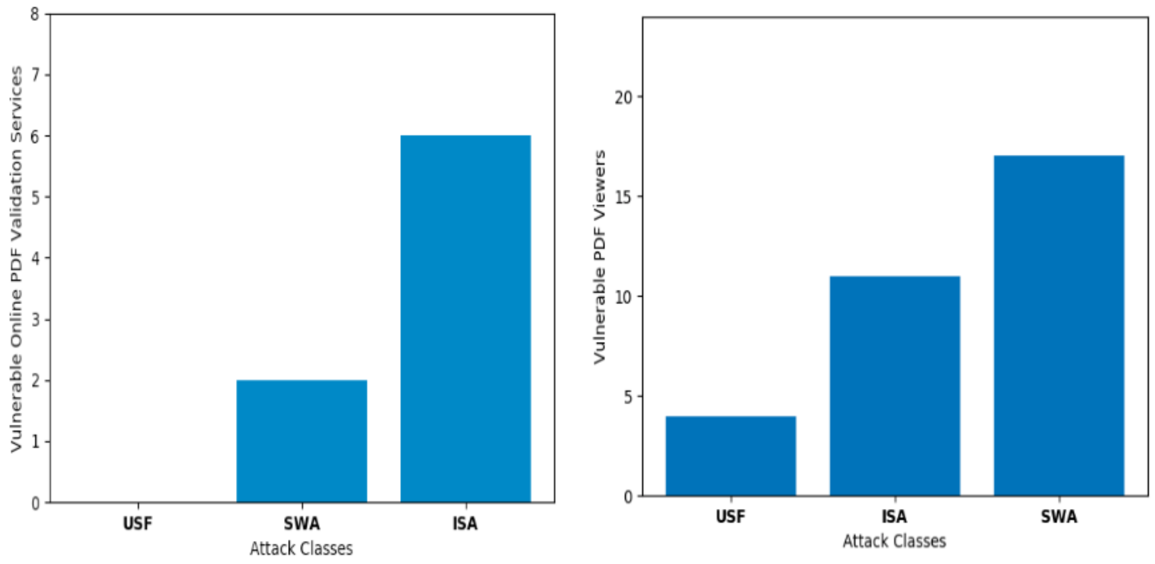


Figure 8: Bar Charts depicting success of USF, ISA and SWA on 22 PDF Readers and 8 Online Validation Services

7 Reasons for such Attacks

1. **Lack of standard specification for PDF signature validation**: The flaws in the specification for the PDF signature validation could be considered as the main reasons for such attacks. The incomplete specification leads to the attacks in which signed content can be moved to a different location. Such attacks are possible because the specification does not specify the signature to cover the whole PDF document.

2. **Leniency in Error validation**: Attack class such as ISA has highlighted the fact that the PDF applications and online PDF validation services are error-tolerant. We have seen in the evaluation part of ISA that 11 out of 22 PDF viewers and 5 out of 8 online PDF validation services have processed the PDF even if the PDF content was not standard complaint [10].

3. **Coding mistakes**: Even if the above two points are taken care there is still a scope of programming errors that can break the complete cryptographic algorithm. In case of USF attack, missing information also leads to a valid signature and displays the user that the content is not modified since the application of the last signature.

8 Possible Solutions

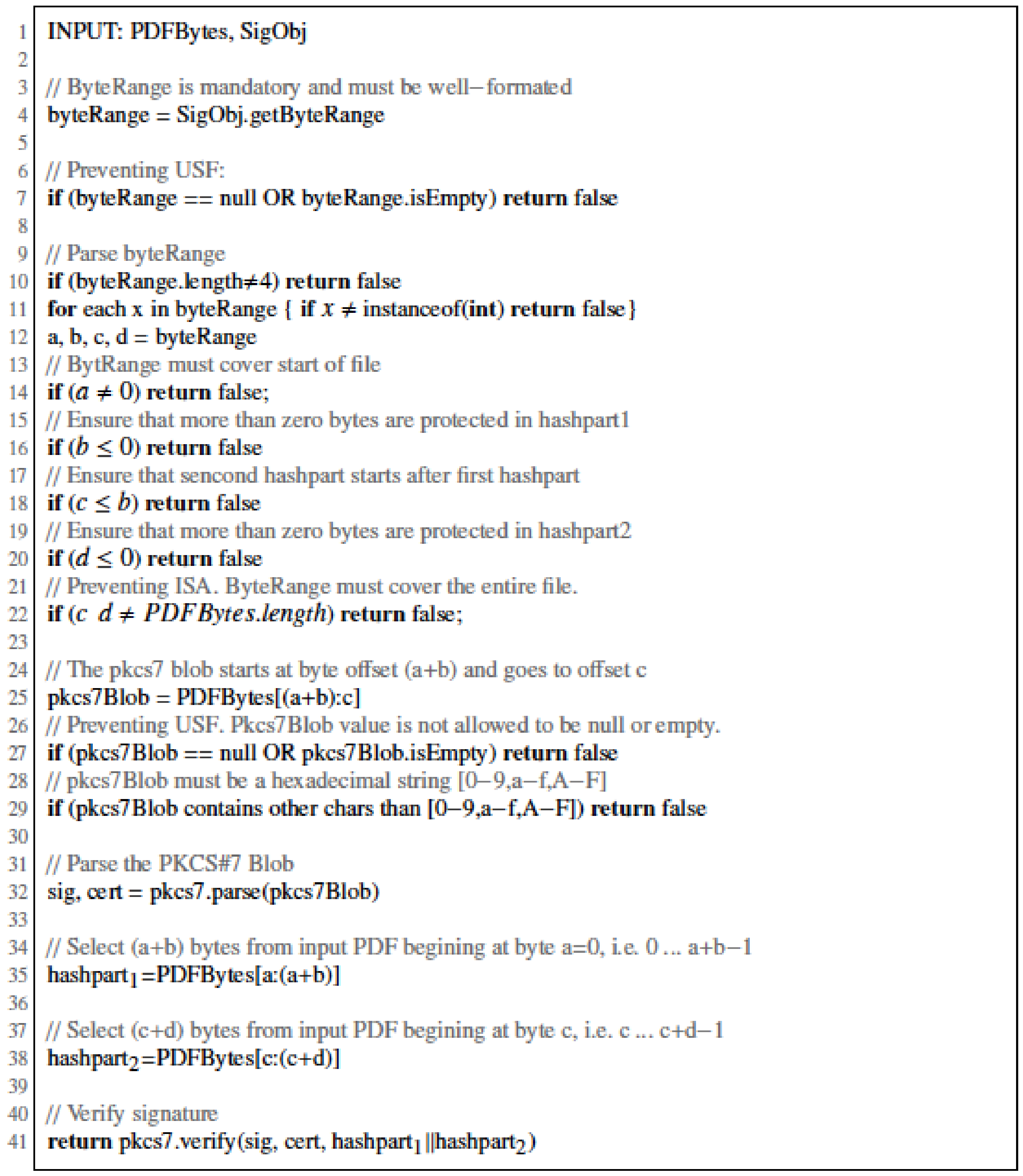
8.1 Algorithm to Prevent PDF Signature Attacks

The following sub-section describes the measures that can be taken to prevent the attacks mentioned in the Problem Description section of this paper. These measures define a definite approach to prevent such attacks without leading to any side-effects concerning the error tolerance of the PDF viewers.

The first approach to counter these attacks is a verification algorithm to detect any modification after the application of the signature in the PDF document. To detect any modification this algorithm must be applied for each signature that is present in the PDF file. The algorithm is a series of checks that takes *PDFBytes* (PDF file as a byte stream) and *SigObj* (signature object) as input arguments as described in the figure 9 at line number 1.

The algorithm focuses on the parameters present inside the *ByteRange*. Thus, the first check in the validation process is to verify that the *ByteRange* should not be null or empty. If found so, the algorithm terminates. This step ensures that the USF attack with the attack vectors such as */ByteRange null*, */ByteRange* -, or removing the */ByteRange* parameter itself should not be allowed. The next validation checks that the length of the *ByteRange* should be exactly 4 i.e. an array with the values like [a,b,c,d]. This is done to check the attack vectors like */ByteRange []*, */ByteRange [0]* should not pass the validation. Line number 11 also ensures that these parameters should be all integers specifying a valid byte offset within the PDF file.

A validation is placed at the line number 14 to check that the first byte offset 'a' must have the value 0 (start of the file). This is followed by the similar checks (from line number 16 to 22) to guarantee that the signature covers the whole file without any overlapping of these parameters. The algorithm then parses the content of the *PKCS#7 blob* and ensures that it must be present and does not contain special characters and must be a hexadecimal string. The last step in the algorithm is to fetch both the hashed parts (refer figure 10) required for the signature verification routine and returns whether the signature is valid or not.



```
INPUT: PDFBytes, SigObj

// ByteRange is mandatory and must be well−formated
byteRange = SigObj.getByteRange

// Preventing USF:
if (byteRange == null OR byteRange.isEmpty) return false

// Parse byteRange
if (byteRange.length≠4) return false
for each x in byteRange { if $x \neq$ instanceof(int) return false }
a, b, c, d = byteRange
// BytRange must cover start of file
if ($a \neq 0$) return false;
// Ensure that more than zero bytes are protected in hashpart1
if ($b \leq 0$) return false
// Ensure that sencond hashpart starts after first hashpart
if ($c \leq b$) return false
// Ensure that more than zero bytes are protected in hashpart2
if ($d \leq 0$) return false
// Preventing ISA. ByteRange must cover the entire file.
if ($c\ \ d \neq PDFBytes.length$) return false;

// The pkcs7 blob starts at byte offset (a+b) and goes to offset c
pkcs7Blob = PDFBytes[(a+b):c]
// Preventing USF. Pkcs7Blob value is not allowed to be null or empty.
if (pkcs7Blob == null OR pkcs7Blob.isEmpty) return false
// pkcs7Blob must be a hexadecimal string [0−9,a−f,A−F]
if (pkcs7Blob contains other chars than [0−9,a−f,A−F]) return false

// Parse the PKCS#7 Blob
sig, cert = pkcs7.parse(pkcs7Blob)

// Select (a+b) bytes from input PDF begining at byte a=0, i.e. 0 ... a+b−1
$hashpart_1$=PDFBytes[a:(a+b)]

// Select (c+d) bytes from input PDF begining at byte c, i.e. c ... c+d−1
$hashpart_2$=PDFBytes[c:(c+d)]

// Verify signature
return pkcs7.verify(sig, cert, $hashpart_1 || hashpart_2$)
```

Figure 9: Algorithm to prevent USF, ISA and SWA attacks [10]

9 Drawbacks

The approach mentioned in section 8.1 has the following two limitations:

1. In the PDF specification 1.7, /ByteRange is not specified as a mandatory parameter[9]. The signature value is calculated only using the signature dictionary. Thus, the

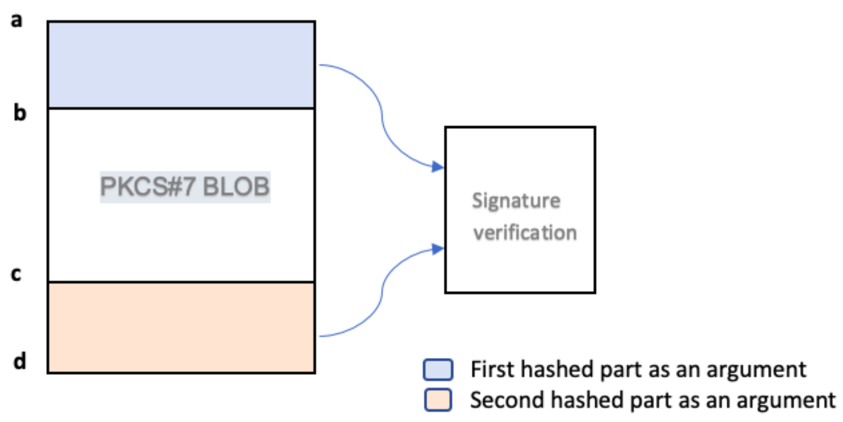

a
b
c
d
PKCS#7 BLOB
Signature verification
First hashed part as an argument
Second hashed part as an argument

Figure 10: Arguments to Signature Verification Routine

/ByteRange parameter is not required to create a signed document. But the algorithm specified relies on the /ByteRange and the values present inside it. As it has strict checks on the presence of /ByteRange on line number 7 (refer figure 9). This problem is solved by making /ByteRange as a compulsory parameter.

2. Another problem arises when multiple signatures are present inside the document and these cover the part of the document and not the whole document (at least one of these covers the whole document). But PDF readers declare a signature valid only if it covers the whole file. This issue could be resolved if each signature also contains the details about the revision number to which it belongs. Finally, the PDF applications could allow users to show only that part which is protected by a specific revision.

10 Possible Attack Extensions

10.1 Dali Attack

PDF can also incorporate JavaScript in it to display dynamic content. This is a supportive feature, but at the same time leaves a window for an attacker to exploit the features of JavaScript to spoof content inside the document body. Because of the dynamic content present inside the file, the file is converted into an image and the content is signed based on a concept of WYSIWYS (What You See Is What You Sign). [7] This attack has got his name from a Spanish artist Salvador Dali who is known for his mysterious paintings.

To carry out this attack, one needs to create a TIFF file and a PDF file. The TIFF file is the image representation of the PDF file. Next step is to embed the TIFF (Tagged Image File Format) header into the initial one Megabyte (MB) of the PDF file. As PDF allows the header to be placed anywhere in the first 1MB of its data. Then, the PDF content is appended followed by the trailer. The last step is to make this multifarious file compatible with the Adobe Reader after shifting the values to the byte-value same as a PDF file format. The parameters that need to be modified are StripOffsets, BitsPer-Sample, XResolution, and more readable fields such as the time and the date of creation of the file. The resulting file can be opened with Adobe Reader or Foxit Reader and does not show any warning message as the signature is applied after the modifications. The catch here is the filename which needs to be changed to '.tif'. This step will not affect the signature as it has no information about the file format.

10.2 Content Masking

In this attack, the actual words of the document are mocked with similar-looking characters that means what is displayed is not what actually it is. It is possible due to the high error-tolerance of the viewer against the fonts used in the document.[3] To carry out such an attack, an attacker needs to first create a new font file defining mapping from characters to glyphs. This is possible with tools like Font-Forge which can directly edit the glyphs for each font. After defining the mapping, the attacker directly makes changes in the PDF file with the help of tools such as LATEX, where one can define new custom fonts. Then, the process of masking can be carried out by replacing each actual word with the targeted word. Content Masking attack has found application in the following areas:

1. Topic Matching in reviewer assignment.
2. Curtail Plagiarism.
3. Document Index Subversion.

Evaluation: The evaluation of this attack is done in section 5.2 of the paper named *'PDF Mirage: Content Masking Attack Against Information-Based Online Services'* where the authors have done the experiment to hide plagiarism using this attack[3]. In the experiment, the authors used 10 published papers from the Web. They targeted a similarity score of in between 5% to 15% by masking words in these documents. The masking of the words was done in descending order of their usage, descending order of their frequency of appearance and in random order. The results from the first approach did not fall in the targeted range of 5% to 15%, but when 20% to 40% words were masked with respect to the frequency of their appearance the desired similarity score was reached. The same results were also achieved by masking 17% to 20% of the words in random order.

11 Possible Solution Extensions

1. These attacks can also be mitigated if the PDF readers carefully adhere to the *PDF Advanced Electronic Signatures specification(PAdES)*[1]. These specifications also ensure the safety if the signer or the verifying party disagrees with the signature in the future.

2. Signed data can be compressed and can be made hidden for PDF applications. Thus, before verifying the signature the signed data should be decompressed. This transformation information could be added to the signature dictionary to prevent such an attack[11].

3. Dali attack could be prevented if the file name is also included in the PKCS#7 blob[7]. If then an attacker tries to change the extension could be traced by the PDF application in the signature verification step.

4. The success of an attack is determined by factors such as how the signature is created and validated. But by following eIDAS(Electronic Identification, Authentication and Trust Services)[8] standards one could limit such vulnerabilities and guarantees the integrity. It puts checks such as the signature must be distinctively link to its signer. The eIDAS also puts restriction on the issuing of the certificates to trusted service providers and certificates are only issued if the identity of the signer is verified.

12 Conclusion

In this paper, I summarized the basics of PDF and the possible attacks that could be performed on the digitally signed PDF files along with their evaluation. This report also includes a pseudo-code to prevent these attacks. The evaluation was done on 22 PDF applications and 8 online validation services. The results were daunting as 95% of the PDF desktop applications and 75% of the online PDF validation services were prone to at least one of these three attacks. I also presented two more similar attacks that could also pose a threat to PDF security along with the evaluation and the possible solution. This research shows that there has been a lack of standards for PDF specification for defining and validating signed PDFs. Furthermore, it also makes us believe that fewer efforts have been put on defining the best practices for signing a PDF file despite the popularity of this file format. 'Better late than never', a new perspective has been born with the introduction of these attacks when it comes to the security and the authenticity of the digital signatures in the PDF documents and it has also given a rise to a new research direction for the existing researchers in this field of study.

References

[1] ETSI EN 319 142-1 V1.1.1 Electronic Signatures and Infrastructures (ESI); PAdES digital signatures . (2016-04).

[2] How to break PDF Encryption (September 2019). *https://www.pdf-insecurity.org/* , 2019.

[3] Yao Liu Ian Markwood, Dakun Shen and Zhuo Lu. PDF Mirage: Content Masking Attack Against Information-Based Online Services . *https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-markwood.pdf* , August 16–18 2017.

[4] Meiko Jensen Jörg Schwenk Nils Gruschka Luigi Lo Iacono Juraj Somorovsky, Mario Heiderich. All Your Clouds are Belong to us – Security Analysis of Cloud Management Interfaces . *https://dl.acm.org/doi/10.1145/2046660.2046664* .

[5] Creative Commons Attribution 4.0 International License. Critical vulnerabilities in JSON Web Token libraries . *https://www.chosenplaintext.ca/2015/03/31/jwt-algorithm-confusion.html* , 2015.

[6] Dejan Lukan. PDF File Format: Basic Structure https://resources.infosecinstitute.com/pdf-file-format-basic-structure/ . , MAY 6, 2018.

[7] Dan-Sabin Popescu. Hiding Malicious Content in PDF Documents . *https://arxiv.org/pdf/1201.0397.pdf* .

[8] Peter Smirnoff and Dawn M. Turne. What are E-Signature Validation Attacks and How to Protect Yourself in the Context of eIDAS . (2019).

[9] Adobe Systems. Adobe Systems Incorporated. 2006. PDF Reference, version 1.7 (sixth edition ed.) . (2006).

[10] Karsten Meyer zu Selhausen Martin Grothe Jörg Schwenk Vladislav Mladenov, Christian Mainka. 1 Trillion Dollar Refund – How To Spoof PDF Signatures . , page 7.

[11] Karsten Meyer zu Selhausen. Security of PDF Signatures, . page 91, (2018).