U/OO/234068-17

Inspection and Sanitization Guidance for Portable Document Format

Version 1.0

2 May 2011

National Security Agency
9800 Savage Rd, Suite 6721
Ft. George G. Meade. MD 20755

Authored/Released by:
Unified Cross Domain Capabilities Office
cds_tech@nsa.gov

DOCUMENT REVISION HISTORY

Date	Version	Description
5/2/2011	1.0	final
12/13/2017	1.0	Updated Contact information, IAC Logo, Cited Trademarks and Copyrights, Expanded Acronyms, and added Legal Disclaimer

EXECUTIVE SUMMARY

This *Inspection and Sanitization Guidance for PDF* document provides guidelines and specifications for developing file inspection and sanitization software for PDF files.

It is well understood that the proprietary binary file formats used by Microsoft (MS) Office®[1] can introduce hidden, potentially sensitive data to a document without the author's knowledge. PDF is often suggested as a "safer" alternative to MS Office formats to avoid such hidden data problems as metadata, macros, and editing comments. PDF is an open international standard from the ISO (32000-1:2008) [1], however, it is important to investigate the PDF file format and standard for hidden data vulnerabilities similar to those that has plagued MS Office.

There are two general hidden-data risks associated with the PDF file format. The first is improper use of the format by some malicious user that can lead to potentially sensitive data becoming obscured in a document. Improper redaction is one example of this, which has occurred numerous times ([2], and [3]). Secondly, the sheer quantities of complex features that have been added to the PDF format over its many revisions provide avenues for both malicious and accidental data hiding [4].

In this report, "hidden data" is defined as data in a document that is not readily visible when the document is printed or displayed with viewing applications. There are two general hidden data types: *hidden content data*, which is content data obscured through the use of formatting, (e.g., text overlaid by an image, overlapping images, text written in a very small font, etc.); and *embedded hidden data*, which is non-content and has to do with document structures. Metadata, revision history data, embedded objects or files, and active scripting or macros are typical forms of embedded hidden data.

This PDF evaluation shows that there are many opportunities for PDF files to contain both hidden content data and embedded hidden data. PDF's flexible content formatting system allows almost any content object be resized, positioned, and colored at will. This flexibility leads to numerous ways in which document content can be obscured from view. The PDF format also contains many other complex, dynamic features that provide avenues for hidden data to enter a document.

To prepare a PDF document safe for distribution, all hidden content data and embedded hidden data risks must be eliminated.

[1] Microsoft Office is a registered trademark of Microsoft Corp.

Content can be hidden in the following ways:

- Size – Text and graphical content can be sized so that it is too small to be visible when normally viewing document or when printed.
- Cropping – Images can be cropped or clipped so that only a portion is visible in the document while the entirety of the image data resides within the PDF file.
- Rendering – Reader software is responsible for displaying and representing the internals of the file to the user. Text and fonts, as well as images, go through a rendering process as they are painted on the document. Methods can be deployed to hide data from a human reader through font or rendering.
- Transparency and Color – PDF's coloring and transparency formatting features can be manipulated to make text and images transparent or impossible to differentiate from the background.
- Positioning – Pieces of content in PDF files can be positioned anywhere on or off the visible or printed portion of a page, rendering it invisible. Additionally, any content could be positioned to overlap other content, obscuring it.

No set of formatting features can be singled out as the cause of hidden content data. Every formatting option has both legitimate and data hiding uses depending on the manner and situation in which it is used. The simplest and most reliable method for finding hidden content data is to identify and extract all textual and graphical content and evaluate it. This report has shown how to identify graphical content in addition to text and fonts. However, this approach is the most cumbersome and labor-intensive. Automated PDF analysis tools could implement rules to check for some types of hidden content data, for example, by searching for abnormally small font and image sizes, the presence of cropping, rendering allowing for hidden information, transparency, and image color settings that introduces obscurity, and abnormal positioning.

Embedded hidden data is comparatively easy to identify versus hidden content data yet poses the most risk since it is not directly part of a document's visible or printed contents. Through a careful feature-by-feature review of the PDF specification, this report has identified the avenues for embedded hidden data, including:

- Metadata
- Annotations and Comments
- Actions, Links, and Scripting
- Embedded Files, Images, Multimedia, and Fonts
- Encryption
- Optional Content and Alternate Views
- Embedded Executables and Code

Using the information provided in this report, reviewers can identify and analyze potential locations where embedded hidden or malicious data may reside in PDF files. Embedded hidden data lends itself more readily to automated removal methods than hidden content data. There are several PDF manipulation packages that could be used to produce a hidden-data removal tool implementing the recommendations set forth in this report.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. SCOPE

1.1 Purpose of this Document

The purpose of this document is to provide guidance for the development of a sanitization and analysis software tool for the Portable Document Format (PDF). It provides inspection and analysis on various elements and objects that are contained within the PDF file structure and how they can be a cause for concern for either hiding sensitive data or attempts to exploit a system. This document provides an analysis of numerous features in PDF and also provides recommendations to mitigate these threats to provide a safer file. Although this report does not mention vulnerabilities related to a specific PDF reader software application, however there were a number of them used in the analysis of the standard.

The intended audience of this document includes system engineers, designers, software developers, and testers who work on file inspection and sanitization applications that involve processing PDF documents.

1.2 Introduction

The Portable Document Format (PDF) has become a widely used and trusted standard of representing text and graphic files for a number of users and platforms. PDF files are organized in such as a manner that lends data to be hidden easily and also allows for malicious executable content to be embedded within the document content, but not rendered as part of its content.

1.3 Background

The usage of PDF files has seen an enormous increase in recent years due to their high level of portability and the increased amount of content now available online. Many people use them because of widely available free PDF reader software to render the documents. However, it is the fact that the International Organization for Standardization (ISO) standard was created for portability, allowing readers to be developed that can operate on a number of platforms. Windows, Mac, and Linux users can freely distribute PDF files as a document format, and there are numerous reader applications widely available for many platforms.

The creation of a PDF file can vary by different users. There are two common use cases for the creation of a PDF file.

- Printing or converting a document (such as Microsoft Word) to a PDF file.

- Scanning a document or image into a PDF file.

Determining how a PDF file was created can be useful to identify what should be present in the content of the file. Since the software generates a partially known pattern of PDF objects, metadata, and document organization. Ideally, filtering software should be capable of attempting to distinguish how the PDF file was crafted between the two methods discussed above. For example, a PDF file constructed from a word document will likely contain streams with text content providing the composition of the word document. A scanned document may contain a PDF entirely composed of image data. It may be rather difficult to determine the precise software tools used to develop the PDF if that information is not obvious in the PDF syntax.

1.4 Document Organization

This document describes the objects and syntax of a PDF document. It refers to these elements as constructs. In addition to describing each relevant object, this document also describes its potential flaws or ways data can be hidden, disclosed, or embedded maliciously. Each construct description begins with the label '*PDF :x.y*' and ends with '*PDF :x.y: END*', where ***x*** represents the section number of this document and ***y*** is the sequential index for each section. They are provided as a reference, and each section of information is written to allow a user to reference a particular feature to analyze concerns and recommendations for that feature.

The following table summarizes the organization of this document.

Table 1-1. Document Organization

Section	Description
Section 1: Scope	This section describes the scope, organization, and limitations of this document.
Section 2: Construct and Taxonomy	This section describes a definition of how the constructs are represented as well as the terms defined in this document.
Section 3: PDF File Overview	This section describes the general overview and description of the PDF file format.
Section 4: PDF High Level Constructs	This section describes the general PDF dictionaries and overall syntax that provide the contents to a PDF file. It also introduces some of the basic data types and special features in the PDF standard.
Section 5: PDF File – Constructs and Metadata	This section describes numerous objects and dictionary information defined in the PDF file that provide rich media content, interactive features, and embedded content.

Section	Description
Appendix A: GeoPDF	This appendix describes the GeoPDF specification and additions to PDF documents.
Appendix B: Summary of Risks	This appendix highlights each concerns section described in each feature and summarizes each type of risk per feature in this document.

1.5 Recommendations

The following subsections summarize the categories of recommendation actions that appear in this document, and associated options.

1.5.1 Actions

Each construct description lists recommended actions for handling the construct when processing a document. Generally, inspection and sanitization programs will perform an action on a construct: *Validate*, *Remove*, *Replace*, *External Filtering Required*, *Review, or Reject*.

The Recommendation section in each construct lists each of these actions and corresponding applicable explanations of the action to take. It notes if a particular action does not apply, indicates actions that are not part of the standard set of actions (listed in the previous paragraph). For example, a program may choose to reject a file if it is encrypted. Additionally, for some constructs, an action may further break down to specific elements of a construct (e.g., for hidden data in a dictionary's syntax) to give administrators the flexibility to handle specific elements differently.

Recommendations such as Remove and Replace alter the contents of the document. PDF documents are structured such that each object can be identified by a byte offset into the file. There are other fields within various objects that also use byte offset information to identify the location of other objects. If one of these recommendations is taken, the byte offset information throughout the entire PDF file will require recalculation to correct the structure of the file.

NOTE

The recommendations in this document are brief explanations rather than a How-To Guide. Readers should refer to the construct description or ISO32000-1:2008 official documentation for additional details.

Table 1-2 summarizes the recommendation actions.

Table 1-2. Recommendation Actions

Recommendation Action	Comments
Validate	Verify the data structure's integrity, which may include integrity checks on other components in the file.(This should almost always be a recommended action)
Replace	Replace the data structure, or one or more of its elements, with values that alleviate the risk (e.g., replacing a user name with a non-identifying, harmless value, or substituting a common name for all authors).
Remove	Remove the data structure or one or more of its elements and any other affected areas.
External Filtering Required	Note the data type and pass the data to an external action for handling that data type (e.g., extract text and pass it to a dirty word search).
Review	Present the data structure or its constructs for a human to review. (This should almost always be recommended if the object being inspected can be revised by a human)
Reject	Reject the PDF file.

NOTE

No recommendations for logging all actions and found data are included here because all activity logging in a file inspection application should occur "at an appropriate level" and presented in a form that a human can analyze further (e.g., the audit information may be stored in any format but must be parsable and provide enough information to address the issue when presented to a human.)

1.5.2 Action Options

The companion to this document, *Data Transfer Guidance for PDF Documents*, specifies four options for each recommended action: *Mandatory*, *Recommended*, *Optional*, or *Ignore*. Depending on the circumstances (e.g., a low to high data transfer versus a classified to unclassified transfer), programs can be configured to handle constructs differently.

Table 1-3 summarizes the recommendation action options.

Table 1-3. Recommendation Action Options

Action Options	Comments
Mandatory	For the given direction (e.g., secure private network to unsecure Internet), the file inspection and sanitization program must perform this recommended action.
Recommended	Programs should implement this action if technically feasible.
Optional	Programs may choose to perform or ignore this recommended action.
Ignore	Programs can ignore this construct or data structure entirely

1.5.3 Naming Convention for Recommendations

Recommendations in this document are numbered sequentially, where applicable, and adhere to a standard naming convention identified by a single number *x*, where *x* is a sequential number following by the recommendation keyword defined in Table 1-2. There may be multiple recommendations of the same type, which remain uniquely identified by its number. There is only one file content under review in this document (PDF).

1.6 Data Transfer Guidance

Each format that is documented for inspection and sanitization analysis has a companion document (i.e., the aforementioned DTG document). The DTG serves as a checklist for administrators and others to describe expected behaviors for inspection and sanitization programs. For example, administrators may only remove certain values in a metadata dictionary. Or, the administrator may decide to remove all hidden data if the document is being transferred to a lower security domain.

The *DTG* gives the administrator the flexibility to specify behaviors for inspection and sanitization programs. The workbook contains a worksheet for each security domain (i.e., the originating domain). Each worksheet lists the numbered constructs from this document and enumerated recommendations in a row. After the recommendations, the worksheet displays a cell for each possible destination domain. This enables an administrator to select the action option for data transfer from the originating domain to the particular destination domain. Each construct row also contains two comment cells: one for low to high transfers and another for high to low transfers.

The recommended actions address two broad risk types: data hiding and data execution. Most data structures are vulnerable to one risk type, while others are

susceptible to both risk types. Each construct row in the DTG worksheet contains a cell for designating the risk type (i.e., data execution, data hiding, or both) and another cell for assessing the risk level for that construct (i.e., high, medium and low). This enables administrators to assign the risk type and risk level to each specific construct.

1.7 Document Limitations

This document covers information from the ISO standard 32000-1:2008. At the time of this writing, the next version of the ISO standard (32000-2) is under development, and contains information about PDF-2.0. This document covers PDF-1.7 and the risks discovered while analyzing the ISO standard 32000-1:2008.

1.7.1 Covert Channel Analysis

It is nearly theoretically impossible to detect or prevent covert channels during communication. It is impossible to identify all available covert channels in any file format. Because PDF documents contain free-form text, searching for hidden data becomes increasingly difficult. No tool can possibly analyze every channel, so this document highlights the highest risk areas to reduce or eliminate data spills and malicious content.

Additionally, this document does not discuss steganography within block text or media files, such as a hidden message that is embedded within an innocuous image or paragraph. Separate file format filters that specialize in steganography should be used to handle embedded content, such as text, images, videos, and audio when attempting to reduce the risk of covert channels.

1.7.2 Character Encoding

The PDF specification indicates a PDF file is a sequence of bytes. Grouping certain bytes together form tokens or keywords that are part of the syntax of the PDF language. PDF files are not completely restricted to American Standard Code for Information Interchange (ASCII); the considerations are listed below [1].

- Tokens that delimit objects and describe the structure of the PDF file shall use the ASCII character set. All reserved words and names used as keys in the PDF standard dictionaries and certain array types shall be defined using the ASCII character set [1].

- Data values of strings and streams objects may be written entirely using UTF-16BE or entirely in binary data. Data that is naturally binary, such as an image, is usually represented in binary for efficiency [1].

- PDF files shall always be transported as a binary file rather than text such that all of the bytes in the file are preserved [1].

PDF files implement three distinct types of characters: regular, delimiter, and white-space. White space characters are NULL, Horizontal Tab, Line Feed, Form Feed, Carriage Return, and Space. They are shown in Table 1 of the PDF ISO standard. The Carriage Return and Line Feed can be combined to be an End of Line (EOL) marker. Delimiter characters are special ASCII characters (<, >, (,), [,], /, and %) that are used to delimit syntax or tokens in the PDF file. Table 2 of the ISO standard displays the delimiter characters. For example, the combined "<<" and ">>" enclose a dictionary defined in the PDF file. Every other character within the ASCII character set and outside of the ASCII character set are referred to as regular characters. The ISO standard defines the full description of syntax in a PDF, for a complete reference this section should be examined in greater detail.

2. CONSTRUCTS AND TAXONOMY

2.1 Constructs

Although this document delves into many low level constructs in the PDF syntax hierarchy, it does not serve as a complete reference to all of the different constructs in the standard. The document covers the overall file format, various graphics and interactive objects, and metadata elements. We have identified these as particular areas of concern for developers of file inspection and sanitization programs; however there is complete detail in the standard that should be examined alongside this documentation.

- **Description**: a high level explanation of the data structure or element
- **Concern**: an explanation of potential problems posed by the element. For example, some metadata elements can cause inadvertent data leakage and others can be used for data exfiltration.
- **Location**: provides a textual description of where to find the element in the document. This can vary by client (or product) or it may apply across the entire product line.
- **Examples**: if applicable, the definition will contain an example of the construct.
- **Recommendations**: as described in Section 2.2.

Recommendations appear within each of the PDF constructs. For the purposes of this document, these recommendations are "alternatives." Some recommendations may seem better than others and some recommendations may be more difficult to implement. Certain recommendations complement each other and can be grouped together (e.g., "Remove action object" and "Remove reference to action object"). Other recommendations may seem contradictory (e.g., "Remove Metadata" and "Replace Metadata").

2.2 Taxonomy

The following table describes the terms that appear in this document.

Table 2-1. Document Taxonomy

Term	Definition
Consistency	A construct state in which object information is set to correct values, and that required objects are implemented as defined in the PDF ISO standard.
Construct	An object in PDF terminology that represents some form of information or data in the hierarchy of the PDF document structure.
DTG	A list of all ISG constructs and their associated recommendations. DTGs are used to define policies for handling every ISG construct when performing inspection and sanitization.
Grammar	A precise, formal, and rigorous syntax for defining constructs.
Inspection and Sanitization	Activities for processing files to prevent inadvertent data leakage, data exfiltration, and malicious data or code transmission
ISG	A document (such as this) that details a file format or protocol and inspection and sanitization activities for constructs within that file format.
Recommendations	A series of actions for handling a construct when performing inspection and sanitization activities.
Referential Integrity	The construct state in which all associated objects are properly referenced in the construct and that construct entries reference existing objects. References may also point to byte offset location into the PDF file as opposed to object number. If this isn't the case the document may not open or may break.

3. PDF OVERVIEW

Describing a PDF and its syntax can be done in four categories. This is noted in the early sections of the ISO standard. First, there are objects, which are the basic data types representing information in the document. A PDF document is a hierarchy or data structure that is composed of objects, everything is an object. Secondly, there is a PDF file structure, which is discussed in greater detail in the next section. It defines how the file is laid out and how the objects are organized. Third, there is Document Structure, also discussed in greater detail in later sections. Document structure defines how each object forms the representative view of the PDF file. This defines how the objects are related and combined created a unique view of the PDF document. Objects such as pages, fonts, and annotations all belong into the Document Structure. Lastly, the ISO standard presents content streams. A stream is a PDF object, and perhaps the most complex one. A content stream is sequence of instructions defined by the ISO standard that describes how the page appears, how text is laid out on the page, and how a graphical image is represented. The content stream itself is different from other streams that contain raw data, since it contains instructions and parameters that impact the appearance of the document. In this section, more details on the PDF file structure and PDF document structure are introduced.

3.1 PDF File Structure

The PDF file structure is composed of four separate sections: the header, the body, the cross-reference table, and the trailer. Figure 3-1 illustrates a representation of a PDF file and basic contents that exist in each section. The header is always the first line of the PDF file and contains a suggested version of the ISO standard used by the file In addition, a second line in the header may contain a series of arbitrary "high order ASCII" values (ISO 32000-1:2008, 7.5.2). A PDF must have the version in the header, however the `/Version` in the Document Catalog object (if present) does override it. These lines are comments which are designated by the character '%'. Comments are discussed later in PDF 4.19:. The figure below depicts the header of a file that conforms to the latest version of the ISO standard.

The Body section of a PDF file contains a sequence of objects that represent the entire hierarchy of the content of the PDF document. Objects were described early as the basic types of data and the building blocks to PDF. The body is a listing of indirect objects, in no particular order, which defines several entities such as the root object (Document Catalog), a particular page of the document, the content of a particular page, an image, etc. The objects in this section comprise the PDF Document Structure, which is presented in the next section.

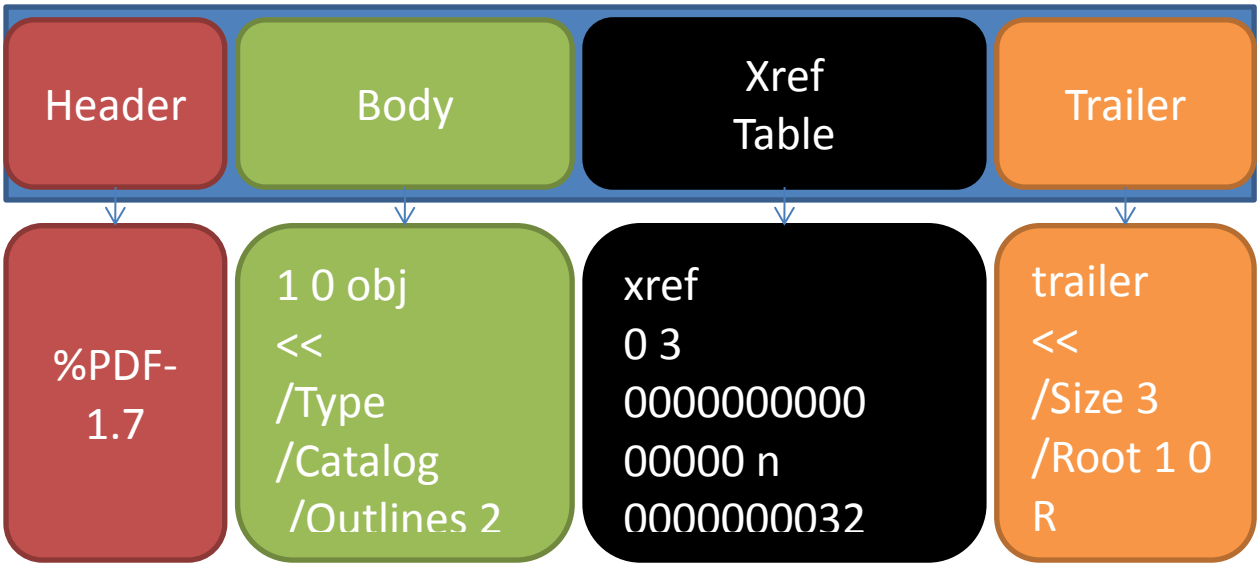

Figure 3-1. Traditional PDF File Format

The cross-reference table (shown above as Xref) contains object information that allow for random access to indirect objects throughout the body section of the PDF file. This is considered the original cross-reference table used in PDF 1.0 – PDF 1.4. New methods of implementing the cross-reference table are discussed later in this section. A cross-reference table allows a PDF viewer application to locate a specific object without having to parse the entire contents of the file. The xref keyword denotes the start of the cross-reference table and second line contains two digits: the first representing the object number of the first object in the subsection and the second representing the number of objects or the size of the table. Following this line, each object in the file is represented by a single line containing three entries: the first is a 10 digit number containing the byte offset into the file of that object's location, the second is a 5 digit number representing the generation or revision number of the object (objects in new file shall have this set to 0), and the third value identifies in the object is in use ('n') or is not in use ('f'). The table can be split and there can be another line of two digits following an entry. This resets the starting object number and list the number of the next sequential object numbers that follow. There might be multiple copies of cross-reference tables, with incremental updates or linearized PDF files, which are discussed in later sections. In a typical PDF file that was created once, there is normally one instance of the cross-reference table. The file trailer provides references to each cross-reference table such that information in each table can be linked together and assembled to represent the full contents of the file. Following the cross-reference table is the file trailer which is located the end of the file.

In version 1.5 of the PDF specification, the traditional cross-reference table can be replaced with a cross-reference stream object. This means that the information and

format of the cross-reference table can be represented and compressed into a block of data, which is later decompressed by a reader and parsed when the PDF file is opened. Cross-reference streams are discussed in greater detail in PDF 4.14:. It should be noted that the traditional cross reference table presented in the previous paragraph is for older versions; readers today support the usage of both formats. An image representing the file structure with a cross-reference stream is shown in Figure 3-2.

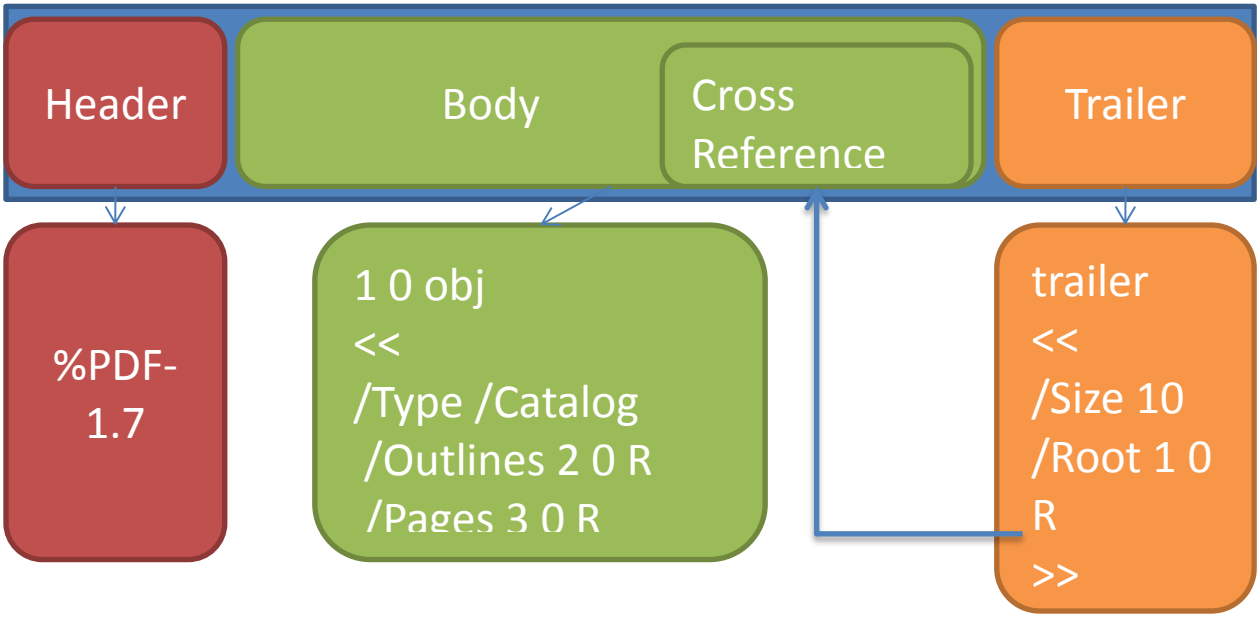

Figure 3-2. PDF with Cross Reference Stream Object

The file trailer is a special type of object that provides a reference to the cross-reference table information and its size, the root indirect object (Document Catalog), and possibly the encryption dictionary when the document is encrypted [5]. The trailer is important because it identifies the location of the cross-reference table. It identifies the root of the PDF document structure. The body of the PDF contains the whole group of objects; the trailer is needed to identify the root object of the document hierarchy.

Certain information regarding the file structure may sometimes be inaccurate, which means the file has been illegally modified in some manner. Information in the trailer is critical to the file structure since it defines the root object and cross-reference information. The contents of the cross-reference table may not always be correct and trusted. Most PDF viewer applications and sanitization applications appear to reconstruct these entries if the file is determined to be damaged or invalid in some way. While if the standard is followed and trusted, a PDF viewer application can parse the file starting at the end of the document, locate each object in the file and build the document structure based upon information that is present in the file.

3.2 Document Structure

The PDF document structure is as important as the file structure, it is another view that may be applied to indicate how the document itself is structured, not the actual underlying PDF file. As discussed before, a PDF file is composed of four sections, and the body is comprised of numerous discrete PDF objects that describe a specific aspect of the document. Objects are linked together through a known hierarchy described ultimately in the ISO standard. Objects may be reused as they can be referenced through other existing objects, as long as it is done properly as defined in the ISO standard. For example, a line of text may reference a font or formatting object that describes how the text should appear. Similarly, other sections of text may reference similar font, or a page may reference the same blocks of text. The document is a hierarchy of objects that are contained within the body section of the PDF file, as shown in Figure 3-2. Each object shown in the figure is a dictionary that describes the object, references and relations to other objects within the body of the PDF file.

The root of the tree is the Document Catalog object, which is a dictionary that can have a number of parameters associated with it, including the page tree object and outline hierarchy object. As a PDF viewer application parses through this hierarchy it has the means to reconstruct the contents of the document on page by page and object by object basis. The trailer in Figure 3-1 identifies the document catalog with field /Root, which indicates where the viewer application should begin parsing the document.

The Document Catalog defines a reference to a Pages object. This object is responsible for defining the number of pages in the entire PDF document, and providing references to each single page object. A page object may define resources, graphics, or contents that are applicable to that single page. Thus beginning with the Document Catalog, the contents of the PDF document can be drilled down into to find the most basic object type. This represents a path through the document hierarchy, and represents details on how the document appears on each page for the reader. All of the objects and references ultimately describe the PDF document structure, which is used by reader software to represent the content of the document to a viewer.

3.3 PDF Embedded Contents

PDF files are a hierarchy of objects and dictionaries defined in the ISO standard. The hierarchy defines detail down to the content of a single page. A significant consideration for inspection and sanitization is the recursion level provided by any utility. One can think of recursion, the ability to sanitize and inspect files contained in a file, in terms of depth and width. For instance, if a PDF file contains a reference to an external file or a file embedded directly within the document, the inspection would need to include recursion that is one level deep. If the embedded PDF file also contains

an embedded PDF file, then the recursion capabilities would need to go two levels deep. Recursion width also can present an issue. For example, a PDF file may contain two un-nested embedded PDF files. This example would result in a recursion depth of one and a width of two. Recursion can impose a significant performance penalty on inspection and sanitization tools, especially as depth and width of recursion increases. Recursive processing potentially may generate complex inspection results also, which may not be intuitive for average users to understand.

PDF files are unique because binary files can be compressed and embedded into a single file. The PDF format allows for a wide variety of file types, as well as images and multimedia to be compressed and stored within the PDF file. Much of the PDF file contents are located in stream objects, which will be discussed later in the report. At a high level, streams may contain raw binary data (embedded file) or content information which instructs reader software to properly render textual and graphic image onto a page in the PDF file. Regardless, all types of stream data can be compressed through numerous iterations of compression algorithms. For software attempting to analyze this information, it is paramount that it is understood that this information must be properly decoded before it can be thoroughly inspected. Content stream data that contains text and graphical information can be further inspected once properly decoded into a readable format.

3.4 Document Coverage

The ISO standard contains numerous references to other documents. This guidance document is written to be standalone to a certain degree, for readers wishing to learn more about concerns and features within a PDF document. Although for readers that want full descriptions and more details, the following documents should be consulted:

- ISO 32000-1 Document Management – Portable Document Format – Part 1: PDF 1.7 [1]
- Adobe®[2] Extensions to ISO 32000-1:2008 [6]
- Open Geospatial Consortium Inc, OGC 08-139r2. GeoPDF Encoding Best Practice Version 2.2 [7]

There are numerous references that are of interest but were not focused on in this report. Developers and readers should consult them to investigate more about their respective functionality in PDF documents. The following documents are additional references that may be of interest to the reader:

[2] Adobe is a registered trademark of Adobe Systems, Inc.

- JavaScript®[3] for Acrobat®[4] API Reference, Version 8.1 [8]
- XML Forms Architecture (XFA) Specification, Version 3.1 [9]
- Map Projections – A Working Manual. USGS. [10]

3.5 Assumptions

This report analyzes various features of the PDF ISO standard, and describes concerns and provides recommendations based upon each feature. PDF supports a wealth of embedded content and rich media features that are beyond the scope of this document. Many inspection and sanitization efforts will rely on the use of external filters that are capable of parsing data that belongs to another format. PDF inspection and sanitize must be able to determine if data in the file is properly encoded, extract this information in a safe manner, and present it for further inspection. Some content within PDF documents, such as encoded content streams, must be decoded and examined by this inspection process, because it uses constructs and commands defined by the ISO standard.

Throughout this document, many recommendations are listed to handle specific features of the PDF standard. Validation of PDF features can generally imply checking for required entries that must be present in the object's dictionary and it must be implemented properly (with the correct object type or reference) as defined in the standard. There are many references to ensuring referential integrity. In PDF dictionaries, there can be indirect references to other PDF objects. Referential integrity ensures that the object referenced must point to a valid object in the PDF file and of the correct object type as specified through the standard.

3.6 PDF and PostScript Conversion

This section summarizes the risks and issues known with PDF and PostScript (PS) conversion [11]. PostScript (PS) is a Page Description Language (PDL) and a printer-specific language that was design to render and represent text and page contents. Since PS is a device-dependent language, Encapsulated PostScript (EPS) was developed to remove the dependency on the device and is a subset of PostScript. Software tools have been created that allow the conversion between PS and PDF, as well as EPS to PDF, and vice versa. Originally, the imaging model in PDF was similar with PostScript, but that has changed with the introduction of PDF-1.4 were many new features were introduced.

[3] JavaScript is a registered trademark of Oracle Corporation
[4] Acrobat is a registered trademark of Adobe Systems, Inc.

The main issue with the conversion between PDF and PS, or EPS, is that there are several features and support for image quality added in later versions of PDF that are not supported in both PS and EPS. Therefore, converting a PDF document to PS or EPS will lose those unsupported features. Converting PS or EPS to PDF is still acceptable, but the process of converting PDF to PS/EPS and then back to PDF can be lossy. The term for this process is called "refrying" a PDF document. It is highly recommended to only use EPS if this process must be done, since it is device independent. This section identifies some of the features that introduce problems in the "refrying" process.

Images in PDF support up to 16-bits per component, which is an increase over PS and EPS which supports 8-bits per component. Therefore, all images in a PDF-PS/EPS conversion will lose half of each pixel. This could be very degrading for the quality of a PDF document. In later versions of PDF, newer image compression such as JPEG2000 and JBIG2 were introduced. PS and EPS do not support these technologies, making the conversion from PDF to PS/EPS problematic with those image types. A solution would require converting the image in the original PDF to a supported format before the conversion to PS or EPS.

Later in this report, a discussion on fonts in PDF files is presented (Section 5.2). In PDF, support was added that allow text to be extracted from the document through the `/ToUnicode` field, which converts the character IDs to Unicode values so they can be extracted from the document. This supports operations such as copy and paste and searching, but would not be supported once converted to PS or EPS.

PDF supports device-independent color spaces, including the International Color Consortium (ICC) profiling technology. However, PS was designed to be device-dependent and will not support these color profiles, which will be lost in the conversion between PDF to PS. EPS does support a single ICC profile for the entire content, and not multiple ICC profiles used in the document.

"Refrying" PDF documents will cause data loss, particularly in image data, but also if more ICC profiles are used in the document, and functionality concerning fonts and text. In general, PS should not be used since its device dependent, and if refrying is ever to be performed, EPS Level 3 should be used instead.

4. PDF HIGH LEVEL CONSTRUCTS

PDF files contain objects that are organized to implement and represent the content in the body of the file. The file structure including the header, cross-reference table, and trailer have a generally expected format, but the PDF body contains the discrete objects that comprise the description of the PDF file. The file structure has been defined in the previous section, but to discuss features in greater detail, the format and syntax of objects needs to be introduced.

Objects in PDF files are represented by a standard format in the document as shown below. In this example, the "%" represents a valid comment that is inline with content describing the purpose of each line of a generic PDF object. Each object in the dictionary is preceded by the character "/" which indicates that it is of type "Name" and is followed by a keyword that is defined in the ISO standard. Objects in the body implement dictionaries, which contain a listing of paired objects, defined with a key and value pair. Objects can be defined by using the name as the key and a value providing an assignment to that object (e.g., strings, integers).

```
1 0 obj %First number is the object number (1), second number is the generation (0)
<<
    /Type /Catalog %Type is a special keyword defining the type of this object.
    /Pages 2 0 R   %This is a reference to object 2, generation 0
>>
endobj %The end of the object
```

Figure 4-1. Example PDF Indirect Object and Dictionary

The PDF file contains a Document Catalog object that defines references to the next level of the objects in a tree-based hierarchy. References in PDF files use the object label, for example "`1 0 R`" would reference the object describe above. This allows for reusability of objects as well as methods to properly organize all objects within the file. For example, the PDF file may contain a font object and several text objects that reference the same font object.

Each indirect object within the PDF file serves a purpose and provides data so the reader application can properly render a document resembling this description. Each indirect object has a defined type. Data within an indirect object might be references to other objects, numerical or string data, or embedded stream data that is encoded and appended to the end of the object. Examples in this document will further describe many more objects.

Understanding the syntax of the PDF internals is most important when analyzing or sanitizing a PDF document. As mentioned before, the body of the PDF is comprised of objects. These objects can be of numerous types: integers, real numbers, name objects, strings, arrays, dictionaries, stream objects, and the null object. All objects other than PDF dictionaries and stream objects resemble what is shown in Figure 4-2. They contain a value between the obj and endobj keywords. These objects can also be implemented directly in another PDF dictionary.

```
1 0 obj
100                    %This is an integer object, with a value of 100
endobj

2 0 obj
(This is a string)     %This is a string object
endobj
```

Figure 4-2. PDF Objects

Dictionaries and stream objects are different than the other objects. They can contain multiple objects and even PDF dictionaries within them. PDF Dictionaries are encapsulated by the characters "<<" and ">>", a difference between the objects shown above as compared to the objects shown below in Figure 4-3. PDF dictionaries may implement other objects in a key-value pair as shown below. The value on the left must be an expected name from the ISO standard, or otherwise it will be ignored. The value must be spelled correctly and it is case-sensitive. Some readers may attempt to fix case-sensitive issues, but any PDF writer should use the correct name and correct case. The value on the right must be one of the different objects or an indirect reference to an indirect object.

This section provides a small example describing the initial contents of the PDF file including the Document Catalog, the Outlines object, and the Pages object. The example is shown in Figure 4-4. The Document Catalog is object 1 0, which is of type "`Catalog`" and contains indirect references to two objects: `Outlines` and `Pages`. The PDF document does contain a single page (as a PDF must have at least one page to be valid) which is described in the Pages object in 3 0. The pages object defines the Page Count attribute, as well as the Kids array which define references to each page. The type of this object is `Pages`. The page object itself is represented in object 4 0. The more details, contents, and objects that are present in the PDF document, the number of objects and references will increase. This introduction describes how there can be numerous objects in the PDF body and how they are referenced and implemented.

```
3 0 obj              %This is a dictionary object
<<
   /Type /Catalog    %This is a Name Object inside a dictionary
   /Lang (en-US)     %This is a String object inside a dictionary
>>
endobj

%This is a stream object
4 0 obj
<<
    %Dictionary is here, stream data follows it.

   /Length 100     %This is an integer object within a stream object
   /Filter [/FlateDecode /DCTDecode] %This is an array of Name objects
>>
stream
…encoded data…
endstream
endobj
```

Figure 4-3. PDF Dictionary and Stream Objects

```
%PDF-1.7
1 0 obj         %Object 1 0 is the Document Catalog, root of the document.
<<
 /Type /Catalog %The type is catalog, a Named object.
 /Pages 3 0 R   %Indirect reference to a Pages object in 3 0
>>
endobj          %End of Object 1 0
3 0 obj         %Object 3 0 is the Pages Object referenced by the Document catalog.
<<
 /Type /Pages   %The type is Pages
 /Kids [4 0 R]  %It has one page in object 4 0, indirect reference to object 4 0.
 /Count 1       %Page count is 1
>>
endobj          %End of Object 3 0
4 0 obj         %Object 4 0 is a Page object referenced from Object 3 0
<<
   %Page 1 contents start with this object
   /Contents 5 0 R  %The contents of this page are defined in this field
>>
endobj          %End of Object 4 0
5 0 obj
<<
    /Filter /FlateDecode  %A name object (can be an array)
    /Length 6 0 R         %An indirect reference to an integer
>>
stream
%100 bytes of encoded stream data
endstream
endobj
6 0 obj
100              %A simple integer object (not a dictionary)
endobj
…                %Remainder of document and indirect objects.
xref
0 9
0000000000 65535 f
0000000032 00000 n
0000000109 00000 n
…
%Rest of xref table
trailer
<<
 /Size 9
 /Root 1 0 R  %Points to the Document Catalog in Object 1 0
>>
startxref
642
%%EOF
```

Figure 4-4. Example PDF Syntax

4.1 PDF File Structure

PDF 4.1: PDF HEADER

DESCRIPTION:

The PDF header is a one-line comment in the beginning of the PDF file that provides a suggestion as to which version of the specification that the file conforms to. Typically a comment can be added to the line afterwards, making the header two lines. The second line comment contains at least four high order ASCII characters. The standard does not indicate a maximum number of characters.

CONCERNS:

A header is typically not a great concern for inspection, but it indicates a suggested version of PDF that this file conforms to. This can be overridden with a `/Version` entry in the Document Catalog object, described later in PDF 4.7:. This may be a concern with some readers if they ignore content not applicable to the latest version of the standard, provided the file indicates an earlier version. The second line of the header may also be of concern since it is a comment and can be several bytes in length. The second line may contain hidden data, data being disclosed in a comment, or malicious data.

Having an invalid header in a PDF file is a tactic used by many malware authors; this would introduce a data attack risk if data came from an untrusted network. Although some readers may discard this as an invalid PDF file, which is correct. However, by placing bytes before the header of arbitrary or empty data, it may be possible for attackers to trick detection methods or insert data at the beginning of the file.

PRODUCT: PDF-1.0

LOCATIONS:

The header is located at the very beginning of the file. It is always of fixed size and must be the first eight bytes in the file, starting with a comment. Additionally, at least five extra bytes (including the comment '%' character) may be added to the next line.

EXAMPLES:

```
%PDF-1.7
%µµµµ
```

RECOMMENDATIONS:

1 **Validate:** Check for consistency by validating the proper length of the first line, and that the content matches %PDF-*x*.y, where x and *y* are both valid numbers, according to the latest standard numbering system. This needs to be on the first line of the file.

2 **Validate**: Check consistency when a comment line exists after the first header line, ensure that it contains four high value ASCII characters preceded by a '%'.

3 **Remove**: Remove the comment on the second line of the file. This may cause older tools or hardware to improperly process the file.

4 **Remove**: If bytes exist before the header, remove any bytes before the header in the file such that the first line in the file matches the %PDF-*x.y*, where *x* and *y* represent valid versions of the ISO standard.

5 **Replace**: Replace the first line with a valid header (%PDF-x.y), where *x* and *y* represent the correct numbers for the latest ISO standard.

6 **Replace**: Replace the second line with a comment (%) and then fixed four characters of high value ASCII.

7 **External Filtering Required**: N/A

8 **Review**: N/A

9 **Reject**: Fail documents that do not provide a correct header according to the first two validation actions.

PDF 4.1: END

PDF 4.2: TRADITIONAL CROSS-REFERENCE TABLE

DESCRIPTION:

The traditional cross-reference table is located near the end of the PDF file as it contains a lookup table of where each object, its generation number, and it's byte offset in the body of the PDF file. It also defines whether or not the object number is in use throughout the file. The object number may be free for others to use since it is not defined in the body. Since the cross-reference table presents an ordered listing, it is observed and possible to list both used and unused object numbers. The cross-reference table may be split into subsections, which indicates that it contains multiple sections of listed objects. For instance, the first subsection may define objects 1 through 9. While the table can still continue listing objects, but the starting object number in the next subsection may begin at object 13 (leaving objects 10 through 12 undefined).

Traditional cross-reference tables can be replaced with a cross-reference stream, which is introduced in Section PDF 4.14:.

CONCERNS:

The cross-reference table is important because it provides the byte location of each object in the PDF file. As observed, this information may be inaccurate but an invalid PDF file can still be rendered correctly. A PDF reader may realize that the byte locations do not point to valid locations where an object begins, but may still be able to parse the file correctly without needing it. The file would be recreated from scratch based on the objects in the document. In this case, an incorrect cross-reference table may be a location for hidden data or malicious data. This information would never be read by a conforming PDF reader.

PRODUCT: PDF-1.0

LOCATIONS:

The cross-reference table is traditionally near the end of the PDF file, and its location is denoted by the *startxref* value in the File Trailer (PDF 4.3:). The cross-reference table begins with the value xref on the first line and is followed on the next line by two digits: the starting object number of the first object in the table, and the second number is the length of the block of objects. This applies to the cross-reference table until it ends or until a new two digit line is displayed which can restart the object numbering block again (Example 2). The actual contents of the table consist of an entry with three items: the first is a 10-digit number indicating the byte offset of that object in the file, the second number is a 5-digit generation number (0 for new objects, may be changed for updates), and a 1 character flag indicating whether the object number is in use or free (n/f). Each entry is 20 bytes long. The first entry, object 0, is always free, and always has a generation number of 65535.

EXAMPLE 1 (NORMAL XREF TABLE):

```
xref
0 9
0000000000 65535 f
0000000032 00000 n
0000000109 00000 n
0000000165 00000 n
0000000234 00000 n
0000000401 00000 n
0000000505 00000 n
0000000538 00000 n
0000000658 00000 n
```

EXAMPLE 2 (XREF TABLE WITH SUB-SECTIONS):

```
%The break in the table indicates the order starts at Object Number 7
%For the next sub-section, object number 6 is not defined.
xref
0 5
0000000000 65535 f
0000000032 00000 n
0000000109 00000 n
0000000165 00000 n
0000000234 00000 n
0000000401 00000 n
7 3
0000000505 00000 n
0000000538 00000 n
0000000658 00000 n
```

RECOMMENDATIONS:

1 **Validate**: Check for consistency by verifying the traditional cross-reference table is syntactically correct.

2 **Validate**: Check for consistency by validating the keyword on first-line, correct second line with starting object number and size. Ensuring that the following lines are in the correct format and correct number of entries, including a free object number 0. In the case of a table with subsections, each subsection should be validated to ensure that the object number space does not overlap and that each field is of the correct length and size.

3 **Validate**: For each entry in the cross-reference table, the byte offset should point to a valid location in the file that contains the proper first line heading for an indirect object in the body of the PDF file (e.g., 1 0 obj).

4 **Validate**: Validation can be extended to ensure that there are no overlapping byte offsets in the cross-reference table. This includes parsing the body to determine each byte offset of each indirect object and validating that they are correct in the cross-reference table.

5 **Remove**: N/A

6 **Replace**: Reconstruct the cross-reference table by parsing the body of the file, producing an accurate representation of the file contents.

7 **External Filtering Required**: N/A

8 **Review**: N/A

PDF 4.2: END

PDF 4.3: FILE TRAILER

DESCRIPTION:

The PDF standard defines a File Trailer that allows the reader software to successfully locate the cross-reference table information and special objects within the PDF file, which means that a PDF reader begins parsing with the file trailer at the end of the file. The trailer contains a reference to the Root object or the Document Catalog.

CONCERNS:

A file trailer is required to be present and its information helps a reader initially parse the file. Some of the fields are required to be correct or else parsing of the document can fail. Information in this dictionary must be validated to ensure there are no data attack risks involved. Since the file trailer is a dictionary just as any PDF dictionary, data can be hidden inside this dictionary if the contents are not officially registered by the ISO standard.

PRODUCT: PDF-1.0

LOCATIONS:

The trailer is a PDF object dictionary that is located at the end of the PDF file just before the end of file marker (%%EOF). It is designated by the keyword `trailer`. Inside the dictionary are the required fields: `Size` which indicates the size of the cross-reference table, `Root` which indicates the object number of the document catalog. Optionally, the fields `Prev` (if there is an incremental update to the file), and `Encrypt` (if document is encrypted) and `ID`, which is required in modern versions of PDF. The ID field is used as part of the document decryption algorithm, and can also be used when importing file content into another PDF file.

Following the trailer dictionary is a line indicated by the keyword `startxref` which indicates the byte location in the file of the cross-reference table.

EXAMPLES:

```
trailer
<<
   /Size 9   %9 objects in xref table
   /Root 1 0 R %Document catalog at Object 1 0 in body
>>
startxref
748 %Byte offset 748 contains cross-reference table
%%EOF
```

RECOMMENDATIONS:

1 **Validate**: Validation of the trailer is to check that it is syntactically correct with the valid "`trailer`" keyword, and the correct required fields in the dictionary.

2 **Validate**: Check that the size field indicates the correct size of the correct cross-reference table and that the root field is an indirect reference to the Document Catalog object. Ensure that following the trailer is the keyword "`startxref`" and that the next line contains the correct byte offset to the correct location of the cross-reference table or cross-reference stream object.

3 **Validate**: Ensure consistency that the file is terminated with the `EOF` comment keyword and that no data exists following this. In the special case where the file contains incremental updates, please refer to that section for additional information on validation.

4 **Validate**: If the document is encrypted, ensure referential integrity by validating that the Encryption dictionary is properly defined and that the document is validated according to the Encryption section (PDF 4.18:). Ensure that the `ID` field is also implemented correctly, as required by the standard when Encryption is used.

5 **Validate**: Ensure referential integrity when the `/Prev` field is defined. This can occur with Incremental Updates (PDF 4.16:) or Linearized PDF files (PDF 4.17:).

6 **Remove**: Remove unregistered objects within the trailer (those not registered with the standard).

7 **Replace**: Replace the trailer with a valid size field and root indirect reference to the Document catalog. This is this least amount required for the document to be viewed correctly by reader software. Replace the `startxref` with the actual location of the cross-reference table or cross-reference stream object.

8 **External Filtering Required**: N/A

9 **Review**: N/A

10 **Reject**: Fail documents that do not supply consistent trailer information.

PDF 4.3: END

4.2 PDF Objects and References

PDF 4.4: BASIC OBJECTS

DESCRIPTION:

There are numerous object data types defined by the PDF ISO standard. The following objects are possible within a PDF file and are identified as the major basic types:

Table 4-1. PDF OBJECT TYPES

Type	Format	Example
Boolean	true or false	/V true
Integer	Integer values	/Count 1
Real Numbers	Real Numbers	/MyReal 2.0
Strings	Strings are identified between '(' and ')'	/Desc (This is a string for descriptive text)
Name Objects (PDF-1.2)	Defined by a '/', then following by [A-Z, 0-9]	/MyNamedObject 1.0
Dictionaries	Contents between '<<' and '>>'	… /Resources << %Dictionary contents in %between >> …
Streams	A dictionary with defined fields, and followed by a block of data in between stream and endstream.	<< /Length 400 /Filter /FlateDecode >> stream <binary data> endstream
The Null object	Null	%Simple example /V null
Arrays	Defined encapsulated with '[' and ']'. Arrays can contain mixed object types	%Array with two indirect references /Kids [1 0 R 2 0 R] %Example with mixed object types /Dest [3 0 R /XYZ null 701 null] %Array includes %3 0 R is an Indirect Reference %/XYZ is a reserved name %null %701 is an integer %null

Streams are more complex and covered later in PDF 4.10:; the remainders of the objects are analyzed in this section. Basic dictionaries were presented earlier in this section to introduce the syntax of the PDF file, this section presents more detail and how they are represented. Dictionary objects can exist as an indirect object in the PDF body or they can be nested within other indirect objects, thus the dictionary type. Booleans, integers, reals, strings, arrays, dictionaries, streams, and null are defined by their values, which correspond to the appropriate value for that data type. A name object is unique because it is essentially a symbol that defines the value for that name, which can be of another type. The name is preceded by the "/" solidus character.

CONCERNS:

Objects and dictionaries are vital to PDF documents and represent the structure of the PDF file. If an object or data type is introduced into a dictionary, there must be a legitimate reason for its existence and a registered name. It must be referenced by another construct in an approved manner or else the data is orphaned and serves no documented purpose. Simply referencing another object does not imply that it is in use, it has to be used by a known data type from the standard. The risk is that when an object, reference, or variable clearly serves no purpose, it can be suspect for hidden data or a hidden message. Especially true since string data can be defined within objects themselves or values of an object.

PRODUCT: PDF-1.0 (DATA TYPES), PDF-1.2 (NAME OBJECTS)

LOCATIONS:

The body of the PDF contains a number of indirect objects, which can be a dictionary object which defines other objects or dictionaries within that indirect object. Objects and name objects can appear in any dictionary or sub-dictionary throughout the body and trailer of the PDF document. Name objects can be defined within objects themselves by introducing a new /NewObject <Value> pair. Examples below demonstrate how objects and dictionaries are defined.

Annex E of the PDF standard should be consulted for a more detailed description of the PDF Name Registry.

EXAMPLE:

```
%It should be noted in this example that all of these objects within
%are essentially unused, since they have no impact on the document structure or
%content. It is more an example showing the different types of variables and
%data types that can be defined.

%All of these are not valid registered names with the ISO standard, but it is
%valid PDF syntax. This information can be mixed with valid names.
1 0 obj
<<
   /MyBoolean true
   /MyInteger 1
   /MyNegativeInteger -1
   /MyRealNum 34.5
   /MyString (StringContents!!!&^})
   /HiddenString [2 0 R 3 0 R]  %Points to objects below
   /MyHexString <4E6F462033696DAF6A206B612…>
   /MyArray [200 5.5 (String) 1 /ARandomName]
   /MyDictionary
   <<
       /MyNameObject 1.0
   >>
```

```
      /MyObfuscatedN#61ME (MyObfuscatedNAME) %#61=A, hex, ASCII substitution
      /MyNull null
      …
   >>
   endobj

   2 0 obj
   (This is a)
   endobj

   3 0 obj
   (Hidden String)
   endobj

   …
   %rest of document
   trailer
   <<
      /Root 1 0 R
      /Size 10
      /TrailerString (A string value)
   >>
   %%EOF
```

RECOMMENDATIONS:

1 **Validate**: Check for consistency by validating that each object is of proper type. Check that it is a first-class name and its value is properly assigned.

2 **Validate**: Check that lengths of each data type value should be validated against constraints of the host system or architectural limit of the data type.

3 **Validate**: Names should also be validated such that there is no ASCII or non-standard substitution in cases where it is not needed to escape for non-ASCII values (e.g., J#61vaScript=>JavaScript). Check that this is only used for legitimate purposes.

4 **Remove**: Remove objects that are not registered by the ISO standard or otherwise publicly documented.

5 **Replace**: Replace typical ASCII substitution if the character is a printable character to deobfuscate the name. This is only used in cases where the escape should not have been necessary and clearly an attempt at obfuscation. In many cases, the escape is necessary and cannot be replaced.

6 **External Filtering Required**: Pass all string objects in any dictionary to an external filter.

7 **Review**: N/A

PDF 4.4: END

PDF 4.5: UNUSED OBJECTS

DESCRIPTION:

Objects can be defined in the body of the PDF and inside dictionaries. In a proper PDF file, all objects are referenced correctly as defined in the ISO standard, and there are no additional objects that are left out of the hierarchy of objects in the PDF structure. It can be observed that objects can be introduced without any purpose or referenced by other objects in the PDF document.

CONCERNS:

If an object or data type is introduced into a PDF, there must be a legitimate reason for its existence. There may also be the case where the object is metadata or part of a workflow and not actually related to the content of the PDF file. Simply referencing another object with valid PDF syntax does not imply that it is in use, it has to be used by a known and registered data object and its usage has to be clearly indicated by the standard. The risk is that when an object or reference which is undefined by the standard, but used in the document, then it clearly serves no purpose, and it can be suspect for hidden data or disclosure risks.

PRODUCT: PDF-1.0

LOCATIONS:

Objects can be located in the body of the document as indirect objects. They can also be defined within indirect object dictionaries arrays or streams, as nested objects. They may also appear, although this is not allowed by the standard, in the trailer object.

EXAMPLE:

```
1 0 obj
<<
    %Not a registered name in ISO, but can exist
    /MyVariable (This object is never used, but a hidden string)
    /Type /Catalog %Expected keywords and types.
    /Pages 2 0 R    %Indirect reference
    …
>>
endobj

%Unused or unreferenced objects that are valid but not referenced anywhere
else.
```

RECOMMENDATIONS:

1 **Validate**: Ensure consistency and validate that each object in the PDF file maps to proper usage in the ISO standard or other public documentation. Ensure that it is not orphaned or unused and that referential integrity exists.

2 **Remove**: Remove objects in the body of the PDF that are orphaned and not referenced by other objects.

3 **Remove**: Remove objects within objects that are not required, registered, or using valid first-class names.

4 **Replace**: N/A

5 External Filtering Required: N/A

6 Review: N/A

PDF 4.5: END

PDF 4.6: INDIRECT REFERENCES

DESCRIPTION:

Indirect references are widely used throughout PDF files. They are similar to pointers that reference other indirect objects that can simply contain a value, or even an entire PDF dictionary. Indirect references are often required by the standard for some objects. For example, the Root object in the File Trailer shall be an indirect reference, which points to the Document Catalog, which is implemented as a separate indirect object. An indirect reference is the value of a particular field in a dictionary or array. They are also useful for reusability, rather than re-implementing objects, indirect references can be utilized by many objects.

CONCERNS:

Indirect references may be a concern since they create a layer of indirection between two objects. Although this is a useful concept for PDF developers as it enables reusability, a sanitizer should be aware of these relationships and ensure that they are not used for malicious purposes (long chain of indirect references, circular references) that could possibly exploit reader software as a data attack risk. Hidden data may also be a risk since references can join or link information together possibly obfuscating data.

PRODUCT: PDF-1.0

LOCATIONS:

An indirect object reference is located across two objects, one containing the reference to the object, and the other the actual object containing a value. An example where it is used is in a stream length in the example below. It is designated by the number and generation of the object, followed by the letter `R`. For example, to reference Object 12 0, the indirect reference would be `12 0 R`.

In the example below, the Length object in the stream dictionary references indirect object 7 0. Object 7 0 is defined later in the file as an integer value, which is not a dictionary. This can be done, as well as indirect references to indirect objects implementing a dictionary.

EXAMPLE:

```
6 0 obj
<<
    /Length 7 0 R   %Indirect reference to object 7 0, which is simply a value
>>
stream
    %stream contents, 200 bytes long as defined in object 7 0
endstream
endobj

%This is an indirect object with no dictionary, just an integer value, which is
```

```
%referenced by object 6 0.
7 0 obj
   200
endobj
```

RECOMMENDATIONS:

1 **Validate**: Check for consistency and that indirect references are used for only approved purposes, in cases where they are mandated by the standard. In many cases, it may not matter and the standard allows either implementation, so these are acceptable. The standard will indicate that it expects an indirect reference as opposed to a nested dictionary.

2 **Validate**: Check for referential integrity and that references are not chained or a link or indirect references are used nonsensically to promote obscurity.

3 **Remove**: Remove indirect references that point to unused indirect objects. In this case the referring object will not be a valid first-class object.

4 **Replace**: Replace indirect reference with the value or dictionary that is being referenced.

5 **External Filtering Required**: N/A

6 **Review**: N/A

PDF 4.6: END

PDF 4.7: DOCUMENT CATALOG

DESCRIPTION:

The document catalog is the root of the PDF file and contains references to several other PDF objects providing actual contents of the document. It is identified by an indirect reference in the trailer of the PDF file, so the trailer indicates which object in the PDF file is the root of all objects.

There are numerous fields in the Document Catalog as shown in Table 28 of the ISO standard. Some of the fields and their respective concerns are addressed in the Concerns section below.

CONCERNS:

The Document Catalog introduces several fields that allow for interaction. Fields such as `/OpenAction` and `/AA` invoke an action to occur when the document is opened. This presents a data attack risk.

The Document Catalog can provide a dictionary for `/URI`, This provides a base URI for the document, in which further URI information can be relative to that base in the document. Any URI or fully qualified path (base or relative) may also be subject to a data disclosure risk.

A field such as `/SpiderInfo` is also a concern since it provides information on how the document was created with external content. This does not perform any action to retrieve information when a PDF is viewed, only at the creation time may software retrieve external

information. This data represents a type of metadata that describes how the information was retrieved and where it came from, which can introduce a data disclosure risk.

The Document Catalog also defines the Names dictionary, which may include interactive objects (such as JavaScript) in a Names tree. In some cases, objects listed in the Names tree through the Document Catalog can be executed when the document is opened. This could possibly introduce a similar data attack risk. The Names tree can also include other objects which need to be individually evaluated. This is discussed in later sections (PDF 4.9:).

The Document Catalog also provides a `/Lang` field which is a free text string that indicates the language identifier. This is similar to many other types of meta-data information that can reside in the document.

PRODUCT: PDF-1.0

LOCATIONS:

The document catalog is a PDF object defined in the body of the file. The trailer of the PDF file will specify a root object, which informs the viewer application to begin parsing the tree of PDF objects at this location. The object is identified by the `/Type` of `/Catalog` and there are numerous parameters that can be included in this object. For a full listing, the ISO standard describes these parameters in Table 28. A few objects of interest are `/OpenAction`, `/AA` (additional-action), `/SpiderInfo`, and `/Names`, although there are numerous others that are mentioned in this report.

The Document Catalog references the Pages tree (PDF 4.8:), the Name Dictionary (PDF 4.9:), and Page Labels, which contain references to a Page Label dictionary, which could contain string information about page information.

The Document Catalog also implements a `/Perm` field which identifies the permissions dictionary, which indicates the user access permissions for this document. The permissions dictionary references a signature dictionary (PDF 5.72:) or a usage rights dictionary. This is not the same as encryption or DRM permissions (PDF 4.18:). This is used as part of the Certified Document feature.

EXAMPLES:

```
1 0 obj
<<
    /Type /Catalog
    /Pages 2 0 R
    /PageMode /UseOutlines
    /Outlines 3 0 R
    /OpenAction 4 0 R %Document open invokes action
    /AA << /O 10 0 R >>        %Additional actions on open

    /Perms 13 0 R %Permissions on the document in another dictionary

    /AcroForm 12 0 R
>>
endobj
… %rest of document
xref
0 7
… %rest of xref table
trailer
<<
```

```
        /Root 1 0 R %Document catalog is at 1 0
        /Size 7
>>
```

RECOMMENDATIONS:

1 **Validate**: Ensure consistency and that the dictionary has been implemented properly with the required fields.

2 **Validate**: Ensure referential integrity such that the trailer points to it through the `/Root` field, and that each indirect reference in the Document Catalog points to the correct corresponding object type.

3 **Remove**: Remove references to OpenAction, Additional Actions, and SpiderInfo objects. This may orphan action objects or spider info information, which means they may be unused and may need to be removed as well.

4 **Replace**: N/A

5 **External Filtering Required**: N/A

6 **Review**: N/A

PDF 4.7: END

PDF 4.8: PAGE TREE AND PAGE OBJECTS

DESCRIPTION:

As objects in PDF have been presented, one of the most important objects is a Page Tree Node, which comprises a structure called the Page Tree. Within the Page Tree Node are references to other Page Tree Nodes or Page Objects. Page Objects define a single page of the PDF document. Each Page Tree Node implements a `/Count` and an array called `/Kids` which define the children of that Page Tree node, which may be more Page Tree nodes or Page Objects (the leaves of the Page Tree).

The Page Tree allows for certain inheritable parameters to be implemented in a Page Tree Node, as opposed to the actual Page Object. If a number have pages have the same page boundaries, this information can be applied at a Page Tree node and used by each Page Object that is inherited from that node.

The Page Tree defines an important parameter to PDF pages, which imply the page boundaries. This is an important structure because each object on the page is hopefully placed within these boundaries. Each of these coordinates is defined in User Space, which is discussed in PDF 5.2:. The Page Tree and Object can implement five different boxes called the Media Box, Crop Box, Bleed Box, Trim Box, and the Art Box. The Media Box is the boundaries of the physical medium of the displayed or printed page. The Crop Box is the visible region of the page, which can be clipped from the Media Box. The Bleed Box is the region of the page which contains the contents and may be clipped when output in a production environment [1]. The Trim Box defines the

finished page dimensions after trimming in a production environment. Lastly, the Art Box defines a rectangle that defines the page's meaningful content, as determined by the PDF creator. If the Art Box, Trim Box, and Bleed Box are not defined, they will default to the Crop Box rectangle value. If the Crop Box is not defined, it will default to the Media Box. The Media Box is a required parameter for the page, and if this is the only one defined, then all of them are equal to one another. Figure 4-5 illustrates these boxes if they were all different in size. The standard does not require each box to be in this arrangement, as they can be defined in any order or size, but clipping can occur in cases where it is not done properly.

The Page Tree also allows for an inheritable field called /Rotate which identifies in degrees how to rotate the page clockwise. There are also additional non-inheritable fields for Page Objects such as Annotations for that page, discussed in Section 5.7.1. Also available are Group attributes for transparency (PDF 5.23:), Metadata for that page (PDF 5.1:) and Page-Piece information (PDF 5.74:) along with a LastModified date.

The Page Object can also implement an additional action object which may trigger an action to occur when opened or closed, as discussed in Section 5.7.3.1.

The Page Object also implements a Duration and Transition field which control how the PDF might be viewed in a slide-show mode, which in some readers when viewed in full screen mode. These values should be examined carefully to ensure information is not hidden if the reader defaults to this mode of viewing. If the Duration is set to a high value, it has been observed in reader software that it quickly transitions to the page after. These values need to be checked such that they are within integer ranges.

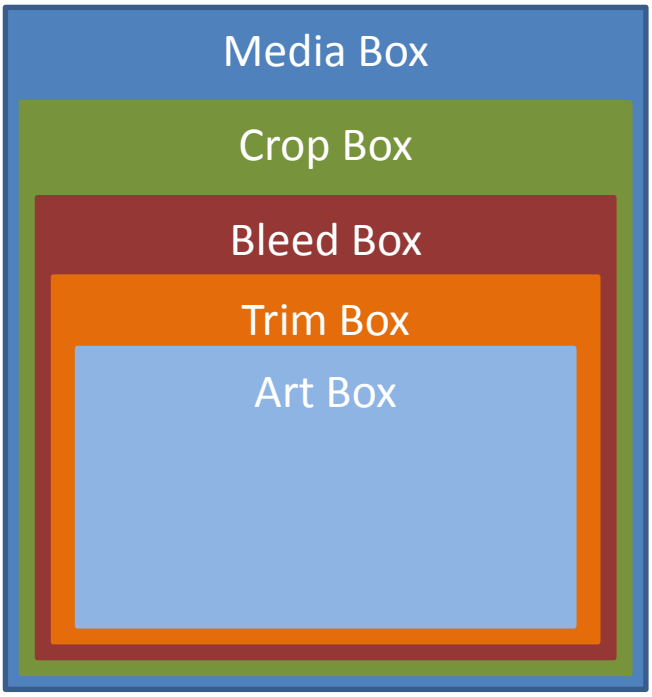

Figure 4-5. Page Boundaries [1]

CONCERNS:

Page Tree Nodes and Page Objects define very important characteristics for the PDF document. Each object referenced from the Page Tree can include a separate risk which is covered in other sections in this report. One of the most important attributes of the Page Tree is that it defines the boxes (or coordinates) for page information, which can ultimately determine if other content is improperly sized or located off the page, a hidden data risk. The Page Object also allows for Metadata, which is considered a hidden data risk and data disclosure risk and is discussed in PDF 5.1:.

There is also another potential hidden data risk if the /Count field and the actual size of the Page Tree are the same (all of the Page Objects in the document). Many readers may examine one or the other to determine the actual page count, which means that pages near the end of the document may not be rendered by the reader if the values are different.

Page Objects can introduce an Additional Action (Section 5.7.3.1) which triggers an action to occur when the page is opened or closed. This is covered in later sections as a data attack risk.

PRODUCT: PDF-1.0

LOCATIONS:

The Page Tree is defined by the Document Catalog (PDF 4.7:). The object referenced is the root of the Pages Tree and can define references to other Page Tree Nodes or Page Objects. Page Tree Nodes have a required set of parameters that make them different from Page Objects. The standard includes Table 29 to describe the required entries in a page tree node, and also Table 30 to list all of the entries in a page object.

A Page Tree object implements a Resources dictionary. Resources are seen in other objects as well, such as XObjects (PDF 4.12:). The Resource dictionary lists references to other objects that will be implemented on that page, such as XObjects, Fonts, Graphics State Parameters, and other objects shown in Table 33 of the ISO standard. Inheritance is possible in the Pages Tree and individual Page Objects may inherit Resources from parent nodes. It is important to ensure that all Resource objects are properly implemented in the page. Otherwise, this leads to an unused object, despite it being referenced by the Resource dictionary.

EXAMPLE:

```
1 0 obj
<<
    /Type /Catalog
    /Pages 2 0 R
>>
endobj

2 0 obj
<<
    /Type /Pages
    %Different Crop and MediaBox implies cropping for each page object
    /CropBox[0.0 72.0 284.1 164.68] %Inheritable feature
    /MediaBox[0.0 0.0 284.1 209.68] %Inheritable feature
    /Count 3
    /Kids [3 0 R 4 0 R 5 0 R]
>>
endobj

3 0 obj
<<
```

```
        /Type /Page  %Type Page = Page Object, leaf node
        /Parent 2 0 R
        /Resources
        <<
            /Font<</F1 6 0 R>>
        >>
        /Contents [7 0 R 8 0 R]
        %Action executed when page opens
        /AA << /O 9 0 R >>
        %Reference to page metadata
        /Metadata 10 0 R

        %This is required if PieceInfo is present, Page Piece dictionary
        /LastModified (date string)
        /PieceInfo 12 0 R

        %Slide show type effect, duration and transition
        %These may only be present in full-screen mode of a reader
        /Dur 2
        /Trans 11 0 R

        %These overwrite what is inherited for this page only.
        /CropBox[0.0 72.0 284.1 160.68] %Inheritable feature
        /MediaBox[0.0 0.0 284.1 190.68] %Inheritable feature

    >>
    endobj

    %Objects 4 and 5 are other Pages Objects
```

RECOMMENDATIONS:

1. **Validate:** Check that the `/Type`, `/Parent` (if not the root node), `/Kids`, and `/Count` fields are all present and implemented correctly.

2. **Validate:** Check that the `/Count` and size of the Page Tree/Objects is accurate.

3. **Validate:** Check that for each page in the document, that the page boundaries are defined, and that all content within that page fits within the viewable area.

4. **Validate:** Check that each Page Object has each required field implemented and if not, check that it is covered under inheritance.

5. **Validate:** Check that all objects in the Resource dictionary are implemented on the page.

6. **Remove:** Remove page objects additional actions from the dictionary. This may orphan action objects, which may need to be removed as well.

7. **Remove:** Remove the `/Dur` and `/Trans` entries and the transition dictionary in the document. This may orphan a transition dictionary which may need to be removed as well. Transition actions can also reference transition dictionaries.

8. **Replace:** N/A

9. **External Filter:** N/A

10. Review: N/A

PDF 4.8: END

PDF 4.9: NAME DICTIONARY AND NAME TREES

DESCRIPTION:

A PDF document implements a document name dictionary which allows references to objects by a "name" string. The name dictionary is comprised of several possible name tree objects. For example, the name dictionary implements a JavaScript name tree, which maps a "name" string to JavaScript actions that apply at the document level. Table 31 of the PDF ISO standard lists all of the possible entries in the name dictionary.

A name tree provides the hierarchy that defines and maps string values to indirect objects. For example, object 10 0 can be defined within the body and can have the associated name (Object10). Each name tree implements a single Root Node of the tree, which references, through indirect references, the children of that node. There can be Intermediate Nodes which define a mapping of names to indirect objects, as well as children of that node. Leaf Nodes, the children of intermediate nodes, implement a similar mapping of names to indirect objects, but define no children.

CONCERNS:

Naming hierarchies in the PDF document can make several layers of references making it possible to hide data the more complex the organization of information. Information embedded within the string name could be susceptible for hidden data risks.

Name trees introduce strings names which could be subject to data disclosure risks since they are free text fields that could provide some amount of description.

One particular concern is the organization of JavaScript in a name tree for the entire document. Selections of JavaScript code or functions can be built into the name tree, which can allow referencing JavaScript code by a name, or the name of the function it provides. When the document opens, it executes all the JavaScript code referenced in the tree. Other name trees provide similar group of information that may be a data attack risk if the document came from an untrusted source. Although, each object a name tree references will be inspected, it's important to realize that its existence in the name tree.

PRODUCT: PDF-1.2

LOCATIONS:

Name dictionaries are defined by the `/Names` entry in the Document Catalog, which identifies the root of the name tree. The document name dictionary entries define Named Destinations, Embedded Files, URLs, JavaScript Actions, Named Pages, and other objects that exist throughout the document. Each of these entries points to the root of a Name Tree for that hierarchy of objects that can be referenced by a name.

Name trees present an ordered list of named objects and their object numbers and generation. A name tree dictionary defines three possible fields: `/Kids`, `/Names`, and `/Limits`. The field `/Kids` define an array of indirect references to the children of this node, which allows the hierarchy to continue. The `/Name` field defines an array of key and value pairs indicating the object and key. A required field `/Limits` defines the least and greatest keys that exist in the `/Names` field, the boundary or limits.

EXAMPLE:

```
%Document Catalog, points to Name Dictionary in object 16
17 0 obj
<<
    /Names 16 0 R
    /Type/Catalog
    /Pages 4 0 R
>>
endobj

4 0 obj
<<
    /Type/Pages
    /Count 3
    /Kids[5 0 R 7 0 R 8 0 R]
>>
endobj
%Name Dictionary
16 0 obj
<<
    /Dests 14 0 R
>>
endobj

%Name tree, simple one node, no kids or limits defined, just an array
%of names and objects
14 0 obj
<<
    /Names[(Page1) 11 0 R(Page2) 12 0 R(Page3) 13 0 R]
>>
endobj

11 0 obj
  [5 0 R/XYZ 100 800 0]
endobj
```

RECOMMENDATIONS:

1 **Validate**: Check for consistency that the hierarchy is properly structured. Check that the level of the tree does not exceed a certain threshold.

2 **Remove**: Remove the dictionary and trees associated with that object. Removal of the entire tree will require fixing references where objects are referenced via name. This may impact objects within normal PDF syntax, content streams, or JavaScript (function calls) that may exist elsewhere in the document. This should be carefully done as it may break certain functions in the PDF.

3 **Remove**: Remove a specific Names Tree from the Names Dictionary. This will not delete the entire tree, just an entry (for example, JavaScript tree can be removed, but Dests may remain). For each Name Tree, references will have to be fixed where an object is referenced

by a name (which may be in an encoded stream). Table 31 of the ISO standard describes each name tree and what object it references.

4 **Replace**: Replace string names with generic information and replace where used as a reference. This may require decoding information in streams or strings where the name may be used. For example, JavaScript code can call a function, which can be the name of the JavaScript object in the tree.

5 **External Filtering Required**: Pass string names defined in the name tree to an external filter.

6 **Review**: N/A

PDF 4.9: END

PDF 4.10: STREAMS

DESCRIPTION:

Streams are used widely in PDF files. This section presents a high level overview of streams, other specific objects such as content streams, and embedded file streams are presented in the next section. The simplest way to define a stream is that it is basically a sequence of bytes. A dictionary exists as part of the indirect object and stream data is appended to the object definition's dictionary. The dictionary implements objects that define the size of the stream and how to properly decode it.

Streams are mentioned throughout this report as they are an important PDF object type. They are mentioned in greater detail in Chapter 5 in the follow sections:

Table 4-2. Stream Types

Stream Type	Section	Description
Content Stream	PDF 5.3:	Content streams comprise of PDF defined operators and operands (commands) that perform drawing or placing text onto the page. These commands are placed in order to represent the content of the page. Examples: Page Content Stream, Form XObject, Annotation Appearance Stream, Patterns, and Type 3 Fonts.
Embedded File Stream	PDF 5.4:	A stream object that contains a file embedded directly in the PDF document. Example: Embedded SWF File.
Embedded Data Stream	PDF 5.6:	A stream object that simply contains data. One example can be a text stream that contains a string encoded inside a stream.

		Examples: Embedded Fonts, ICC Profiles, Text Stream (JavaScript Stream).

CONCERNS:

Streams are a major cause for concern in PDF files simply because their contents can be encoded preventing applications and humans from easily examining data. Data can be hidden, disclosed, and can be used to store malicious shellcode or binaries that would otherwise not be part of the visible PDF file. To be completely aware of the risks with stream data, all stream objects and their data should be extracted and analyzed very carefully. It should also be understood that specially crafted stream data could be built to target the decoding algorithm, such that applications that decode data could be the target of an attack.

Streams also must be terminated and formatted correctly. This means that the /Length field must be correct, the number of filters used must be bounded, and the stream and endstream operators align properly with the /Length field. In addition, the object must be terminated following the `endstream` operator with `endobj`. Reader software may parse this object differently which may lead to a hidden data risk or possibly data attack risk if it is not properly implemented.

In reader software, if the value of `/Length` is negative or larger than what is allowed by an integer; the reader may not correctly parse the stream object and may not render the information onto the page. This could be both a hidden data risk and data attack risk.

If the `endstream` and `endobj` operators are not correctly used, readers may have different behavior determine how to properly terminate the stream object. Extra data could be inserted, although technically outside of the object space, it should be a concern when sanitizing PDF files to ensure these objects are properly implemented [12].

PRODUCT: PDF-1.0

LOCATIONS:

A stream object is defined in the PDF body of the file. Following the dictionary implementing the stream object, but before the `endobj` keyword, there is a keyword `stream` that indicates the stream contents immediately follow. The `endstream` keyword terminates the stream contents. The dictionary contains a length field which indicates the length of data between stream and endstream. It also includes an array for stream filters which is covered in more detail in the next section. There is an additional optional field for the decoded length (`/DL`), which serves as a hint to the approximate size of the decoded stream data. It should not be relied upon since it provides an approximation and is not often used.

EXAMPLES:

```
1 0 obj
<<
     %% A stream object of 534 bytes in length, an embedded stream, no filter
     /Length 534
>>
stream
% …534 Bytes of data…
endstream
endobj

9 0 obj
534           %Integer object
```

```
endobj

%Indirect reference used for Length
8 0 obj
<<
    /Length 9 0 R
>>
stream
%...Bytes of data specified in Object 9 (534)
endstream
endobj

10 0 obj
<<
    /Length -34
>>
stream
%Information is here even though Length is wrong, stream may be ignored.
endstream
endobj

11 0 obj
<<
    /Length 10000000000000000000000000000000000000000000000000000000000
>>
stream
%Information is here even though Length is wrong, stream may be ignored.
endstream
endobj
```

RECOMMENDATIONS:

1. **Validate**: Check that each required field is implemented within the stream dictionary.

2. **Validate**: Check that the length and decoded length field, when present, reflects the length of the actual stream data and approximate decoded stream data and that the value does not exceed its integer bounds. The decoded length field is often not used, but if present, ensure that is does not exceed its maximum integer value that could possibly create an overflow.

3. **Remove**: Remove the object dictionary and stream contents as well as any reference to this indirect object throughout the file. References to stream objects may reside in numerous locations through the PDF file including Content Arrays, Embedded Files, Font Files, XObjects names, 3D, Embedded Flash®[5], and many others. Removing stream objects can have an adverse effect on the file.

4. **Replace**: Replace stream dictionary values with correct parameters if they can be determined (perhaps length or decoded length if it can be determined accurately).

5. **External Filtering Required**: If the usage of the stream can be identified and there is a different file embedded within the stream, use an external filter for the appropriate content within the stream.

[5] Flash is a registered trademark of Adobe Systems, Inc.

6. **External Filtering Required**: Pass all text string information to an external filter. Decode parameters may include string information in the stream object.

7. **Review**: N/A

PDF 4.10: END

PDF 4.11: STREAM FILTERS

OVERVIEW:

Stream objects within the PDF syntax are designed such that they are compressed and encoded within the document, rendering them unreadable by humans, which can enable the placement of hidden and malicious data. Several filters can be applied to data streams, forcing a viewer application to decode, decompress, or decrypt in the opposite order to view the contents properly. PDF supports a wide number of filters than can be applied to any data stream.

- ASCII filters that encode data into text (ASCIIHexDecode, ASCII85Decode)
- Generic binary compression filters (LZWDecode, FlateDecode)
- Image compression filters (RunLengthDecode, CCITTFaxDecode, JBIG2Decode, DCTDecode, and JPXDecode).
- Encrypt Filter (Crypt, covered later in detail in PDF 4.18:).

Filters can be referenced in the dictionary preceding the `stream` and `endstream` contents. This is an array of filter types that the data was encoded in originally and in that order. PDF viewer applications will be required to follow the reverse to order to reproduce the data in a readable format.

The Filters can be implemented as an array, which means that the decoding process needs to be done in that order (left to right) as defined in the array. The output of the decoding process is the input to the next stage of the decoding process. Thus, data can be encoded with several algorithms.

Filter names in stream objects must be completely spelled out, unlike the abbreviations of the filter name, which is allowed in Inline Images (PDF 5.20:). Although not allowed in the standard, some reader implementations allow for abbreviated filter names in stream objects. Table 94 of the ISO standard defines these abbreviations.

CONCERNS:

Content within a PDF file that is not human readable causes concern because data can be easily hidden that is sensitive or malicious. The lack of restriction on the type of data could also lead to hiding arbitrary data. While it seems that decoding, decryption, or decompressing the data seems the safest method to examine the contents of the stream for hidden or malicious data. Parameters associated with decoding information must be properly sanitized, or they may be introduced as a data attack vector on the algorithms used to decode stream information.

Parameters may also introduce string or free text information which can lead to a hidden data risk or a data disclosure risk.

The ISO standard defines a set of supported filters for use and does not mention custom filters. For compatibility reasons, if a reader does not know of a filter, the ISO indicates there is little choice but to give an error because it would not be able to read that information [1]. If a PDF attempts to utilize a non-standard filter, many reader applications will not know how to decode the stream and it will not render the decoded output of the stream, but the rest of the document can be viewed provided other stream objects were encoded with a standard filter. If a specially designed reader was implemented to support additional non-standard filters, and PDF files were specially crafted to utilize those filters and encoded the data appropriately, it could possibly become a hidden data risk.

PRODUCT: PDF-1.0

LOCATION:
Streams in a PDF file contain a dictionary that can identify the `/Filter` type of the object in the data stream portion. These objects can be referenced by other objects throughout the document.

EXAMPLE:

```
1 0 obj
<<
     %% A data object of 534 bytes in length, an embedded stream, required
     /Length 534
     %% Filter shows the series of filters used, required
     /Filter [/ASCII85Decode /LZWDecode]
     %% Parameters for filters, optional
     /DecodeParms [ <<%Dictionary for ASCII85DecodeParms
                    >>
                    <<%Dictionary for LZWDecodeParms
                    >>]
>>
stream
% …534 Bytes of data that is not human readable…
endstream
endobj
```

RECOMMENDATIONS:

1 **Validate**: Check for consistency and that a standard (listed above) decoding algorithm(s) is used.

2 **Validate**: Check that the filter type is consistent with its appropriate usage, depending on the type of stream (content, image, embedded file). Some decoding algorithms are more appropriate with certain data types.

3 **Validate**: Check that the decoding array is terminated and has an acceptable number of filters used for the decoding process.

4 **Validate:** Check for consistency by examining the content of the stream for header or checksum information and check against the stream filter selected. Certain encoding algorithms may have a fixed header (for example, flate compression is observed to have a 2-byte header), which could be inspected to determine if the stream is valid according to the selected stream filter.

5 **Remove**: Remove stream filter, and if possible, insert decoded data. This will increase the size of the PDF file.

6 **Remove:** Remove the entire stream object, which includes the filter information. See removal information in the previous section (PDF 4.10:)

7 **Replace**: Replace abbreviated filter names with the full version as required in the standard for stream objects. They may be permitted in Inline images.

8 **External Filtering Required**: Pass all string information to an external filter. String information may reside in stream parameters included in the dictionary.

9 **Review**: N/A

PDF 4.11: END

PDF 4.12: XOBJECTS

OVERVIEW:

External objects, or XObjects, are graphical objects that have its contents in a stream, an embedded data stream. There are three different types of XObjects: image XObject, PostScript XObjects, and Form XObjects, which are covered in more detail in PDF 5.15:, PDF 5.16:, and PDF 5.17:. There are other defined types of XObjects called Group XObjects and Reference XObjects, which are defined in PDF 5.18: and PDF 5.19:. XObjects are referenced through other existing Content Streams, and although a XObject is technically a stream object, it is slightly different because a XObject defines the image but is referenced in another stream to draw the image (thus, an External Object).

XObjects are defined by name such that they can be used in content streams. There is a similar method used for font names. One method is to include a Resource dictionary, defined in the Page Object (PDF 4.8:) or in the stream dictionary itself. Inheritance is used to determine how to use that XObject. For example, a XObject defined in a Page Object can be utilized in every content stream referenced by the `/Contents` array of that Page Object.

CONCERNS:

XObjects are susceptible for hidden data since they can be manipulated in many ways to obscure content. This is possible with any stream data or any image data.

XObjects may also introduce a data attack risk. Vulnerabilities in the past have used malformed XObjects to attack the algorithm of the filter responsible for decoding the stream data [14].

PRODUCT: PDF-1.4

LOCATION:

XObjects implement their own dictionary in the indirect object defining this data type, which is indicated by `/Type /XObject`. They are referenced in other existing content streams through the `Do` operator to paint the named XObject (with external data from the stream) onto the document

at that point. XObjects must be referenced in a Resource Dictionary for a page through the keyword `/XObject`. This contains a reference to the actual XObject image defined elsewhere in the PDF body.

EXAMPLE:

```
1 0 obj
<<
    /Type /Catalog
    /Pages 3 0 R
>>
endobj

3 0 obj
<<
   /Type /Pages
   /Kids [4 0 R]
   /Count 1
   /Rotate 0
>>
endobj

4 0 obj
<<
    /Type/Page
    /MediaBox [0 0 612 792]
    /Rotate 0
    /Parent 3 0 R
    /Resources<</ProcSet[/PDF /ImageC /Text]
                /ExtGState 11 0 R
                /XObject 12 0 R     %Part of the resources for this Page
                /Font 13 0 R
            >>
    /Contents 5 0 R
>>
endobj

%This is the name of the XObject, just points to object 10 0
12 0 obj
<<
    /R10 10 0 R %Name is R10, XObject is defined there.
>>
endobj

%Actual XObject
10 0 obj
<<
    /Subtype/Image
    /ColorSpace/DeviceRGB
    /Width 360
    /Height 360
    /BitsPerComponent 8
    /Filter /DCTDecode
    /Length 14056
>>
stream
…stream data…
endstream
endobj
>>
%Content stream defined in 5 0 obj, implements the XObject through the Do
%operator.
```

RECOMMENDATIONS:

1 **Validate**: Check for consistency and validate the dictionary of the XObject by ensuring that the parameters are correct (e.g. Height and Width align with the size of the stream data).

2 **Validate**: Check for consistency and perform the same validation as a normal stream object. Also ensure that standard filters are used based on the file type of the XObject, if possible to determine.

3 **Remove**: Remove the XObject dictionary and stream and anything that references the XObject in the PDF file. Remove references to the XObject from the Resource Dictionary, the name definition and reference to the XObject, as well as any references through the `Do` operator in content streams (that will require decoding) elsewhere throughout the document. Removing this will have an impact on the document and other objects that utilize it may need to be removed as well. Cross-reference information will need to be reconstructed.

4 **Replace**: Replace the dictionary and stream contents with approved content. Cross-reference information will need to be reconstructed.

5 **External Filtering Required**: Determine the file type embedded within the XObject. This may be determined by the decoding type or peeking into the decoded data stream. Once determined, decode and extract the stream contents from the file for external review for that specific file type.

6 **External Filtering Required**: Pass all string data associated with a XObject to an external filter (e.g., Image name).

7 **Review**: Decode and extract the contents of the XObject stream and reconstruct an external file. If it is an image or content that can be viewed, have the file examined by a human reviewer for acceptance.

PDF 4.12: END

4.3 Advanced PDF File Structure

PDF 4.13: OBJECT STREAMS

OVERVIEW:

A stream was discussed in earlier examples in this chapter is a means to embed binary data and compressed data into PDF files. This allows for greater portability as data is condensed and efficiently stored within the document.

Object streams are a different type of stream than the previously mentioned embedded streams. They do not contain data for embedded files or multimedia file types such as images, audio, or flash. Object streams are different because they contain other PDF indirect objects which can still be referenced elsewhere in the PDF file structure through the same indirect references.

The PDF standard indicates that there are four types of indirect objects that cannot be embedded within an object stream. They are:

- Stream Objects
- Objects with a generation number other than zero.
- A document's encryption dictionary.
- An object that represents the value of the Length entry in an object stream dictionary. (This could introduce a bad circular reference, for example the length of the object stream cannot be contained within the encoded object stream itself, and thus the length could never be accurately determined).

Therefore, only basic PDF indirect objects can be included within an object stream. However, when this is used it can reduce the size of the file by compressing a number of PDF indirect objects. Object streams have reduced the file size of PDF documents since there is a restriction on the byte offset length of the traditional cross reference table. Modifying a PDF file or replacing Object streams with their actual content may not be possible in very large files. Also, it is very important to understand that changes to Object streams will result in changes to Cross-Reference stream objects (PDF 4.14:).

CONCERNS:

Object streams face similar risks as normal embedded streams; they are just used for a different purpose. Their contents are unknown until they are fully decoded and decompressed. Since their contents contain actual PDF objects themselves, this adds an extra layer of complexity for inspecting the contents of the PDF file. Before with stream filters, data was compressed or encoded, and now with object streams, actual PDF indirect objects and dictionaries are no longer human readable.

For instance, JavaScript actions (and code) or Launch actions may reside in an Object Stream, hiding their contents from the outermost level of the file.

PRODUCT: PDF-1.5

LOCATION:

Object streams are located as a specific object in the body of the PDF file. They are identified by the `/Type` field with the value of `/ObjStm`. They have additional entries for length and filters that are used in the stream, just as any other stream object. They contain the important field `/N` indicating the number of objects that are encapsulated within the object stream and the field `/First` indicating the byte offset into the stream of the first embedded object.

The stream contents of the object, once decompressed, must be in a standard format. It is a listing of *N* pairs of integers that are separated by whitespaces. The first integer represents the object number (all objects have their generation number set to 0), the second integer represents the byte offset into the decoded object stream. The objects can be in any order and since bytes offsets are used the `obj` and `endobj` keywords are not used to separate them. Following the N pairs of integers, which defines the header for the rest of the section, each object content starting with the '<<' and ending with the '>>' delimiter are included in the stream.

EXAMPLE:

```
%PDF-1.7
%âãÏÓ
4 0 obj
<<
```

```
        /Length 64
        /Filter/FlateDecode
    >>
    stream
    …binary data…
    endstream
    endobj

    7 0 obj
    <<
        /Length 65
        /Filter
        /FlateDecode
    >>
    stream
    …binary data…
    endstream
    endobj

    10 0 obj
    <<
        /Length 65
        /Filter/FlateDecode
    >>
    stream
    …binary data…
    endstream
    endobj

    14 0 obj
    <<
        /Names 13 0 R
        /Type/Catalog
        /Pages 5 0 R
    >>
    endobj

    15 0 obj
    <<
        /Producer(producer)
        /ModDate(date)
        /CreationDate(date)
    >>
    endobj

    %Object stream containing nine objects, launch, JavaScript action embedded
    %Encoded and Decoded placed side-by-side for reference
    %Length fields inaccurate, due to formatting.
```

`%Actual file content`
`3 0 obj`
`<<`
`  /Type/ObjStm`
`  /N 9`
`  /Length 405`
`  /First 55`
`  /Filter/FlateDecode`
`%Encoded, not readable`
`>>`
`stream`
`…binary data…`
`endstream`
	`%Decoded stream`
	`3 0 obj`
	`<<`
	`  /Type/ObjStm`
	`  /N 9`
	`  /Length 405`
	`  /First 55`
	`  /Filter/FlateDecode`
	`>>`
	`stream`
	`%Objects 2, 6, 8, 9, 11, 1, 5, 12, 13 are all`
	`%located in this object stream`
	`%Decoded here for reference`

endobj	%Comments not originally in stream, just for %reference in this example. 2 0 6 126 8 282 9 424 11 474 1 617 5 690 12 751 13 788 %Annotation with a Launch command << /C[0 0 1]/Border[0 0 0] /A<</Win<</D()/F("C:\\text.txt")/O()/P()>>/S/Launch >> /Subtype/Link/Rect[35 785 78 798]>> %Page 1 <</Parent 5 0 R/Contents 4 0 R/Type/Page/Resources<</ProcSet [/PDF /Text /ImageB /ImageC /ImageI]/Font<</F1 1 0 R>>>>/MediaBox[0 0 595 842]/Annots[2 0 R]>> %Page 2 <</Parent 5 0 R/Contents 7 0 R/Type/Page/Resources<</ProcSet [/PDF /Text /ImageB /ImageC /ImageI]/Font<</F1 1 0 R>>>>/MediaBox[0 0 595 842]>> %JavaScript action and code <</S/JavaScript/JS(app.alert\({cMsg:'JavaScript'}\) ;)>> %Page 3 <</Parent 5 0 R/Contents 10 0 R/Type/Page/Resources<</ProcSet [/PDF /Text /ImageB /ImageC /ImageI]/Font<</F1 1 0 R>>>>/MediaBox[0 0 595 842]>> %Font used <</BaseFont/Helvetica/Type/Font/Encoding/WinAnsiEnc oding/Subtype/Type1>> %Pages <</Type/Pages/Count 3/Kids[6 0 R 8 0 R 11 0 R]>> %Name Tree, pointing to JavaScript <</Names[(0000000000000000) 9 0 R]>> %JavaScript in Name Tree <</JavaScript 12 0 R>> endstream endobj

```
16 0 obj
<<
    /Type/XRef
    /W[1 2 2]
    /Root 14 0 R
    /Index[0 17]
    /ID [<##>
         <##>]
    /Length 62
    /Info 15 0 R
    /Size 17
    /Filter/FlateDecode
>>
stream
…data stream…
endstream
endobj

startxref
1095
```

```
%%EOF
```

RECOMMENDATIONS:

1 **Validate**: Check that all object streams are encoded and compressed with approved and terminated list of filters.

2 **Validate**: Check the length of each object stream aligns with the cross-reference table information and consistent with the file size and actual stream contents.

3 **Validate**: Decode the object streams contents and perform the same validation method on each of the embedded objects separately (as if they were separate indirect objects outside the object stream).

4 **Remove**: Remove invalid objects from the object stream, fix the header of the object stream, re-encode the stream, fix the applicable cross-reference stream information and object stream dictionary contents to reflect the new changes. Indirect references to the removed object shall also be removed.

5 **Remove**: Remove the object stream dictionary and stream contents, and remove corresponding data from cross-reference streams and updating main cross-reference table shall be performed. Removal of object streams will require synchronization between the cross-reference stream object.

6 **Replace**: Decode the object stream and replace the object stream with the actual indirect objects at the outermost level of the PDF file. Update the cross-reference table and cross-reference stream information. This will force the file size to grow much larger. Due to fixed size of the byte offset in a traditional cross-reference table, this may not be possible.

7 **External Filtering Required**: N/A

8 **Review**: N/A

PDF 4.13: END

PDF 4.14: CROSS-REFERENCE STREAMS

OVERVIEW:

Cross-reference streams provide the ability to efficiently store cross-reference information and access compressed objects in object streams, as presented in PDF 4.13:. They are stream objects just like any other stream object previously defined in this chapter, but they contain information that is equivalent to the actual cross reference table at the end of the document, and can be used to replace the cross-reference table and trailer. It is possible to create PDF files that contain only a sequence of PDF objects if the cross-reference table information is contained within a stream. The keywords xref and trailer do not need to be used, and the only non-object-based data in the PDF file is the line "*startxref address %%EOF*", which points the cross-reference stream object (address) and the EOF comment simply terminates the file. Object streams and cross-reference

streams also allow for larger PDF files, since the traditional cross-reference table had a fixed byte offset length. It should be noted that changes in either object streams or cross-reference streams will have an impact on both types of objects.

CONCERNS:

Cross-reference streams present the same risks as embedded streams in PDF file; their contents are unknown until they are decoded, and this stream contains important information about how the file is organized.

PRODUCT: PDF-1.5

LOCATION:

The Cross reference stream object is located in the body of a PDF file and can be used to replace the traditional cross reference table. The Cross-Reference Stream object contains a dictionary that identifies its Type as `XRef`, and the size which is an integer indicating the number of objects in the cross-reference stream content.

The cross-reference stream dictionary provides an optional field called `/Index` which is an array of a pair of integers that define the first object number in the subsection (first integer in array) and the number of entries in the subsection (second integer in array).

The dictionary provides a required field called `/W` for the representation of the cross-reference information in the stream. The value is an array of 3 integers; each value in the array represents the byte length of each field in the cross-reference information embedded within the stream. For example [1 2 1] means that the first value is 1 byte long, the second 2 bytes long, and the third is one byte long.

The entries in a cross-reference stream data contain 3 different permutations of data representation, each defined as an entry with 3 integers. Therefore, the `/W` field in the previous paragraph can be used for each type. The first integer defines the type: 0, 1, and 2.

- Type 0 defines the free objects. The second integer in the array represents the object number of the next free object and the third integer represents the generation number (similar to the traditional cross-reference table contents with a ‘f’ value as the in use parameter).

- Type 1 defines objects that are in use in the document but are not compressed into an object stream. The second integer represents the byte offset into the file, and the third number is the generation number (similar to the traditional cross-reference table contents with an ‘n’ value as the in use parameter).

- Type 2 defines objects that are compressed into object streams. The second integer represents the object number of the object stream, and the third integer represents the index of this particular object into the object stream.

The combination of each type allows for objects to be defined through the traditional cross-reference table with byte offsets, and now with Type 2 a way to index into each object stream data to find definitions of PDF objects.

EXAMPLE:

```
1 0 obj  %At byte offset 0x56
<<
```

```
        %object 1 0 contents
    >>
    endobj
    … %objects 2 and 3

    %Object stream
    11 0 obj
    <<
        /Type /ObjStm
        /Length 1856
        %Number of objects in stream
        /N 3
        /First 0
        %Filters used
        /Filter [/FlateDecode /ASCII85Decode]
    >>
    stream
        %This is encoded according to above, and not normally visible.
        %Embedded objects content
        %Contains object numbers (1,2,3) and their byte offsets (0,547,665)
        1 0 2 547 3 665
        << 1 object contents >>
        << 2 object contents >>
        << 3 object contents >>
    endstream
    endobj

    %Object cross-reference stream
    10 0 obj  %Located at byte 642 in this example
    <<
        /Type /XRef
        /Size 4
        /W [1 2 1]
    >>
    %First object is at byte offset 0x56 in file
    %Second object is first object in object stream (0xB), 11 0 (above)
    %Third object is the second object in object stream 11 0 (above)
    %Fourth object is the third object in object stream 11 0 (above)
    stream
    01 0056 0
    02 000B 0
    02 000B 1
    02 000B 2
    endstream
    endobj
    …
    %No xref table or trailer, not needed

    startxref
    642      % pointing to object 10 0, rather than xref
    %%EOF
```

RECOMMENDATIONS:

1 **Validate**: Check for consistency the dictionary has implemented all of its required fields.

2 **Validate**: Check for consistency and that the encoding makes sense with the /W array and size of the table from the dictionary.

3 **Validate**: Check for consistency the characters in the stream are of proper type (a hexadecimal representation according to the /W array).

4 **Validate**: Check for consistency and referential integrity and that normal indirect objects have the correct type and byte offset into the file.

5 **Validate**: Check for consistency and referential integrity such that embedded objects in object streams have the correct type, referencing a valid object stream indirect object, and have a unique number or offset into that object stream.

6 **Remove**: Remove the cross-reference stream object after replacing the Object streams that are referenced in the cross-reference stream object (if that's possible to determine), by taking them out of each object stream and place them at the outermost level of the PDF file. The main cross reference table will need to be corrected. If the cross-reference stream was in place because of size limitations of the traditional cross-reference table, then this is possible. Otherwise, the cross-reference stream will be required.

7 **Replace**: N/A

8 **External Filtering Required**: N/A

9 **Review**: N/A

PDF 4.14: END

PDF 4.15: OBJECT REVISIONS

OVERVIEW:

Objects in PDF files are referenced in a hierarchy of objects comprising the content of the file; they are referenced through indirect references as mentioned in PDF 4.6:. In many examples that are presented in this document, the generation number has remained zero, meaning the initial creation of the object. Objects can be modified with a more recent generation, or higher generation number. This section focuses on having multiple objects with the same number, but different generation numbers. This is not supported by the standard, but it could exist in the body of the PDF file. This is equivalent to an orphaned object, because it's effectively unused, but is different because it shares a valid object number.

CONCERNS:

The main risk in this section is a hidden data risk where older content can exist in the document and may be left out of the hierarchy of objects in the PDF file, similar to an orphaned object. If the most recent object revision is referenced in the cross-reference table information and by other indirect references, an older object with a lesser generation number may essentially become orphaned. This means that the reader will only display the most recent object.

This may also provide a data disclosure risk if the file was updated, older data somehow remained in the PDF file, but not rendered. It may be possible for users to examine the file to extract the data that was not rendered.

PRODUCT: PDF-1.0

LOCATION:

PDF indirect objects are designated by the first line in their description. For example, "`3 0 obj`", means that object 3, generation 0 of the object will be defined until the keyword `endobj` terminates the content. Another object defined as "`3 1 obj`" uses the same object number but a more recent revision. As long as all references in the document point to "`3 1 R`", and as well as the cross-reference table pointing to the latest object, the object "3 0 "can remain in the document but its content has been replaced by the more recent version of object 3.

EXAMPLE:

```
3 0 obj
<<
    %Hidden data
>>
endobj

3 1 obj
<<
    %Visible data since it replaces object 3 0, Cross-reference should point
    %here, along with any other indirect references.
>>
endobj

%Object 3 below indicates generation 1; this value needs to be correct.
xref
0 10
0000065535 00000 n
0000000009 00000 n
0000000017 00000 n
0000000023 00001 n
… rest of xref table
trailer
<<
…
>>
%%EOF
```

RECOMMENDATIONS:

1 **Validate**: Check for referential integrity and validate that each object is properly referenced, and that there are no orphaned objects even with same object number in the hierarchy of PDF objects.

2 **Validate**: Check for referential integrity and that each used object in the cross-reference table information is mapped to the matching object and generation in the body of the PDF file.

3 **Remove**: Remove objects that have been completely replaced with newer objects.

4 **Replace**: N/A

5 **External Filtering Required**: N/A

6 **Review**: N/A

PDF 4.15: END

PDF 4.16: INCREMENTAL UPDATES

OVERVIEW:

Incremental updates allow for large PDF files to be updated quickly without rewriting the entire contents of the file. To do this, object changes to the file are appended to the end of the document, leaving the original contents unchanged. The delta changes to the cross-reference table, a new trailer that indicates where to find the previous trailer are also added to the existing PDF document.

CONCERNS:

Incremental updates append changes to objects at the end of the document by adding the updated objects, which then render the old objects unused. This causes a hidden data or a data disclosure concern in the contents of the older objects that appear in the earlier revisions of the file. Numerous updates can also be applied to a file, which could lead to a data attack risk if there are a large number of updates in the file.

PRODUCT: PDF-1.0

LOCATION:

When a PDF file is updated, updated indirect objects of the PDF document are essentially appended to the existing file. An updated body section of the document containing the deltas between the different versions. A new cross-reference table and trailer are also appended, which contain information only regarding the delta that has been added. The figure below demonstrates how the file appears after *N* revisions. This process can occur several times and reader applications must account for multiple updates which may result in multiple objects that remain hidden. It is important not to allow too many updates, which can introduce a significant amount of hidden data and potential exploit to reader software if there is an overflow or error processing that information.

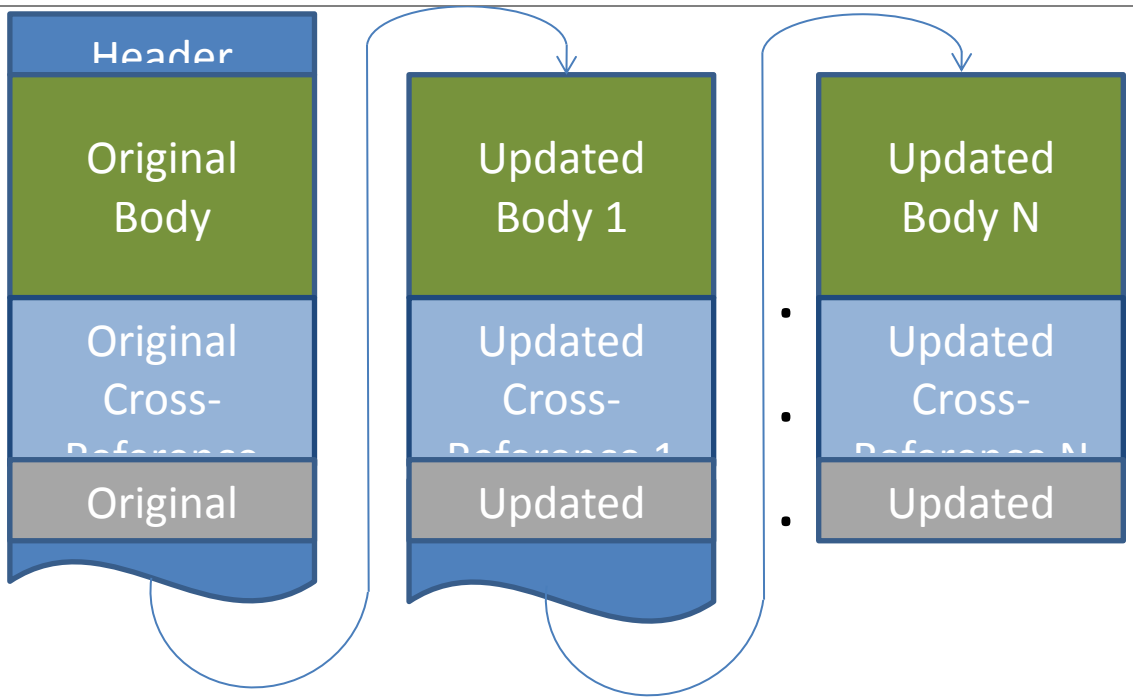

Figure 4-6. Incremental Updates

EXAMPLE:

```
%PDF-1.7
1 0 obj
<<
 /Type /Catalog
 /Pages 2 0 R
 /OpenAction 8 0 R
>>
endobj

2 0 obj
<<
 /Type /Pages
 /Kids [3 0 R]
 /Count 1
>>
endobj

3 0 obj
<<
 /Type /Page
 /Parent 2 0 R
 /MediaBox [0 0 612 792]
 /Contents 4 0 R
 /Resources
 << /ProcSet 5 0 R
    /Font << /F1 6 0 R >>
 >>
>>
endobj

4 0 obj
<< /Length 46 >>
```

```
stream
BT
/F1 24 Tf
100 700 Td
(Hello World)Tj
ET
endstream
endobj

5 0 obj
[/PDF /Text]
endobj

6 0 obj
<<
 /Type /Font
 /Subtype /Type1
 /Name /F1
 /BaseFont /Helvetica
 /Encoding /MacRomanEncoding
>>
endobj

7 0 obj %OLD OBJECT, no longer displayed by reader
<<
 /Type /Action
 /S /URI
 /URI (http://www.google.com)
>>
endobj

xref
0 8
0000000000 65535 f
0000000012 00000 n
0000000109 00000 n
0000000165 00000 n
0000000234 00000 n
0000000401 00000 n
0000000505 00000 n
0000000538 00000 n
trailer
<<
 /Size 8
 /Root 1 0 R
>>
startxref
748
%%EOF

%New URI goes to CNN, rather than Google.com
7 0 obj
<<
 /Type /Action
 /S /URI
 /URI (http://www.cnn.com)
>>
endobj

xref
7 1
0000001007 00000 n
```

```
trailer
<<
      /Size 1
      /Root 1 0 R
      /Prev 939
>>
%%EOF
```

RECOMMENDATIONS:

1 **Validate**: Check for consistency and validate that the PDF document should inspect for multiple copies of same object number. Only one object number should exist in the file and it shall be properly referenced in the remainder of the body of the PDF file.

2 **Remove**: If the incremental update introduces the replacement of objects, then remove old objects from the original body and possibly the older updated sections. Reconstruct the file such it is no longer an incremental updated file with multiple body, trailers, and cross-reference table segments, but that the latest objects are now in the original body where the older objects existed. The PDF file should be reconstructed with the most up to date indirect object information and reconstructed cross-reference table and trailer information. Reconstruction will break files that are digitally signed.

3 **Replace**: N/A

4 **External Filtering Required**: N/A

5 **Review**: N/A

PDF 4.16: END

PDF 4.17: LINEARIZED PDF FILES

DESCRIPTION:

Linearizing a PDF file allows for efficiently reading a large file over the Internet (http) in increments at at ime. The goal of this design to is to display the first page of the document as quickly as possible.

The specification outlines these four goals [1]:

- When a document is opened, display the first page as quickly as possible. The first page to be viewed may be an arbitrary page of the document, not necessarily page 0 (though opening at page 0 is most common).
- When the user requests another page of an open document (for example, by going to the next page or by following a link to an arbitrary page), display that page as quickly as possible.

- When data for a page is delivered over a slow channel, display the page incrementally as it arrives. To the extent possible, display the most useful data first.

- Permit user interaction, such as following a link, to be performed even before the entire page has been received and displayed.

To accommodate this feature, there are additional rules on how objects are to be ordered in the PDF file, and additional optional data structures, known as hint tables, which is encoded into a stream object. These allow improved methods to navigate the file. According to the ISO standard, Linearized PDF files are divided into specific sections shown in Figure 4-7. The Linearized PDF file utilizes the incremental updates feature in the PDF standard, but does so in a way that it does not introduce orphaned objects.

CONCERNS:

Linearized PDF files introduce a hidden data risk. This is a concern because the structure of the PDF file is slightly different from typical PDF files presented earlier in this report. For example, in hint stream objects, if they are incorrect, there is still enough information in the file to render it properly, so the hint data could be ignored by a reader. This presents a hidden data risk.

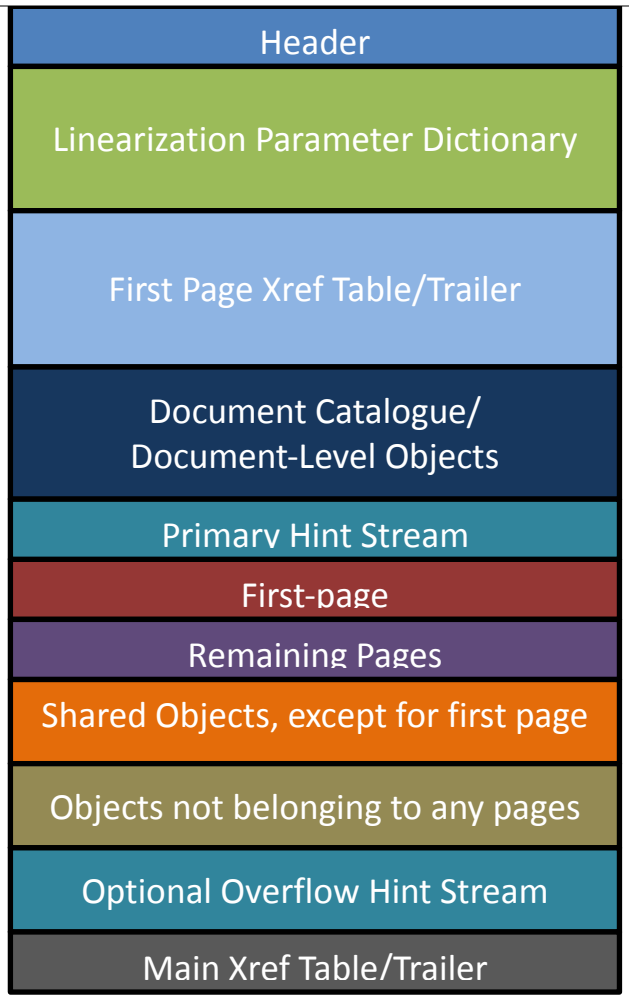

Figure 4-7. Linearized PDF

PRODUCT: PDF-1.2

LOCATIONS:

The location of linearized PDF file specifics are in Section 2 and Section 5 of the PDF file (refer to example below). Section 2 contains a Linearization Parameters dictionary. This defines the length of the document, the object number of the first page, the byte offset of the end of the first page, the total number of pages in the document, and the offset of the first entry in the main cross-reference table at the end of the document. Most importantly, the Linearization parameters dictionary contains information regarding the hint stream (byte offset and length), which is defined in Section 5 as an indirect object implementing a stream object.

Section 5 implements a stream object containing the hint stream for the Linearized PDF file. The stream object contains all the normal stream object parameters such as Length and Filter. The stream dictionary also implements other fields that are not normally present in standard stream objects. The only way to differentiate is the fact that this stream object was referenced through the linearization parameters dictionary by byte offset (not object number) from the beginning of the file. The stream dictionary implements the required `/S` parameter defined the byte offset into the stream of the shared object hint table. There are other numerous hint tables that are only

required if certain functionality is present within the PDF file. For example, hint tables can also exist in the stream if certain features such as thumbnails, outlines, article thread information, named destinations, and interactive forms are implemented within the current file.

The hint table stream information is usually encoded and not readable, just as any other stream. However, when decoded, the table is encoded in a format that would need to be translated for a user to read (encoded in bit information in values). Header information, bit sizes for each field and entries in the table are defined in Annex F.4 of the ISO standard.

EXAMPLE (3DFILE.PDF):

```
%PDF-1.3%âãÏÓ

%Section 1: Header above.

%Section 2: Linearization dictionary
651 0 obj
<<
    /Linearized 1 %Version of Linearization (1)
    /L 3635354    %File length of the document in bytes.
    /O 653        %Object number belonging to the first page in Section 6
below.
    /E 99633      %Offset of end of first page
    /N 13         %Number of pages in document.
    /T 3622286    %Offset of first entry in main xref table.
    /H [1752 652] %Primary hint stream byte offset and length.
>>endobj
%Section 3: First-page Xref table and trailer.
xref
651 71
0000000016 00000 n
…
trailer
<<
    /Size 722
    /Prev 3622274
    /XRefStm 2404
    /Root 652 0 R
    /Info 138 0 R
    /ID[<##><##>]
>>
startxref
0
%%EOF
…

%Section 4: Document Catalog, Document-level objects.
652 0 obj
<<
    /MarkInfo <</Marked true>>
    /ViewerPreferences <</Direction/L2R>>
    /Outlines 127 0 R
    /Metadata 137 0 R
    /Pages 134 0 R
    /StructTreeRoot 139 0 R
    /Type/Catalog
>>endobj

%Section 5: Primary Hint Stream, this is a new addition for Linearized files.
721 0 obj
<<
```

```
      /Length 558
      /C 709  %Logical structure hint table at Byte 709
      /Filter/FlateDecode
      /I 737  %Information Dictionary hint table at Byte 737
      /O 693 %Outline Hint Table at Byte 693.
      /S 504 %Only required field. Shared Object Hint table at byte 504
   >>
   stream
   %Encoded contents of hint stream. Information in hexadecimal, not printable.
   endstream
   endobj

   %Section 6: Object for Page 1.

   %Section 7. Remaining pages, Page 2.

   %Section 8: Shared objects for all pages except the first.
   %Section 9: Objects not associated with any pages.
   %Section 10: Overflow hint stream (optional).
   %Section 11. Main Xref table and trailer.
```

RECOMMENDATIONS:

1 **Validate**: Check for consistency and check that the file length, the array pointing the location (pointing to an indirect object with the hint stream) and size of the hint table stream are accurate.

2 **Validate**: Check for referential integrity and check that the first page object number can be validated if the number is a valid page object and is referenced in the `/Pages` object as the first in the `/Kids` array. The offset of the end of the first page can also be validated. The number of pages in the document can be validated to ensure that it equals the count in the `/Pages` object as well as matches the total number of `/Page` object in the document.

3 **Validate**: Check for consistency and validate the hint table streams. They need to be decoded to its original form and checked for correctness. Depending on the type of hint table, the format for the header, and each entry field should be validated according to the format defined in the standard.

4 **Remove**: Linearization information can be removed from the document by restructuring the document such that everything is included in the main cross-reference table (removing the first-page xref table/trailer). Linearization Parameter dictionary and hint table stream shall be removed. This will not maintain the user experience of the document, as the entire document will need to be downloaded before the first page can be displayed.

5 **Replace**: Replacement methods that may occur could be fixing byte offsets or object numbers if they can be determined properly.

6 **External Filtering Required**: N/A

7 **Review**: N/A

PDF 4.17: END

4.4 PDF Encryption

PDF 4.18: ENCRYPTION

OVERVIEW:

PDF encryption allows for the document's contents to be protected and only read by an authorized user. Encryption exists at multiple levels in a PDF document. Document-level encryption applies to all strings and streams in the PDF document, except for the ID entry in the trailer, strings that are defined in the Encryption dictionary, and strings that are inside content streams and object streams, which are encrypted anyways. Document-level encryption does not include other object types such as integers and Boolean values, which are used to convey information about the document's structure, as opposed to strings and streams which contain the content of the PDF file.

The Crypt Filter (PDF-1.5) presents a method for a PDF document to contain select stream objects to be encrypted, while the remainder of the PDF document can be unencrypted. Just as data can be encoded with a particular algorithm or stream filter, the Crypt filter is used to protect the contents of an individual stream.

Regardless of the type of encryption used, the first indication that the file contains encrypted content is in the file trailer, which provides an indirect reference to the Encryption dictionary, identifying the file's security handler. PDF supports a standard security handler (which is password-based), public-key security handlers, and also allows developers to define their own security handler. Currently, the ISO standard lists RC4, a symmetric-key stream cipher, and AES, a symmetric-key block cipher, as encryption algorithms.

Standard Security Handler:

The PDF standard identifies a Standard Security Handler, which is a password-based encryption of the document. There are two passwords: a user password and an owner password. Permissions can be set on the document to handle the capabilities of each user. The standard security handler uses the password to compute the encryption key (used for both encryption and decryption).

Public-Key Security Handler:

The PDF specification allows a Public-Key Security Handler to be defined. The public-key handler is more complicated as it requires defining a recipients list (in Public Key Cryptographic Standard Number 7 (PKCS#7) encoding syntax) that defines individual access permissions for the document. This means that only those recipients can open the document, as opposed to the Standard Security handler, where it was restricted to those with the correct password. This requires the public key of each recipient to be available in order for the document to be encrypted. Decryption of the document content uses the recipient's private key, which may require a password on the user's end to retrieve.

Crypt Filter:

Crypt filters, used in a similar manner as other stream filters, are used to selectively encrypt a particular stream in the PDF document. A stream object also contains a field for Decode

Parameters, which is used to identify the name of the crypt filter. A list of all crypt filters used in the document is defined in the Encryption dictionary. The Encryption dictionary implements Crypt Filter dictionaries which define the parameters for the security handler.

PDF supports an Identity crypt filter which performs no operation on the stream data. This allows information to be unencrypted in a document that is encrypted. For example, the standard uses a metadata stream as an example

CONCERNS:

Encryption of a PDF could present a data attack risk on the algorithms that are used in the process. The standard indicates the usage of RC4 or AES, along with the MD5 hash algorithm.

Data disclosure may be a risk as well when utilizing Crypt filters. By selectively encrypting streams and possibly using the Identity crypt filter (which performs no encryption), there may be a risk to disclose information.

PRODUCT:

Document encryption: PDF-1.1
Public-Key Cryptography : PDF-1.3
Crypt Filters: PDF-1.5
AES: PDF-1.6

LOCATION:

Common Document Encryption – Dictionary Information:

Document encryption is specified by an Encryption dictionary that is linked to by the `/Encrypt` entry in the document's File Trailer dictionary. Regardless of the type of Encryption there are common entries in the dictionary which are described in this section.

The Encryption dictionary provides a required `/Filter` entry which specifies the security handler, which defines the software module that implements the encryption process. There is a built-in password-based security denoted by `/Filter /Standard`, which is the Standard Security Handler described earlier.

The Encryption dictionary specifies the optional `/SubFilter` entry which completely describes the software module. `/SubFilter` is used to allow security handlers other than the one in `/Filter` to decrypt the document. It is not required for the `/Standard` security handler. This field was mainly introduced to support Public-Key Security Handlers.

All Encryption dictionaries implement an optional field called `/V` which indicates the algorithm used for encryption and decryption. Table 4-3 uses information from the ISO standard to describe how this is defined [1].

Table 4-3. Specifying Encryption Algorithms [1]

/V	Meaning
0	Undocumented algorithm. Should not be used.
1	General Encryption Algorithm. Key Length = 40 bits. RC4 or AES algorithm.
2	General Encryption Algorithm. Key Length h > 40 bits. RC4 or AES algorithm.
3	Unpublished algorithm. Key Length between 40 and 128 bits. Should not be used.
4	Custom defined Handler, defined further in `/CF, /StmF, and /StrF.`

An optional field is defined to quantify the key length is called /Length. This is used when V is either a 2 or a 3 since they have a variable key length according to Table 4-3. Its value ranges from 40 to 128, in multiples of 8.

When the value of /V is 4, the security handler can be custom defined. In this case, the /CF field implements a Crypt Filter dictionary. This dictionary is defined in this section further below. In addition to when /V is 4, the /StmF field defines a name for the crypt filter used for decrypting streams; this is a name that is present in the CF dictionary. The /StrF field is similar to StmF except that it defines the name for the crypt filter used for decrypting strings in the document. Finally, there is a third name field called /EFF, which is the name of the crypt filter for decrypting embedded files within the document. All of these fields in this paragraph are only used when the value of /V equals 4. The definition is meaningless if the value of /V is any other value.

Standard Security Handler:
The Standard security handler implemented by the specification is a basic password-protected PDF file that can be decrypted with either a user password or an owner password. It contains the above common entries in the Encryption dictionary but also implements some custom fields of its own. The fields /O and /U are required and provide a 32-bit string that is used in conjunction with the entered password to obtain the correct encryption key. There is a separate User and Owner password. There is a field called /P, which is an integer that defines the operations allowed for a user. A table in the ISO specification is defined in Table 22, and defines based on the revision of the security handler, whether the document can be printed, modified, copy or extract text, and fill-in forms. Bits 3-12 are important with this 32-bit integer value, the remainder of bits are 1's, making this a signed twos-complement number, implying that this will be a negative integer in the dictionary, which is shown in the example below.

Public-Key Security Handler:
The use of a Public-Key Security handler in the document implements all of the common Encryption dictionary fields plus a few more. This handler is identified through the /Filter field in the Encryption dictionary, the name must be a valid public-key security handler. Valid values are Entrust.PPKEF, Adobe.PPKLite, and Adobe.PubSec. The /SubFilter may contain the names adbe.pkcs7.s3, adbe.pkcs7.s4, and adbe.pkcs7.s5. The latter one is used for Crypt filter where only a stream is encrypted in an unencrypted document. The encryption dictionary is extended with a field called /Recipients which is an array of byte-strings indicating a PKCS#7 object.

A field entitled /P, an integer, defines a set of flags indicating permitted operations when opened as a user. The listing of permissions is similar to the Standard Security handler, but with slightly different bit positions so they are not the same.

Crypt Filters:
When the /CF field is defined in the Encryption dictionary it defines a dictionary of a listing of crypt filters. Crypt filters offer individual encryption of streams when the whole document itself is unencrypted. Crypt Filter are defined in their own dictionary and then referenced by /StmF and /StrF in the Encryption dictionary. When a stream is encrypted, the /Filter selected is /Crypt. In the stream dictionary there is a /DecodeParms dictionary that selects the Crypt filter by name (a defined name or the keyword /Identity). Identity is a standard crypt filter that does not process the data.

A Crypt Filter dictionary is defined by the optional field /Type, which is /CryptFilter. Another optional field is /CFM, which identifies the method used to decrypt data. The valid values for this are None, V2 (RC4 algorithm), or AESV2 (PDF-1.6, AES algorithm).

EXAMPLE 1(STANDARD SECURITY HANDLER – PASSWORD-BASED):

```
52 0 obj
<<
   /Filter /Standard %Password based
   /V 2 %Algorithm 1, 40-bit-128-bit encryption
   /R 3
   /Length 128 %128-bit Encryption
   /P -4   %All permissions granted
   /O <hex string with hash>
   /U <different hex string with hash>
>>
endobj

trailer
<<
   /Root 1 0 R
   /Info 7 0 R
   /Encrypt 52 0 R
   /ID [<##> <##> ]
   /Size 53
>>
startxref
48506
```

EXAMPLE 2 (PUBLIC-KEY SECURITY HANDLER):

```
52 0 obj
<<
   /Filter /Adobe.PPKLite %Defined
   /SubFilter /adbe.pkcs7.s3 %Defined
   /V 2 %Algorithm 1, 40-bit-128-bit encryption
   /Length 128 %128-bit Encryption
   /P 5
   /Recipients [ <byte strings> ]
>>
endobj

trailer
<<
   /Root 1 0 R
   /Info 7 0 R
   /Encrypt 52 0 R
   /ID [<##> <##> ]
   /Size 53
>>
```

EXAMPLE 3 (CRYPT FILTER):

```
3 0 obj %% 1st page
<<
    /Type /Page
    /Parent 2 0 R
    /MediaBox [0 0 612 792]
    /Contents 4 0 R
```

```
>>
endobj

4 0 obj %% Page contents
<<
    /Length 35
>>
stream
... Encrypted Page-marking operators ...
endstream
endobj

6 0 obj
<<
    /Type /Metadata
    /Subtype /XML
    /Length 15
    /Filter /Crypt %% Uses a crypt filter with these parameters
    /DecodeParms
     <<
          /Type /CryptFilterDecodeParms
          /Name /Identity %% Indicates no encryption, MyFilter0 could be used.
     >>
>>
stream
XML metadata %% Unencrypted metadata
endstream
endobj

8 0 obj %% Encryption dictionary
<<
    /Filter /MySecurityHandlerName
    /V 4 %% Allow crypt filters
    /CF %% List of crypt filters
     <<
         /MyFilter0
         <<
                /Type /CryptFilter
                /CFM V2 %% Uses the standard algorithm
         >>
     >>
    /StrF /MyFilter0 %% Strings are decrypted using /MyFilter0
    /StmF /MyFilter0 %% Streams are decrypted using /MyFilter0
     ... %% Private data for /MySecurityHandlerName
    /MyUnsecureKey (12345678)
    /EncryptMetadata false
>>
endobj
```

RECOMMENDATIONS:
Standard Encryption

1 **Validate**: Encryption dictionaries should be validated to verify that mandatory or required fields are present and implemented properly with correct data types.

2 **Validate**: Valid references to encryption information in the trailer of the document should also be present to ensure referential integrity.

3 **Validate**: Attempt to decrypt the document (no user password or default weak passwords) to further validate the remainder of the document.

4 **Remove**: N/A

5 **Replace**: N/A

6 **External Filtering Required**: N/A

7 **Review**: N/A

8 **Reject**: Reject encrypted PDF documents.

9 **Reject**: Reject encrypted PDF documents with no user password.

Public-Key Encryption

1 **Validate**: Encryption dictionaries should be validated to verify that mandatory or required fields are present (for Public-Key) and implemented properly with correct data types.

2 **Validate**: Valid references to encryption information in the trailer of the document should also be present to ensure referential integrity.

3 **Remove**: N/A

4 **Replace**: N/A

5 **External Filtering Required**: For recipient information in public-key dictionaries, external filters can be responsible for examining PKCS#7 objects to ensure correctness and suitability if it shall remain in the document.

6 **Review**: N/A

7 **Reject**: Reject encrypted PDF documents.

Crypt Filter

1 **Validate**: Encryption dictionaries should be validated to verify that mandatory or required fields are present and implemented properly with correct data types.

2 **Validate**: Valid references to encryption information in the trailer of the document should also be present for referential integrity.

3 **Validate**: Ensure referential integrity and ensure that the crypt filter is utilized in other streams throughout the document.

4 **Remove**: Remove streams that use one of the mentioned crypt filters.

5 **Replace**: N/A

6 **External Filtering Required**: For recipient information in public-key dictionaries, external filters can be responsible for examining PKCS#7 objects to ensure correctness and suitability if it shall remain in the document.

7 **Review**: N/A

8 **Reject**: Reject PDF documents that are specifying a Crypt Filter and using it as a stream filter.

PDF 4.18: END

4.5 Non Rendered PDF Content and Malformed Documents

PDF 4.19: COMMENT STRINGS

OVERVIEW:

PDF files contain comments throughout the document that are not interpreted by the viewer application. Comments in PDF files are similar to comments in any computer programming language, and have no impact on the end result. Normally, comment strings are not critical to PDF files and applications that read the files do not need to interpret them. However, there are exceptions to this rule. The header contents of the document (`%PDF-1.7`) contain the version that this file supports. The end of the document (`%%EOF`), indicates that there is no more data after this in the file. It is also recommended to include a comment string immediately following the header. The comment string should include high order ASCII characters.

CONCERNS:

Comments provide a field for text in the document and introduce a hidden data risk, since the contents are not displayed by a reader. Comments may also introduce a data disclosure risk as well. If the contents of the comments are binary, they could also introduce a data attack risk.

PRODUCT: PDF-1.0

LOCATION:

A comment string in the PDF file begins with the character '`%`' and continues until the end of the line. The exception of this rule is when the character appears within a string or a stream, then it relies on the character ')' to terminate the string, or the keyword endstream to terminate the stream contents.

EXAMPLE:

```
%PDF-1.7
%âãÏÓ

% This is a comment
1 0 obj
<<
    /Type /Catalog %This and the rest of the line is a comment
…
```

```
>>

2 0 obj
<<
    /Y (This is a string and % not a comment) %But this is a comment
    /Length 4
>>
%This is a comment but the % below in the stream is not a comment since its in
a %data stream
stream
12%3
endstream
endobj

5 0 obj
<<
     /Length 160
>>stream
BT
     %Comments are allowed here, which can sometimes be encoded
     1 Tr /F1 30 Tf 350 750 Td (base.pdf) Tj
ET
BT
     0 Tr /F1 15 Tf 186 690 Td (Empty test file) Tj
ET
endstream
endobj
```

RECOMMENDATIONS:

1 **Validate**: N/A

2 **Remove**: Remove comments within the file by removing the '%' sign and the entire comment until the end of the line.

3 **Replace**: Replace comment values with new desired text.

4 **External Filtering Required**: Extract all comments from the document and send the text to an external filter responsible for checking against known keywords or values that may need to be removed from the document.

5 **Review**: N/A

PDF 4.19: END

PDF 4.20: CONTENT OUTSIDE PDF OBJECT SPACE

DESCRIPTION:

PDF files are a listing of indirect objects, a header, a trailer, and a cross-reference table. If the trailer and cross-reference table information are correct, objects in the PDF file can reside in any order and at any byte location. If the reader software can correctly parse the file given the correct byte offsets of each object, then there could be space in between each object where data can reside. This is similar to the description about unreferenced objects in PDF 4.15:. However, in

this section, the focus is on raw data that is assumingly not parsed by the reader since the reader has enough information from the cross-reference table information and trailer to render the file appropriately, not just a valid PDF object left out of the file hierarchy. Although this convention does not follow the ISO standard, some reader implementations will continue to render the document. The content outside the PDF object space will not be displayed.

CONCERNS:

The primary two concerns for this section are hidden data and data attack. A PDF file can be crafted to contain hidden information in spaces between objects. PDF files can also be written to embed malicious code in this location. If the file is crafted properly, the document can still be rendered without the user's knowledge of this data's existence or activity that it may cause.

PRODUCT: PDF-1.0

LOCATIONS:

There are a few places where data can be hidden in a PDF file. This section describes possibilities where data can exist outside the PDF object and name hierarchy but remains in the file, and the file can be rendered appropriately to a user. These areas include:

- Hiding data between objects in the body of the PDF file, this can be immediately following the header as well. This implies that the reader parses the xref table correctly to get the correct byte offsets for each object.

- Hiding data between the traditional xref table and the Trailer since the trailer indicates where the xref starts, and the xref indicates where the objects exist. The PDF reader will look at the end for the trailer then jump to the byte offset of the xref table, something can hide in between.

Data can also be hidden in places that invalidate the PDF file, such that is cannot be read by PDF reader software. Locations such as

- Following the `%%EOF` marker that indicates the end of the file. Data can be appended following this as normal with incremental updates, but PDF reader software may fail in parsing the document if the updated trailer is not available and the file terminates with invalid content.

EXAMPLE 1 (CONTENT BETWEEN OBJECTS IN BODY):

```
%Hidden data can exist here too, but it's not a valid PDF according to the
%standard.
%PDF-1.7
%Hidden data

4 0 obj
<<
 /Type /Page
 /Parent 3 0 R
 /MediaBox [0 0 612 792]
 /Contents 5 0 R
 /Resources
 << /ProcSet 6 0 R
    /Font << /F1 7 0 R >>
 >>
>>
endobj
```

```
...
…File data…
...
5 0 obj
<< /Length 46 >>
stream
BT
/F1 24 Tf
100 700 Td(Hello World)Tj
ET
endstream
endobj

%xref table and trailer must be correct to individually access 4 0 and 5 0
```

EXAMPLE 2 (CONTENT BETWEEN XREF AND TRAILER):

```
xref
0 9
...
...

…End of Xref table
…File data…
…
trailer
<<
 /Size 9
 /Root 1 0 R
>>
%This needs to be correct for this to work.
startxref
32708
%%EOF
```

EXAMPLE 3 (CONTENT AFTER THE END OF FILE):

```
%PDF-1.7

xref
0 9
...
...

trailer
<<
 /Size 9
 /Root 1 0 R
>>
startxref
32708
%%EOF

%extra data at the end of the document, fails to open with some readers.
```

RECOMMENDATIONS:

1 Validate: N/A

2 **Remove**: Remove invalid and hidden contents between indirect objects, before the valid header, at the end of the file, and in between traditional cross-reference table and trailer information.

3 **Replace**: Replace invalid contents in all possible areas with a desired text.

4 **External Filtering Required**: Pass extra data in the PDF file to an external filter.

5 **Review**: N/A

6 **Reject**: Fail documents that contain illegal content within the document.

PDF 4.20: END

PDF 4.21: MALFORMED PDF FILES

DESCRIPTION:

PDF syntax can be represented in a variety of ways. Due to its complexity, reader software can also grow in complexity, leaving room for bugs and errors that can be uncovered due to various fuzzing techniques. While this section does not explicitly mention vulnerabilities or software errors in particular readers, it simply mentions that many PDF files may have syntax errors and it introduces a risk. The ISO standard indicates that these files are illegal and are not properly structured. Although each feature mentioned in this report should check for syntax and correct structure of the PDF object that is under review according to the standard, it's important to highlight and provide an example of an invalid PDF file.

CONCERNS:

PDF files that contain errors primarily are the result of a malicious author, whether to launch malicious content, embedded an executable, or to force the reader into an error state. Malformed PDF files that contain syntax or invalid formatting can introduce a data attack risk.

PRODUCT: PDF-1.0

LOCATIONS:

Throughout the PDF file, syntax errors may exist in a number of areas. The specification allows many fields to be optional and represented in a number of different ways (indirect reference or dictionary). Through indirect references, nested objects, the specification does not explicitly provide a strict standard to allow a 1 to 1 mapping of rendered document to underlying syntax. Additionally, objects can have missing fields, or missing terminators such as a ">>", or "endobj", or "endstream". All of these types of malformed syntax may introduce problems with parsing the file.

An example of malformed PDF is shown below, which indicates one of the smallest PDF files that can be created. It contains no cross-reference table, an incomplete header, no body, and the file trailer implements a minimal document catalog within the object that simply launches

JavaScript code. The result of this shows how reader software can be difficult to develop given a specification that provides so much flexibility.

EXAMPLE [13]:

```
%PDF-1.
trailer<</Root<</Pages<<>>/OpenAction<</S/JavaScript/JS(app.alert({cMsg:'JavaSc
ript'});)>>>>>>
```

RECOMMENDATIONS:

1 **Validate**: Check for consistency and referential integrity and perform strict syntax checking on the PDF document according to all rules of the standard. Ensure proper syntax as well as appropriate referencing of objects. Major items to consider are the proper termination of objects, dictionaries, and streams. Also strictly enforcing the referencing syntax (mandated use of indirect reference versus dictionary) may also be applied.

2 **Remove**: Remove objects with errors (such as those not properly terminated). Reconstruct the cross-reference table, trailer and other byte offset related fields.

3 **Replace**: Some syntax errors can be corrected or replaced to some minimal extent. Ensuring missing endobj/endstream to terminate contents or replacing nested objects to an indirect object in the PDF file. The document could be restructured to a certain degree if objects are not defined properly (nested object as opposed to indirect reference). This option may be limited, and as with any restructuring, the cross-reference table, trailer, and other byte offset related fields will need to be updated.

4 **External Filtering Required**: N/A

5 **Review**: N/A

6 **Reject**: Fail illegal PDF documents that contain invalid syntax and errors.

PDF 4.21: END

5. PDF FILE – CONSTRUCTS AND METADATA

PDF 5.1: METADATA

DESCRIPTION:

Metadata provides information about the document that is not directly visible in PDF reader software. Information such as document properties of the PDF client is normally used and represented in document metadata. Document metadata can be represented through a Document Information dictionary, linked to by the `/Info` field. Table 5-1 shows the fields available in the Document Information dictionary. There may also be additional custom fields in this dictionary which are used by other tools.

Table 5-1. Document Metadata Fields

Entry Name	Description
`/Title`	The title of the document.
`/Author`	The author of the document.
`/Subject`	The subject of the document.
`/Keywords`	Keywords associated with the document.
`/Creator`	The name of application software that created the original document. For example, Microsoft Word could have initially created the document.
`/Producer`	The name of the application software that produced the PDF file. For example, Acrobat Distiller could be a producer.
`/CreationDate`	The date and time the document was created, in human-readable form (ISO 8601).
`/ModDate`	The date and time the document was most recently modified, in human-readable form (ISO 8601).
`/Trapped`	A name object indicating whether the document has been modified to include trapping information.

CONCERNS:

Metadata is not always visible through PDF reader software. Its contents are not read by the end user unless the PDF reader software allows you to view attributes of the document. The PDF reader software only presents document properties that is chooses to display to the user. Therefore, a data disclosure risk may be possible since metadata may contain personal or sensitive information. Author information can be considered a sensitive field in metadata. Metadata could also be encrypted or encoded in an XML stream that may not be human-readable. Metadata may also introduce a hidden data risk if it is used to hide information rather than the actual metadata, and since it is not part of the document content.

PRODUCT: PDF-1.4

LOCATIONS:

Metadata can be represented in a PDF object by itself and referenced by the trailer of the document to specify additional information. The fields in the Document Information Dictionary will be applied to the entire PDF document. Example 1 demonstrates embedding metadata as a distinct object. Metadata can also be represented in a stream object by specifying the optional `/Metadata` field in the document catalog. The ISO standard recommends the usage of this method rather than

the information dictionary. This can be shown in Example 2. It must be followed by an XML stream that identifies the metadata. Metadata can also be attached to any PDF object through the `/Metadata` key added to its dictionary, which is not accessible like the Document metadata in Example 1.

Metadata can be referenced from a number of objects in PDF, summarized in Table 5-2.

Table 5-2. References to Metadata in ISO Standard

Dictionary/Location	Purpose	ISO Reference
Document Catalog	Metadata for the entire document.	Table 28
Page Object	Metadata for the page.	Table 30
ICCBased Color Spaces	Metadata for the Color Space.	Table 66
Image Dictionary	Metadata for an image.	Table 89
Type 1 Form Dictionary	Metadata for a Form XObject.	Table 95
Embedded Font Stream Dictionary	Metadata for an Embedded Font Program.	Table 127
Stream/Dictionary	Metadata for a specific component.	Table 316
XFA	Metadata can be included in XFA information.	XFA Specification.

EXAMPLES:

Example 1 (Document Information Dictionary):

```
1 0 obj
<<
     /Title (PostScript Language Reference, Third Edition)
     /Author (Adobe Systems Incorporated)
     /Creator (Adobe® FrameMaker® 5.5.3 for Power Macintosh®)
     /Producer (Acrobat® Distiller™ 3.01 for Power Macintosh)
     /CreationDate (D:19970915110347-08'00')
     /ModDate (D:19990209153925-08'00')
>>
endobj
```

Example 2 (XML Metadata Stream):

```
1 0 obj
<<
…
     /Type /Metadata  %Object type is Metadata
     /Subtype /XML    %Stream is an XML type
     /Length 3292     %Length of the stream embedded below
>>
stream
…XML Metadata Stream…
endstream
endobj
```

RECOMMENDATIONS:

1 **Validate**: N/A

2 **Remove**: In the document context, an indirect object such as Example 1 can be removed and any references in other objects that may refer to the object should also be removed. Fields with `/Metadata` should be removed within an existing object. Any attached data

associated with metadata should be removed as well. XML Data streams implementing metadata can also be removed (indirect object and stream data), as well as the reference to that Metadata from another indirect object. Since both options remove an indirect object, cross-reference, trailer, or other byte offset information will need to be reconstructed.

3 **Replace**: Find all existing Metadata objects or `/Type /Metadata` dictionaries and replace with a generic string of text indicating replacement.

4 **External Filtering Required**: If the metadata is in a human readable form, the text can be extracted and sent to an external filter. If the metadata is in an XML form, it should be passed to an XML filter.

5 **Review**: Examine the human readable contents of the metadata to determine acceptance. If the metadata is encoded or encrypted it would need to be decoded or decrypted and then parsed by a reviewer for acceptance.

PDF 5.1: END

PDF 5.2: PDF COORDINATE SPACES

DESCRIPTION:

To first understand how text and graphics are arranged in a PDF file, it is important to understand coordinate systems that are used in PDF. PDF supports and uses a number of coordinates' spaces and uses transformations from one space to another, eventually transforming to the device space, where it can be printed or displayed. The PDF standard mentions the following Coordinate Spaces [1]:

- Device Space: The Device's Coordinate System, depends on a printer or a physical display.
- User Space: Since having a PDF file dependent on a device is not portable, a User Space coordinate space was introduced to be device independent. User space can be defined per page of the PDF document, by specifying the coordinates for the rectangle or shape of the page. Transformations perform translating user space to a specific device space.
- Text Space: When using text operators in a content stream, which will be discussed later, there is a separate coordinate space that is used. There is an operator to define how this transforms into user space.
- Character Glyph Space: Each Glyph for a font has its own space, a 1000 unit map or square grid that which equals 1 unit of text space. So that it the transformation between Glyph Space to Text Space.
- Image Space: Images have their own space and the transformation between image space and user space is predefined.

- Form Space: Form XObject represent a graphic image and it contains its own coordinate space. Transformations are possible to convert it to user space.

- Pattern Space: A pattern is defined within a content stream, and is considered a type of color which can be used to be drawn on an area of shading. It contains its own pattern space.

- 3D: Later sections of this report define 3D artwork, which provide a 3D coordinate system.

Transformations are done via a transformation matrix which is available in many of the text or graphics operations in PDF. A transformation is represented as a 6 element array, which when combined in specific combination providing some kind of operation such as scaling, rotating, or skewing. The standard enumerates each type of transformation.

CONCERNS:

While coordinate spaces are important in PDF files, sanitization tools should be aware of page boundaries which were discussed in PDF 4.8:, and other boundaries of each coordinate space. Sanitization tools should also be aware of the transformation processes since some of them can be defined in the PDF. In addition, the complexity of transforming data from space to space may lend itself to an exploit or data attack risk. Ensuring that this feature is used properly and is used within reasonable parameters is important.

PRODUCT: PDF-1.0

LOCATIONS:

Transformation matrices are located in numerous locations in many graphics and text objects. They are embedded into content streams with the remainder of operands and operators that draw the content stream onto the page.

EXAMPLE [1]:

```
1 0 obj
<<
    /Length 42
>>
%Font size of 1, but scaled to 12
stream
BT
/F1 1 Tf
12 0 0 12 100 600 Tm
(Hello) Tj
ET
endstream
endobj
```

RECOMMENDATIONS:

1 **Validate**: Validate each transformation matrix in the PDF document to ensure that parameters are implemented properly.

2 **Remove**: N/A

3 **Replace**: Replace coordinate information with values within range, but ensure that it does not overlap other existing content. Replace transformation matrix information with an

approved matrix that does not obscure the contents and display information accurately in user and device space.

4 **External Filtering Required**: N/A

5 **Review**: N/A

PDF 5.2: END

5.1 Streams and File Content

The content of the actual page, images, embedded files, rich content, are all ultimately embedded within a stream object and stored in the PDF file. This section extends the summary from Section 4 on Stream Objects (PDF 4.10:) and Stream Filters (PDF 4.11:), and goes into further detail on how streams are used and what they exactly contain.

This section introduces Content Streams, which are referenced from a variety of places including the Page dictionary through the `/Contents` array. These streams comprise the actual content on a PDF page, including text, graphics, and can also contain references to draw images (XObjects) in the location on the page. Content Streams contain operators which set the color, position, font size, actual text to insert, draw a line between points, and many other features. For a full list of Content Stream operators, see the ISO standard, Annex A.2.

In addition to Content Streams, a stream object may contain any type of embedded data, and also embedded files. In this section, file specifications and external file references are discussed and how files can be attached to the document.

PDF 5.3: CONTENT STREAMS

OVERVIEW:

Content streams are PDF objects that contain stream data which represents a sequence of instructions indicating how a graphic is to be painted or represented on the page [1]. Content streams provide instructions in the stream data to perform or draw an operation onto the display area. The contents may be text rendering in a particular manner with a specific font, a graphical object, or a form (Form XObject). Content streams appear to look like any other stream in a PDF file that contains embedded data but within the stream contains operators and operands that represent data in the PDF syntax.

A page's content array references one or more content streams, which lists the content for that page. A conforming reader will be able to distinguish a content stream from a file stream through the PDF hierarchy.

CONCERNS:

Content streams introduce hidden data risks because their contents are not human readable. They may also be used as a data attack if the embedded stream content contains shellcode as opposed to valid content data.

PRODUCT: PDF-1.0

LOCATION:

A content stream is a PDF object in the file that is similar to any other type of stream object. It contains a Length field, a Filter field, and the contents of the stream are enclosed by the `stream/endstream` keywords that follow the stream's dictionary. Inside a content stream, there are number of operators and operands that provide the sequence of operations on the content data. The ISO Standard provides an Appendix of possible operators that can be used in a PDF file.

EXAMPLE:

```
13 0 obj
<<
    /Length 102
    /Filter /FlateDecode
%This text is normally encoded, shown here decoded to show example operators
%This displays the string "Test" on the page.
%Tm is a transformation matrix discussed earlier.
>>
stream
  /GS1 gs
  BT
      /TT2 1 Tf
      100 0 0 100 100 100 Tm
      0 g
      0 Tc
      0 Tw
      (Test) Tj
  ET
endstream
endobj
```

RECOMMENDATIONS:

1 **Validate**: Check for consistency and that the stream object has been validated accordingly as discussed in previous sections (PDF 4.10:, PDF 4.11:).

2 **Validate**: Extract and, if necessary, decode the content stream and analyze the operators and operands to ensure that they are permitted. Examine the operand content for sensitive data.

3 **Validate**: Check for referential integrity and that the stream is an actual content stream and was referenced from the one of the five possible locations in the PDF file (Page Content, Form XObject, Annotation Appearance Stream, Patterns, and Type 3 Fonts).

4 **Remove**: Remove content streams that cannot be decoded properly or if they can be decoded, those that contain invalid syntax.

5 **Replace**: The entire content stream can be replaced indicating a replacement of text or graphics. For a more detailed approach, operands within the content stream can be replaced to ensure text or graphics are visible.

6 **External Filtering Required**: N/A

7 **Review**: Extract and decode the content stream, extract the text that is to be displayed or the end result of a drawing in a separate PDF file, and analyze the contents manually.

PDF 5.3: END

PDF 5.4: EMBEDDED FILE STREAMS

OVERVIEW:

Binary data can be embedded into PDF files and indirectly referenced through the use of streams. This data can be embedded files or other types of information. This is unlike content streams where the encoded data is interpreted as a sequence of operators and operands representing basic data representation in the PDF syntax.

CONCERNS:

Embedding data into PDF files causes concern because there can be sensitive and/or malicious data contained within these objects. Data disclosure is a risk because the embedded content may contain sensitive material. Data attack can be a risk because the contents of the embedded file may be malicious and since its contents are encoded, they can be hidden. Hidden data is also a potential risk since the contents are encoded, which makes them unreadable until decoded.

PRODUCT: PDF-1.4

LOCATION:

A stream in a PDF file is contained within the keywords `stream` and `endstream`, and can be defined after the stream object's dictionary, which has been presented earlier. There is an optional `/Type` field that can indicate the stream is an embedded file, and `/Subtype` that indicates the type of file.

Embedded files may be referenced in the Name Dictionary of the PDF document. They are referenced through the field `/EF` entry in a file specification dictionary, shown as object 4 0 in the example below. This object points to the embedded data stream that contains the file data, shown in object 5 0.

Embedded files may also be included through a File Attachment Annotation, which can associate the file to a particular location in the PDF document.

EXAMPLE:

```
%File specification dictionary
4 0 obj
<<
   /Type /Filespec
   /EF 5 0 R
>>
endobj
```

```
%File stream
5 0 obj
<<
    /Type /EmbeddedFile
    /Length 4000
    /Filter /LZWDecode
    %Object specific dictionary entries.
>>
stream
%4000 byte file data
endstream
endobj
```

RECOMMENDATIONS:

1 **Validate**: Check that the stream object has been validated as discussed in earlier sections (PDF 4.10:, PDF 4.11:). Further validate the object by ensuring consistency of the file specification dictionary.

2 **Remove**: Remove the contents of the stream (between `stream` and `endstream`), along with the entire indirect object dictionary and references to that object throughout the document.

3 **Replace**: Embedded files can be replaced with some known data.

4 **External Filtering Required**: Extract and decode the stream contents and pass to an external filter.

5 **Review**: Extract and decode the stream contents for manual review.

PDF 5.4: END

PDF 5.5: EXTERNAL FILE SPECIFICATIONS

OVERVIEW:

A PDF file refers to contents of another using a file specification dictionary. The dictionary can specify the path of the file and numerous operating system specific fields that fully define the file. This file specification is represented by a separate dictionary. A file specification can point to an external file or an embedded file located within the PDF file, in this section we focus on external files.

CONCERNS:

Referencing another file from a PDF causes concern because this file may not have been properly sanitized. In the case of an external file, this allows the context of the current PDF file to extend beyond the scope of the current file. Data disclosure is a risk through the fully qualified path names of the external file, or the contents of the external file itself. External files referenced from

PDFs may also present a data attack risk if the contents of the actual referenced file are malicious.

PRODUCT: PDF-1.0

LOCATION:
File specifications are referenced by a File Specification dictionary, which provides attributes to define the file. The dictionary specifies a number of parameters to identify the location of the file. A file is denoted by the `/F` field which identifies the referenced filename.

EXAMPLE:

```
1 0 obj
<<
    %Required if /EF or /RF is present, shall be Filespec.
    /Type /Filespec

    %Optional Filesystem parameter, only valid entry is URL, URL based file
ref.
    /FS URL

    %Obsolete for writer software, use /F, /UF instead
    %/DOS
    %/Mac
    %/Unix
    %This is required if DOS, Mac, Unix are absent
    /F  (C:\\externalfile.txt)

    %Recommend if F exists in dictionary
    /UF (encoding)

    %ID used as a file identifier
    /ID [ <#> <#> ]

    %V, flag indicating if the file specification is volatile, to acquire
    %the file each time it is accessed.
    /V false

    /Desc (Description of this file)
>>
endobj
```

RECOMMENDATIONS:

1 **Validate**: Check that the mandatory fields are present and of the correct type per the standard.

2 **Validate**: Check for the `/V` field, if present, must always be set to false.

3 **Remove**: The file dictionary can be removed from the document. References to this file information should be removed as well.

4 **Replace**: N/A

5 **External Filtering Required:** Pass the pathname and description of the file specification to an external filter to examine the string for sensitive content.

6 **Review**: Extract the external file reference and its pathname for manual review.

PDF 5.5: END

PDF 5.6: EMBEDDED DATA STREAMS

OVERVIEW:

Stream data in PDF documents can be arbitrary and not necessarily content streams (with operators and operands) or point to an embedded file. They simply can contain data that has been compressed and stored within a stream object in a PDF. This section covers miscellaneous data that may reside solely in a PDF data stream.

CONCERNS:

Compressed and filtered data is a concern for hiding sensitive or malicious content. This poses a hidden data risk simply because data can be hidden through numerous stream filters. This could also be a data attack risk because malicious code may be embedded into a stream and encoded several iterations to avoid detection.

PROUCT: PDF-1.0

LOCATION:

Stream data looks similar to an embedded file stream except that the dictionary and hierarchy does not point to an embedded file or contain a file reference. It points to an arbitrary block of data, which once decoded, represents the content of the entity that referenced it originally.

EXAMPLE:

```
%FreeText Annotation with rich text in an embedded data stream, or text stream
9 0 obj
<<
    /Type /Annot
    /Subtype /FreeText
    /DA (/Font1 10 Tf 0 g)
    /RC 10 0 R %Rich text stream
>>
endobj

%Stream dictionary that contains text stream
10 0 obj
<<
     /Length 500
     /Filter /ASCII85Decode
>>
stream
<stream data>
endstream
endobj
```

RECOMMENDATIONS:

1 **Validate**: Ensure consistency and validate the stream object and its dictionary parameters (PDF 4.10:, PDF 4.11:). Extract and decode stream information to determine if the content is correct according to the referring object.

2 **Remove**: The stream object and data can be removed from the document. This involves removing the stream dictionary and data, along with references to that object. Depending on the type of data, if removed, it may impact the remainder of the document adversely.

3 **Replace**: N/A

4 **External Filtering Required**: Extract and decode the stream data and pass to an external filter.

5 **Review**: N/A

PDF 5.6: END

PDF 5.7: COLLECTIONS

OVERVIEW:

PDF files have the capability to attach files to the document, and starting in PDF 1.7, documents can specify to a reader how to present a collection of attached (embedded) files, otherwise known as a portable collection. The collection is used to sort, present, and search through relevant documents that are embedded into the PDF document. If a PDF document contains a collection, when the reader opens the file, it must show the contents of the original document, as well as the other documents in the EmbeddedFiles name tree. It's intended for the original document to contain information or a summary about the collection of documents.

CONCERNS:

Attaching other files to a PDF document is a concern because it is an embedded file that requires further examination before it can be revealed to the reader. A data attack risk with malicious content and hidden data risks can both be present through this capability. A data disclosure risk may also exist if certain data is revealed through a collection.

PRODUCT: PDF-1.7

LOCATION:

A collection is defined by its own dictionary and contains the `/Type /Collection` field, an optional `/Schema` dictionary object, an optional byte string `/D` that specifies the entry in the EmbeddedFiles name tree to display to the user. It also contains a field for the initial view (`/View`) which allows the view to show details, titles, or hidden which means that the user must perform an action to reveal the contents of the files. The Schema dictionary contains most of the information about the collection object.

EXAMPLE:

```
102 0 obj<<
```

```
        /MarkInfo <</Marked true>>
        /PageMode/UseOC
        /Names 2715 0 R
        /Outlines 11 0 R
        /Metadata 2787 0 R
        /AcroForm 141 0 R
        /Pages 98 0 R
        /OCProperties 2786 0 R
        /Collection 2712 0 R
        /StructTreeRoot 16 0 R
        /Type/Catalog
    >>endobj

    2712 0 obj<<
        /Sort 2743 0 R
        /Schema 2742 0 R
        /Type/Collection
    >>endobj

    2743 0 obj<<
        /A[true true true true]
        /S[/field3/field0/field2/field1]
        /Type/CollectionSort
    >>endobj

    2742 0 obj<<
        /Type/CollectionSchema
        /field0 2707 0 R
        /field1 2708 0 R
        /field2 2709 0 R
        /field3 2710 0 R
        /field4 2711 0 R
    >>endobj

    2707 0 obj<</Subtype/S/E false/N(To)/O 1/V true/Type/CollectionField>>endobj
    2708 0 obj<</Subtype/S/E false/N(Subject)/O 4/V
    true/Type/CollectionField>>endobj
    2709 0 obj<</Subtype/D/E false/N(Date Received)/O 2/V
    true/Type/CollectionField>>endobj
    2710 0 obj<</Subtype/S/E false/N(From)/O 0/V true/Type/CollectionField>>endobj
    2711 0 obj<</Subtype/Size/E false/N(Size)/O 3/V
    true/Type/CollectionField>>endobj
```

RECOMMENDATIONS:

1 **Validate**: Validate the collection scheme by ensure each required field is present and visible, and also validate the data types for correctness. Check that the embedded content (embedded files) has also been validated.

2 **Remove**: Remove the embedded file reference and collection entries that reference these documents.

3 **Replace**: Replace each embedded document or object with a flattened content that is visible as part of the context of the main document.

4 **Replace**: Check that its visibility is always true by replacing the view field in its dictionary (`/View /D`).

5 **External Filtering Required**: Extract and decode each embedded file and provide to external filter.

6 **Review:** N/A

PDF 5.7: END

5.2 Font and Text Objects

Fonts are an important topic in PDF files, because there are a wide variety of fonts that can be used, embedded within the file, and how they are referenced and utilized is important to understand. In PDF 5.8:, Content Streams are discussed again to illustrate how a block of text references specific fonts. The Content Stream ultimately defines how the text appears at a specific location on the page, including referencing a font.

PDF 5.8: TEXT CONTENT STREAMS

OVERVIEW:

Content streams have been presented earlier as a means to include text and graphics into the content of the PDF file. To introduce fonts in PDF files, it's first important to understand how they are referenced by the text using that font. As discussed before, text data is embedded within a content stream, which may or may not be readable by humans since it can be encoded. The content stream then introduces a number of operators and operands that define the text in its entirety.

CONCERNS:

Content streams and their concerns were discussed previously in PDF 5.3:, and mentioned that both text and graphic information can be stored in a content stream. There is a hidden data risk since their contents can be encoded.

PRODUCT: PDF-1.0

LOCATION:

A Text block is located in a PDF content stream and is enclosed by the keywords `BT` (Begin Text) and `ET` (End Text). Figure 5-1 shows an example of a text block.

```
BT
        1 Tr
        /F13 12 Tf
        288 720 Td
        (ABC) Tj
ET
```

Figure 5-1. Text Block

The example in Figure 5-1 shows the font (`Tf`), the positioning of the text (`Td`), the string (`Tj`) and the text rendering mode (`Tr`). These are text state parameters; a more detailed listing is in ISO32000-1:2008, Section 9.3, and the standard includes an appendix of all valid operands. The

font selected is represented by the field /F13 which is a reference to the names of the entries in the /Font dictionary of the current resource dictionary at the time the operator is invoked. With Figure 5-1, there will be an entry elsewhere in the document similar to /Font << /F13 23 0 R>> which links the font name /F13 to object 23 0. The font size of 12 is also shown after /F13. In the line that positions the text, coordinates are given to place the text on the page. The next line indicates the character IDs relative to the specified encoded for the font in the text block. The text rendering mode can be any of the 8 defined modes of representing text shown in the ISO32000-1:2008, Section 9.3.6.

EXAMPLE:

```
%PDF-1.5
%µµµµ
…
3 0 obj
<<
    /Type/Page
    /Parent 2 0 R
    /Resources
    << /Font
       <<
          /F1 5 0 R
       >>
    >>
    /MediaBox[ 0 0 612 792]
    /Contents 4 0 R
>>
endobj
%Contents from Object 3 0
4 0 obj
<<
    %Stream is decoded for example here
    /Filter/FlateDecode
    /Length 208
>>
stream
BT
/F1 12 Tf
1 0 0 1 72.024 709.2 Tm
0 g
0 G
[(Example Text)] TJ
ET
endstream
endobj

%Font Dictionary
5 0 obj
<<
    /Type/Font
    /Subtype/TrueType
    /Name/F1
    /BaseFont/Times#20New#20Roman
    /Encoding/WinAnsiEncoding
    /FontDescriptor 6 0 R
    /FirstChar 32
    /LastChar 120
    /Widths 15 0 R
>>
endobj
```

```
%Font Descriptor dictionary
6 0 obj
<<
   /Type /FontDescriptor
   /FontName /Times#20New#20Roman
   /Flags 32
   /ItalicAngle 0
   /Ascent 891
   /Descent -216
   /CapHeight 693
   /AvgWidth 401
   /MaxWidth 2568
   /FontWeight 400
   /XHeight 250
   /Leading 42
   /StemV 40
   /FontBBox [ -568 -216 2000 693]
 >>
endobj
```

RECOMMENDATIONS:

1 **Validate**: Check for consistency and that the stream object has been validated accordingly as discussed in previous sections (PDF 4.10:, PDF 4.11:).

2 **Validate**: Extract and, if necessary, decode the content stream and analyze the operators and operands to check that they are valid.

3 **Validate:** Ensure referential integrity and check that the text stream utilizes an existing font for that page.

4 **Remove**: If there is no font dictionary defined, then remove the text drawing operators and associated operands.

5 **Replace**: If there is no font dictionary defined, substitute a generic simple font dictionary.

6 **External Filtering Required**: Extract and decode the text strings from the content stream and pass to an external filter.

7 **Review**: Extract the end result of the content and present for manual review.

PDF 5.8: END

5.2.1 Simple Fonts

To describe PDF fonts, the differences between the terms character and a glyph need to be discussed first. A character is a symbol (much like the letters A, B, and C), which are universally recognized. A glyph is a graphical representation of that character, which can be done in a variety of ways. A collection of these glyphs form a font, or a font program. Rather than representing characters in a document, a PDF reader more appropriately draws the glyphs on the page that represents the characters. Some of these

fonts are traditional, standard, and well known; while some font programs and descriptions may be entirely embedded and defined within the current PDF file.

This section presents simple fonts such as Type 1 Fonts, which are known as PostScript or PostScript Type 1 fonts, which is a format for single-byte character codes, and a single mapping of character to glyph. This section also covers TrueType fonts, which was introduced by Apple, Inc. and Microsoft Corporation as a competitor to Type 1 fonts. This section also covers Type 3 fonts, which is a way to use PDF drawing operators that allows more complex glyphs to be defined. The PDF hierarchy of simple font structures and how they map to content on a page is shown in Figure 5-2.

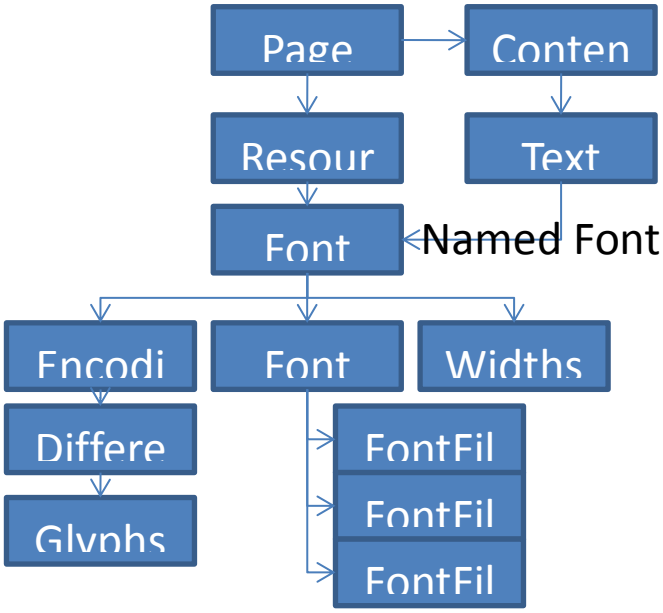

Figure 5-2. Simple Font PDF Object Hierarchy

PDF 5.9: TYPE1 AND TRUETYPE FONTS

OVERVIEW:

A Type 1 font is a PostScript font. Like all simple fonts, it maps single-byte characters to a specific glyph representation. A Type 1 font uses a more simplistic and condensed glyph representation.

Type 1 fonts feature a well known set of fonts called the Standard 14 fonts, which are Times-Roman, Helvetica, Courier, Symbol, Times-Bold, Helvetica-Bold, Courier-Bold, ZapfDingbats, Times-Italic, Helvetica-Oblique, Courier-Oblique, Times-BoldItalic, Helvetica-BoldOblique, and Courier-BoldOblique. These fonts are typically available to reader software; however a reader is free to substitute.

CONCERNS:

Fonts introduce a hidden data risk because font information can be modified in a manner to introduce obscurity or completely hide data from the reader. Data disclosure may also be a risk since the content may contain sensitive information.

Embedded fonts also provide a data attack risk since they are an embedded file that is parsed by the reader software. This is applicable when an embedded font program is used.

PRODUCT: PDF-1.0

LOCATION:
Type 1 fonts are represented at the highest level through a Font Dictionary as shown in the example below. A font dictionary implements the required `Type` field, which should be `Font` for the font dictionary. A required `Subtype` field is defined as `Type1` for Type 1 fonts. The `Name` field is only required in PDF-1.0, but is considered obsolete by the standard. This is largely because the font dictionary is a named object that is originally referenced from the `Page` object in its `Resource` dictionary, which has an entry for `Font`. In this examples it "names" the font `Xi1` in Object 1 0 through the resource dictionary.

The Font Dictionary implements a required `BaseFont` field which is the PostScript name of the font. In this example below, it is the Helvetica font.

The Font Dictionary provides two more required fields called `FirstChar` and `LastChar`. The fields define the first and last character code represented by the font, respectively. The Font dictionary also defines a required field called `Widths` (1000 units = 1 unit of text space) which should be an indirect reference to an array. This array maps width values to each glyph of the font that is associated with of the characters defined by `FirstChar` and `LastChar`.

A Font dictionary also must define a `FontDescriptor` which is an indirect reference which defines the font's metrics besides the Widths which is defined through the Width array.

The Font dictionary also implements the optional `Encoding` field which is either the values of `MacRomanEncoding`, `MacExpertEncoding`, or `WinAnsiEncoding`. Encoding can be defined because differences may exist from the default built-in encoding. Encoding may implement an embedded dictionary within the PDF file.

The final optional field in the Font dictionary is the `ToUnicode` field which is a stream that has the conversion from the character codes to Unicode values. This is useful for extraction of text content and it references a stream containing an embedded CMap file. The purpose of this file is to translate between the character codes to Unicode values.

Many of the features of the Font dictionary are defined to be entirely present in PDF-1.5. However, this is indicated only for writers, so PDF readers must be able to handle missing entries for standard fonts.

TrueType fonts are simple fonts that implement a font dictionary as well as a font descriptor, just like Type1 fonts. Refer to PDF 5.9: for more details on the font dictionary specific information. There are some differences in the BaseFont and how it is defined and the encoding field.

EXAMPLE – TYPE1 [1]:

```
%Page object
1514 0 obj
<<
    /Type/Page
    /Contents[204 0 R 1515 0 R 205 0 R]
    /Parent 1516 0 R
    /Resources
```

```
        <<
            /Font<</C2_0 1520 0 R
                   /TT1 1521 0 R
                   /TT0 1522 0 R
                   /Xi1 1 0 R
                  >>
         >>
         /CropBox[0.0 0.0 595.0 842.0]
         /MediaBox[0.0 0.0 595.0 842.0]
         /Rotate 0
    >>
    endobj

    %Font Dictionary, given the name Xi1
    1 0 obj
    <<
        /Type/Font
        /BaseFont/Helvetica
        /Subtype/Type1
        /Encoding/WinAnsiEncoding   %Standard, not external.

        %Optional ToUnicode
        /ToUnicode 2 0 R
    >>
    endobj

    %CMap to Unicode translation
    2 0 obj
    <<
        /Length 200
    >>
    stream
    %CMap File data
    endstream
    endobj
```

EXAMPLE – TRUETYPE [1]:

```
    %Page number 1, pay close attention to resources dictionary which lists the
    font
    1514 0 obj
    <<
        /Type/Page
        /Contents[204 0 R 1515 0 R 205 0 R]
        /Parent 1516 0 R
        /Resources
        <<
            /Font<</C2_0 1520 0 R
                   /TT1 1521 0 R
                   /TT0 1522 0 R
                   /Xi1 1 0 R
                  >>
         >>
         /CropBox[0.0 0.0 595.0 842.0]
         /MediaBox[0.0 0.0 595.0 842.0]
         /Rotate 0
    >>
    endobj

    %This is font dictionary for TT0
    1522 0 obj
    <<
```

```
    /FirstChar 32  %Starting character is the space key
    /Type/Font
    /FontDescriptor 7970 0 R
    /BaseFont/HDHCMJ+Arial,Bold
    /Subtype/TrueType
    /LastChar 201 %End character is beyond low value ASCII
    /Encoding /WinAnsiEncoding
    /Widths[278 0 0 0 0 0 0 0 333 333 0 0 0 333 278 278 556 556 556 556 556 556
0 556 556 556 333 333 0 0 0 0 0 722 0 722 722 667 611 778 722 278 0 0 611 833
722 778 667 0 722 667 611 722 667 0 0 667 0 0 0 0 0 0 0 556 611 556 611 556 333
611 611 278 0 0 278 889 611 611 611 0 389 556 333 611 0 778 556 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1000 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 333 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 667]
>>
endobj

%This is TT0's FontDescriptor dictionary
7970 0 obj
<<
    /Type/FontDescriptor
    /CapHeight 718
    /XHeight 515
    /Flags 32
    /FontStretch /Normal
    /Descent -211
    /FontBBox[-628 -376 2000 1010]
    /StemV 144
    /FontName/HDHCMJ+Arial,Bold
    /FontFamily(FamilyName)
    /FontFile2 9968 0 R %A font file, this will be discussed later.
    /Ascent 905
    /FontWeight 700
    /ItalicAngle 0
>>
endobj
```

RECOMMENDATIONS:

1 **Validate**: Check consistency and that Type 1 fonts and their font dictionaries, encoding, and descriptors should be validated in their entirety. Font dictionaries should implement all of their required fields with correct values.

2 **Validate:** The FirstChar and LastChar should be accurate and correct values, not outside the range for characters.

3 **Validate:** Check consistency with the values of the Widths array, ensuring that the values at each array position do not obscure or hide glyphs and that the values match those that are contained within the font itself.

4 **Validate:** Validation should continue into the Font Descriptor dictionary to ensure that all values in that dictionary are valid. If Encoding is defined, it should be one of the 3 valid possibilities, or to be Custom with a Differences array.

5 **Remove**: Remove Font Descriptor indirect reference.

6 **Remove**: The ToUnicode dictionary and stream can be removed with a standard font where Unicode mappings are known.

7 **Replace**: A substitution should be made on the entire font dictionary and font descriptor. These should be predefined fonts that are considered acceptable.

8 **Replace**: If the original Font Dictionary has been validated, then replace the stream contents with an approved ToUnicode CMap for this particular font.

9 **External Filtering Required**: Validation that extends into the Font Descriptor dictionary should validate font files if they are embedded. Font Descriptor dictionaries are discussed in PDF 5.13:.

10 **External Filtering Required**: Extract and decode the ToUnicode data stream and have inspected by an external filter for fonts (CMaps).

11 **Review**: N/A

PDF 5.9: END

PDF 5.10: TYPE3 FONTS

OVERVIEW:

Type 3 fonts are different from Type 1 and TrueType fonts because in Type 3 fonts, each glyph is defined within the PDF file in a separate content stream. The content stream is composed of graphical operators and operands, and any graphical operator is valid in a Type 3 font. Therefore, a user could design a font and embed it within the PDF file.

CONCERNS:

Type 3 fonts present a concern because the glyph information and drawing operations are included in the PDF file. The risk for hiding data and obscuring data is higher because of the flexibility to define custom glyphs. Two major options are possible; glyphs could be created to represent characters that are encoded inside a content stream. The rendered content may not make sense but the encoded content is hidden to the user. The second option is to have the text content and characters within the content stream not make any sense to the filtering application but the glyphs could be designed in such a way that they reveal content once rendered to a user. Both of these are examples of a data hiding risk.

Data attacks can be a concern when rendering font file information, particularly when embedded into the file, which has been used as an attack vector recently. Data disclosure attacks can be possible simply due to sensitive information obscured due to custom font information.

PRODUCT: PDF-1.0

LOCATION:

Type 3 fonts follow the simple font hierarchy of PDF objects as the other fonts presented in this section. The Font dictionary indicates a `Type3` font, the `FontDescriptor` describes the font metrics and the `Encoding` dictionary must be present and define the complete character encoding for the font. The `BaseFont` is not present like Type1 fonts, so this field will not

indicate the base standard font, which means more investigation is needed into the nested structures to uncover the font description.

In other simple fonts, encoding was typically a default generic encoding that is understood by the reader software. In Type 3 fonts, character encoding is supplied in the PDF file. The `Encoding` dictionary is supplied as a reference that must define a `Differences` array. Encoding dictionaries provide three fields: a Type which is Encoding, a BaseEncoding, which is a defined name (`MacRomanEncoding, MacExpertEncoding`, or `WinAnsiEncoding`), and a `Differences` array.

The `Differences` array can be present in fonts and can be misused. It should be inspected in all cases where it exists. The array identifies the starting code for the character and then follows a sequential list of names of the glyphs. In the example below, the differences array specifies that starting with character 97 ('a'), the glyphs for 97 should be the 'tee' object and 98 should be the 'x' object. These glyphs are defined in external content streams that are referenced through the CharProcs field in the Font dictionary. The glyph content streams both contain operators and operands that draw the image on the screen.

EXAMPLE (PDF RENDERED):

Figure 5-3. Example PDF Rendered

EXAMPLE (PDF SYNTAX):

```
%PDF-1.7
…
3 0 obj
<<
    /Type /Page
    /Parent 2 0 R
    /Resources
     <<
         /Font <</F1 4 0 R>>
         /ProcSet[/PDF/Text/ImageB/ImageC/ImageI]
     >>
    /MediaBox[ 0 0 612 792]
    /Contents 5 0 R
>>
endobj

4 0 obj
<<
    /Type /Font
    /Subtype /Type3
    /FontBBox [0 0 750 750]
    /FontMatrix [0.001 0 0 0.001 0 0]
    /CharProcs 10 0 R
    /Encoding 9 0 R
    /FirstChar 97
    /LastChar 98
    /Widths [1000 1000]
```

```
>>
endobj

5 0 obj
<<
    /Length 50
    %Text is abab, not txtx
>>
stream
BT
    /F1 1 Tf
    100 700 Td
    (abab)Tj
ET
endstream
endobj

%Encoding object supplied by Font Dictionary
9 0 obj
<<
    /Type /Encoding
    %ASCII character 97(a) is a 't', and 98(b) is an 'x'
    /Differences [97 /tee /x]
>>
endobj

%Associates object 11 and 12 with a Named Object
10 0 obj
<< /tee 11 0 R
/x 12 0 R
>>
endobj

%Stream of Tee glyph
11 0 obj
<< /Length 79 >>
stream
1000 0 0 0 750 750 d1
350 0 m
350 600 l
0 600 l
0 700 l
700 700 l
700 600 l
400 600 l
f
endstream
endobj

%Stream of X glyph
12 0 obj
<< /Length 51 >>
stream
1000 0 0 0 750 750 d1
0 750 m
750 0 l
f
750 750 m
0 0 l
f
endstream
endobj
```

RECOMMENDATIONS:

1 **Validate**: Validation for Type 3 fonts should be similar to Type 1 fonts, but more involved. The font dictionary, encoding and differences array, and font descriptor should all be validated in this process. Type fields should be correct for Type3 fonts; BaseFont should not be present.

2 **Validate**: Check consistency and that `FirstChar` and `LastChar` are valid values.

3 **Validate**: Check consistency and validate the Widths array is accurate and is not hiding particular characters. (Note: Difficult to do when they are your own glyphs)

4 **Validate**: Check consistency and validate the Font descriptor dictionary for correctness. If the differences array is defined, it should be validated as well, along with each content stream that represents each defined glyph.

5 **Remove**: Remove the Differences array since it modifies the encoding. This means characters will be represented by the base encoding, which could not be desired.

6 **Replace**: Replace the Type 3 font structures with an approved Type 1 font that is approved and valid and does not obscure any content.

7 **External Filtering Required**: Extract and decode all content streams related to this font hierarchy. This process involves analyzing glyphs per each character to ensure its acceptability.

8 **Review**: Manually review the rendered text versus the actual text within the content stream.

PDF 5.10: END

PDF 5.11: CHARACTER ENCODING

OVERVIEW:

Character encoding has been presented in Type 1, TrueType, and Type 3 fonts. In the previous section, Character encoding was shown how to manipulate an existing font with custom glyphs. The standard defines encoding as the association between character codes and glyph descriptions [1]. Every simple font, except for Type 3 fonts, has a built-in encoding. This is preferred in most cases but allowing the PDF to define character coding provides flexibility as it may be required in some instances.

CONCERNS:

Character encoding can be used for legitimate purposes when applying differences to certain characters, but this feature could be used in a malicious manner. It can be a hidden data risk, if the differences array is defined to obscure the original contents of the document.

PRODUCT: PDF-1.0

LOCATION:

Character encoding provides an encoding dictionary to override the built-in encoding that is part of each simple font. Font dictionaries can simply reference the base encoding name in the Encoding entry in their dictionary. If this is not done, then a dictionary must be implemented defining the character encoding.

The character encoding dictionary implements an optional `/Type /Encoding`, and an optional entry for `/BaseEncoding`, which is either `/MacRomanEncoding`, `/MacExpertEncoding`, or `/WinAnsiEncoding`. If this is not defined, the `/Differences` array defines the differences from an implicit standard base encoding, or the encoding defined from an embedded font file. The third field in the encoding dictionary is the `/Differences` field which is an array that defines a mapping of character codes and known character names that will be used to override a base encoding. The differences array has an organization such that it can define a character code as the first entry, followed by the character name in a sequential order. The array can be disjointed and restart the sequential sequence followed by the next set of sequential character names.

EXAMPLE [1]:

```
25 0 obj
<<
    /Type /Encoding
    %No base encoding, assume implicit standard, or can be defined in Font File
    /Differences
      [ 39 /quotesingle
        96 /grave
        128 /Adieresis /Aring /Ccedilla /Eacute /Ntilde /Odieresis /Udieresis
            /aacute /agrave /acircumflex /adieresis /atilde /aring /ccedilla
            /eacute /egrave /ecircumflex /edieresis /iacute /igrave
/icircumflex
            /idieresis /ntilde /oacute /ograve /ocircumflex /odieresis /otilde
            /uacute /ugrave /ucircumflex /udieresis /dagger /degree /cent
            /sterling /section /bullet /paragraph /germandbls /registered
            /copyright /trademark /acute /dieresis
        174 /AE /Oslash
        177 /plusminus
        180 /yen /mu
        187 /ordfeminine /ordmasculine
        190 /ae /oslash /questiondown /exclamdown /logicalnot
        196 /florin
        199 /guillemotleft /guillemotright /ellipsis
        203 /Agrave /Atilde /Otilde /OE /oe /endash /emdash /quotedblleft
            /quotedblright /quoteleft /quoteright /divide
        216 /ydieresis /Ydieresis /fraction /currency /guilsinglleft
            /guilsinglright /fi /fl /daggerdbl /periodcentered /quotesinglbase
            /quotedblbase /perthousand /Acircumflex /Ecircumflex /Aacute
            /Edieresis /Egrave /Iacute /Icircumflex /Idieresis /Igrave /Oacute
            /Ocircumflex
        241 /Ograve /Uacute /Ucircumflex /Ugrave /dotlessi /circumflex /tilde
            /macron /breve /dotaccent /ring /cedilla /hungarumlaut /ogonek
            /caron
       ]
>>
endobj
```

RECOMMENDATIONS:

1 **Validate:** Check that the `/Differences` array only covers non-alphanumeric characters, which could be used to introduce obscurity.

2 **Validate:** Check that the `/Differences` array is terminated properly and does not contain invalid mapping beyond the character space.

3 **Remove:** Remove `/Differences` array and check that a base encoding is defined.

4 **Replace:** Replace Character Encoding dictionary with standard encoding name in Font Dictionary.

5 **External Filtering Required:** N/A

6 **Review:** N/A

PDF 5.11: END

5.2.2 Composite Fonts

Composite fonts are used to represent characters in which there is a larger character space than simple fonts. Recall in simple fonts, there was a single byte or character mapping to a glyph. Composite fonts were created to extend fonts to contain more than 256 glyphs, common for many foreign languages and Unicode where the character space is larger than 8 bits. Composite fonts are referred to as Type 0 fonts. To understand terminology in the composite font section, there a few items to introduce. First, as discussed in simple fonts, the character which is an abstract term that represents a value that has a universal meaning, like a letter in the alphabet is an example. There is a term called a Character Identifier, or CID, which maps an integer value to each character. The combined characters create a character collection, each with a unique CID. A certain type of font called a CID-keyed font describes how CID values of characters can be used to map to glyph data.

Composite Fonts or Type 0 fonts obtains glyphs from a CIDFont, which is considered a descendant of the root Type 0 font. A CID-keyed font, or a CIDFont, uses CID as a character selector to obtain glyphs since the organization of CIDFonts are by the integer CID value, as opposed to names of characters. A Character Map (or CMap) specifies the relationship between character codes and CID numbers that are used to obtain glyphs. CMaps are analogous to encoding for single byte simple font, and also have similar risks as presented before. The key difference, CMaps can map a larger space from multiple byte codes to a larger number of glyphs that belong to a CID-keyed font. In PDF terminology, there are dictionaries for CMaps and CIDFonts, along with font dictionaries for Type 0 fonts.

PDF 5.12: TYPE 0 FONTS

OVERVIEW:
Type 0 font dictionaries introduce a composite font in a PDF file. Due to its complexity, there are related dictionaries that accompany this one such as the descendant CIDFont dictionary and

CMap that maps character codes to a CID, which then get used to select a glyph for representation on the page of the PDF file.

CONCERNS:

Type 0 fonts can introduce similar hidden data risks as Type1 and TrueType fonts. Encoding can be modified in CMap information which can hide information. Since font data can be embedded into the PDF file, a data attack is also possible on the software rendering the embedded font. As with any font, data disclosure risks are possible when sensitive information is contained within text content.

PRODUCT: PDF-1.0

LOCATION:

CIDFont Dictionary:
CIDFonts are implemented as their own dictionary. They are similar to a font dictionary since the `/Type` must be `/Font`, but the `/Subtype` is `CIDFontType0` (or `CIDFontType2`). The CIDFont dictionary implements a required `BaseFont`, which is a name value for the CIDFontName. A required dictionary called `CIDSystemInfo` must be present in a CIDFont dictionary because it is used to describe the character collection of the CIDFont. Similar to simple fonts, a CIDFont dictionary implements an indirect reference to a FontDescriptor, which contains other metrics about the font. Optionally, CIDFonts may provide a `DW` field, which indicates the default widths of the glyphs for this font. Also, optional as well, a `W` widths array can be used to specify the widths, and if unused, the `DW` field shall be used. Optionally, a `DW2` field can be present which is similar to `DW` but applies for vertically written. Likewise, an optional `W2` field is similar to `W` but for vertical fonts. Finally, but only for Type2 fonts, there is `CIDToGIDMap`, which maps CIDS to glyph indices, only when a Type2 CIDFont, which is based on TrueType fonts, is embedded within the PDF file.

CMap Dictionary:

CMap dictionaries are used to map character codes to character selectors, which known in PDF as CIDs [1], which select from a CIDFont. This replaces the Encoding dictionary that can be used in Simple Fonts presented earlier. CMaps are represented through a named object that is a standard predefined CMap, or a custom CMap in a stream object. The ISO standard provides a long list of named standard CMaps. To implement a custom CMap, the stream object is defined with additional parameters to a stream dictionary. A required field for `Type` is `CMap`, along with a required `CMapName`, which is the name of the CMap file. There is an additional required dictionary called `CIDSystemInfo`.

Type 0 Font Dictionary:

Type 0 fonts are defined just as every other font, including a CIDFont, starting with the Font dictionary, with type defined as `Type0`. A required field called `BaseFont` defines the name of the Type 0 font. In Adobe's implementation, the name is the BaseFont name of the `CIDFont`, a hyphen, and the `CMap` name, but this is not a required naming convention. Type 0 font dictionaries require an `Encoding` field that defines the name of the `CMap`. The CMap will map each character codes to font numbers and CIDs. It may be a name of a standard CMap or a custom one.

Similar to Simple Fonts, Type 0 fonts implement a required `Encoding` field in the dictionary. This is either the name of the pre-defined `CMap` or reference to the stream that contains the custom CMap mapping character codes to font numbers and CIDs.

Also in the Type 0 font dictionary is a required array called `DescendantFonts` which specifies a one-element array contain the CIDFont dictionary. This is needed because the Type0 font is the root font, and the CIDFont, a descendant.

Finally, the Type 0 font dictionary implements an optional `ToUnicode` stream that contains a cmap file (basically a mapping) which converts the character codes into Unicode values. This was also presented with simple fonts.

EXAMPLE [1]:

```
1520 0 obj
<<
    /Type /Font
    /DescendantFonts [7968 0 R]
    /ToUnicode 1523 0 R
    /BaseFont /LELHFP+Symbol
    /Subtype /Type0
    /Encoding 10 0 R
>>
endobj

7968 0 obj
<<
    /Type /Font
    /DW 1000
    /FontDescriptor 7969 0 R
    /CIDSystemInfo << /Supplement 0
                      /Ordering (string_value)
                      /Registry (string_value)
                   >>
>>
%Custom CMap in a stream (Referenced from ISO Standard)
10 0 obj
<<
        /Type /CMap
        /CMapName /90ms-RKSJ-H
        /CIDSystemInfo << /Registry (Adobe)
                          /Ordering (Japan1)
                          /Supplement 2
                       >>
        /Length 11 0 R
>>
stream
…data…
begincmap
1 beginnotdefrange
<00>  <1F> 231
endnotdefrange
…
endstream
enbobj
```

RECOMMENDATIONS:

1 **Validate**: Validation of Type 0 and other composite fonts requires thorough inspection of the font dictionary of the CIDFont, the Type0 (or other) font dictionary, Font Descriptor dictionary, and the CMap for the encoding equivalent of the font.

2 **Remove**: Remove embedded CMap information and replace with a predefined character encoding if possible.

3 **Replace**: Replace the CIDFont, Type0, and CMap dictionary with a standard and defined composite font structure.

4 **External Filtering Required**: Extract embedded CMap information and pass to an external filter.

5 **Review**: Extract and decode the text represented in the content stream (in a standard font) and compare against the rendered visible content. Manually inspect for review if this is possible.

PDF 5.12: END

5.2.3 Font Descriptors

PDF 5.13: FONT DESCRIPTORS

OVERVIEW:

A font descriptor is an object that is referenced from a font dictionary. It defines additional attributes of the font other than the Widths that is defined in an array of the font dictionary. Font Descriptors are common to both simple fonts and composite fonts. Font Descriptor dictionaries are interesting because they allow for inserting Embedded Font Programs through Font files.

CONCERNS:

The largest concern with Font Descriptor objects is that fact that they reference embedded font files which have been known for data attack risks. Through embedding custom fonts in the PDF file, a hidden data risk may also be present.

PRODUCT: PDF-1.0

LOCATION:

A Font Descriptor implements its own dictionary, which is referenced by the Font Dictionary. The Font Descriptor dictionary is identified by the `/Type /FontDescriptor` and the required `/FontName`, the value of the BaseFont or CIDFont that referred to this dictionary. Optionally, for Type 3 fonts in PDF-1.5, a byte string indicating the /FontFamily.

Another optional field in the Font Descriptor dictionary, used for Type 3 fonts in Tagged PDF, is a `FontStretch` field which indicates a name of `UltraCondensed, ExtraCondensed, Condensed, SemiCondensed, Normal, SemiExpanded, Expanded, or`

UltraExpanded. Also, a FontWeight number is present to define the thickness or weight which ranges for valid values from 100-900, in increments of 100.

The Font Desciptor dictionary requires a Flags integer to be present, which is a 32-bit value with each bit indicating a characteristic of the font. Table 123 of the ISO32000-1:2008 indicates the breakout of bits and characteristics.

The Font Descriptor requires a FontBBox rectangle to be defined which is the bounding box for the glyphs coordinate system. This is not necessary for Type 3 fonts.

The remainder of fields in the font descriptor dictionary indicate the italicAngle, Ascent, Descent, Leading, CapHeight, XHeight, StemV, and StemH parameters that identify the thickness, the height, and other detailed parameters about glyphs of this font.

There are 3 optional entries for stream objects in a Font Descriptor dictionary called FontFile, FontFile2, and FontFile3. The FontFile stream entry contains a Type 1 font program. The FontFile2 stream entry contains a TrueType font program. The FontFile3 entry contains any other embedded font program, which is identified as a Subtype in the stream dictionary. Only one of these three fields should be present in the dictionary.

In CIDFonts, a Font Descriptor can contain additional entries, one that specifies the Language of the font, called /Lang. This is a named object which its value is a code from RFC 3066, which can identify the language of the font if it is not apparent in the font name.

EXAMPLE [1]:

```
%This is TT0's FontDescriptor dictionary, a TrueType font
7970 0 obj
<<
    /Type/FontDescriptor           %A FontDescriptor
    /CapHeight 718
    /XHeight 515
    /Flags 32                      %Nonsymbolic
    /FontStretch /Normal
    /Descent -211
    /FontBBox[-628 -376 2000 1010] %Glyph rectangle bounding box
    /StemV 144
    /FontName/HDHCMJ+Arial,Bold    %The name of the Font referred to
    /FontFamily(FamilyName)             %The Font Family name,
    /FontFile2 9968 0 R            %A font file for TrueType embedded font.
    /Ascent 905
    /FontWeight 700
    /ItalicAngle 0
>>
endobj

%FontFile2 stream, contains embedded content.
9968 0 obj
<<
    /Filter/FlateDecode
    /Length 27588
    /Length1 47856
>>
stream
<font file contents>
endstream
endobj
```

RECOMMENDATIONS:

1 **Validate**: Validation of a Font Descriptors includes ensuring consistency and that the required fields are present and correct, along with font names and the flags. There should only be a single FontFile indirect reference to a stream object, which requires validation as well.

2 **Remove**: Remove embedded font file, replacement may be needed on the higher level Font Dictionary if this font information is not available to the reader.

3 **Replace**: Replace Font Descriptor dictionary with a standard descriptor information, along with a standard font file if one is embedded in a stream.

4 **External Filtering Required**: Extract and decode the font file stream and review the font program by an external filter.

5 **Review**: N/A

6 **Reject:** Reject files that contain an embedded font file.

PDF 5.13: END

5.3 Graphical Objects and XObjects

PDF 5.14: GRAPHICS OBJECTS

OVERVIEW:

The PDF ISO 32000-1:2008 identifies five different types of graphics objects:

- Path Object: An arbitrary shape composed of straight lines, rectangles, and cubic Bezier curves [1]. It is a content stream containing operators and operands that sequentially draw these items on the page.
- Text Object: A string of characters that identify a sequence of glyphs to be drawn in the document.
- External Object (See PDF 5.15:): A named resource object that is defined outside of the content stream data. For more information, various types of External Objects, or XObjects, are covered in later sections.
- Inline Image Object (See PDF 5.20:): An image contained directly within a content stream. Inline images are covered in later sections.
- Shading Object: A shape whose color is a function of position within the shape.

CONCERNS:
Images and graphical objects are a concern in many documents because they represent data that can be embedded or manipulated in ways that are not obviously apparent to a human reader. There is a concern with any image the possibility of hidden data. There is also a possible data attack risk if the graphic targets the rendering software.

PRODUCT: PDF-1.0

LOCATION:
Graphics objects are represented in content streams. In a content stream, both text and path images can be represented. Content streams are referenced in the PDF hierarchy indicating that the stream data includes sequences of instruction.

One important set of operators is the `q` and `Q` operator, which means to save and restore the graphics state, respectively. This is similar to stack push and pop operations, where information is stored and then restored. Graphics state parameters and information should be examined when they are used in content streams. Table 51 of the ISO standard has more operators that can be used and how they are related.

EXAMPLE (PATH OBJECT) [1]:

```
%PDF-1.4
…
5 0 obj
<<
    /Length 883
>>
stream
    % Draw a black line segment, using the default line width.
    150 250 m
    150 350 l
    S

    % Draw a thicker, dashed line segment.
    4 w % Set line width to 4 points
    [4 6] 0 d % Set dash pattern to 4 units on, 6 units off
    150 250 m
    400 250 l
    S
    [] 0 d % Reset dash pattern to a solid line
    1 w % Reset line width to 1 unit
    1.0 0.0 0.0 RG % Red for stroke color
    0.5 0.75 1.0 rg % Light blue for fill color
    200 300 50 75 re
    B

    % Draw a curve filled with gray and with a colored border.
    0.5 0.1 0.2 RG
    0.7 g
    300 300 m
    300 400 400 400 400 300 c
    b
endstream
endobj
```

RECOMMENDATIONS:

1 **Validate**: Check that the stream has been validated just as any other object stream in the document. Decode the stream and ensure that valid commands and possible valid

operands exist in the decoded content stream. Ensure the operands are correct for the coordinate space of the object.

2 **Remove**: Remove the content stream containing graphics and references to the content stream in the document.

3 **Replace**: N/A

4 **External Filtering Required**: N/A

5 **Review**: Extract and decode the stream and reconstruct the graphic and manually inspect the graphic for acceptability.

PDF 5.14: END

PDF 5.15: IMAGE EXTERNAL OBJECTS (XOBJECTS)

OVERVIEW:

An external object (XObject) is a data object that contains information solely in a data stream and referenced by name in a separate content stream in which it is used. This section focuses on basic Image XObjects, where sampled image data is stored within the stream. A XObject implements a dictionary which defines certain parameters of the image, such as height or width. Image XObjects are used in content streams through the "Name" of the XObject (requiring a name for the object) and the `Do` operator in the context of the content stream. This command essentially paints the sampled image from the "external object", allowing for greater reusability.

CONCERNS:

Image XObjects are susceptible for hidden data since they can be manipulated in many ways to obscure content. There have also been several vulnerabilities regarding the processing of external objects and images as a way to have arbitrary code execution on the reader's side, which is a data attack risk.

PRODUCT: PDF-1.0

LOCATION:

Image XObjects contain an image dictionary that identifies the `/Type /XObject` and `/SubType /Image`, along with the `/Width`, `/Height`, and `/ColorSpace` for the image. The dictionary for an image also contains characteristics for alternate images and image masks. An alternate image is indicated by the `/Alternates` field and allows a different image to be rendered in certain occasions. The `/Mask` field specifies another separate image to act as a "mask" that changes how the image is rendered. The `/Mask` entry can be a specified color, an array of colors, or a stream that can be applied to the image. The image dictionary also allows an optional decode entry that indicate how to decode the image using a known stream filter. For more information on masking, refer to PDF 5.26:.

EXAMPLE:

```
%Here is the page object; the XObject references object 12, which is R10, which
%references object 10, which is the actual XObject.
20 0 obj
<<
    /Type/Page
    /MediaBox [0 0 612 792]
    /Rotate 0
    /Parent 3 0 R
    %Resource dictionary for Page references XObject, object 12 contains name.
    /Resources<</ProcSet[/PDF /ImageC /Text]
                /ExtGState 11 0 R
                /XObject 12 0 R
                /Font 13 0 R
    >>
    /Contents 21 0 R
>>
endobj

%Content stream that draws the XObject
21 0 obj
<<
    /Length 94
    %uses the Do command to paint the XObject R10
>>
stream
/R10 Do
endstream
endobj

%The name of the XObject is R10, that's important because the content stream
%invokes it by this name R10.
12 0 obj
<<
    /R10 10 0 R
>>
endobj

%The XObject has an image dictionary plus data stream
10 0 obj
<<
    /Subtype/Image
    /ColorSpace/DeviceRGB
    /Width 360
    /Height 360
    /BitsPerComponent 8
    /Filter/DCTDecode
    /Length 14056
>>
stream
<image data>
endstream
endobj
```

RECOMMENDATIONS:

1 **Validate**: Check for referential integrity and that the PDF hierarchy contains all the necessary objects and references required to include a XObject. Ensure that a name exists for the XObject such that it can be referenced by other content streams (such that it is not an unused XObject).

2 **Validate:** Check for referential integrity and that the document contains a reference through the `Do` operator to the name of the XObject.

3 **Validate:** Check for consistency and that the image dictionary contains all the required fields and that their values are valid (valid height, width, colorspace, bits per component (if applicable)). Ensure that boundary boxes or information with coordinates are valid, which applies for all XObjects. Ensure that approved filter sequences are used for encoding the data. Ensure that the length is valid with stream contents.

4 **Validate:** Check for consistency and that if Alternates are defined, first that the image itself is not already an Alternate, and second that the Alternate images have been validated.

5 **Remove**: If the image is an alternate, remove the Alternates array. If the alternates are not valid, remove references to them.

6 **Remove:** Remove the Alternates array and every alternate XObject.

7 **Remove:** Remove mask information that obscures the original image. This will change the visual appearance of the document.

8 **Remove:** The entire hierarchy of the XObject in the document should be removed, including the XObject dictionary and stream data, along with references in other content streams in the document.

9 **Replace**: N/A

10 **External Filtering Required**: Extract and decode the content and pass to an external filter.

11 **Review**: Extract, decode, and pass data for review.

PDF 5.15: END

PDF 5.16: POSTSCRIPT XOBJECTS

OVERVIEW:

A PostScript XObject is a fragment of code written in the PostScript description language contained within a stream. They are rendered only when printing to a PostScript output device, which may not be a conforming PDF reader.

According to the ISO standard for PDF-1.7, PostScript XObjects should not be used. In PDF-1.5, it is mentioned initially that PostScript XObjects are no longer recommended to be used [15].

CONCERNS:

PostScript objects are deprecated and are normally used with caution and for legacy purposes. According to the ISO standard, their use should be avoided. They present a hidden data and data disclosure risk since many readers may not be able to interpret their contents or they may be printed incorrectly. There is also the possibility of a data attack risk on the software interpreting the stream content since it is a deprecated feature.

PRODUCT: PDF-1.1

LOCATION:

A PostScript object is represented as an object in the body of the PDF document with the `/Subtype /PS`, or it may contain the fields `/Subtype /Form` and `/Subtype2 /PS`. A PostScript object may contain the `/Type /XObject` and may contain the field `/Level1` that indicates the PostScript stream.

EXAMPLE:

```
13 0 obj
<<
    /Type /XObject
    /Subtype /Form
    /Subtype2 /PS
    /BBox [0.0000 0.0000 191.9996 275.9998]
    /Length 1502032
>>
%%This stream contains the PostScript fragment.
stream
%!PS-Adobe-3.0
%%EndComments
userdict begin
/b4_Inc_state save def
/dict_count countdictstack def
/op_count count 1 sub def
/showpage {} def
0 setgray 0 setlinecap
1 setlinewidth 0 setlinejoin
10 setmiterlimit [] 0 setdash newpath
/languagelevel where
{pop languagelevel
1 ne
     {false setstrokeadjust false setoverprint
     } if
} if
-313 -96 translate
%%BeginDocument: C:\document.eps
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 313 96 505 372
%%LanguageLevel: 1
%%Creator: VITScript 11
…
…remainder of PostScript…
```

RECOMMENDATIONS:

1 **Validate**: Validate the dictionary of the PostScript XObject and checking that it contains accurate parameters, similar to any stream object.

2 **Remove**: PostScript XObjects should be removed in their entirety and any references to them shall be removed. If a Level1 stream is present, remove this object as well.

3 **Replace**: Replace stream data with known PostScript functionality.

4 **External Filtering Required**: Pass stream data to an external filter.

5 **Review**: N/A

PDF 5.16: END

PDF 5.17: FORM XOBJECTS

OVERVIEW:

Form XObjects are content streams (with operators and operands) that represent collections of graphics objects, including text objects, path objects (drawings) and sampled images [1]. Form XObjects are self-contained descriptions of a sequence of graphics objects. It is used primarily for graphics that are used repeatedly and to cache the rendered images to save processing time.

There exists a special Form XObject called a group XObject which is used to group graphical objects together for different purposes. This type of XObject is discussed in PDF 5.18:.

A Reference XObject is another special type of a Form XObject that can be used to import content and data from one PDF Document into another [1]. The document where the reference occurs is called the containing document and it imports content from the target document. This will be covered in PDF 5.19:.

Form XObjects may provide the basis for a page in the PDF file. They may also be used for repeatedly used images and in conjunction with group and reference information.

CONCERNS:

Form XObjects can be susceptible to hidden data problems as discussed with other images within PDF documents. Form XObjects can contain embedded content or external content, which may result in a data attack risk. Any content stream can refer to a Form XObject or Reference XObject.

PRODUCT: PDF-1.0

LOCATION:

There are two parts to the form XObject: the first is its dictionary which describes attributes about the Form XObject and the second is the content stream immediately following the dictionary, which describe the graphical objects. As any XObject, it is a Named resource and accessed by name through the `Do` operator in another content stream, the content that actually draws the set of images. The entire list of dictionary fields (optional and required) is in the ISO standard, but a few of interest are described here.

The Form XObject dictionary is defined by the optional `/Type /XObject`, and the required `/Subtype /Form`, which indicates a Form XObject. There is an optional code indicating form XObject type called `/FormType`, which is an integer always equal to 1. A required Bounding Box, or `/BBox` is a rectangle indicating the area that contains and clips the Form XObject. There

is also an optional /Matrix array with transforms the space between the form and the user space.

The Form XObject dictionary contains an optional field to implement a Group dictionary through the /Group field. This defines the Group attributes dictionary which is covered in later sections.

The Form XObject dictionary also contains an optional field to implement a Reference dictionary through the /Ref field. This indicates that this XObject is a Reference XObject, which is covered in more detail in later sections.

There is an optional stream object in the Form XObject dictionary called /Metadata, which contains metadata for this content. As discussed before, there are inherent problems with metadata due to its content not being represented in the rendered PDF content.

EXAMPLE:

```
6 0 obj
<<
    /Type /XObject
    /Subtype /Form

    %Transparency only defined for Group XObjects
    %Device color space (devicergb in this case)
    /CS /DeviceRGB

    %Coordinates outlining the location of the XObject
    /BBox [0 0 1000 1000]
    /Filter [/LZWDecode]
    /Length 58
    /Metadata 7 0 R %Stream object with metadata.
>>
stream
<stream data>
endstream
endobj
```

RECOMMENDATIONS:

1 **Validate**: Check for consistency and validate the XObject has each required field defined and correct just as any other stream object in the PDF document.

2 **Remove**: Removing the XObject and its references. It is important to note that XObjects may reside in the body of the PDF, named through a Resource dictionary on a certain page, or referenced in a content stream. Removal requires checking all of these possible areas and removing references to the deleted XObject.

3 **Replace**: N/A

4 **External Filtering Required**: N/A

5 **Review**: Extract and decode the stream data for manual review.

PDF 5.17: END

PDF 5.18: GROUP XOBJECTS

OVERVIEW:

A group XObject is a type of Form XObject that groups graphical objects together. The specification provides a definition for a Group dictionary. Currently, the only type of group is a Transparency group, or Transparency Group XObjects. For more discussion on Transparency, see PDF 5.23:.

CONCERNS:

Since the only group currently supported is Transparency, this feature can introduce a data disclosure risk. If used intentionally it could be viewed as a hidden data risk when dealing with graphical objects. As any XObject, there is a potential for data attack risk with embedded contents.

PRODUCT: PDF-1.4

LOCATION:

A group XObject, or a transparency group XObject has a base Form XObject, but there is a Group field in the XObject dictionary that points to a Group attributes dictionary. In this Group dictionary, an optional `/Type` field and value of `/Group` can be present to indicate a group attributes dictionary. A required field `/S` (for subtype) is needed and its value can only be `/Transparency`.

In addition, the ISO standard indicates that a page object can have a group entry similar to a Form XObject. The group attributes dictionary should be treated similar as the Form XObject.

The stream content of a group attributes dictionary contains graphics objects that are used by an external `Do` operator that paints the objects onto the page.

EXAMPLE:

```
6 0 obj
<<
    /Type /XObject
    /Subtype /Form

    %Coordinates outlining the location of the XObject
    /BBox [0 0 1000 1000]
    /Filter /LZWDecode
    /Length 58
    /Metadata 7 0 R %Stream object with metadata.

    /Group 8 0 R %This is a Group XObject now
>>
stream
<stream data>
endstream
endobj
>>

%Group dictionary
```

```
8 0 obj
<<
    /Type /Group
    /S /Transparency
>>
stream
<stream data>
endstream
endobj
```

RECOMMENDATIONS:

1 **Validate**: Validation of Group XObjects require validation of the higher level XObject (just as any normal XObject) and ensuring consistency of the Group Attributes dictionary and stream content. Check that the required parameters in each dictionary are implemented and are valid.

2 **Remove**: Remove the Form XObject through its normal method, but also remove the Group Attributes dictionary and stream if unused or referenced by any other objects in the PDF file.

3 **Replace**: N/A

4 **External Filtering Required**: Extract and decode the original form XObject content for an external application review.

5 **Review**: Extract and decode the original form XObject and the group attributes dictionary for manual review.

PDF 5.18: END

PDF 5.19: REFERENCE XOBJECTS

OVERVIEW:

Reference XObjects allow for a PDF document to import content from another PDF document. Similar to a Group XObject, it descends from a Form XObject and is considered to be a Form XObject. The target document may reside in a file external to the containing document or may be included within it as an embedded file stream.

CONCERNS:

Reference XObjects are a serious concern due to their ability to bring content from one PDF file into another, whether it contains hidden data or malicious content.

PRODUCT: PDF-1.4

LOCATION:

Reference XObjects are originally defined in a Form XObject (dictionary and stream), with a `/Ref` entry in the Form XObject dictionary that points to a Reference dictionary.

EXAMPLE:

```
6 0 obj
<<
    /Type /XObject
    /Subtype /Form

    %Coordinates outlining the location of the XObject
    /BBox [0 0 1000 1000]

    %Reference dictionary identifying a page to be imported from another PDF
    %file.
    /Ref 8 0 R
    /Length 58
>>
stream
<stream data>
endstream
endobj
>>

%Reference dictionary
8 0 obj
<<
    %reference to imported file, this may be an external file or embedded file.
    %Page and ID are left as generic values.
    /F 9 0 R
    %Page Index (integer) or Page Label Dictionary
    /Page 1
    /ID [ <##> <##> ]
>>
endobj
```

RECOMMENDATIONS:

1 **Validate**: Check for consistency of the XObject dictionary as discussed earlier. Validation should also examine the Ref object and file specification stream of the other PDF file. The File Specification and if present, the embedded file stream, should also be validated as discussed earlier.

2 **Validate**: Validate the page information (index as a valid integer), or as a page label.

3 **Remove**: Remove the reference to the other PDF file and its file specification and possible stream data.

4 **Remove:** Remove the XObject dictionary and data stream, along with the reference to the other PDF file including its file specification dictionary and possible stream data.

5 **Replace**: N/A.

6 **External Filtering Required**: Validate the referenced PDF data as a separate PDF file. This is a separate PDF file that should undergo the same recommendations as any other PDF file.

7 **External Filtering Required**: Pass all string data in the file specification or page label dictionary, if present, to an external filter.

8 **Review**: N/A

PDF 5.19: END

PDF 5.20: INLINE IMAGES

OVERVIEW:

In PDF documents images can be defined directly in a content stream with the use of Inline Images objects. They are considered an alternative to image XObjects since they are defined directly in the content stream of an object, as opposed to an external object referenced through the name of the object.

Inline images can be encoded data within an existing stream object. This means there is a filter list identifying how the image data is encoded, as well as additional entries that define the same properties to those in an Image XObject. The structure of the inline image looks very similar to an Image XObject, except that it is embedded in an existing content stream. Abbreviations (as defined in Table 93 and Table 94 of the ISO standard) are permitted and may be used with Inline Images, and not anywhere else in the PDF document.

CONCERNS:

The threat of an inline image is similar to that of a XObject, but the image data is embedded differently within a content stream. Inline image data is encoded in an already encoded stream object. The numerous iterations of encoding and decoding could be the subject of a data attack. As with any image data, hidden data and disclosure may also be a risk.

PRODUCT: PDF-1.0

LOCATION:

Inline images exist in the content stream of an object and represented by the operator `BI` (begin image), `ID` (image data), and `EI` (end image). In the `BI` section, key-value pairs are described that document the image width, height, the color space, bits per component, and filters that are used. The image data contains the stream of data and the end image simply terminates the inline image.

EXAMPLE:

```
5 0 obj
<<
    /Length 200 %200 bytes in content stream
>>
stream
q                   %Save graphics state.
17 0 0 17 298 388 cm %Scale and translate coordinate space
%Now begin the actual inline image
BI                  %Begin inline image object
    /W 17           % Width in samples
    /H 17           % Height in samples
    /CS /RGB        % Color space
    /BPC 8          % Bits per component
    /F [/A85 /LZW]  % Filters
ID                  % Begin image data
… Image data …
```

```
EI                  %End inline image object
Q                   %Restore graphics state
endstream
endobj
```

RECOMMENDATIONS:

1 **Validate**: Ensure consistency and that the containing content stream and its dictionary has been validated. Decode the content stream to examine information, determine that valid operators and operands are used.

2 **Remove**: Remove the actual stream object and references to the stream object throughout the document.

3 **Replace**: N/A

4 **External Filtering Required**: Extract the image data from the content stream and pass to an external filter.

5 **Review**: Extract the image data from the content stream and pass for review.

PDF 5.20: END

5.4 Formatting, Rendering, and Transparency

Objects within the body of a PDF can be sized, cropped, colored, rendered, and positioned differently. This allows for the creation of a PDF file with numerous hidden or obscured objects including both text and image contents. This section presents some of the risks associated with the advanced formatting, rendering, and transparency group features. Sizing and cropping information was presented earlier in the Page sections, where page boundaries as discussed. In general, many objects are defined to a location on the page and it must fall within defined page boundaries in the page object.

PDF 5.21: TEXT COLORING

OVERVIEW:

Data can be hidden easily by manipulating colors and transparency levels of objects in the document. Techniques with color involving setting the foreground and background colors similarly which results in the object not being as visible. Color attributes are common to all objects within the PDF file, and their usage should be reviewed.

CONCERNS:

Objects that have their color attributes modified and can create a level of obscurity in the document, which introduces a hidden data risk. Colors can be changed such that data becomes invisible or difficult for a human to render.

PRODUCT: PDF-1.0

LOCATION:
Colors can be manipulated throughout the document. Methods such as setting the color can be present in operators of a content stream that identify the color value. Images often have parameters in dictionaries that directly impact this feature.

EXAMPLE: (BASELINE, BLACK TEXT ON WHITE RECTANGLE):

```
4 0 obj
<</Filter/FlateDecode/Length 207>>
%Stream is decoded below into plaintext.
stream
/P
<</MCID 0>>
BDC
1 g % White rectangle = 1, Black rectangle = 0
-0 -0.099976 612.1 792.1 re  %This covers the entire Media box
f*
EMC

/P
<</MCID 1>>
BDC
BT
/F1 12 Tf
1 0 0 1 72.024 709.2 Tm
0 g %Black text
0 G
[(There is black text on top of a white rectangle)] TJ
ET
EMC
endstream
endobj
```

EXAMPLE (HIDDEN BLACK TEXT ON BLACK RECTANGLE):

```
4 0 obj
<</Filter/FlateDecode/Length 207>>
%Stream is decoded below into plaintext.
stream
/P
<</MCID 0>>
BDC
0 g %Black rectangle = 0, White rectangle = 1
-0 -0.099976 612.1 792.1 re %Rectangle covers the Media Box
f*
EMC

/P
<</MCID 1>>
BDC
BT
/F1 12 Tf
1 0 0 1 72.024 709.2 Tm
0 g %Black text = 0, White text = ~1
0 G
[(There is black text on top of a black rectangle)] TJ
ET
EMC
endstream
endobj
```

EXAMPLE (HIDDEN WHITE TEXT ON WHITE RECTANGLE):

```
4 0 obj
<</Filter/FlateDecode/Length 209>>
%Stream is decoded below
stream
/P
<</MCID 0>>
BDC
1 g  %White rectangle = 1, black rectangle = 0
-0 -0.099976 612.1 792.1 re %Rectangle covers the Media Box
f*
EMC

/P
<</MCID 1>>
BDC
BT
/F1 12 Tf
1 0 0 1 72.024 709.2 Tm
0.973 g %This is close to white color, such that it is not visible
0.973 G
[(There is white text on top of a white rectangle)] TJ
ET
EMC
endstream
endobj
```

RECOMMENDATIONS:

1 **Validate**: Ensure consistency on the content stream and validate the operators and operands.

2 **Validate:** Ensure the text color is visible.

3 **Remove**: N/A

4 **Replace**: Replace attributes that hide an element with values that do not, such as setting contrasting color values for foreground and background.

5 **External Filtering Required**: N/A

6 **Review**: N/A

PDF 5.21: END

PDF 5.22: IMAGE COLORING

OVERVIEW:

Images can be represented in PDF in a number of ways. Some of them have been presented in this report with Graphics in Content Streams, and many types of XObjects. The standard discusses two topics in the Color Spaces section, which is color specification and color rendering. Color specification is defining abstract colors that are used for certain color, in a device-

independent manner. Color rendering is the process and transformation in a device-depended manner to produce colors on a particular output. With Color Specification, the definition of color space families is important.

There are numerous color space families that are used in PDF files. There are Device Color Spaces called `DeviceGray`, `DeviceRGB` (red, green, blue), and `DeviceCMYK` (cyan, magenta, yellow, black). These color spaces are the most simple and do not supply any additional parameters once they are used. There are also CIE-based color spaces, which is a standard created by the International Commission on Illumination better known as the ICC). The names of these color spaces are `CalGray`, `CalRGB`, `Lab`, and `ICCBased`. These color spaces require additional parameters in separate custom PDF dictionaries. The third type of color space is a Special Color Space which adds to an existing color space. Valid names for these types of spaces are `Pattern`, `Indexed`, `Separation`, and `DeviceN`.

CONCERNS:

Coloring with graphics and images can present a hidden data risk since the color space can be modified to possibly obscure content.

PRODUCT: PDF-1.1

LOCATION:

Changing color settings can occur in content streams with various operators. The `CS` and `cs` commands set the color space, and there are commands to set the stroking (`SC/SCN`) and nonstroking (`sc/scn`) color. Similar commands, `G`, `RG`, and `K` can also impact the color.

XObjects may also specify a color space that was discussed above, which is a predefined PDF named object. XObjects specify this through the `/ColorSpace` field in the XObject dictionary.

Color Spaces can also be introduced into PDFs by the Graphics State Parameters dictionary and they are specified within a content stream through certain operators. This is different from how XObjects explicitly specify color spaces.

ICCBased color spaces are implemented as a stream object with a dictionary that is defined by Table 66 in the ISO standard. This dictionary provides the number of color components in the color space by the required `/N` field. It also provides an optional `/Alternate` field which is an array or name, which defines an alternate color space in case the ICCBased color space defined in the stream object cannot be used, or is not supported. This should be investigated as it may be used by non-conforming readers.

EXAMPLE:

```
%PDF-1.4
…
5 0 obj
<<
    /Length 883
>>
stream
    % Draw a line segment, using the default line width and color (black).
    150 250 m
    150 350 l
    S

    % Draw a thicker, dashed line segment.
    4 w % Set line width to 4 points
```

```
        [4 6] 0 d % Set dash pattern to 4 units on, 6 units off
        150 250 m
        400 250 l
        S
        [] 0 d % Reset dash pattern to a solid line
        1 w % Reset line width to 1 unit
        1.0 0.0 0.0 RG % Red for stroke color
        0.5 0.75 1.0 rg % Light blue for fill color
        200 300 50 75 re
        B

        % Draw a curve filled with gray and with a colored border.
        0.5 0.1 0.2 RG
        0.7 g
        300 300 m
        300 400 400 400 400 300 c
        b
endstream
endobj
```

RECOMMENDATIONS:

1 **Validate**: Check that only valid color spaces and color values are used through the document.

2 **Validate**: Check alternative color spaces if they are defined for a valid name or correct values in the array specified by the `/Alternate` field.

3 **Remove**: N/A

4 **Replace**: Replace color space definition with standard device color spaces.

5 **External Filtering Required:** In the case of an external image or object that might have obscurity through coloring, extract and decode the information for external analysis.

6 **Review:** Pass the file for manual review.

PDF 5.22: END

PDF 5.23: TRANSPARENCY

OVERVIEW:

Imaging in PDF files allow for transparency, where objects blend with other overlapping objects. The blending process is called compositing. There is more information in the ISO standard regarding the actual specifics of transparency but in this section the focus is on how to specify or use transparency in PDF and transparency groups. In a PDF each image or painted object can be arranged in a transparency stack, where the objects are listed in order and each contributes to the visibility at a given point. The colors of each painted object at a particular point are done so by compositing rules.

In a stack of objects (in the transparency stack), one or more consecutive objects can be combined into a transparency group, which was briefly discussed in the Group XObjects section (PDF 5.18:). However, transparency can be achieved without the use of Group XObjects. Objects within the group are treated as a separate transparency stack or group stack. There is a backdrop, a composite color, shape, and opacity defined for the group. A group backdrop is the result of compositing all elements up to but not the first element in the group [1]. An initial backdrop is the one selected for compositing the group's first element. [1]. The immediate backdrop is the result of compositing all of the elements in the group up to and not including the current element [1]. Groups are not always needed for transparency, but they do allow for more control.

A transparency group can have several attributes listed in Section 11.4.1 of the ISO standard. These attributes include:

- Input variables affecting compositing computation, including mask and constant shape, mask and constant opacity, and blend mode.
- Isolated or non-isolated groups. An isolated group contains objects that are composited onto an initial backdrop rather than the group's backdrop. The opposite is true for a non-isolated group, where each element is composited onto the group's backdrop. An isolated group can specify its own blending color space, which doesn't have to be the same as the group's backdrop [1].
- Knockout or non-knockout groups, which determine if the objects in a stack are blended or composited with each other or only with the group's backdrop. In a knockout group each object is composited with the group's initial backdrop, as opposed to the other elements before this object. Therefore, each object "knocks-out" the other preceding objects if they overlap one another. The opposite is true for a non-knockout group.
- A group's result, rather than being displayed on a current page, can be used as a soft mask.

CONCERNS:

Transparency introduces a hidden data risk because the blending of content may introduce obscurity hiding particular content from a user. An unintentional blending of elements may also lead to a data disclosure risk, which is important to sanitize each element in the stack. Each layer or painted object must be inspected and sanitized, and then same process again after the compositing or blending has been performed.

There is a concern with knockout groups since they can hide data from preceding objects in the stack.

Due to its complexity, a data attack risk may be possible with many of the blending and operations that can be introduced.

PRODUCT: PDF-1.4

LOCATION:

Specifying transparency in PDF files is located in Section 11.6 of the ISO standard. Transparency is used with graphic elements, such as XObjects, which is located in the example below.

Group Transparency dictionaries, located in a Form XObject in the /Group field identify the Transparency group (/S /Transparency), the color space of the group (a name or array, see PDF 5.22:) a Boolean flag called /I which is true for isolated groups, false for non-isolated groups. There is another Boolean flag called /K which is true for knockout groups, false for non-knockout groups.

EXAMPLE [16]:

```
%BMult name used for graphics state parameters
32 0 obj <</Type /ExtGState  /BM /Multiply    /CA 1.0  /ca 1.0>> endobj
% G2Y XObject
% ===========
42 0 obj
<</Type /XObject
  /Subtype /Form
  /FormType 1
  /BBox [-.77 -.77 .77 .77]
  /Group <<
             /S /Transparency
             /CS /DeviceRGB
             /I false
             /K false
         >>
>>
stream
  /BMult gs
  q 1 0 0 1 -.25  .25 cm    /Circle Do Q
  q 1 0 0 1  .25  .25 cm    /Circle Do Q
endstream
endobj
% Circle XObject
% ==============
% 1-unit circle centered at (0,0)
22 0 obj
<<
     /Type /XObject
     /Subtype /Form
     /FormType 1
     /BBox [-.57 -.57 .57 .57]
     /Matrix [1 0 0 1 0 0 ]
>>
stream
  .7 .7 .7 rg
  0 w
  .5 0 m
  .5 -.276 .276 -.5 0 -.5 c
  -.276 -.5 -.5 -.276 -.5 0 c
  -.5 .276 -.276 .5  0 .5 c
  .276 .5 .5 .276 .5 0 c
  b
endstream
endobj
```

RECOMMENDATIONS:

1 **Validate**: Check that each XObject in each stack has been previously sanitized.

2 **Remove**: Remove the stack of objects, including the removal of each XObject and element that is used.

3 **Replace**: N/A

4 **External Filtering Required:** Pass the XObject data for external filtering.

5 **Review:** Review the stack of objects and each individual object to ensure data is neither hidden nor obscured.

PDF 5.23: END

PDF 5.24: TEXT RENDERING

OVERVIEW:

Text, images, and objects within a PDF file are rendered, which can impact an object's visibility. Text rendering modes can be specified in a content stream through the operator `Tr`. There are several modes which are valid in Section 9.3.6 of the ISO standard.

CONCERNS:

Certain text rendering modes may introduce a hidden data risk. In particular, text rendering can be performed such that information is not properly displayed to an end user.

PRODUCT: PDF-1.0

LOCATION:

Rendering modes for text operations can be found within the operators of a content stream. The `Tr` operator indicates the rendering mode with several optional from the specification. The example below demonstrates a content stream with text that is hidden due to its selected rendering mode.

EXAMPLE:

```
20 0 obj
<<
    /Length 40
    %This text is now hidden since the rendering operator Tr (3) is invisible.
>>
stream
BT
        /F13 12 Tf
        288 720 Td
        3 Tr
        (ABC) Tj
ET
endstream
endobj
```

RECOMMENDATIONS:

1 **Validate**: Ensure that text operands in the content stream do not utilize a rendering mode (3 – Invisible) that hides information from the user.

2 **Remove**: If the contents under inspection contain hidden data or information that will not be revealed to the reader software, remove a text block within the content stream that uses a rendering mode that hides the text from the viewer.

3 **Remove:** If the contents under inspection contain hidden data or information that will not be revealed to the reader software, remove the entire content stream and references to the content stream throughout the document.

4 **Replace**: If the contents under inspection contain hidden data or information that will not be revealed to the reader software, its rendering mode should be changed to a value that provides the highest visibility to the reader software.

5 **External Filtering Required**: Extract the text and pass to an external filter.

6 **Review**: N/A

PDF 5.24: END

PDF 5.25: POSITIONING

OVERVIEW:

PDF objects can be arranged in such a way to move content away from the visibility of the reader. This could happen when an object is placed outside of a page boundary, which is a similar risk as cropping presented earlier. It may also occur when objects overlap one another to obscure or hide contents. Images can also be placed in noisy areas and rendered such that they look like an image with poor quality.

CONCERNS:

The ability to position data and content out of the visibility of the reader, or human visibility, is a great concern for hiding data within a PDF file. Positioning data overlapping with other information may also hide data as well.

Boundary conditions in the position of any object may present the risk of a data attack if they are not checked properly. These values should be examined and checked carefully.

PRODUCT: PDF-1.0

LOCATION:

All graphic objects can be positioned freely throughout the document. For example, the Td operator on text objects places the text at predefined coordinates. In the example below, the y-coordinate places the string out of view of the reader, thus hiding data in the document.

EXAMPLE (OUT OF PAGE BOUNDARY):

```
5 0 obj
<<
    /Length 46
    %Tx=100 (valid), Ty=800 (invalid) puts the text out of view of the
    %reader/printer
>>
stream
BT
```

```
/F1 24 Tf
100 800 Td
1 Tr
(Hello World)Tj
ET
endstream
endobj
```

EXAMPLE (OVERLAPPING XOBJECTS):

```
%This XObject has been named Im1
31 0 obj
<<   /Type/XObject/Subtype/Image/Width 360
     /Height 360/ColorSpace/DeviceRGB
     /BitsPerComponent 8
     /Length 50000/Filter/FlateDecode>>
stream
…image data…
endstream
endobj

%This XObject has been named Im2, same size as Im1
33 0 obj
<<   /Type/XObject/Subtype/Image/Width 360
     /Height 360/ColorSpace/DeviceRGB
     /BitsPerComponent 8
     /Length 48564/Filter/FlateDecode>>
stream
…image data…
endstream
endobj

2 0 obj
<<
    /Length 204
>>
%Contains a text and two XObject that overlap
stream
BT
/DeviceRGB cs 0 0 0 scn /DeviceRGB CS 0 0 0 SCN /ColorName 10 Tf 174.319
     677.164 TD[(This is example text)]TJ
ET
q 360 0 0 360 142.56 297.39 cm /Im2 Do Q
q 360 0 0 360 46.5195 326.658 cm /Im1 Do Q
endstream
endobj
```

RECOMMENDATIONS:

1 **Validate**: Check the location of all image and text objects to ensure that they fit within the page boundaries and margins.

2 **Validate**: Ensure that the size of each object and the location does not overlap more than an allowable threshold with any other content.

3 **Remove**: Remove content that exceeds page margins, boundaries, or overlaps other content more than an allowable threshold.

4 **Replace**: N/A

5 **External Filtering Required**: Extract, decode, and pass objects to an external filter.

6 **Review**: Extract, decode, and pass objects for manual review.

PDF 5.25: END

PDF 5.26: MASKING

OVERVIEW:

In the ISO standard, there is a section covering Masked Images in 8.9.6. There are several types of masks that are defined in this section and how they are located in a PDF. The ISO defines stencil masking, explicit masking, color key masking, and soft masking.

There exists a `/ImageMask` entry in an Image XObject dictionary, which defines a stencil mask. A stencil mask is used to either mask out pixels in an image or allow the current color from the image to appear (just as a normal stencil is used). Stencil masks are used often in Type 3 fonts.

The `/Mask` entry in an Image XObject dictionary defines an explicit mask. An explicit mask indicates which areas of the image to paint and which to mask out [1]. An explicit mask and the original image are overlapped (they do not need to be the same dimensions) in the PDF. The mask indicates which parts are painted and which parts are left the same as the original image.

The `/Mask` entry may also be used for color key masking, which indicates a color range that will be masked out in any location in that image. The `/Mask` entry is an array of integers specifying the range of colors. The ISO standard indicates that it is similar to the chroma-key technique. The range of colors not in range will show through.

Soft masks are used in conjunction with transparency and is defined through the `/SMask` entry in an object, which was defined in PDF-1.4. There is a Soft-Mask dictionary defined in the graphics state dictionary. This dictionary defines the subtype (Alpha/Luminosity), which is the method to derive mask values. The dictionary defines the transparency group XObject as the source for the mask. Optionally, there is a PDF function which defines the transfer function that computes the mask values.

In PDF-1.5, the field `/SMaskInData` can also be used for soft masking in an image XObject dictionary for JPEG2000 images, encoded the `/JPXDecode` filter. This field is an integer that defines if the soft-mask information with the image is to be used.

CONCERNS:

Masking can introduce a data hiding risk or a data disclosure risk. Information can be hidden by introducing a stencil mask, an explicit mask, and even a color key mask to block out certain information. Sanitization requires examining the original content, masking content, and the final output of the process. As with any image object, there may also be a risk of data attack due to the unknown format of the image or mask in this case.

PRODUCT:

Stencil Mask: PDF-1.0
Explicit Mask: PDF-1.3
Color Key Mask: PDF-1.3
Soft Mask: PDF-1.4
Soft Mask (`/SMaskInData`): PDF-1.5

LOCATION:

Image masks can be identified in Image XObject dictionaries through the combination of `/ImageMask` (Boolean) and `/Mask` fields. The `/Mask` field is a stream object or an array of colors for a color key mask. Soft-masks are defined with the `/SMask` (stream object) and `/SMaskInData` (integer) fields.

Graphics State Parameter dictionaries can also define soft-mask information, as well as references to other transfer functions.

EXAMPLE:

RECOMMENDATIONS:

1 **Validate**: Validate the Image XObject that references mask information just as any other stream object.

2 **Validate**: Validate the color key range to ensure all values are within range.

3 **Remove**: Remove all masking information from the image dictionaries to reveal the original image. This may have an adverse effect on the image in the document.

4 **Replace**: Replace the mask data with content that displays the original image. This may have an adverse effect on the image in the document.

5 **External Filtering Required**: Pass all image data (for each layer) to an external filter.

6 **Review**: Have the original content manually reviewed, the mask content reviewed as a separate layer, and have the final output manually reviewed.

PDF 5.26: END

5.5 Optional Content

The contents of a PDF file can be layered such that certain content can be viewed or hidden on demand. This functionality is called Optional Content. It can be useful for engineering or architectural diagrams that contain multiple layers of drawings or blueprint information. PDF files that utilize the optional content feature are often created via external applications such as CAD programs that use the layering effect for representing data. Optional Content may also be useful for viewing content at different zoom levels or even languages because information can be selectively viewed.

In this section, there are many different features presented: Optional Content Groups (OCG), Optional Content Membership Dictionary (OCMD), Optional Content Usage Dictionaries, using Optional Content, and Optional Content Configuration. All of these objects contain references to each other and the object hierarchy is illustrated in Figure 5-4.

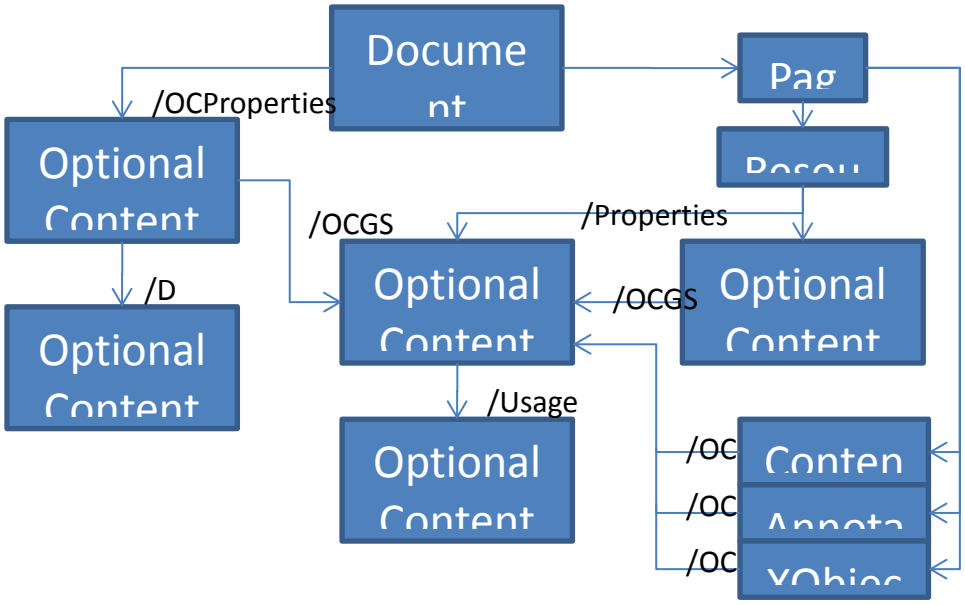

Figure 5-4. Optional Content Object Hierarchy

PDF 5.27: OPTIONAL CONTENT GROUPS

OVERVIEW:

Optional content is implemented such that groups of objects and graphics can be hidden or made visible based upon the user interaction with a PDF document. User interaction events include viewing, printing, and exporting the PDF document. Groups are represented through Optional Content Groups and Optional Content Membership Dictionary objects. Many objects in the PDF syntax such as content stream data, image XObjects, Form XObjects, and annotations can reference an optional content group as part of their dictionary to set their visibility behavior to a particular optional content group. The magnification and language can be used to change visibility or a group of related objects.

Optional Content Groups:

Optional Content Groups, or OCGs, define a group, through a group name text string. This dictionary contains an entry for Usage Dictionaries, which define when content appears to the user. Entries in the OCG dictionary are shown in Table 98 of the ISO standard. Content that belongs to the group reference the OCG in their respective dictionaries; this is defined in this section under "Using Optional Content."

Optional Content Membership Dictionary:

Optional Content Membership Dictionaries (OCMDs) are used to define more complex visibility policies [1]. In this case, content such as a content stream, XObject, or an annotation can declare itself to belong to an OCMD rather than an OCG. An OCMD can list OCGs which it can control the visibility. As opposed to an OCG dictionary, which defines a Usage Dictionary, the visibility policy of this group is defined by an entry `/P`, which is `/AllOn`, `/AnyOn`, `/AnyOff`, or `/AllOff`, as it relates to the list of OCGs. There is another method to control visibility of the

groups mentioned in this dictionary, which is to define a Visibility Expression (/VE) in the dictionary. This allows a Boolean expression to be defined, which computes the visibility based upon the states of the other OCGs defined in this membership dictionary.

Usage Dictionaries:
Usage Dictionaries are defined from the OCG dictionary. They are dictionaries themselves that define the nature of when the content controlled by the group is presented [1]. Usage dictionaries define the creator application and a language dictionary, which contains a text string indicating the language or locale. It also defines the ExportState, which is either ON or OFF, which defines the visibility when the document is exported to a format that does not implement optional content [1]. It defines a Zoom dictionary, which define the min and max values for magnification at which this content will be visible. It also defines a `/Print` entry which defines the state (ON or OFF) when the document is printed. The Usage dictionary defines a View entry which define the ViewState (ON or OFF), which defines if members of the group are visible when the document is simply viewed. A User dictionary is also an entry which can define the user that this content group is intended. This information defines the name, title, organization of a person.

Using Optional Content:
Optional content visibility groups can be applied to content streams, images and form XObjects, and annotations. The marked content feature in PDF allows these objects to reference optional content dictionaries through the field `/OC`. Examples in this section demonstrate how to reference the optional content group used for that particular PDF content.

Optional Content Properties:
Configuring Optional Content defines the default state and how events are used to change the states of OCGs. The document catalog defines an optional entry called `/OCProperties` which points to an Optional Content Properties dictionary. This dictionary defines indirect references to all the OCGs that exist in the document. It also defines configuration dictionaries, one is for the default configuration dictionary, and it also provides alternate configuration dictionaries. These configuration dictionaries are defined in the following paragraph.

Optional Content Configuration:
Configuration dictionaries define the initial state of OCGs when the document is being opened [1]. The default configuration dictionary defined by the `/D` entry in the Optional Content Properties dictionaries specifies this default behavior. There can also be alternate configuration dictionaries which are used in other cases.

Configuration dictionaries define the name and creator application of the configuration, and define the `/BaseState` (ON, OFF, or Unchanged), which is used to initialize the OCGs. There are also separate array entries for ON or OFF, which define which OCGs are ON or OFF initially. They also contain references to Usage Application Dictionaries, which define the Usage Dictionaries that shall be used. It also contains an entry called `/Locked`, which is an array that defines which OCGs are locked and cannot change under this configuration.

CONCERNS:

Optional content introduces a hidden data risk since actions may trigger different behavior in the viewing of content, and there may not always be a consistent view of the content. Some of the dictionaries described above discuss creator, language, and intended audience information which may be a source of a data disclosure risk.

PRODUCT: PDF-1.5

LOCATION:

The OCG and OCMD information, each separate dictionary objects, are linked from a Resources dictionary for the pages in the PDF document on which they are referenced from a page's content stream. In the case of XObject or Annotations, those OCGs are referenced directly there. From each OCG, a Usage Dictionary can be implemented to describe the visibility of objects in that group. The configuration information, or initial state information, is referenced from the document catalog and utilized when the document is opened. As mentioned before, Content Streams, XObjects, and Annotations can reference OCGs to be included in that group.

EXAMPLE:

```
%PDF-1.7
1 0 obj         %Object 1 0 is the Document Catalog, root of the document.
<<
    /Type /Catalog     %The type is catalog.
    /Pages 3 0 R       %It references a Pages object 3 0
    /OCProperties 30 0 R
>>
endobj          %End of Object 1 0
…
%Optional Content Properties Dictionary
30 0 obj
<<
    %Required to relate to OCGs in the file
    /OCGS [10 0 R 11 0 R]
    %Required for Optional Content Configuration Dictionary
    /D 40 0 R
    %Optional: array for alternate configuration dictionaries that could be
used.
    /Configs [41 0 R]
>>
endobj

%Optional Content Configuration Dictionary
40 0 obj
<<
    %Optional text name for this configuration
    /Name (Configuration for Hiding Data)
    %Optional. Metadata describing the application that created this
    %configuration.
    /Creator (ApplicationName)
    %Sets the initial and default visibility state of the OCGs
    /BaseState /ON
    %Arrays of references to OCGs and their visible with this configuration
    /ON [11 0 R]
    /OFF [10 0 R]
    %Usage Application Dictionaries Array.
    /AS [42 0 R]
    %OCGs that are locked and cannot be changed.
    /Locked [10 0 R]
>>
endobj

%This is the Optional Content Group (OCG) object
10 0 obj
<<
    % Required type as Optional Content Group
    /Type /OCG
    % Required Name of this content group
```

```
        /Name (HiddenText)
        %Optional Content Usage Dictionary.
        /Usage 50 0 R
    >>
    endobj

    %Usage Dictionary of OCG, indicates view state
    50 0 obj
    <<
        /View << /ViewSate /OFF >>
    >>
    endobj

    60 0 obj
    <<
        /Type /OCMD
        %Optional
        %Visibility of content in the OCGs is AllOn, so content associated with
    this
        %group is visible when both OCG 10 0 and 11 0 are also visible.
        /OCGs [10 0 R 11 0 R]
        /P /AllOn
    >>
    endobj

    %This is a Resources Dictionary for a page, other fields may be included
    %This is needed to reference an OCG by name in the content stream object.
    5 0 obj
    <<
      %Every field is optional, properties allow for the name /HiddenText point to
      %object 10 0
      /Properties << /HiddenText 10 0 R >>
    >>
    endobj

    %Stream data where it uses BDC and EMC to enclose the contents belong to Hidden
    %Text OCG, since HiddenText is defined in a Resources dictionary, this can be
    %used in a content stream by its name.
    6 0 obj
    <<
        /Length 86
    >>
    stream
    /OC /HiddenText BDC
        BT
            /F13 12 Tf
            288 720 Td
            (ABC) Tj
        ET
    EMC
    endstream
    endobj

    %XObject that assigns itself to an OCG
    8 0 obj
    <<
        /Type /XObject
        /Subtype /Image
        %Belongs to Optional content group 10 0
        /OC 10 0 R
    >>
    stream
```

```
<Xobject stream data>
endstream
endobj
```

RECOMMENDATIONS (OCG AND OCMD):

1 **Validate**: N/A

2 **Remove**: Optional Content Group dictionaries and Optional Content Membership Dictionary objects can be removed from the document. Usage directories referenced by the OCG shall be removed as well. References to those objects from other objects should be removed since they are no longer in use. This will have an adverse effect on the document.

3 **Replace**: The `/P` field should be set to `/AllOn` in OCMD dictionaries. The original intention of the document may be distorted.

4 **External Filtering Required**: Pass all text strings to an external filter.

5 **Review**: Pass each layer or member of the optional content for review.

RECOMMENDATIONS: (USING OPTIONAL CONTENT)

1 **Validate**: Check for consistency and that all objects that contain references to an OC object should ensure that it points to an approved optional content group where hidden data is not possible.

2 **Remove**: Remove all `/OC` fields from content streams, XObjects, and annotations.

3 **Replace**: N/A

4 **External Filtering Required**: N/A

5 **Review**: N/A

RECOMMENDATIONS: (OPTIONAL CONTENT PROPERTIES AND CONFIGURATION)

1 **Validate**: N/A

2 **Remove**: Remove the entry from the Document Catalog (`/OCProperties`), remove the Optional Content Properties dictionary, and remove the Optional Content Configuration dictionary, the Usage Application dictionaries since they change states of objects, along with the remainder of the optional content group information hierarchy presented in this section.

3 **Remove:** Remove the `/Locked` entry in the properties dictionary.

4 **Replace**: The `/BaseState` can be set to ON, however; this may change the original content with layering content.

5 **External Filtering Required**: Pass all text strings to an external filter.

6 **Review**: Pass each layer or object for manual review.

PDF 5.27: END

PDF 5.28: SET-OCG-STATE ACTIONS

OVERVIEW:

Set Optional Content Groups (OCG) state actions change the visibility of these groups, discussed earlier in PDF 5.27:. This action can specify On, Off, or Toggle actions on certain data to change its visibility. Certain objects in the PDF file belong to an Optional Content Group which has a dynamic impact on its visibility. This action can change that visibility to a different state than what it was originally defined.

CONCERNS:

This action creates a dynamic feature invoked by the PDF reader software that is a hidden data risk, since it can change the visibility of content. A data disclosure risk is also present because there is a concern for leaking sensitive data when data can be toggled or when the visibility is set to off.

PRODUCT: PDF-1.5

LOCATION:

The SetOCGState action is an object in the PDF file that defines several items in its dictionary. The type of action is always defined; along with the State field which is an array that indicates which objects in the PDF file are to be `ON`, which objects are `OFF`, and which objects are set to `TOGGLE`. These items are read from left to right in the array, this allows for objects to have their state changed when they are referenced multiple times in the array.

EXAMPLE:

```
1 0 obj
<<
     /S /SetOCGState
     %Turns off object 2 and 3
     %Toggles object 16 and 19
     %Turns on object 5
     /State [/OFF 2 0 R 3 0 R /Toggle 16 0 R 19 0 R /ON 5 0 R]
>>
endobj
```

RECOMMENDATIONS:

1 **Validate**: Check for consistency with action dictionary and also that all actions specify `ON`, and that the remainder of the dictionary contains the correct and required fields with correct data types of their values.

2 **Remove**: If validation fails because it is hiding data, remove the action object in its entirety and remove any references to that object throughout the document.

3 **Replace**: If validation fails because it is hiding data, force the action object to a default `ON` state such that data is not hidden or obscured. Therefore, `TOGGLE` and `OFF` entries should be replaced with `ON` entries, or added to the list of `ON` objects.

4 **External Filtering Required**: N/A

5 **Review**: N/A

PDF 5.28: END

5.6 Multimedia objects

PDF 5.29: SOUND AND MOVIE OBJECTS

OVERVIEW:

Sound and Movie objects are the original mechanisms for embedding multimedia in a PDF. These objects are used in conjunction with the Sound and Movie annotations. The use of these multimedia objects has been deprecated as of PDF 1.5 in favor of the new Renditions feature described below.

Sound objects are included in a PDF document's content through either sound actions or sound annotations. A sound annotation contains a `/Sound` entry which references a sound object. Similarly, a sound action references a sound object via its `/Sound` field in its dictionary. A sound object is a stream object containing the sound clip or sample, which is embedded within the file.

Movie objects are obsolete and are not recommended by the standard. Movies are embedded into a document through the use of a movie annotation, and a file specification is required for a movie object.

CONCERNS:

Multimedia objects are a concern because they contain information that could be used for a data attack (malformed file), particularly for movie files. In addition, movies and sounds may also contain hidden information embedded within the file format, and may include author or source information that could be subject to a data disclosure risk.

PRODUCT: PDF-1.2

LOCATION:

A sound object is a special case of an embedded stream with a number of additional, multimedia-specific characteristics. Sound object have two varieties that store the sound data in different ways. One possibility is for the sound object to directly include sound data in its stream data section.

Movie objects differ from sound objects in that they must use a file specification referenced from its `/F` entry to link to video data. Movie objects are display as part of a document's content when referenced by movie annotations. The movie annotations `/Movie` entry references a movie

object. Movie annotations also contain an optional /A field which references a movie activation dictionary. The configuration settings described in a movie activation dictionary allow a movie annotation to play only a selected portion of a video, while the remainder stays hidden from view.

EXAMPLE:

```
14 0 obj<<     /Type /Page     /Parent 9 0 R     /Contents 108 0 R
/MediaBox [ 0 0 612 792 ]     /CropBox [ 0 0 612 792 ]     /Rotate 0
/Annots 113 0 R >> endobj

%Referenced movie file111 0 obj<<     /Type /Filespec     /F (C:\media.mov)>>
endobj
%Movie annotation object
112 0 obj<<
    /Type /Annot     /Subtype /Movie     /Rect [ 75.62988 424.77484 325.2085
612.16888 ]     /T (media)    /F 1
    %References movie file spec    /Movie
    <<
        /F 111 0 R
        /Aspect [ 320 240 ]
    >>     /C [ 0.93724 0.12157 0.11372 ]     /Border [ 0 0 0 ]     /A <<
/ShowControls true /Mode /Open >> >> endobj
```

RECOMMENDATIONS:

1 **Validate**: Ensure referential integrity and that the sound or movie object is referenced through a valid annotation. Ensure consistency and validate the dictionary of each object.

2 **Validate**: Ensure consistency and validate the file specification if exists for either type of object.

3 **Remove**: Remove movie objects from the document and the annotation that included it. Further remove references to each within the file.

4 **Remove:** Remove sound objects from the document and the annotation that included it. Further remove references to each within the file.

5 **Replace**: Replace movie object with a flattened and harmless image.

6 **Replace:** Sounds objects can be replaced with text indicating this feature has been removed. This might be an issue for Section 508 compliance.

7 **External Filtering Required**: Extract the sound and movie files (if embedded) and have them reviewed by an external filter.

8 **External Filtering Required:** Extract path information for the file specification and pass to an external filter.

9 **Review**: Extract the file in its original format and manually inspect the contents by viewing or listening to the media.

PDF 5.29: END

PDF 5.30: SOUND ACTIONS

OVERVIEW:

A sound action is capable of playing a sound object, a sound object in a stream of data. Sound Objects were presented earlier in PDF 5.29:. A Sound Action object contains additional parameters that control the playback of the sound file.

CONCERNS:

Multimedia and sound clips are generally a concern for documents since they can provide a PDF document a place to hide data, and even mix data with other multimedia streams to produce obscured or hidden audio data. Due to the multimedia player, they may be a data attack risk (this is more so with Movie files).

PRODUCT: PDF-1.2

LOCATION:

A Sound action contains a dictionary including the type of action, and a sound stream containing the data. The dictionary of a Sound Action also contains the optional entries such as the volume of the sound clip, a synchronous flag to indicate whether or not any action can occur until the sound clip finishes, a repeat flag to indicate if the sound clip should loop, and a mix flag to indicate to mix the sound clip with other sound clips that are already playing.

EXAMPLE:

```
5 0 obj
<<
    /S /Sound
    /Sound 6 0 R %stream data
    /Volume /1.0
    /Synchronous true
    /Repeat true
    /Mix true
>>
endobj
```

RECOMMENDATIONS:

1 **Validate**: Ensure the Volume and the Mix fields within the Sound dictionary have approved values such that sound information is not distorted or obscured from the user.

2 **Validate**: Ensure referential integrity and that the Sound Action object references an approved and validated Sound Object.

3 **Remove:** Remove the sound action object in its entirety and remove all references to that object throughout the document. Sound annotations and sound dictionaries may be removed along with this process.

4 **Replace**: Replace each required field for the volume, synchronous, repeat, and mix settings to appropriate values.

5 **External Filtering Required**: N/A.

6 **Review:** N/A.

PDF 5.30: END

PDF 5.31: MOVIE ACTIONS

OVERVIEW:

A movie action can be used to dynamically start playing a movie file within the document based on some sort of user action. It can be in the form of a Movie Annotation that can be embedded within the document. Movie Objects were covered earlier in PDF 5.29:.

CONCERNS:

Similar to risks with Sound Actions, Sound Objects, and Movie Objects, this can introduce a data attack risk. Due to multimedia formats, a hidden data risk is also possible. Also, since file specifications are used by movie objects to identify the location of the file, this may also present a data disclosure risk.

PRODUCT: PDF-1.2

LOCATION:

Movie action objects can be located in the body of the PDF document and contain the `/S /Movie` field indicating a Movie Action. In addition, they can include an optional annotation if this is a reference to a Movie Annotation. It may also contain a text string that identifies the title of the movie annotation. The last optional entry in the Movie Action dictionary is the Operation, which may be stop, pause, resume, or play.

EXAMPLE:

```
5 0 obj
<<
    /S /Movie
    /Annotation 6 0 R
    /T (Movie Title)
    /Operation /Resume
>>
endobj
```

RECOMMENDATIONS:

1 **Validate**: Ensure consistency that the operation parameters has an approved value, along with Start and duration contain approved values.

2 **Remove**: Remove the action object in its entirety and remove any references to that object throughout the document. Objects associated with this movie action (annotation, embedded file) may already be selected for removal.

3 **Replace**: Replace operation parameters, start, and duration in the dictionary with approved parameters.

4 **External Filtering Required**: If present, pass free text string of the movie title for external filtering.

5 **Review**: N/A

PDF 5.31: END

PDF 5.32: RENDITION AND MEDIA CLIPS

OVERVIEW:

The PDF standard improved the multimedia functionality (sound and movie objects) with rendition objects and media clip objects. Rendition objects provide a tree structure in which Media Clip objects reside.

There are two types of Rendition objects: media rendition which is an object that specifies what media to play, how to play, and where to play it [1]. A second rendition is a selector rendition that contains an ordered list of selector renditions and media renditions which basically forms a tree structure with leaves of media rendition objects.

There are also two types of Media clip objects: Media Clip Data (MCD) and Media Clip Section (MCS). Media Clip Data provides a definition for the data of the media object that can be played in the document. Media Clip Section describes a continuous section or another media clip object, which may be either a MCD or a MCS. Essentially, media clips can be arranged through this manner in a linked list, provided that it terminates with an MCD object.

CONCERNS:

Multimedia objects represent a risk due to their ability to hide data. Media clip data objects can also point to external files or URL which can be from an un-trusted source. In addition, since they point to files or path information, this may result in a data disclosure risk. There may also be a data disclosure risk due to the media clip configured to only play segments of the multimedia. There may be a data attack risk associated with the multimedia player; however that is out of scope.

PRODUCT: PDF-1.5

LOCATION:

The Media rendition objects contain all the parameters for playing a media clip. A Media Rendition object is an object located in the body of the PDF file that contains references to the media clip to be played, along with viability parameters and other display settings. It contains the field `/S /MR` to define a media rendition. The optional field (`/C`) defines the media clip dictionary. If that is not present, the field `/P` must be defined which describes the media play parameters to indicate how the media shall be played. There is also an optional screen parameters dictionary that can be added to a media rendition object. A media rendition also specifies certain viability parameters through the entries `/MH` (must honor) and `/BE` (best effort).

A Selection Rendition object specifies an array or rendition objects through the field /R in its dictionary. It can reference other Selection Rendition objects or Media rendition objects in its array. The first viable media rendition found will be selected and played.

Media Clip Data objects define the actual data to be played within the PDF document. This is denoted by the /D field in the dictionary, which is either an embedded stream as an object within the PDF file or a file specification referencing the external media file or a URL.

A Media clip selection object defines the section of an existing MCD or MCS. Its dictionary references other MCD or MCS objects through the /D field. There are separate sub-dictionaries in the MCS that define the beginning (/B) and end (/E) of the offset into the media that it references.

EXAMPLE:

```
%Widget Annotation, invokes rendition action
103 0 obj
<<
    /F 4
    /H/P
    /Type/Annot
    /Parent 76 0 R
    /Rect[46.070038 784.728638 117.971008 810.139038]
    /BS <</W 1/S/B>>
    /Subtype/Widget
    /A 186 0 R %Rendition Action
    /P 99 0 R  %Parent
    /AP <</D 104 0 R/N 105 0 R>>
    /MK <</BG[0.847061 0.847061 0.847061]/BC[0.0 0.0 0.0]/CA(Play)/IF<<>>>>
>>
endobj

%Rendition action
186 0 obj
<<
    /R 187 0 R %Reference the rendition
    /S/Rendition
    /JS(\nthis.myPlayer.open\(\);\r)
>>endobj

%Media Rendition
187 0 obj
<<
    /C 188 0 R
    /N (Music)
    /S /MR
    /SP
    <<
        /BE << /W 2 /B[1.0 1.0 1.0] >>
    >>
>>endobj

%MCD
188 0 obj
<<
    /D 185 0 R
    /N(Media clip from JSBach_TwoPartInvention13.mov)
    /P<< /TF(TEMPACCESS) >>
    /S/MCD
    /CT(video/quicktime)
```

```
>>endobj

%File specification, points to embedded file in 81 0 obj (not shown)
185 0 obj
<<
    /F(<embedded file>)
    /EF<</F 81 0 R>>
    /Type/Filespec
>>endobj
```

RECOMMENDATIONS:

1 **Validate**: In general validation can be performed on each dictionary described in this section to ensure correct values and references to other objects.

2 **Remove**: Remove rendition objects, media clip objects, screen annotations, rendition actions, and any other reference to the removed media shall also be removed.

3 **Replace**: Replace media clips with video with a flattened image.

4 **External Filtering Required**: The embedded content containing the multimedia shall be passed to an external filter.

5 **Review**: Pass the multimedia file embedded within the document and have them reviewed manually.

PDF 5.32: END

PDF 5.33: RENDITION ACTIONS

OVERVIEW:

Rendition actions control the playback of multimedia objects that are located in the PDF document. The action is used by a screen annotation (PDF 5.45:) to play the rendition object. The screen annotation can indicate where the rendition is played on the page. In addition to playing multimedia content, an action could stop, pause, or resume the rendition object. It can also allow executing JavaScript that could perform a custom operation on the media control.

CONCERNS:

Due to its control over multimedia types, multimedia players may be subject to data attack risks. This action's potential execution of JavaScript raises more concerns for a data attack risk.

PRODUCT: PDF-1.5

LOCATION:

The Rendition Action is an object that exists in the body of the PDF file. It is defined through a dictionary that contains the Rendition type, and the operation `/OP`, identified by an integer, it intends to perform. If there is no operation defined, a section of JavaScript can be executed when the action is triggered. If both are specified, then the operation defined by `/OP` is a fallback if the JavaScript cannot be executed. In this case the operation is ignored. The operation requires two

additional parameters: a rendition object and a screen annotation object to be specified in the dictionary.

EXAMPLE:

```
%Rendition action
186 0 obj
<<
    /R 187 0 R
    /S/Rendition
    /JS(\nthis.myPlayer.open\(\);\r)
    %OP is not defined here.
>>endobj

%Rendition
187 0 obj
<<
    /C 188 0 R
    /N (Music)
    /S /MR
    /SP <<
        /BE << /W 2 /B[1.0 1.0 1.0] >>
    >>
>>endobj
```

RECOMMENDATIONS:

1 **Validate**: Validate the rendition action by checking for correct values of /OP and correct references for the rendition and annotation.

2 **Validate**: If JavaScript is defined, validate the JavaScript similar to JavaScript actions (PDF 5.60:).

3 **Remove**: Remove the action object in its entirety and remove all references to the object. Examine references in the object such as the annotation or rendition object and remove those as well.

4 **Remove**: Remove the JavaScript entry.

5 **Replace**: Replace the dictionary with playback modes that are approved and without JavaScript.

6 **External Filtering Required**: N/A

7 **Review**: N/A

PDF 5.33: END

PDF 5.34: 3D ARTWORK AND PRESENTATIONS

OVERVIEW:

The PDF file specification allows for embedded 3D artwork into the document. It is stored in a Universal 3D, (U3D) or a Product Representation Compact, (PRC) file format and rendered similarly as sound and movie objects. 3D artwork objects are shown through 3D annotation objects. In PDF 1.7, views can determine how 3D artwork is rendered without JavaScript.

CONCERNS:

3D artwork can be susceptible to hidden data since it is a rendered object, similar to sound and movie objects. 3D artwork also provides a portion of JavaScript code that can be executed when the 3D stream is read, which may provide a data attack risk. In general, the 3D artwork renderer may be subject to data attack; however this is out of scope. CVE-2009-2990 discusses a data attack vector that was used when parsing a U3D or PRC file. Data disclosure risks are possible since the representation is complex and may contain sensitive information.

PRODUCT: PDF-1.6

LOCATION:

A 3D object and annotation object exist in the body of the PDF file. A 3D annotation object is identified by the subtype field `/Subtype /3D`. The annotation simply references the 3D artwork stream object. The artwork stream object is identified by the `/Type /3D` and `/Subtype /U3D` entries in its dictionary. The 3D artwork stream object also contains a field called /OnInstantiate which specifies a JavaScript stream that executes when the artwork stream is read. The 3D artwork stream object also identifies various 3D view dictionaries that define different camera views. These views are often used in conjunction with Go-To 3D view actions (PDF 5.35:).

EXAMPLE:

```
%Annotation for 3D artwork.
1 0 obj
<<
    %Location of annotation
    /Rect[5.000000 14.000000 590.275574 590.275574]
    /Subtype /3D %3D annotation
    /F 68
    /Contents(3D Model)
    /P 4 0 R
    %Activation dictionary
    /3DA <<
        /DIS /I    %instantiate
        /A /PV     %activate as soon any part of page is visible.
    >>

    %References the 3D artwork stream object
    /3DD 2 0 R
    %
    /3DV 11 0 R
    /Type/Annot
    /AP <</N 13 0 R>>
>>endobj

%3D artwork stream object
2 0 obj
<<
    % The proceeding lines define this object as a 3D stream
    % Note: the only valid value for subtype is U3D.
    /Type /3D
    /Subtype /U3D
```

```
        % Length of artwork
        /Length 5000

        % Optional. A list of 3D View dictionaries
        /VA [ ... references to 3D View dictionaries ... ]

         % A JavaScript script that should be executed when the 3D stream is
         % read to create an instance of the artwork.
         /OnInstantiate 3 0 R %Indirect reference to JavaScript stream
    >>
    stream
    ... U3D DATA ...
     endstream
    endobj
```

RECOMMENDATIONS:

1 **Validate**: Ensure consistency and referential integrity and extract the annotation and stream dictionary and validate based upon the required fields and that the reference is correct. Validate the 3D artwork stream as any other data stream within the document.

2 **Remove**: 3D artwork object streams, 3D annotation objects, and JavaScript that is referenced only by that object should be removed entirely. References to the 3D artwork should be removed as well.

3 **Replace**: Replace with standard image, and remove the JavaScript associated with the 3D artwork.

4 **External Filtering Required**: Pass the 3D data to an external filter.

5 **Review**: Pass the 3D data for manual review.

PDF 5.34: END

PDF 5.35: GOTO-3D-VIEW ACTIONS

OVERVIEW:

This action specifies the view for a 3D annotation to use. The action targets an annotation and can change the view of how the 3D artwork appears to the user. A view object implements various parameters such as lightning, rendering, and projection, which define how the virtual camera is used to illustrate the 3D artwork.

CONCERNS:

This action could be a concern when 3D artwork is approved in the document. This action can modify the view such it presents a hidden data disk. This is aside from the risks presented with 3D artwork, which contains a data attack risk due to the inclusion of JavaScript in 3D artwork.

PRODUCT: PDF-1.6

LOCATION:

A GoTo3D View action is implemented in a dictionary that contains possibly four objects. The first is optional and defines the type of action, `/Type /Action`, the second is the subtype defined as `/S /GoTo3DView`. The third field indicates the target annotation is a dictionary type that references the annotation which view is changed. The final object defines the view information to define how the 3D artwork appears.

EXAMPLE:

```
5 0 obj
<<
    %Optional
    /Type /Action

    %Required, defines the action
    /S /GoTo3DView

    %Required, dictionary defining target annotation
    /TA
    <<
       %Target annotation
    >>
    % Required View information
    /V <view information, can be a dictionary, an integer, a text string, or a
        name>
>>
endobj
```

RECOMMENDATIONS:

1 **Validate**: Ensure consistency and validate the dictionary and target annotation for correct, required fields and their respective values.

2 **Validate**: Ensure referential integrity and that the 3D artwork is valid and has been approved.

3 **Remove**: Remove action object and references to the action object throughout the document.

4 **Replace**: N/A

5 **External Filtering Required**: N/A

6 **Review**: N/A

PDF 5.35: END

PDF 5.36: ALTERNATE PRESENTATIONS

OVERVIEW:

An alternative presentation allows a PDF document to be viewed in a different and alternative way. The only supported alternative presentation is a slideshow. JavaScript is used to initiate the slideshow through an interactive form. According to the standard, readers are not required to

implement this functionality, but must arrange the content in such a way that everything can be viewed in a static manner. This feature was deprecated in ISO 32000-1, and not implemented by any supported viewer.

The PDF standard provides another method to present a document as a slideshow that uses the actual page content. This is not considered an Alternate Presentation and is considered a completely different mechanism, but intends to provide a similar slideshow implementation. A page object may use the optional `/Dur` and `/Trans` entries to specify a Transition dictionary and a page's display duration, respectively. The Transition dictionary contains optional entries that define the style and duration of the visual transition to use when moving from another page to the given page during a presentation. In addition, a sub-navigation feature allows navigation not only between pages but between different states of the same page; this latter functionality is related to optional content.

CONCERNS:

Alternate presentations present a risk because the content in the images associated with the slideshow does not necessarily match that of the pages they represent and because they may be present but not displayed. If PDF files are not designed such that it can be viewed in a static manner, this can present a hidden data risk.

PRODUCT: PDF-1.4

(Only supported Alternate Presentation: Slideshow, PDF-1.5)

LOCATION:

An alternate presentation is specified by a name tree referenced by the optional `/AlternatePresentations` entry in the document's name dictionary; the name tree maps strings to the alternate presentations object, which is available for the document. The slideshow definition includes required `/Type /Slideshow` and `/Subtype /Embedded` entries, and also includes a required `/Resources` entry. All slideshow resources must be either image XObjects or embedded file streams.

EXAMPLE [1]:

```
1 0 obj
<<
    /Type /Catalog
    /Pages 2 0 R
    /Names 3 0 R % Indirect reference to name dictionary
>>
endobj

3 0 obj % The name dictionary
<<
    /AlternatePresentations 4 0 R
>>
endobj

4 0 obj % The alternate presentations name tree
<<
    /Names [(MySlideShow) 5 0 R]
>>
endobj

5 0 obj % The slideshow definition
<<
```

```
          /Type /SlideShow
          /Subtype /Embedded
          /Resources <</Names [ (mysvg.svg) 31 0R
                                (abc0001.jpg) 35 0 R
                                (abc0002.jpg) 36 0 R
                                (mysvg.js) 61 0 R
                                (mymusic.mp3) 65 0 R ]
                     >>
          /StartResource (mysvg.svg)
      >>
      endobj

      31 0 obj
      <<
          /Type /Filespec % The root object, which
          /F (mysvg.svg) % points to an embedded file stream
          /EF <</F 32 0 R>>
      >>
      endobj

      32 0 obj % The embedded file stream
      <<
          /Type /EmbeddedFile
          /Subtype /image#2Fsvg+xml
          /Length 72
      >>
      stream
      <?xml version="1.0" standalone="no"?>
      <svg><!-- Some SVG goes here--></svg>
      endstream
      endobj
```

RECOMMENDATIONS:

1 **Validate**: Ensure that Alternate Presentations is not implemented in the file as its functionality has been deprecated.

2 **Remove**: Remove the `/AlternatePresentations` entry in the Name dictionary. This means that each page will be presented statically. Remove objects or filestreams that are associated with this functionality and no longer in use in the document.

3 **Replace**: N/A

4 **External Filtering Required**: Pass all pathnames or strings to an external filter.

5 **Review**: N/A

PDF 5.36: END

PDF 5.37: RICH MEDIA

OVERVIEW:

Support was added to enable interaction with SWF files to provide rich media content to the PDF document with Adobe's Extensions to the ISO standard. Flash applications, video, audio, as well as other multimedia can be embedded into the PDF file. Similar to 3D artwork discussed earlier, a rich media annotation was added to support this functionality.

CONCERNS:

Rich media has a great concern for containing malicious content. SWF files and flash have been the source of numerous exploits and introduce a data attack risk. There are also numerous hidden data risks and data disclosure risks that are possible with any multimedia file type.

PRODUCT: PDF-1.7

LOCATION:

Rich media requires a rich media annotation to display the contents. A rich media annotation is identified by the `/Subtype /RichMedia`. A rich media annotation features a RichMediaSettings dictionary that is specific for each annotation, it defines the activation dictionary and deactivation dictionary. A rich media annotation also implements a RichMediaContent dictionary that can be shared and referenced among different rich media annotations. It provides the configuration and views of the content. The RichMediaConfiguration dictionary contains RichMediaInstances. Then the RichMediaInstances contain RichMediaParameters and Assets, which ultimately refer to the SWF file specification that is used in the annotation. Figure 5-5 shows a general hierarchy, there are more details in other dictionaries but this highlights the general structure.

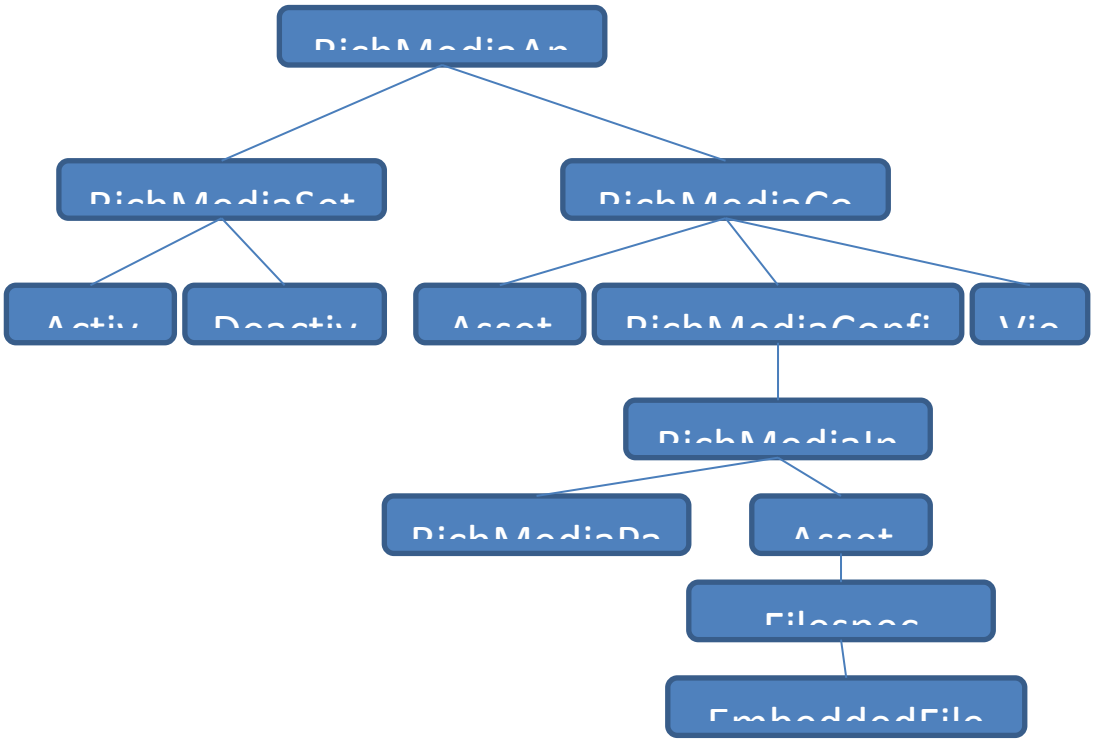

Figure 5-5. RichMedia Annotation Hierarchy

EXAMPLE:

```
% RichMedia annotation (root of the tree)
9 0 obj
 <<
```

```
        /Type /Annot
        /Subtype /RichMedia
        /NM (RichMedia001) % Annotation name
        /AP << /N 10 0 R >> % Appearance dictionary
        /F 68 % Annotation flags
        /P 5 0 R % Parent
        /Rect [ 50 50 742 500 ] % Rectangle
        /Border [ 0 0 0 ] % Border
        /BS % Border Style dictionary
        << /Type /Border
              /W 0 % Width (0 points)
              /S /S % Border style (Solid)
        >>
        %Next level down is RichMediaContent
        /RichMediaContent 12 0 R
        %Next level down is RichMediaSettings
        /RichMediaSettings 22 0 R
    >>
    endobj

    % RichMediaContent dictionary
    12 0 obj
    <<
        /Type /RichMediaContent
        /Configurations 13 0 R
        /Views 15 0 R
        /Assets 29 0 R
    >>
    endobj

    % RichMediaConfigurations array
    13 0 obj
    [ 14 0 R ]
    endobj

    % RichMediaConfiguration dictionary
    14 0 obj
    <<
        /Type /RichMediaConfiguration
        /Name (Configuration01)
        /Instances 15 0 R
    >>
    endobj

    % RichMediaInstances array
    15 0 obj
    [ 16 0 obj
    17 0 obj
    ]
    endobj

    % RichMediaInstance dictionary
    17 0 obj
     <<
        /Type /RichMediaInstance
        /Subtype /Flash
        /Asset 31 0 R
        /Params 18 0 R
    >>
    endobj

    % File specification dictionary for SWF file
```

```
31 0 obj
<<
    /Type /Filespec
    /F (Flash.swf)
    /UF (Flash.swf)
    /EF
    <<
        /F 41 0 R
    >> % Stream containing the SWF file
>>
endobj

% Embedded file stream for SWF file
41 0 obj
<<
     /Type /EmbeddedFile % Flash.swf
     /Length 5000 %bytes long
     /Filter /FlateDecode
>>
stream
...Data for Flash.swf...
endstream
endobj
```

RECOMMENDATIONS:

1 **Validate**: Ensure consistency and referential integrity and that dictionaries in the hierarchy of rich media can have their data types and values validated. Ensure that the embedded file stream can be validated just as any other data stream in the PDF document.

2 **Remove**: Remove the entire rich media object hierarchy in the document. This includes every object from the annotation down the file specification.

3 **Replace**: N/A

4 **External Filtering Required**: Pass the embedded SWF content to an external filter.

5 **External Filtering Required:** Pass the text of the file path or free text information for the file specification to an external filter.

6 **Review**: Pass the embedded SWF content for manual review.

PDF 5.37: END

PDF 5.38: RICHMEDIAEXECUTE ACTIONS

OVERVIEW:

A Rich Media Execution action is a command that is sent to the handler of a rich media annotation. These actions are directly linked with an annotation and implement a

RichMediaCommand dictionary which is ultimately executed. A RichMediaCommand is a simple ActionScript or a JavaScript function name.

CONCERNS:
In addition to the concerns created by the existence of rich media such as SWF content for an annotation, the interface to send commands to this object extends its functionality. This presents an additional data attack risk.

PRODUCT: PDF-1.7

LOCATION:
As discussed before, Rich Media has a hierarchy of objects and dictionaries that define its existence in the PDF, where it ultimately resides through an annotation. A RichMediaExecute Action is a separate object that references the annotation and also implements a RichMediaCommand dictionary.

A RichMediaExecute Action object is defined by the field `/S /RichMediaExecute`. Another required field is the `/TA` entry which defines the targeted annotation for the rich media command. An optional `/TI` entry can also define an indirect object reference to a RichMediaInstance dictionary associated with a current annotation.

The RichMediaCommand dictionary defines the object by the value `/Type /RichMediaCommand`. A required field the `/C` entry which is a text string that specifies the script command (ActionScript or JavaScript function name). An optional entry `/A` defines the arguments to the command which can be a single value, an array of values, a string, or Boolean value.

EXAMPLE:

```
9 0 obj % RichMedia annotation
<<
    /Type /Annot
    /Subtype /RichMedia
    /NM (RichMedia001) % Annotation name
    /AP << /N 10 0 R >> % Appearance dictionary
    /F 68 % Annotation flags
    /P 5 0 R % Parent
    /Rect [ 50 50 742 500 ] % Rectangle
    /Border [ 0 0 0 ] % Border
    /BS % Border Style dictionary
    <<
         /Type /Border
         /W 0 % Width (0 points)
         /S /S % Border style (Solid)
    >>
    /RichMediaContent 12 0 R
    /RichMediaSettings 22 0 R
>>
endobj

… %rest of RichMedia hierarchy

18 0 obj % RichMedia Params dictionary
<<
    /Type /RichMediaParams
    /Binding /Foreground
    /FlashVars (name=FirstName+LastName&address=1+Main+St.&city=Hometown)
    /CuePoints
```

```
          [
            << /Type /CuePoint
               /Name (Cue Point 1)
               /Subtype /Nav
               /Time 5100
            >>
            << /Type /CuePoint
               /Name (Cue Point 2)
               /Subtype /Event
               /Time 4020
               /A 19 0 R %Specifies JavaScript action to get called with flash
            >>
          ]
      >>
      endobj

      19 0 obj % JavaScript action dictionary
      <<
          /Type /Action
          /S /JavaScript
          /JS (app.alert\("Cue Point Reached"\);)
          /Next 20 0 R %Transition action to RichMediaExecute
      >>
      endobj

      20 0 obj % RichMediaExecute action
      <<
          /Type /Action
          /S /RichMediaExecute
          /TA 9 0 R
          /CMD
           <<
              /A (Argument String)
              /C (CommandName)
           >>
      >>
      endobj
```

RECOMMENDATIONS:

1 **Validate**: Check each object in the RichMedia action object and nested objects are implemented correctly.

2 **Remove**: Remove the action object in its entirety and any references to that object throughout the document.

3 **Replace**: Replace the action object with commands and argument that does not impact the rich media. This assumes the rich media is acceptable but the action associated with it is not approved.

4 **External Filtering Required**: Extract the rich media and actions associated with it and test its operating with the command and argument from the action object through an external filter.

5 **Review**: N/A

PDF 5.38: END

5.7 Interactive Objects

5.7.1 Annotations

Annotations allow a user to interact with the document in several ways. For example, a user can insert comments into a PDF document, which can be useful for document review purposes. An annotation can be in the form of a visible icon, such as a paperclip, caret, underline, or drawing object. An annotation with a visible icon is called a markup annotation. Many annotations have been discussed already with sound, movie, and rich media objects.

Annotations have the capability to contain hidden data. Sometimes the annotation is only revealed on action by the user such as the mouse hovering over a certain area to expose the text. An annotation can be rendered such that it is not visible to the user or it can be made illegible. It can also be deleted from the screen in the viewing application but the actual object may remain in the document. In this case, there is an object that is not referenced in the PDF document. Annotations that feature rich media content may also contain vulnerabilities in the software interpreting the file.

In a PDF document, annotations are represented by Annotation dictionaries. A page object will contain the optional `/Annots` entry to indicate the presence of an annotation on that page. This entry contains an array of Annotation dictionaries that represent all of the annotations associated with the particular page. An annotation dictionary can only be referenced from the `/Annots` entry of a single page. An Annotation dictionary is identified by the optional `/Type /Annot` entry and the required `/SubType` entry that specifies the annotation type. Also required, is a `/Rect` array which specifies the area of the annotation on the page. Annotations can have several different subtypes and they are shown in Table 5-3. The following sections discuss specific annotations that are not defined elsewhere in this report.

Table 5-3. PDF Annotations [1]

SubType Name	Description	Product	Markup	ISG Section
`/Text`	Markup annotation that displays a "sticky note" attached to a point in the document.	PDF-1.0	Yes	PDF 5.39:
`/Link`	A link to another location in the document, another document, or an action to be performed.	PDF-1.0	No	PDF 5.68:
`/FreeText`	Displays text directly in the document with no associated popup box.	PDF-1.3	Yes	PDF 5.39:
`/Line, /Square, /Circle, /Polygon, /PolyLine`	Markup annotations that display the associated graphic.	PDF-1.3 PDF-1.5 (Polygon, Polyline)	Yes	PDF 5.40:
`/Highlight, /Underline, /Squiggley, /Strikeout`	Text markup annotations that display the associated markup.	PDF-1.3 PDF-1.4 (Squiggly)	Yes	PDF 5.41:

/Stamp	Markup annotation that displays text or graphics.	PDF-1.3	Yes	PDF 5.42:
/Caret	Markup annotation that displays a symbol indicating the presence of text edits.	PDF-1.5	Yes	PDF 5.42:
/Ink	Markup annotation that displays as freehand "scribble".	PDF-1.3	Yes	PDF 5.42:
/Popup	Object associated with all markup annotations.	PDF-1.3	No	PDF 5.43:
/FileAttachment	Markup annotation that contains to a reference to a file, typically embedded in the PDF document.	PDF-1.3	Yes	PDF 5.44:
/Sound, /Movie	Annotations that contain sound or animated graphics and sound.	PDF-1.2	Yes–Sound No-Movie	PDF 5.29:
/Widget	Used with interactive forms to represent the appearance of fields and manage user interactions.	PDF-1.2	No	PDF 5.63:
/Screen	Specifies a region of a page upon which media clips may be played. It also serves as an object from which actions can be triggered.	PDF-1.5	No	PDF 5.45:
/PrinterMark	Specifies graphic symbols or text added to a page to assist in printed document production. Its dictionaries can include text strings that may include sensitive data that is not human-visible.	PDF-1.4	No	PDF 5.46:
/TrapNet	Defines the trapping characteristics for a page of a PDF document.	PDF-1.3	No	N/A
/WaterMark	Used to represent graphics that are expected to be printed at a fixed size and position on a page, regardless of the dimensions of the printed page.	PDF-1.6	No	PDF 5.47:
/3D	Used to represent 3D artwork in a PDF document; the artwork is contained in one or more streams.	PDF1.6	No	PDF 5.34:
/Redact	Used to represent data that is to be removed from the document.	PDF-1.7	Yes	PDF 5.48:
/RichMedia	Used to represent a RichMedia object such as playing a SWF file. Flash applications, video, audio, can be used in this annotation.	PDF-1.7	No	PDF 5.37:
/Projection	Used to provide a way to save 3D and other measurements and comments as markup annotation, which then persist in the document.	PDF-1.7	Yes	PDF 5.49:

PDF 5.39: TEXT AND FREETEXT ANNOTATIONS

OVERVIEW:
A text annotation is a "sticky note", which is attached to the PDF document at a specific location in the document. It can be opened and closed by the user when the document is viewed. When opened, it displays text, and when closed, it displays a small icon which simply indicates its presence.

Freetext annotations are different because they have no open or closed state; therefore, the text is always visible. The text can be manipulated and contain references to Rich Text discussed in PDF 5.51:, which require sanitization.

CONCERNS:
Text annotations are a concern for hidden data since their visibility can be controlled by the user. When the annotation is closed, the text is hidden. This could also be a data disclosure risk if comments are left on the page.

Freetext annotations are a concern for hidden data if their content may overlap other content on the page. This is possible with many objects in PDF and not specific to freetext annotations. These annotations are not as risky as Text Annotations since there is no closed state, but just as any content that can be placed on the page.

PRODUCT:
Text – PDF-1.0
FreeText – PDF-1.3

LOCATION:
Annotations are located within their own dictionaries and references by a Page Dictionary's `/Annots` array, which defines the annotations for that page. Each annotation specifies the `/SubType`, in this case `/Text` or `/FreeText`, and its location on the page. In the simple Text annotation a string is in the annotation dictionary to represent the text content of the annotation. In the FreeText annotation, there is a reference to a text string or text stream for Rich Text which is used for the defining the appearance of the string.

EXAMPLE:

```
%Text annotation
2 0 obj
<< /Type /Annot
   /Subtype /Text
   /Rect [266 116 430 204]
   /Contents (Here is the text that will be shown.)
   /T (This is the title of the annotation.)
   /Subj (This is the subject text string.)

>>
endobj

%Freetext annotation
3 0 obj
<<
    /Type /Annot
    /Subtype /FreeText
    %Appearance string DA
    /DA (/F1 10 Tf 0 g)  %Font name = F1, size = 10
    /Q 0 %Left justified
    /DS (font: 18pt Arial)
    /RC ((<?xml version="1.0"?><body xmlns="http://www.w3.org/1999/xtml"
          xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/"
```

```
              xfa:contentType="text/html" xfa:APIVersion="Acrobat:8.0.0"
              xfa:spec="2.4">
              <p style="text-align:left">
              <b>
              <i>
             Here is some bold italic text
             </i>
             </b>
             </p>
             <p style= "font-size:16pt">
             This text uses default text state parameters but changes the font size
             to 16.
             </p>
             </body>)
    %Rich text string example from ISO 32000-1:2008 example [1].
    >>
    endobj
```

RECOMMENDATIONS:

1 **Validate**: Check the boundaries and location of the annotation and that it does not overlap existing content.

2 **Validate**: Check that the Rich Text String entries are validated.

3 **Remove**: Remove annotation and reference to annotation in the Page dictionary.

4 **Replace**: Replace Text annotations with FreeText annotations that are always visible.

5 **External Filtering Required**: Pass all text strings to an external filter.

6 **Review**: N/A

PDF 5.39: END

PDF 5.40: LINE MARKUP ANNOTATIONS

OVERVIEW:

A Markup annotation is a generic term for an annotation as Text and FreeText are technically considered a markup annotation. This section covers the line markup annotations, and other drawing type annotations such as square, circle, polygon, and polyline annotations.

A line annotation displays a straight line on the page. Like Text annotations, Line Annotations can be opened and closed. When opened, a pop-up window is display containing text. When closed, the line remains in the document and can be opened when a user clicks on the line.

Square and circle annotations draw a rectangle or ellipse on the page similar to the line annotation. When opened, a similar pop-up window is displayed showing the text of the annotation.

Polygon annotations draw a closed polygon on the page which can be clicked on to open the annotation text pop-up window. Polyline annotations are similar, but since it is not a closed polygon, the first and last vertexes are not connected [1].

CONCERNS:

Similar to text annotations, line markup annotations and the other annotations presented in this section represent both a hidden data risk, because annotation text can be hidden unless the user clicks on the annotation, and a data disclosure risk, if the user creates the PDF file inadvertently leaving annotation text in the document.

PRODUCT: PDF-1.0

LOCATION:

Annotations in this section are implemented in their own dictionary and referenced by the `/Annots` array in the Page dictionary.

Line annotations contain information specifying the coordinates for starting and ending the line. It also contains information about the line ending styles and border appearances.

Square and circle annotation dictionaries implement a unique Subtype, and a border style dictionary. It also implements an array entry to designate the color of the interior space of the shape. Optional information can also be included to describe the border effect of the annotation.

Polygon and polyline annotation dictionaries provide a unique Subtype name, and an array of vertices outlining the polygon or polyline (in the polyline the first and last vertices are not connected). A similar Border style dictionary is also optional in this dictionary along with the interior color and border effect information.

EXAMPLE:

```
1 0 obj
<<
    /Contents(Text)
    /Subtype/Line
    /P 4 0 R
    /L[200 300 400 600]
    /Rect[200 300 400 600]
    /T(This is the title of the annotation)
>>
endobj

2 0 obj
<<
  /Contents(Text)
  /Subtype/Square
  /P 4 0 R
  /Rect[200 300 400 600]
  /T(This is the title of the annotation)
>>
endobj
```

RECOMMENDATIONS:

1 **Validate**: Check that the boundary of each type of annotation, and each vertex for the polygon/polyline, falls within the page and does not overlap existing content.

2 **Remove**: Remove the annotation dictionary and references to it in the Page dictionary.

3 **Replace**: N/A

4 **External Filtering Required**: Pass all text string information to an external filter.

5 **Review**: N/A

PDF 5.40: END

PDF 5.41: TEXT MARKUP ANNOTATIONS

OVERVIEW:

Text markup annotations such as highlights, underlines, strikeouts, and squiggly appear in the text of a document. Similar to other markup annotation, they can have an open and closed state. When opened, a pop-up window contains free text that is the contents of the annotation. When closed, the annotation simply appears on the page according to the shape or appearance as the annotation name suggests.

CONCERNS:

As with any markup annotation, the fact that it can be in an open and closed state introduces a data hiding risk, and possibly a data disclosure risk. Since its coordinates can be defined in the annotation, it needs to be visible on the page.

PRODUCT:

Highlight – PDF-1.3
Underline – PDF-1.3
Strikeout – PDF-1.3
Squiggly – PDF-1.4

LOCATION:

Text markup annotations are defined in the Page dictionary as an annotation that exists on that page. Besides the normal required entries for all annotations, text markup annotations introduces a required entry for `/Subtype (/Highlight, /Underline, /Strikeout, or /Squiggly)`. The text markup annotation second required field is an array called `/QuadPoints` which is array of 8xn numbers which specify the coordinates of the quadrilaterals, in user space, which is the location of the annotation. It shall cover a word or group of words in the text. Also, it must also be considered that it falls within the page boundaries.

EXAMPLE:

```
%Strikeout annotation, contains link to popup annotation
1237 0 obj
<<
    /Type/Annot
    /Subtype/StrikeOut
    /P 1 0 R
    /M(date)
    /F 4
```

```
        /Rect[ 77.1055 614.415 172.372 643.659]
        /CreationDate(D:20101103104800-05'00')
        /T(title)
        /Subj(Strikeout)
        /NM(0884f2a2-e40c-4066-b1e2-287cff302b89)
        /AP<</N 1238 0 R >>
        /Popup 1239 0 R
        /QuadPoints[ 77.1055 643.659 172.372 643.659 77.1055 614.415 172.372
    614.415]
        /C[ 1 0 0]
        /CA 0.8
        /Border[ 0 0 1]
    >>
    endobj

    %Popup annotation
    1239 0 obj
    <<
        /Type/Annot
        /Subtype/Popup
        /P 1 0 R
        /M(date)
        /F 28
        /Rect[ 520.77 523.714 700.77 643.714]
        /Open false   %Hidden
        /Parent 1237 0 R
    >>
    endobj
```

RECOMMENDATIONS:

1 **Validate**: Check that the annotation exists on the page and that it covers text content.

2 **Remove**: Remove the annotation dictionary, and if possible orphaned objects such as pop-up annotations or Form XObjects, as well as remove references to the annotation in the Page dictionary.

3 **Replace**: N/A

4 **External Filtering Required**: Pass all text strings to an external filter.

5 **Review**: N/A

PDF 5.41: END

PDF 5.42: GRAPHICAL MARKUP ANNOTATIONS

OVERVIEW:

There are several markup annotations that contain visual symbols such as a caret, a stamp, and an ink annotation.

A caret annotation is a symbol on the PDF page that means there is a text edit at that location.

A stamp annotation places a stamp on the page just as if it were actually stamped on paper. This annotation can be opened to display a pop-up window containing text.

An ink annotation is scribble, or freehand text. When the annotation is in an open state, a pop-up window is displayed that contains text.

CONCERNS:

Each of these annotations can potentially introduce a hidden data risk, when the annotation is closed and the note is not visible. In addition, the graphical nature of these annotations may overlap other existing content as well. Data disclosure risks are also possible in markup annotations where comments are left in the document.

PRODUCT:

Caret – PDF-1.5
Stamp – PDF-1.3
Ink – PDF-1.3

LOCATION:

These annotations are located as separate objects, initially linked from the Page object in the `/Annots` array. Each annotation has a required `/Subtype`, which is either `/Caret`, `/Stamp`, or `/Ink`.

Caret annotations provide an entry called `/RD` which is a rectangle that describes the differences between the `/Rect` array that is required in all annotations, and the actual caret graphical object itself.

A stamp annotation provides a `/Name` entry which is the name of the icon that is used. This is a predefined name which is listed in Table 181 of the ISO standard, or it could be a custom name.

An ink annotation provides an entry called `/InkList` which is an array of arrays that represents the path in user space that the scribble exists. It also provides a border style dictionary identifying the line characteristics.

EXAMPLE:

```
1 0 obj
<<
    /Type /Annot
    /Subtype /Caret
    /Rect [10 20 30 40]
    /Contents (Annotation text.)
    /RD [ ] %Differences between actual caret and /Rect
>>
endobj

2 0 obj
<<
    /Type /Annot
    /Subtype /Stamp
    /Rect [ 10 20 30 40]
    /Name /NotApproved
    /Contents (Annotation text.)
>>
endobj
```

```
3 0 obj
<<
    /Type /Annot
    /Subtype /Ink
    /Rect [10 20 30 40]
    /InkList []    %Array of paths
    /Contents (Annotation text.)
>>
endobj
```

RECOMMENDATIONS:

1 **Validate**: Check that the rectangle and annotation space does not obscure other existing content.

2 **Remove**: Remove the annotation and any other pop-up that may reference this annotation.

3 **Replace**: N/A

4 **External Filtering Required**: Pass all text strings and content to an external filter.

5 **Review**: N/A

PDF 5.42: END

PDF 5.43: POP-UP ANNOTATIONS

OVERVIEW:

A pop-up annotation displays a pop-up window which contains text in a window that can be used for entry and editing. A pop-up annotation is the child of a markup annotation, which is considered its parent annotation. Many of the markup annotations can have a `/Popup` entry which points to this type of annotation.

CONCERNS:

Pop-up windows can be a concern for hidden data and data disclosure risk since they can be open or closed and contain text information.

PRODUCT: PDF-1.7

LOCATION:

A pop-up annotation dictionary contains a `/Subtype /Popup` entry to identify the annotation, and an indirect reference to its parent annotation. The reference to the parent annotation can be optional. If this is present, the pop-up annotation can declare its own `/Contents`, `/M`, `/C`, and `/T` entries to override the annotation. These values should be inspected when examining this annotation. The annotation dictionary also implements a Boolean entry called `/Open` which specifies if the annotation is initially open and displayed.

EXAMPLE:

```
1 0 obj
<<
    /Type /Annot
```

```
        /Subtype /Popup
        /Parent 2 0 R
        /Open false
        /Contents (This is the contents.)
        /T (This is the title.)
        /Rect [w x y z ]  %Coordinates in user space, w,x,y,z would be correct
    values
        /M (modification date)
    >>
    endobj

    2 0 obj
    <<
        %Parent annotation
    >>
    endobj
```

RECOMMENDATIONS:

1 **Validate**: Check that, if implemented, that it references a valid parent annotation.

2 **Remove**: Remove the pop-up annotation from the document and the Page information.

3 **Replace**: Replace open entry with a true value to ensure visibility of the pop-up window.

4 **External Filtering Required**: Pass all text strings in the annotation to an external filter.

5 **Review**: N/A

PDF 5.43: END

PDF 5.44: FILE ATTACHMENT ANNOTATIONS

OVERVIEW:

File attachment annotations simply appear to be an icon on the page that references a file. The standard indicates that this typically shall be an embedded file, in an embedded file stream (PDF 5.4:). The contents entry in the dictionary, as present in other annotations, is used for the description of the file.

CONCERNS:

The text in the annotation represents a freetext description of the file which may be subject to hidden data risk or data disclosure risk. The reference to a file specification also introduces concerns addressed with file streams addressed in PDF 5.4: and PDF 5.5:.

PRODUCT: PDF 1.0

LOCATION:

The file attachment annotation dictionary implements the other entries as other annotations like `/Contents`, but also provides a few new entries. A unique Subtype of FileAttachment is used to denote this specific type of annotation. Another field, `/FS`, is used to reference a file specification object. Lastly, a `/Name` field is used to display the icon on the page, which can be either `/GraphPushPin, /PaperclipTag, and /PushPin`.

EXAMPLE [9]:

```
6813 0 obj
<<
    /CreationDate(D:20070608165554-04'00')
    /Rect[492.975 544.436 506.975 564.436]
    /NM(b97eaf70-014a-4e48-a5a4-1d367fdb27c5)
    /Subtype/FileAttachment
    /C[0.25 0.333328 1.0]
    /F 156
    /Contents(config.rng)/M(D:20070608170351-04'00')
    /P 6810 0 R
    /T(Adobe Systems Inc., iComm)
    /Subj(File Attachment)
    /RC(<?xml version="1.0"?><body xmlns="http://www.w3.org/1999/xhtml"
         xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/"
         xfa:APIVersion="Acrobat:7.0.8" xfa:spec="2.0.2" >
     <p>config.rng</p></body>)
    /FS 6827 0 R
    /Type/Annot
    /AP <</D 8593 0 R/N 8594 0 R>>
>>endobj
%Embedded file
6827 0 obj
<<
    /F(config.rng)
    /EF<</F 8595 0 R>>
    /Type/Filespec
>>endobj
%Stream object with file data
8595 0 obj
<<
    /Length 674
    /Filter/FlateDecode
    /DL 1411
    /Params<<
              /CreationDate(D:20060616150420-04'00')
              /ModDate(D:20061201165920-04'00')
              /Size 1411
              /CheckSum(checksumstring)
            >>
>>
stream
…stream data…
endstream
endobj
```

RECOMMENDATIONS:

1 **Validate**: Check that the annotation references a valid file and contains a valid name for the icon.

2 **Validate**: Check the location of the annotation does not overlap existing content.

3 **Remove**: Remove the file attachment annotation from the document and references to that object from the Page dictionary. If the file specification becomes orphaned, remove this object as well.

4 **Replace**: N/A

5 **External Filtering Required**: Pass contents text and any other descriptive text in the annotation to an external filter.

6 **Review**: N/A

PDF 5.44: END

PDF 5.45: SCREEN ANNOTATIONS

OVERVIEW:

Screen annotations indicate an area of the page where media clips can be played. Screen annotations also allow a trigger for an action, through an action entry, which invokes an action when the annotation is activated, and an additional action field which triggers events based upon certain actions with the annotation. In the Rendition Actions section, there is mention of a Screen Annotation (PDF 5.33:).

CONCERNS:

Screen annotations can be a concern because they reference action objects and can serve as a trigger, which is a data attack risk, especially if the annotation is not used for its intended purposes to play renditions. Since it is an annotation can be located on any region of the page, it could be a hidden data risk if its visibility is obscured.

PRODUCT: PDF 1.5

LOCATION:

Screen annotations are referenced through the Pages dictionary, which is used to identify which annotations exist on which page. The screen annotation is not a markup annotation, but has common entries with other annotations. New entries such as a unique `/Subtype /Screen` is used, and a title of the annotation is specified by the `/T` field, which is a text string.

EXAMPLE (ADOBE MUSICAL SCORE.PDF):

```
101 0 obj<</F 6/Type/Annot/Rect[498.316437 702.674866 549.301331
735.843201]/Border[0 0 1]/BS<</W 1/Type/Border/S/S>>/Subtype/Screen/A 225 0 R/P
99 0 R/T(Music)/AP<</N 102 0 R>>/MK<<>>>>endobj


225 0 obj<</R 193 0 R/S/Rendition/OP 0/AN 101 0 R>>endobj

193 0 obj<</C 194 0 R/N(Music)/S/MR/SP<</BE<</W 2/B[1.0 1.0 1.0]>>>>>>endobj

194 0 obj<</D 195 0 R/N(Media clip from
JSBach_TwoPartInvention13.mov)/P<</TF(TEMPACCESS)>>/S/MCD/CT(video/quicktime)>>
endobj

195 0 obj<</F(<embedded file>)/EF<</F 80 0 R>>/Type/Filespec>>endobj
```

RECOMMENDATIONS:

1 **Validate**: Check that the screen annotation references a valid rendition action.

2 **Remove**: Remove the action related entries in the screen annotation dictionary.

3 **Remove**: Remove the screen annotation and objects that become orphaned upon its removal. Remove the annotation references in the document, in the Pages dictionary.

4 **Replace**: N/A

5 **External Filtering Required**: Pass text strings in the annotation to an external filter.

6 **Review**: Manually review the location and playback of the rendition.

PDF 5.45: END

PDF 5.46: PRINTER'S MARKS ANNOTATIONS

OVERVIEW:

Printer marks are graphics or text that is added to a page to help with people who use a multi-step process to complete a finished printed product. Example from the standard are registration targets for aligning plates and drawing cut marks where the final output medium is to be trimmed. A printer mark is not included in the actual page content. Printer marks usually are located outside of page boundaries since they are not in the content of the document.

Although listed in the standard, Printer's Marks annotations have never been used by any product.

CONCERNS:

Since this information contains graphics or text that typically are outside of the page boundaries and are not associated with the actual content, this could be considered a hidden data risk. If the information was meaningful but remained in the document, it may also be considered a data disclosure risk. Depending on the location of the mark, boundaries should be carefully monitored to ensure that it does not lead to a data attack risk.

PRODUCT: PDF-1.4

LOCATION:

A Printer's Mark annotation includes the typical annotation fields, but also introduces two more. The `/Subtype /PrinterMark` is required to identify the annotation. The other field is called `/MN` which supplies a Name as an argument, which identifies the type of printer's mark. The standard lists examples such as `ColorBar` or `RegistrationTarget`.

If the mark is a Form XObject (PDF 5.17:), then the Form XObject form type dictionary can include additional entries. These entries include a `/MarkStyle`, which is a text string denoting the printer's mark in a readable format. It also implements a `/Colorants` dictionary which provide color space information for the Form XObject.

EXAMPLE:

```
1 0 obj
<<
```

```
        /Type /Annot
        /Subtype /PrinterMark
        /Rect [ ] %location of content
        /Contents (PrinterMark Contents)
        /MN /RegistrationTarget
        /AP 2 0 R
    >>
    endobj
```

RECOMMENDATIONS:

1 **Validate**: Check that Printer's Marks annotations are not present in the document, they are not supported.

2 **Remove**: Removal all printer's marks annotations from the document and references to that annotation, as well as orphaned objects that remain due to its removal (Form XObjects).

3 **Replace**: N/A

4 **External Filtering Required**: Pass all text string information in the annotation hierarchy to an external filter.

5 **Review**: Review the printer's mark annotations outside the page content.

PDF 5.46: END

PDF 5.47: WATERMARK ANNOTATIONS

OVERVIEW:

A watermark annotation displays graphics at a fixed size and location on the page. Watermarks are not markup annotation and have no pop-up window. They are not interactive and simply display a watermark on the screen. A watermark annotation uses an appearance stream (PDF 5.50:) for the content of the watermark. Then it also provides a fixed print dictionary that indicates the size and position of the annotation on the page.

CONCERNS:

Watermark annotations may introduce a hidden data risk if the fixed print dictionary and appearance stream prevent the information from being visibly displayed on the page. Although the point of the FixedPrint dictionary is to ensure that the annotation is on the page, it should be checked that it is visible. And if the annotation is visible within the document, the content itself should be checked for a data disclosure risk.

PRODUCT: PDF 1.6

LOCATION:

A watermark annotation implements a dictionary that contains a `/Subtype /Watermark` and a `/FixedPrint` reference to a dictionary, which identifies how the annotation is drawn to the page.

The Fixed Print dictionary contains a `/Type /FixedPrint` to identify the object. There is a `/Matrix` array used to transform the annotation's rectangle array. There is also a `/H` field that translates the content horizontally and a `/V` field which translates the content vertically. The translation is a percentage of the width or height of the media.

EXAMPLE:

```
%Watermark annotation
9 0 obj
<<
    /Rect [10 20 30 40]
    /Type /Annot
    /Subtype /Watermark
    /FixedPrint 10 0 R
    /AP <</N 8 0 R>>
>>
endobj

%Form XObject used in Appearance Stream for Watermark contents.
8 0 obj
<<
    /Length 48
    /Subtype /Form
    /Resources …
    /BBox …
>>
stream
BT
   /F1 1 Tf
    36 0 0 36 0 -36 Tm
    (Watermark) Tx
ET
endstream
endobj

%Fixed Print Dictionary
10 0 obj
<<
    /Type /FixedPrint
    /Matrix [1 0 0 1 72 -72]
    /H 0
    /V 1.0
>>
endobj
```

RECOMMENDATIONS:

1 **Validate**: Check each matrix and transformation information to ensure its appearance is visible on the page.

2 **Validate**: Check that it references a valid appearance stream that has been validated.

3 **Remove**: Remove the watermark annotation and other orphaned objects as a result. This will have a negative impact on the document since that information is removed and the watermark will not be displayed.

4 **Replace**: Convert it to normal page content.

5 **External Filtering Required**: Pass all string fields in the annotation to an external filter.

6 **Review**: Review the watermark appearance on the page with the remainder of the content.

PDF 5.47: END

PDF 5.48: REDACTION ANNOTATIONS

OVERVIEW:

Redaction annotations denote content that should be removed, or is intended to be removed from the document. The annotation was created to allow for content to be identified in the document and after that has been done, the actual act of redaction removes the content and replaces with a marking to indicate removal. Once removed, the actual redaction annotation itself is removed from the document. This is often used in review processes where information in the document is to be removed.

Reader software is responsible for implementing how the content can be removed. The standard indicates that clipping or image masks cannot be used to hide data and that all traces of the content shall be removed.

CONCERNS:

Redaction annotations are a concern since they provide a user a mechanism to modify the document and possibly remove content. Although the removal process should truly remove the content in its entirety from the document, there are additional fields that contain annotation text and information which could be a hidden data risk or a data disclosure risk.

PRODUCT: PDF-1.7

LOCATION:

A redaction annotation introduces new entries including a `/Subtype /Redact` for identification. There is an additional optional field called `/QuadPoints` which is an array of coordinates in user space that covers the region of the removable content. It's optional, since the `/Rect` entry that is required for all annotations can be used to denote the region. There is also an `/IC` array which indicates an array of three color values (ranging 0.0 to 1.0) for red, green, and blue. If the color value is not present, then a Form XObject (PDF 5.17:) can be specified in the `/RO` entry of the redaction annotation dictionary. This Form XObject is replaced when the previous content has been removed. An optional `/OverlayText` can specified the text string that covers the region once the content is removed, this is not used when /RO form XObject is specified. There is a `/Repeat` boolean value that if true takes the OverlayText value and repeatedly writes to fill the redacted region. If the `/OverlayText` field is present, then a `/DA` byte string must be supplied which specifies the formatting of the variable text in `/OverlayText`.

EXAMPLE:

```
1 0 obj
<<
    /Type /Annot
    /Subtype /Redact
    %w, x, y, and z are coordinates in user space
    /Rect [w x y z ] %Coordinates of annotation
```

```
        /QuadPoints [w x y z ] %Coordinates covering the region of removable
    content.
        %This form XObject replaces the content once it has been removed
        /RO 2 0 R

        %Optional entries if Form XObject (RO) is not used
        %/IC [0.5 0.5 0.5]
        %/OverlayText (Content has been removed)
        %/Repeat true
        %/DA (/Font1 10 Tf 0 g)
    >>
    endobj

    2 0 obj
    <<
        /Type /XObject
        /Subtype /Form
        %image data
    >>
    stream
    …Xobject data…
    endstream
    endobj
```

RECOMMENDATIONS:

1 **Validate**: Check that the annotation dictionary is implemented correctly and covers an existing amount of content (annotation is used properly).

2 **Remove**: Remove the redaction annotation from the document, as well as any references to the object, and objects that are orphaned because of its removal (Form XObjects)

3 **Remove**: Remove the redaction annotation and the content that is contained within the region specified by the annotation.

4 **Replace**: Replace the annotation with specified XObject or text from the annotation.

5 **External Filtering Required**: Pass all text strings in the annotation to an external filter.

6 **Review**: N/A

PDF 5.48: END

PDF 5.49: PROJECTION ANNOTATIONS

OVERVIEW:

Projection annotations are defined in the Adobe Extensions of the ISO 32000 specification. It is defined as a markup annotation, which is used in associated with 3D content. Projection annotations are used to markup measurements or save comments on 3D contents.

CONCERNS:

As with any markup annotation, there is a hidden data risk due to the fact it can be activated or deactivated. In addition, it may provide a data disclosure risk if the annotation was not supposed to remain in the document.

PRODUCT: PDF-1.7

LOCATION:
A Projection Annotation has all of the fields of a markup annotation and general annotations that have been discussed in this section. Project annotations introduce a new field called `/ExData` which defines an external data dictionary, that introduces a `/Subtype` of `/3DM` for measurements of 3D data. It can also use `/Markup3D` for a 3D comment or `/MarkupGeo` for a geospatial markup.

EXAMPLE:

```
1 0 obj
<<
    /Type /Annot
    /Subtype /Projection
    %W, x, y, and z are coordinates in user space
    /Rect [w x y z ] %Coordinates of annotation
    /Contents (Contents)
    /ExData <<
              /Type /ExData
              /Subtype /3DM
              /M3DREF 2 0 R %Indirect reference to 3D measurement dictionary
            >>
>>
endobj

2 0 obj
<<
    /Type /3DMeasure
    /Subtype /RD3  %Many other Named objects in Extensions document exist.
    /TRL (A name string)  %Could be ignored by some readers
>>
endobj
```

RECOMMENDATIONS:

1 **Validate**: Validate the markup annotation just as any other markup annotation in this report.

2 **Validate**: Validate the `/ExData` field and dictionaries for 3D measurement, to ensure the right name objects and correct string data are implemented.

3 **Remove**: Remove the annotation from the document and references to it throughout the rest of the document.

4 **Replace**: N/A

5 **External Filtering Required**: Pass all string text in any of the dictionaries mentioned to an external filter.

6 **External Filtering Required**: Pass 3D data, if included, to an external filter.

7 **Review**: N/A

PDF 5.49: END

PDF 5.50: APPEARANCE STREAMS

OVERVIEW:

Appearance streams are Form XObjects that can be specified by an annotation object as an alternative to border or coloring properties associated with annotations. This allows the appearance of an annotation to be different with user interaction.

An annotation may define three different appearance streams which reference a Form XObject. The three possible appearance streams are Normal, which is used when the user is not interacting with the annotation. The second appearance stream is Rollover, which is used when the user's mouse cursor enters the active area of the annotation without pressing the cursor button. The third appearance stream is Down, when the user pressed down on the mouse button inside the annotation's active area.

CONCERNS:

The main concern for an appearance stream, besides the risks associated with the Form XObject, are an additional hidden data risk. Having the appeareance altered based upon a user interacting with the document introduces a hidden data risk. However, each appearance stream and Form XObject that is referenced must undergo similar sanitization according to PDF 5.50: and PDF 5.17:.

PRODUCT: PDF-1.2

LOCATION:

Appearance streams exist as a Form XObject, but they are referenced by a validation through the `/AP` field. Inside the Appearance stream dictionary there are 3 possible entries: `/N` (normal), `/R` (rollover), and `/D` (down). The `/N` field is the only required object, the other two are optional.

EXAMPLE:

```
%Example Widget annotation used in a form document
14 0 obj
<<
    /F 4
    /Type/Annot
    /Rect[34.0 669.0 580.0 685.703674]
    /FT/Tx
    /Subtype/Widget
    /P 12 0 R
    /T(FillText)
    /V(Value)
    /AP
    <<
        /N 15 0 R %Normal appearance
        /R 16 0 R %Rollover appearance
        /D 17 0 R %Down click appearance
    >>
    /DA (/F1 12 Tf 0 g)
    /Maxlen 16000
```

```
>>endobj

%Form XObject for normal appearance
15 0 obj
<<
    /Length 75
    /BBox[0.0 0.0 546.0 16.703674]
    /Resources <</Font<</F1 16 0 R>>
                 /ProcSet[/PDF]
               >>
>>
stream
/Tx BMC
BT
/F1 12 Tf 0 g
2 4.2598 Td
13.392 TL
(Text) Tj
ET
EMC

endstream
endobj

%Other two Form XObjects for Rollover and Down
```

RECOMMENDATIONS:

1 **Validate**: Check to ensure that only a normal appearance exists and that it references a valid Form XObject. This is the only required item.

2 **Remove:** Remove both "Rollover" and "Down" appearance streams only from the annotation.

3 **Replace**: N/A

4 **External Filtering Required**: Pass XObject data to an external filter.

5 **Review**: N/A

PDF 5.50: END

5.7.2 Rich Text Strings

PDF 5.51: RICH TEXT STRINGS

OVERVIEW:

Rich text strings are associated with the text contents of variable text form fields. They include formatting information and define the style of the text that is to be used. Therefore, it is useful to define this in free-text annotations and markup annotations. They are used in field dictionaries to define characteristics of the text format and appearance. The format of rich text strings conforms

to the XML Forms Architecture (XFA) specification (PDF 5.67:), which is a subset of the XHTML 1.0 specification.

CONCERNS:
Rich text strings introduce a concern because it is another format type (XHTML) which is parsed to interpret the contents, which may lead to a data attack risk.

PRODUCT: PDF-1.5

LOCATION:
The ISO standard defines several fields where rich text strings can be referenced. They are in the form of a string or a text stream, which is an embedded data stream containing a possibly encoded form of the rich text string.

- `/RV` entry in a FDF Field Dictionary (PDF 5.62:)
- `/RV` entry in field dictionaries (PDF 5.63:)
- `/RC` entry in Markup annotations (PDF 5.40:)
- `/RC` entry in Free Text annotation (PDF 5.39:)
- `/DS` entry in Free Text annotation (PDF 5.39:)
- `/DS` entry in fields containing variable text (Appearance Streams, PDF 5.50:)
- `/RV` entry in fields containing variable text (Appearance Streams, PDF 5.50:)

EXAMPLE [1]:

```
/DS (font: 18pt Arial) % Default style string using an abbreviated font
% descriptor to specify 18pt text using an Arial font
/RV (<?xml version="1.0"?><body xmlns="http://www.w3.org/1999/xtml"
      xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/"
      xfa:contentType="text/html" xfa:APIVersion="Acrobat:8.0.0"
xfa:spec="2.4">
      <p style="text-align:left">
      <b>
      <i>
      Here is some bold italic text
      </i>
      </b>
      </p>
      <p style= "font-size:16pt">
      This text uses default text state parameters but changes the font size to
      16.
      </p>
      </body>
     )
```

RECOMMENDATIONS:

1 **Validate**: Check for valid XHTML tags and attributes as defined in the standard.

2 **Remove**: Remove the entire `/RV`, `/RC`, or `/DS` entry and references to it throughout the document.

3 **Replace:** Replace the contents of the field with harmless text data indicating that the information has been removed.

4 **External Filtering Required**: Pass all rich text strings to an external filter handling XHTML data.

5 **Review:** N/A

PDF 5.51: END

5.7.3 Triggers and Actions

5.7.3.1 Triggers

PDF 5.52: TRIGGERS

OVERVIEW:
At a high level, trigger objects are responsible for invoking an action. There are numerous objects within the PDF file that may contain a field named `AA` that is reserved for additional-actions dictionary, which can trigger an action to occur on certain events. Objects such as the entire page, entire document, annotations, and interactive forms can have an additional action dictionary. Another example, the simplest trigger event is the `/OpenAction` entry in the Document Catalog that invokes an action when the document is opened by the PDF reader application. This can reference any of the action objects identified in the previous section.

CONCERNS:
Trigger events can invoke actions that will execute on the users sytem. They present a combined risk of each action presented earlier. Most likely the data attack risk is a high concern. Data hiding and data disclosure risks are also possible since actions can hide information, and many action references fully qualified path names, which may be considered sensitive information.

PRODUCT: REFER TO TABLE 5-4

LOCATION:
Trigger event dictionaries can be attached to annotations, interactive forms, and parts of the document catalogue. Generic trigger events have the form shown in the figure below, where TRIGGER represents one of the trigger events in Table 5-4 below.

```
<<
…DictionaryEntries…
   /AA <<
       /<TRIGGER> <<…DICTIONARY…>>
   >>
>>
```

Figure 5-6. Generic Trigger

Table 5-4. PDF Trigger Types

Entry Name	Description	PDF Version
Annotation Action Triggers		

/E, /X	Triggers an action to be performed when the cursor enters (/E) or exits (/X) the annotation's active area.	PDF-1.2
/D, /U	Triggers an action to be performed when the mouse button is pressed (/D) or released (/U) the annotation's active area.	PDF-1.2
/Fo, /Bl	Triggers an action when the PDF viewer's keyboard input focus enters (/Fo) or leaves (/Bl) a widget annotation input area.	PDF-1.2
/PO, /PC	Triggers an action when the page containing the annotation is opened (/PO) and when it is closed (/PC). The action will be executed in either after the /O or /C action that is in the additional-actions of the dictionary in the page and after the /OpenAction entry.	PDF-1.5
/PV, /PI	Triggers an action when the page containing the annotation becomes visible (/PV) or invisible (/PI).	PDF-1.5
Additional-Actions Entries in a Page Object		
/O, /C	Triggers an action when a user navigates to (/O) or leaves (/C) a page that has an additional action.	PDF-1.2
Form Field JavaScript Action Triggers		
/K	Triggers a JavaScript action when the user modifies a character in a combo box or text fields, or modifies the selected item in a list box.	PDF-1.3
/F	Triggers a JavaScript action before a form field's visible value is formatted for display when it is changed.	PDF-1.3
/V	Triggers a JavaScript action when a form field's value is changed, designed to support validity checking.	PDF-1.3
/C	Triggers a JavaScript action to recalculate the value of the field when another field changes.	PDF-1.3
Document-level JavaScript Action Triggers		
/WC	Triggers a JavaScript action just before the document is closed.	PDF-1.4
/WS, /DS	Triggers a JavaScript action just before (/WS) or just after (/DS) the document is saved.	PDF-1.4
/WP, /DP	Triggers a JavaScript action just before (/WP) or just after (/DP) the document is printed.	PDF-1.4

EXAMPLE:

```
%Document catalog, contains a pages object in 2 0
1 0 obj
<<
    /Pages 2 0 R
    /Type /Catalog
>>
endobj

%Pages object, contains 1 page in object 3 0
2 0 obj
<<
    /Kids [ 3 0 R ]
    /Count 1
    /Type /Pages
>>
endobj

%First and only page object, opens up javascript through /AA /O trigger
3 0 obj
<<
    /Parent 2 0 R
```

```
        /Resources
        <<
             /Font
              <<
                /F1 5 0 R
               >>
        >>
        /AA <<
            /O <<
                    /JS (app.alert\("This page triggered JavaScript"\))
                    /S /JavaScript
                >>
        >>
        /Contents 4 0 R
        /Type /Page
    >>
    endobj
```

RECOMMENDATIONS:

1 **Validate**: Check that the associated action object has been validated. Check that the trigger dictionary has been properly defined.

2 **Remove**: Remove the trigger event and the corresponding action object if not referenced anywhere else.

3 **Replace**: N/A.

4 **External Filtering Required**: N/A

5 **Review**: N/A

PDF 5.52: END

5.7.3.2 Actions

PDF files can contain actions that cause an event to occur on the system reading the file through the viewer application. Actions include linking or resolving to web sites, accessing embedded files, launching applications, and executing scripts. Actions can be attached to the document itself, individual pages, annotations, outline items, and interactive form fields and triggered through a variety of events.

The structure of an action in a document is specified by the `/Type` field with the value of `/Action`. This is following by the `/S` field specifying the type of action. Actions may also specify a `/N` field which indicates that there is a "Next" action that will trigger automatically after the original action is completed. A generic example structure of an action object is shown in Figure 5-7.

There are concerns with any action because they present an interactive version of a PDF document. Transition actions are unique because they chain actions together and are only dangerous because they are action themselves that can force the execution of another action. Generally actions are not necessary but if one is approved to exist in the file, it can act as a starting point to execute a chain of actions. Therefore, this can introduce a data attack risk. For example, an approved may execute initially and then

transition to a malicious action. Regardless, each action should still be inspected, but this possible link between two of them should be carefully examined. Removal and replacement recommendations in this section must consider that a transition action may be defined and accounted for when manipulating a chain of actions.

```
<<
     /Type /Action         %% Optional to define the type action

     %%Required field here to indicate the action
     %% Specifies the type of action described by this dictionary.
     /S    /Launch    %% Required

     /Next << …action dictionary…>>
     %% Alternatively.
     /Next << …more…>>
>>
```

Figure 5-7. Generic Action

This section provides a summary of possible action objects that can be implemented in the PDF file, with examples of each one, and possible recommendations for dealing with each feature. Table 5-5 below indicates each type of action and where it is located in this report. Certain actions have been mentioned previously in this report, where they can be combined with particular content.

Table 5-5. Action Types

Action Name	ISG Section
GoTo Action, GoToR Action, GoToE Action	PDF 5.53:
Launch Action	PDF 5.54:
Thread Action	PDF 5.55:
URI Action	PDF 5.56:
Sound Action	PDF 5.30:
Movie Action	PDF 5.31:
Hide Action	PDF 5.57:
Named Action	PDF 5.58:
Set-OCG-State Action	PDF 5.28:
Rendition Action	PDF 5.33:
GoTo3D View Action	PDF 5.35:
RichMedia Execute Action	PDF 5.38:
JavaScript Action	PDF 5.60:
Submit Form Action	PDF 5.64:
Reset Form Action	PDF 5.66:
Import Data Action	PDF 5.65:
Transition Actions	PDF 5.59:

PDF 5.53: GOTO ACTION

OVERVIEW:

There are three different GoTo actions: `GoTo` (to the same file), `GoToR` (Remote GoTo), and `GoToE` (Embedded File GoTo).

A `GoTo` action causes the PDF viewer software to change the current view to another location within the same document. This is used often in conjunction with Link Annotations. The location is specified in the `/D` field.

A `GoToR` action causes the PDF viewer software to change the current view to another location outside the original PDF document in an external PDF. The `/F` field specifies the external target file. This cannot be used with a file embedded in the document; it must be a destination in a completely separate PDF. This can also be used to access a remote file through a URL.

A `GoToE` action extends the `GoToR` action but the destination is a PDF embedded within the current document or an external document. The file can be represented by the `/F` and `/T` fields. The `/F` field specifies a file target in the same manner as the `GoToR` action. The `/T` field references a target dictionary which can point to an embedded file within the document.

CONCERNS:

GoTo actions cause the PDF reader software to perform an action that may not be wanted by the end user. GoTo actions can also cause the PDF reader to view data outside of the PDF document in another file that may not have been previously sanitized and could be a malicious document, or view embedded files which may not have been analyzed or sanitized. This is a data disclosure risk. The path of the file itself, if an external file was used, could introduce a data disclosure risk. If the referenced file happens to be malicious, this could also be a data attack risk.

PRODUCT: PDF-1.0, GOTOE PDF-1.6

LOCATION:

GoTo action object and their dictionaries are defined by the value `/S /GoTo`*X*, where *X* is an E or R, or doesn't exist. GoTo actions define a destination and if located in another file, a reference for the file.

EXAMPLE 1 (GOTO):

```
%Document catalog of PDF file
1 0 obj<<
       /Pages 6 0 R
       /OpenAction 2 0 R  %%calls object action below
       /Type /Catalog
>>endobj…

2 0 obj <<
       /D[9 0 R/Fit] %%destination is object 9 in this document.
       /S/GoTo
 >>endobj
```

EXAMPLE 2 (GOTOR):

```
5 0 obj
<<
      /S /GoToR
      /D [ 0 /Fit ]
      /F
       <<
            /F
(http://www.adobe.com/devnet/acrobat/pdfs/adobe_supplement_iso32000_1.pdf)
           /Type /FileSpec
```

```
        >>
      /NewWindow false
>>
```

EXAMPLE 3 (GOTOE):

```
5 0 obj % Link from an embedded file to a normal file
<<
     /Type /Action
     /S /GoToE
     /D (Chapter 1)
     /T
     <<
         /R /C %Embedded document is a child doc
         /N (EmbeddedFileName) %Embedded FileName resides in a name tree
     >>
>>
endobj
```

RECOMMENDATIONS:

1. **Validate**: Check for consistency for simple `GoTo` objects, which can be validated if the destination is valid and within the same file. Referential integrity should be check such and the destination should be examined and validated for correctness.

2. **Remove**: Remove the entire object and any references in other objects pointing to the `GoTo` object. In the case of external files or embedded files, these references should be removed throughout the document.

3. **Replace**: Replace the `GoTo` reference to a known location such as the beginning of the file so that it is not malicious or continuously performs a GoTo. `GoToR/GoToE` could be replaced with generic `GoTo` actions that reference a safe location within the same document.

4. **External Filtering Required**: Pass embedded files for external filtering.

5. **External Filtering Required**: Pass the text to an external filter.

6. **Review:** N/A

PDF 5.53: END

PDF 5.54: LAUNCH ACTIONS

OVERVIEW:

This action provides the capability to launch an external application or open a document that has an associated application for that file type. In either case, the document provides the capability to execute an external application.

CONCERNS:

This functionality causes a great amount of concern due to its ability to launch another process or application that could be used maliciously. Most viewer applications may present a warning that it is executing this application. This type of functionality presents a data attack risk. In addition, launch actions may specify a path or filename which could present a data disclosure risk.

PRODUCT: PDF-1.0

LOCATION:

A launch object is represented in an action object dictionary with several other parameters that contain the operating system used for launch, as well as the path of either the filename to be opened or the application itself on the host's filesystem. Host specific information can be represented in a dictionary contained within the Launch action object. The example below is tailored for a Windows environment. The example demonstrates how `OpenAction` can be used to invoke a `Launch` action when the document is first opened.

EXAMPLE:

```
%PDF-1.7
%Document catalog, once document is opened, launch occurs
1 0 obj
<<
    /Type /Catalog
    /Pages 3 0 R
    /OpenAction 8 0 R
>>
endobj

… %rest of document

%Launch action object
8 0 obj
<<
    /Type /Action
    /S /Launch
    /Win %Windows
    <<
       %Filename could contain sensitive information.
       /F ("C:\\text.txt") %Open the file with notepad in Windows
    >>
>>
endobj
```

RECOMMENDATIONS:

1 **Validate**: Check for consistency and that the launch dictionary contains the correct parameters and well formed values as defined by the standard.

2 **Remove**: Remove the action object in its entirety and remove any references to that object throughout the document.

3 **Replace**: N/A

4 **External Filtering Required**: Pass the path (text string) of the launch file to an external filter.

5 **Review**: N/A

PDF 5.54: END

PDF 5.55: THREAD ACTIONS

OVERVIEW:

A Thread action is a special case that jumps to specific article thread within the document (or a different document), which is similar to GoTo actions. Rather than a traditional GoTo that jumps to a particular location within a page, this action forces a jump to a location within an Article Thread, or bead.

CONCERNS:

Thread actions present similar risks as GoTo actions. The remote article thread (only in another PDF) may be the source of malicious content, which would be a data attack risk. Through external files and pathnames, the file specification may also introduce a data disclosure risk.

PRODUCT: PDF-1.1

LOCATION:

Similar to GoTo objects, the Thread action object is defined within the body of the PDF file. A thread action requires the `/S /Thread` entry to be present to indicate a Thread Action, an optional file specification if the thread is in another file, or entry indicating the destination thread.

EXAMPLE:

```
5 0 obj
<<
    /Type /Action
    /S /Thread
    /F (C:\\examplepdf.pdf)
    /D (location)
>>
endobj
```

RECOMMENDATIONS:

1 **Validate**: Examine the end destination of the action and determine if it truly points to a harmless location within the same file.

2 **Validate**: Ensure the dictionary has the correct fields and values implemented.

3 **Remove**: Remove the action object in its entirety and the references to this action.

4 **Replace**: N/A

5 **External Filtering Required**: Pass the path of the file specification for external filtering.

6 **Review**: N/A

PDF 5.55: END

PDF 5.56: URI ACTIONS

OVERVIEW:

The PDF standard supports the action of resolving and opening a Uniform Resource Identifier (URI). This is a file that is a resource located on a network. An action on a URI causes it to be resolved and accessed. For example, a URI for a website will use or launch the default web browser to open that website.

CONCERNS:

URIs can be opened when the PDF document is loaded or when a user clicks on a hyperlink that is embedded into the document. The action that references the URI object specifies how it is opened. This can potentially lead to a malicious PDF file pointing a URI that is an exploit or site containing an exploit, which would result in a data attack risk. Similar to risks associated with file specification, a URI identifies the location in a network, which could lead to a data disclosure risk if the pathname contains sensitive information.

PRODUCT: PDF-1.0

LOCATION:

Throughout the PDF file, there can be several triggers, such as `OpenAction`, that invokes the URI object when that action occurs, or a subtype of Link that allows the user to click on a hyperlink. In both cases, the URI action object is present in the PDF file and is referenced through another object, another action or text that invokes the URI action.

EXAMPLE:

```
8 0 obj
<<
  /Type /Action
  /S /URI
  /URI (http://www.google.com)
>>
endobj
```

RECOMMENDATIONS:

1 **Validate**: N/A

2 **Remove**: Remove the action object in its entirety and remove any references to that object throughout the document.

3 **Replace:** N/A

4 **External Filtering Required**: Pass the text of the path name of the `/URI` object through an external filtering application.

5 **Review**: N/A

PDF 5.56: END

PDF 5.57: HIDE ACTIONS

OVERVIEW:

Hide actions can change the visibility of annotations of interactive forms fields. The `/T` field is used to denote which annotations or form fields should be made visible or invisible when the action is triggered.

CONCERNS:

This action allows for a significant data hiding risk since it controls the visibility of form field objects.

PRODUCT: PDF-1.2

LOCATION:

The Hide Action is represented by an object that first defines the action type of Hide, and then defines a dictionary, text string, or array that lists which items shall be hidden when this action is invoked. Its dictionary is defined by the field `/S /Hide`. Another required field is the annotation or annotations to be hidden or shown in the document through the `/T` field.

EXAMPLE:

```
5 0 obj
<<
    /Type /Action
    %Hide action
    /S /Hide
    %Target annotation is in object 6 0
    /T 6 0 R
    %Hides the annotation in 6 0
    /H true
>>
endobj

6 0 obj
<<
%Annotation object
>>
endobj
```

RECOMMENDATIONS:

1 **Validate**: Check for referential integrity and make sure the action object points to a valid object.

2 **Remove**: Remove the action object in its entirety and remove any references to that object throughout the document.

3 **Replace**: Replace the /H field with false.

4 **External Filtering Required**: N/A

5 **Review**: N/A

PDF 5.57: END

PDF 5.58: NAMED ACTIONS

OVERVIEW:

Named actions are predefined actions similar to GoTo that are implemented in several PDF viewer applications. There are four known named actions: `NextPage, PrevPage, FirstPage,` and `LastPage` which perform a `GoTo` like action to that position in the document.

CONCERNS:

Named actions, just as GoTo actions, can force the PDF viewer application to invoke an action that might adversely impact the system. Commands such as GoTo the next page, previous page, first page, and last page, may seem acceptable, but unknown named actions may introduce an exploit or a data attack risk. In the ISO standard there is a note that indicates that readers may support additional and nonstandard named actions, which is not portable. For sanitized documents, only the supported Named actions and internal GoTo commands should be included and validated.

PRODUCT: PDF-1.2

LOCATION:

The dictionary for this type of action requires the field `/S /Named`, and `/N (name)` for the name of the action. The name of the action is one of the previous defined names (`NextPage, PrevPage, FirstPage,` and `LastPage`), or could be a named destination that indicates which area to view next.

EXAMPLE:

```
5 0 obj
<<
    /S /Named
    /N /NextPage %Goes to next page
>>
endobj
```

RECOMMENDATIONS:

1 **Validate**: Determine which type of Named action is present in the document and accept only the four valid ones discussed above.

2 **Validate**: If the user defined Named Destination is utilized, validate the destination as a harmless location in the current file.

3 **Remove**: Remove the action object in its entirety if the name is nonstandard and remove any references to that object throughout the document.

4 **Replace**: N/A

5 **External Filtering Required**: N/A

6 **Review**: N/A

PDF 5.58: END

PDF 5.59: TRANSITION ACTIONS

OVERVIEW:

Transition actions are used in conjunction with the /Next entry in an action dictionary. Although any action can define a /Next entry which defines the action to execute next in series. A special action can be used in between two actions to handle the transition appearances in the sequence. This can be used for drawing operations that may occur in a sequence. The action specifies a transition dictionary which defines the type of transition effect and the effect's duration in the number of seconds.

CONCERNS:

Transition actions reference a Transition Dictionary which can render the display of the document. Since this can be done as a sequence, there is a hidden data risk and data disclosure risk where information may be lost in between sequences of actions.

PRODUCT: PDF-1.5

LOCATION:

A Transition Action object is an object with only two required entries: a `/S /Trans`, defines the type of action and a dictionary object defined by the key `/Trans`. Optionally, as any action object, the field `/Type /Action` may be present. The Transition Dictionary is defined in Table 162 of the ISO standard. It defines the name of the transition style, the duration of the transition effect

EXAMPLE:

```
10 0 obj
<<
    /Type /Action  %Optional
    /S /Trans      %Required
    /Trans 11 0 R  %Required
>>
endobj

11 0 obj
<<
   /Type /Trans   %Optional
   /S /Split      %Effect is to split the screen
```

```
      /D 10          %10 second effect
      /Dm /V         %Vertical Split
      /M /I          %Inward from the page edge
   >>
   endobj
```

RECOMMENDATIONS:

1 **Validate**: Check that a `/Next` field is present to ensure that the Transition object is necessary.

2 **Validate**: Check the Transition dictionary has the required fields and valid named objects according to Table 162 of the ISO standard. Ensure the field `/D` for the duration of the transition effect is equal to 1 (for one second).

3 **Remove**: Remove the Transition action object and the entire transition dictionary.

4 **Replace**: Replace the transition dictionary with the simplest transition `/R` (for replace) and ensure that `/D` is not present since the effect will be ignored with this transition effect.

5 **External Filtering Required**: N/A

6 **Review**: Review the document to ensure that all pages and content are visible.

PDF 5.59: END

5.7.4 JavaScript

JavaScript is located in numerous locations in a PDF file and can potentially introduce harm for the end users system and is the source of numerous data attack risks. Many interactive features or rich content support the use of JavaScript. Many PDF dictionaries and objects include entries for a JavaScript reference which can invoke an action to occur based upon some type of event when the user was interacting with the PDF document. The JavaScript API document outlines the interface and functions that are available to a PDF designer. There are numerous functions that immediately introduce concern such as functions that remove content and information from the document, access databases through the ADBC plug-in, emailing the PDF document to other recipients with or without user action and numerous others that are discussed in the API document. Functionality like this may be useful in some context, but more than likely is subject to an attack risk. There may also be malicious JavaScript designed to exfiltrate data from the system. In addition to the data attack risk it presents, there may also be a data disclosure risk. Content from the PDF document can be accessed in JavaScript, thus making the hidden data risk greater if JavaScript changes the appearance. Therefore, all the functionality of JavaScript should be considered a high risk from hidden data, data disclosure, and data attack.

Furthermore, certified PDF documents can automatically execute JavaScript since it is a trusted document. Secure methods that are higher privilege than most methods in JavaScript can execute without restriction according to the JavaScript API document.

All JavaScript code should be inspected thoroughly when the file is sanitized and knowing where each type of JavaScript reference as identified in the standard is important. Table 5-6 summarizes the inclusion of JavaScript in the ISO standard and provides a purpose for referencing JavaScript.

Table 5-6. JavaScript References in ISO Standard

Dictionary/Location	Purpose	ISO Reference
Name Dictionary	Contains Document-Level JavaScript actions.	Table 31
Form-Field Additional Actions	JavaScript actions when there is interaction with the field.	Table 196
Document Catalog	Action objects and triggers that can invoke JavaScript.	Table 197
Rendition Action	JavaScript code can be executed when rendition is activated.	Table 214
JavaScript Action	Default JavaScript action that executes a JavaScript script when invoked.	Table 217
JavaScript Dictionary – FDF	FDF JavaScript that can be executed during import of forms data.	Table 245
Legal Attestation	Simply references the number of JavaScript Actions in the document.	Table 259
Requirement Dictionary	Document Requirements which indicate is JavaScript is a requirement for this file.	Table 264
Rendition Dictionary	Name tree lookup for JavaScript actions.	Table 266
3D Stream Dictionary	When the 3D stream is instantiated, a JavaScript script can be executed.	Table 300
Trigger	Any trigger can invoke an action, including a JavaScript action.	Section 12.6.3

PDF 5.60: JAVASCRIPT ACTIONS

OVERVIEW:

Actions can be in the form of JavaScript. When the action is invoked, the JavaScript code will be executed in the reader environment by its interpreter. This is useful for adding dynamic content and changing the contents of the PDF document, particularly when dealing with forms that provide this type of content.

CONCERNS:

Any document that in turn executes a script is of concern to the end user due to a data attack risk. Embedded JavaScript in the document could potentially be malicious if the code is not processed as part of the sanitization process. JavaScript action objects are often obfuscated in PDF syntax (with ASCII substitution) and sometimes obfuscated in the JavaScript code itself. JavaScript code details will require external validation.

PRODUCT: PDF-1.0

LOCATION:

A JavaScript action object can be defined within the body of the PDF file. The JavaScript code can be implemented a string enclosed by parenthesis, shown in the first example. The action object implements a `/JS` field which defines the string of JavaScript or an indirect reference to the JavaScript code.

EXAMPLE:

```
%PDF-1.7
```

```
%Document catalog features OpenAction, invokes JavaScript upon document open.
1 0 obj
<<
    /Type /Catalog
    /Pages 3 0 R
    /OpenAction 7 0 R
>>
endobj

%JavaScript code as string
7 0 obj
<<
    /Type /Action
    /S /JavaScript
    /JS (app.alert({cMsg: 'Alert Message', cTitle: 'PDF
                        JavaScript', nIcon: 3});)
>>
endobj
```

EXAMPLE (JAVASCRIPT IN STREAM):

```
%PDF-1.7
%Document catalog features OpenAction, invokes JavaScript upon document open.
1 0 obj
<<
    /Type /Catalog
    /Outlines 2 0 R
    /Pages 3 0 R
    /OpenAction 7 0 R
>>
endobj

7 0 obj
<<
    /Type /Action
    /S /JavaScript
    /JS 8 0 R
>>
endobj

8 0 obj
<<
    /Length 98 %Length of embedded stream in bytes
    %Possible to encode this so it is not readable to make analysis more
    %difficult.
>>
stream
(app.alert({cMsg: 'Alert Message', cTitle: 'PDF JavaScript', nIcon: 3});)
endstream
endobj
```

RECOMMENDATIONS:

1 **Validate**: Ensure referential integrity and that the usage of JavaScript is consistent with the contents of the file.

2 **Validate**: Ensure consistency that each object is implemented properly and that the JavaScript code is implemented correctly.

3 **Remove**: Remove the JavaScript object in its entirety and remove all references to that object throughout the document. If JavaScript code is defined outside this action object, then remove the object that contains the code under analysis.

4 **Replace**: N/A

5 **External Filtering Required**: Extract and decode the `/JS` contents (from the internal string or other object) to an external filter.

6 **Review**: N/A

PDF 5.60: END

5.8 PDF Functions

PDF 5.61: FUNCTIONS

OVERVIEW:

Function objects permit the representation of a numerical transformation. It is designed to take multiple input values and output any number of values. All of the possible values are numbers with defined ranges or domains. Clipping occurs when values are identified or used outside of the domain. There are four categories of functions in the standard: Sample function, Exponential Interpolation function, Stitching function, and a PostScript calculator function. The definitions of each function are located in the standard.

Examples of function may be a transfer function, which can be referenced by a Graphics State Parameter dictionary, or a halftone function to produce certain visual effects. This transfer function would be present in a halftone dictionary.

There are four classes or types of functions in the standard:

- **Type 0 Functions:** A Sampled function that uses a table of values to define the function.
- **Type 2 Functions**: An Exponential interpolation function which uses a set of coefficients.
- **Type 3 Functions**: A stitching function which is a combination of other functions, partitioned across a domain [1].
- **Type 4 Functions**: A PostScript calculator function which uses operators from the PostScript language to define an arithmetic expression [1].

Functions are used in PDF files in three places. Two locations are in the PostScript's device dependent layer with a transfer function and spot functions. The third location is a shading

pattern (e.g., gradient effects where the color changes over distance). Each shading type in ISO Section 8.7.4.5 allows a function to be specified.

CONCERNS:
While functions may seem harmless, they can manipulate input and generate output. Functions also utilize arrays of values for inputs and outputs, which must absolutely implement safe error and boundary checking. Values embedded into stream data should also have strict validity and boundary checks in place. As such, this could present a data attack risk.

Type 4 functions introduce a concern for data attack risk, because it involves arithmetic operations with user supplied operands that can be performed. Invalid instructions, overflows, or illegal mathematical operations could be used in a malicious function. The standard lists stack overflow, stack underflow, type error, range error, and an undefined result as types of errors that can occur in the PostScript language.

PRODUCT: PDF-1.2 (SAMPLING FUNCTION), PDF-1.3 (OTHERS)

LOCATION:
According to the standard, functions are represented in either completely in a dictionary or a stream, which depends on the type of function. All dictionaries of each function define a `/FunctionType` which is 0, 2, 3, or 4 (Sampling, Exponential Interpolation, Stitching, and PostScript calculator, respectively). A `/Domain` field is also required which defines an array of 2x*m* numbers, which define the range (min/max) of valid input values, of *m* input values. An optional field called `/Range` is a similar array but defines the ranges for the *n* output values.

Each type of function may have additional parameters in its dictionary or required data inside of a data stream. The first example below from the ISO standard represents a basic Sample Function that an input of 2 values, output of 1 value, with a stream full of sample values.

The second example shows a Type 4 function from the ISO standard that uses a series of operators and operands. For a full list of operands, Table 42 of the ISO standard describes each operator.

EXAMPLE:
```
19 0 obj<< /ProcSet [ /PDF /Text /ImageC /ImageI ] /Font << /F1 28 0 R >>
/XObject << /Im1 39 0 R /Im2 40 0 R >> /ExtGState << /GS1 41 0 R /GS2 42 0 R >>
/ColorSpace << /Cs15 20 0 R /Cs8 21 0 R /Cs9 22 0 R /Cs11 25 0 R /Cs12 24 0 R
/Cs14 23 0 R >> /Properties << /MC1 43 0 R /MC2 44 0 R >> >> endobj

%Cs14 color space
23 0 obj[ /DeviceN [ /Black /PANTONE#20354#20CVC /None /None /None ] 22 0 R 29
0 R 36 0 R ]
endobj

%Function Type 4, contents decoded to show operations
29 0 obj<< /FunctionType 4 /Domain [ 0 1 0 1 0 1 0 1 0 1 ] /Range [ 0 1 0 1 0 1
] /Length 28 /Filter /FlateDecode >>
stream
%Stream contents here shown for reference
{5 3 roll pop pop }
%end of contents
endstream
endobj
```

RECOMMENDATIONS:

1 **Validate**: Validation of a function dictionary and stream object requires extensive checking of numerical values and lengths of values inside stream objects. Each value and array should be properly defined to avoid risks in parsing of computing values of the function.

2 **Remove**: The function dictionary and data stream can be removed. References to the function dictionary may need to be removed or replaced, depending on how the function is utilized and what it impact it may have. Removal will break the other object that utilizes the function and changes the visible appearance of the content.

3 **Replace**: N/A

4 **External Filtering Required**: N/A

5 **Review**: N/A

PDF 5.61: END

5.9 Interactive Forms

PDF 5.62: FORMS DATA FORMAT (FDF)

OVERVIEW:

A Forms Data Format (FDF) file is used to store the elements that are required for exporting and important forms and annotation data [1]. A FDF file is very similar to a PDF file in structure. FDF files contain a one-line header, which has the suggested version of the PDF specification in the form of "%FDF-1.2". It contains a body, just as PDF, that contains objects that represent the actual content. It contains an optional cross-reference table that identifies objects in the body. It also contains an optional trailer that is used to locate the cross-reference table and other objects in the body, such as the FDF Catalog. Similar to PDF, but the cross-reference table and trailer are now optional.

The body consists of a FDF Catalog (similar in purpose to the Document Catalog), which provides a version and a required FDF dictionary. The FDF dictionary provides information and references to the remainder of the content in the file. Information regarding the corresponding PDF file, an ID field, a Fields array, a Pages array for FDF Pages, an Annotation array for FDF annotations, a Differences stream which includes incremental update information made to the PDF document since it was opened. The FDF dictionary also includes a Target string identifying the browser frame in which the PDF was opened. It also contains an array for Embedded FDF files within the file. Lastly, it contains a JavaScript dictionary, which can introduce JavaScript in text strings in the FDF file.

In the FDF Field Dictionary, information is listed about each field in the PDF such that content can be replaced with the FDF information into the PDF file. An array of fields is defined in the FDF dictionary. Information regarding each field in the PDF is represented in a field dictionary. The field name (`/T`), and the value (`/V`), flags that denote the state of the field. An appearance

stream for pushbutton fields, as well as other fields that define the appearance of the field in the PDF. Also of interest, action and additional-action fields can be implemented just as in a PDF file.

In the FDF Page Dictionary, there is a templates array which can introduce a template for a page. In addition, there is an optional page information dictionary.

Annotations can exist in an FDF file, all of the annotations mentioned in Section 5.7.1 can be used except for the Link, Movie, Widget, PrinterMark, Screen, and TrapNet annotations. Annotations in FDF files resemble the same as in PDF files, except there is an additional entry called `/Page` which is an integer for the page number that the annotation will appear.

In the JavaScript Dictionary, there are four possible entries for including JavaScript. A `/Before` entry can include JavaScript that gets executed before the FDF file is imported. A `/After` entry which includes JavaScript to be executed following the import. Another field `/AfterPermsReady` which is used to contain JavaScript that is executed when data is imported and usage rights have been determined. The last field is an array called `/Doc` which can include additional JavaScript code in a similar manner to the Names tree, so these get executed before the code in the `/Before` entry.

CONCERNS:

FDF files represent a concern because they can contain JavaScript for form interaction, which introduces a data attack risk.

FDF files provide data that is useful with interactive forms and could be a data disclosure risk if data is imported from the file with values of the form that were previous filled in. The Differences stream can contain all incremental updated information which may also be a data disclosure risk.

Structurally, a FDF file is similar to PDF which introduce a hidden data risk with information around objects not belonging to the FDF object hierarchy.

PRODUCT: PDF-1.2

LOCATION:

The FDF description above represents a separate file that is used in conjunction with PDF. Information regarding the structure and dictionary information is presented in the overview.

EXAMPLE:

```
%FDF-1.2
1 0 obj
<<
    /FDF
    <<
        /F <<
           /Type/Filespec
           /F(/C/pdffile.pdf)
           /UF(/C/ pdffile.pdf)
           >>
           /Fields[<</T(Text1)/V(Value1)>>
               <</T(Text2)/V(Value2)>>
               <</T(Text3)/V(Value3)>>
               <</T(Text4)/V(Value4)>>
               <</T(Check Box42)/V/Off>>]
```

```
                /JavaScript
                <<
                        …JavaScript dictionary possibly containing code …
                >>
        >>
>>
endobj
trailer
<</Root 1 0 R>>
%%EOF
```

RECOMMENDATIONS:

1 **Validate**: Validation of a FDF file should consist of a similar validation of PDF files, particularly when object types are the same.

2 **Remove**: Remove interactive references such as the JavaScript dictionary from the FDF file and action information from field dictionaries.

3 **Replace**: N/A

4 **External Filtering Required**: Pass all text strings and values to text fields in the fields array to an external filter.

5 **Review**: N/A

PDF 5.62: END

PDF 5.63: ACROBAT FORMS

OVERVIEW:

An interactive form is a collection of fields used to gather information from the user. These fields may appear on any page, and all of these fields make up a single, global interactive form spanning the entire document. An interactive form is defined using one of following methods (they are mutually exclusive):

- Acrobat Form (or AcroForm): based on Widget Annotations with data import and export. Import and export functionality is not limited to FDF and can be done in a variety of ways. This section focuses on AcroForms.
- Adobe XML Forms Architecture (XFA) Form: based on separate template, configuration, and dataset specification with data import/export as XML documents. The handling of the full features of XFA forms mentioned in this section but readers should consult the XFA specification for more details.

Interactive forms can be configured to export or import the contents of arbitrary subsets of its fields using form-specific actions. Interactive forms support the following actions:

- Submit-form actions (PDF 5.64:): transmit the names and values of selected interactive form fields to a specified URL, the standard indicates that this is presumably the address of a Web server that will process them and send back a response.

- Reset-form actions (PDF 5.66:): reset selected interactive form fields to their default values.

- Import-data actions (PDF 5.65:): import Forms Data Format (FDF), XML-based Forms Data Format (XFDF), or XML data into the document's interactive form from a specified file.

Fields that are defined in the document are contained within an object called Field Dictionaries; there is a hierarchy so inheritance of fields can be utilized. Child fields include Widget Annotations (PDF 5.39:). Field Dictionaries define the type of field, the value of field when reset, as well as an additional action to certain trigger events, which are defined in PDF 5.52:.

CONCERNS:

The interactive nature of form actions can import content from or export content to external resources introduces many risks. A data disclosure risk is present if information is being transferred that may contain sensitive information. Field names may also contain sensitive information which is another data disclosure risk. Field descriptions or string fields within the dictionary may also be a source for hidden data since some are optional. Actions can be linked with forms, which have been presented earlier as a data attack risk. There are several trigger events that can occur with widget annotations (lost focus, receive focus) that can invoke an action object to occur.

Content may be encoded as rich text strings (PDF 5.51:), which are XML documents that conform to the XFA specification, which is itself a subset of the XHTML 1.0 specification. Rich text strings may include hyperlinks to arbitrary URIs (via the <a> XHTML element), which again may present a similar data disclosure risk.

PRODUCT: PDF-1.2

LOCATION:

An interactive form in a document is identified by an Interactive Form dictionary specified in the optional `/AcroForm` entry in the Document Catalog. This dictionary contains a required `/Fields` entry as well as optional fields such as, `/DA` (attribute for variable text), and `/DR` (resource dictionary for the field appearance streams). XFA Forms are identified by the optional `/XFA` field in the AcroForm dictionary.

Individual form fields are identified by Field dictionaries linked to by the `/Fields` entry in the Document Catalog's `/AcroForm` entry. The Field dictionaries may be organized hierarchically into a tree structure. Each dictionary contains a required `/FT` entry that specifies the field type; however, this entry may not be present in fields that have other fields as descendants. A field's children may also include Widget Annotations that define the appearance on the page. When Field contains a single Widget Annotation the contents of both dictionaries may be merged into a single dictionary. Supported field types include:

- Button fields: represent interactive controls on the screen that the user can manipulate with the mouse. They include pushbuttons, check boxes, and radio buttons.

- Text fields: boxes or spaces in which the user can enter text from the keyboard. The text may be restricted to a single line or may be permitted to span multiple lines. Rich text strings are identified by the `/RV` entry.
- Choice fields: contain several text items, one or more of which may be selected as the field value. The items may be presented to the user in one of two forms: a scrollable list box, or a combo box consisting of a drop-down list optionally accompanied by an editable text box in which the user can type a value other than the predefined choices. NOTE: Since the contents of these choice fields is not displayed or may not be immediately apparent, data can be hidden in this type of field very easily.
- Signature field: represents a digital signature.
- Barcode field: from the Adobe Extensions document.

EXAMPLE:

```
%Document catalog
1 0 obj
<<
    /Pages 6 0 R
    /Type/Catalog
    /AcroForm 2 0 R
    /Metadata 7 0 R
>>endobj

%Forms dictionary, referenced through AcroForm
2 0 obj
<<
     /Fields[3 0 R 4 0 R 5 0 R]
    /DR 1 0 R
    /DA(/Font1 10 Tf 0 g)
>>endobj

%Field and widget annotation, referenced by Fields array in object 2
3 0 obj
<<
    /F 4
    /Type/Annot          %An annotation
    /Rect[32.0 622.0 566.0 675.0]
    /FT/Tx               %Field type is text
    /Subtype /Widget     %Widget annotation
    /P 8 0 R
    /T(FillText)
    /V(DefaultValue)
    /AP <</N 9 0 R>>     %Appearance stream
    /DA (/Font1 12 Tf 0 g)
    /Maxlen 16000
>>endobj
```

RECOMMENDATIONS:

1 **Validate**: Check for consistency and verify that the Field data is correct and required fields are implemented in the dictionary.

2 **Validate**: Check for referential integrity and that all Fields are properly referenced. If annotation is combined, check that it is a `/Widget` annotation.

3 **Remove**: Remove the entire dictionary of the field type and annotations, including references to it throughout the document. This may remove the entire content of a file.

4 **Remove**: Remove additional actions in Field dictionaries.

5 **Remove**: Remove field attributes and field dictionary but keep widget intact, which would freeze the form in a final view.

6 **Replace:** Replace the contents of the field with harmless text data indicating that the information has been removed.

7 **Replace:** Put the field value into the page content as normal text.

8 **External Filtering Required**: Pass all field text to an external filter.

9 **Review**: N/A

PDF 5.63: END

PDF 5.64: SUBMIT-FORM ACTIONS

OVERVIEW:

When a submit-form action is invoked, the names and values of selected interactive form fields are sent to a URL. The standard assumes that the URL points to a valid server that will properly process the request and respond accordingly.

CONCERNS:

This action can be a cause for concern due to its exfiltration of data contained within the PDF. It also means that PDF document is interactive and has the ability to interface with an unknown website or URL where information can be exchanged. This can be a data disclosure risk for information contained within the form. In addition, since the PDF interacts with a remote server, the name of the server could present another data disclosure risk, and if the remote server is malicious, then it is a data attack risk.

PRODUCT: PDF-1.2

LOCATION:

The Submit-Form action implements a dictionary with the field `/S /SubmitForm`. It also implements a required field `/F`, which is a URL file specification indicating the script of a website that will process the submission of form data. An optional field in the Submit-Form action dictionary is `/Fields`, an array identifying which fields to include or exclude in the submission to the URL. The second optional field is `/Flags` which is an integer value which indicates information such as the Include/Exclude feature, the GetMethod for the HTTP request, ExportFormat of the forms data.

EXAMPLE:

```
3 0 obj
<<
      /Parent 2 0 R
      /AA <<
          /O <<
```

```
                    /JS (app.alert\("Submit form example”))
                    /Next <<
                     %This is the SubmitForm Action Object
                      /Flags 8  %Get METHOD
                      /Fields [ ] %none specified
                      /F <<
                           /FS /URL
                           /F (http://www.google.com)
                      >>
                      /S /SubmitForm
                >>
                /S /JavaScript
                >>
          >>
          /Contents 4 0 R
          /MediaBox [ 0 0 795 842 ]
          /Type /Page
     >>
     endobj
```

RECOMMENDATIONS:

1 **Validate**: Ensure fields within the SubmitForm action dictionary are properly formed.

2 **Remove**: Remove the SubmitForm action in its entirety and any references to that object.

3 **Replace**: N/A

4 **External Filtering Required:** Pass the URL information text to an external filter.

5 **Review**: N/A

PDF 5.64: END

PDF 5.65: IMPORTDATA ACTIONS

OVERVIEW:

The import-data action imports Forms Data Format (FDF), XFDF, or XML data into the document’s interactive form from a file.

CONCERNS:

Since data is introduced through an extra file that may not be visible in the current file, this introduces unknown data that may be sensitive or malicious in nature. It presents a data disclosure risk and it also presents a potential data attack risk if the data is not properly formatted correctly.

PRODUCT: PDF-1.2

LOCATION:

The ImportData dictionary is defined as an object with the field `/S /ImportData`. The only additional entry is the file specification represented by the field `/F`. File specifications are presented earlier in PDF 5.4: and PDF 5.5:.

EXAMPLE:

```
1 0 obj
<<
    /Type /Action
    /S /ImportData
    /F 2 0 R
>>
endobj

%File specification dictionary
2 0 obj
<<
   /Type /Filespec
   /EF 3 0 R
>>
endobj

%File stream
3 0 obj
<<
    /Type /EmbeddedFile
    /Length 4000
    /Filter /LZWDecode
    %Object specific dictionary entries.
>>
stream
%4000 byte file data
endstream
endobj
```

RECOMMENDATIONS:

1 **Validate**: Ensure consistency and that all the valid fields in the dictionary are present and implemented properly.

2 **Validate**: Ensure the file specification information does not contain sensitive data in the path name or related information.

3 **Remove**: Remove the action object and references to the object throughout the file.

4 **Replace**: N/A

5 **External Filtering Required**: Examine the contents of the file specification. Use an external filter to examine this according to the correct file type.

6 **External Filtering Required**: Pass all path information or text information to an external filter.

7 **Review**: N/A

PDF 5.65: END

PDF 5.66: RESET-FORM ACTION

OVERVIEW:

Interactive form fields in a PDF file contain default values, or no initial value. A reset-form action can reset the data within the form fields to those values. The action defines the fields that it is considering resetting, and then implements a flag. When the first bit in flags is clear, the Fields specified will be reset, otherwise the Fields will be excluded from the reset.

CONCERNS:

Resetting form fields may revert to default values, which are defined as text strings in the form information. This information could represent a data disclosure risk if not properly sanitized.

PRODUCT: PDF-1.2

LOCATION:

The Reset-Form action object dictionary is defined as an object with the field `/S /Resetform`. There is a Fields array which contains indirect references pointing to fields that are impacted by this action. Lastly, there is a flag field which only implements one flag in bit 1. A value of 0 indicates that the Fields in the previously defined Fields array are to be reset; meanwhile a value of 1 indicates that these fields are excluded from the action.

EXAMPLE:

```
5 0 obj
<<
    /Type /Action
    /S /ResetForm
    %References Fields to reset
    /Fields [ 6 0 R 7 0 R]
    %Flags, either 0 or 1, 0 = Reset Fields, 1 = Exclude Fields from reset
    /Flags 0
>>
endobj
```

RECOMMENDATIONS:

1 **Validate**: Ensure consistency and that all the valid fields in the dictionary are present and implemented properly.

2 **Validate**: Ensure referential integrity and that each indirect reference in the Fields array points to a valid field.

3 **Remove**: Remove the action object and references to the object throughout the file.

4 **Replace**: Replace the flags such that it changes the include/exclude functionality appropriately. Depends on the fields referenced.

5 **External Filtering Required**: N/A

6 **Review**: N/A

PDF 5.66: END

PDF 5.67: XML FORMS ARCHITECTURE

OVERVIEW:

The data that supports Adobe's XML Forms Architecture (XFA) represent several significant challenges to the inspection and sanitization effort. The XFA specification includes a description of the data structures contained in PDF streams associated with the *XFA* attribute. The XFA specification also provides the overall context for programs that process the XFA data structures and includes references other documents that are needed to fully document the expected behavior of programs that process XFA data. The XFA specification provides a common XML-based grammar for describing interactive forms. In the simplest case, XFA defines the appearance of a form and identifies the fields where data maybe entered by the user. For more complex business applications, XFA supports the following capabilities:

- User Interactions: Input data may be validated or dynamically calculated. Other servers may be queried to provide context dependent default values.
- Dynamic Layout: The layout of fields on the form may be rearranged in response to user input or data from an external server.
- Workflow: Control of sequencing of data presentation, editing, printing, submission to the form server.

The XFA data structures are XML-based and are defined in schemas written in the RELAX: Next Generation (RNG) language. This schema specification extends XML-Schema 1.0 to allow for additional directives (e.g., the "interleave" directive which allows children to appear in any order). While XFA Forms PDF Forms (*AcroForm*) shares common functionality, they are independent from each other but may be combined in several different ways. The XFA data structures may be packaged alone in a file (Full XFA mode) but are more commonly enclosed in a PDF file as a stream and utilize the PDF stream compression and certification (i.e., digital signing). PDF may be used to specify the form background with XFA only managing the presentation of data in fields (XFA Foreground mode). When Full XFA is packaged in a PDF file (shell PDF), the fonts and images in the *AcroForm* resource directory (*DR)* may be referenced from XFA. The *NeedsRendering* flag in the *AcroForm* dictionary is set to disable PDF rendering. The XFA specification also permits a PDF file with the form background to be embedded as an XFA element (*xdp.pdf*) as either an external reference or as encoded data (the *chunk* element). The *xdp* package is the highest level element. The following are its more significant sub elements:

- Template (*template*): This element provides the overall the data presentation logic and the most extensive functionality may be accessed via its sub elements.
- Data Specification (*datasets* and *dataDescription*): This element describes the data structure of the data presented in the form's fields. The sub elements may be drawn from another XML name space (via the *xmlns* attribute).
- External Connections (*connectionSet*, *sourceSet, xsdConnection, wsdlConnection,* and *xmlConnection*): These elements support access to external servers via web services (SOAP message exchange via HTTP POST request) and Microsoft's ActiveX Data Object (ADO) interfaces.

- Configuration (*config*, *localeSet,* and *stylesheet*): The data in these elements modify the behavior of other elements. For example, the data presentation specific to a particular language and country may be specified.
- PDF interoperation (*pdf, xfdf,* and *xmpmeta*): These element support interaction with data formatted as PDF, XML Forms Data Format (XFDF), and Extensible Metadata Platform (XMP).

The data presentation approach in XFA is different from PDF and includes many of the core features of XHTML. The subset of XHTML supported by XFA is called Rich Text. Rich Text uses a restricted set of Cascading Style Sheet (CSS) attributes plus some non-standard attributes that have been proposed for a future CSS version. In Rich Text, the *span* element supports embedded object accessed via a *xfa:embed* attribute. The embedded object's value is inserted as XHTML text. Embedded objects provide access to form fields and data retrieved from external servers. If a URI is specified Rich Text from an external server may be inserted. Rich Text also supports hyperlinks (an *a* element with a *href* attribute), but does not expect the XFA reader to follow the link when clicked. The XFA template element contains a hierarchical set of containers each of which may include elements that display text, images, or graphic primitives (i.e., arcs, lines, and rectangles). XFA decouples the data value from its presentation and supports elements for common data types (e.g., float, decimal, date, and time). For these items the presentation can be specific to specific locale (e.g., date format).

XFA includes support for the JavaScript and Adobe's FormCalc scripting languages in support of its data presentation and user interaction capabilities. These scripting language reference data items in the XFA package using its Domain Object Model (DOM) which provides a naming convention that allows access to the XML elements in the XFA package as well as data imported from the user or external servers. XFA scripts can access the PDF DOM for the PDF that it resides inside, in the case of an XFA foreground document. XFA scripts are assigned to object properties and invoked by the XFA processor when a specific event occurs (e.g., field *validate* or *calculate* properties). The DOM naming convention is also used to link different elements of the XFA package with each other. This allows much of XFA dynamic functionality to be accomplished without scripts. For example, the default value for an input field can be automatically retrieved from a database by linking the field to the appropriate *sourceSet* element. The "LiveCycle® Designer ES Scripting Reference" provides a description of the objects, properties and methods available to the scripting languages. The specific capabilities available to a script depend on the location (client or server) and the vendor's implementation of security restrictions. While, the scripting languages do not represent a significant risk by themselves, they do provide alternate methods for building and modifying the XFA DOM structures that control XFA processing which makes it difficult to reliably implement any inspection and sanitization restrictions. The following scripting objects/methods may represent a more significant threat when used for malicious purpose:

- The *hostPseudoHost* object provides several methods that may be used to initiate an application level action (e.g., *importData*, *exportData*, *gotoURL*, and *MessageBox*).
- The *wsdlConnection.execute* method initiates a request via SOAP/WSDL.
- The *source.open* and *source.update* methods initiates request to an external server via ADO.
- The *node.loadXML* and *node.applyXSL* methods initiate XML processing.
- The *eventPseudoModel.emit* method generates user interface events.

Both XFA and PDF Forms support user interaction while entering data into a form. The data in the XFA elements that supports this capability may not be displayed on the final screen or in

hard copy output, but could still represent a sanitization risk: In particular, XFA's support for dynamic form layout allows the data in a field or from an external server to control the visual appearance of the form. The following are potential sanitization issues:

- Choice List (*choiceList*), tooltip hints (*toolTip*), and descriptive labels (*caption*) support the user interface and may contain sensitive data.
- The XML elements for fields and data structures include names that may contain sensitive data.
- The comments, variable names, and string constants in scripts may contain sensitive data.
- Image selection widget (*imageEdit*) allows a user to enter an image via its URL.

The example located in this section is directly referenced from the ISO standard.

CONCERNS:

XFA forms have similar risks to AcroForms. Contents within each packet and stream data in the PDF file need to be validated to ensure they are properly formatted; this could introduce a data attack risk on software parsing this information. XFA also contains scripting object or methods that could be used for data attack. Since this information is stored in stream data and in a different format, it needs to be thoroughly examined for a hidden data or disclosure of data risk.

The following are also potential sanitization issues:

- Adobe products support the JPEG, PNG, and TIFF image formats.
- Fonts may be defined in the PDF file.
- Values retrieved from external servers might change after processing.
- Format of some data fields depends on the locale info in effect at time of processing.
- A hyperlink's URL would not be displayed but could be sensitive.
- Containers may be hidden either explicitly or by position in the hierarchy (Z-order).
- The XFA supports inclusion of meta-data that supports add-in products.

Some malicious PDF files are commonly embedding JavaScript code into XFA forms or use XFA has a container for other malicious content. This can reside in a stream object defining the XFA, which uses a different object model than the PDF-based JavaScript discussed in this report (PDF 5.60:). Sanitization should inspect and decode stream objects containing this information and examine it looking for script information.

PRODUCT: PDF-1.5

LOCATION:

XFA forms specify an interactive form dictionary. The dictionary implements a stream itself containing XFA resource, or it identifies an array of individual streams. It can be done either way; the array of streams can be combined into a single stream. The stream data represents the state of the form according to the ISO standard.

EXAMPLE [1]:

```
%XFA entry in interactive form dictionary, same as AcroForms, but specifies
%an XFA field, which is defined in the standard.
1 0 obj
<<
    /XFA [(xdp:xdp) 10 0 R XFA %resource specified as individual packets
          (template) 11 0 R
          (datasets) 12 0 R
          (config) 13 0 R
```

```
            (/xdp:xdp) 14 0 R ]
    >>
    endobj

    10 0 obj
    stream
        <xdp:xdp xmlns:xdp="http://ns.adobe.com/xdp/">
    endstream

    11 0 obj
    stream
        <template xmlns="http://www.xfa.org/schema/xfa-template/2.4/">
        ...remaining contents of template packet...
        </template>
    endstream

    12 0 obj
    stream
        <xfa:datasets xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/">
        ...contents of datasets packet...
        </xfa:datasets>
    endstream

    13 0 obj
    stream
        <config xmlns="http://www.xfa.org/schema/xci/1.0/">
        ...contents of config node of XFA Data Package...
        <config>
    endstream

    14 0 obj
    stream
        </xdp:xdp>
    endstream
```

RECOMMENDATIONS:

1 **Validate:** Validate the form dictionary similar to validation methods presented earlier for form dictionaries (ensuring consistency and that they are correct). Validate the /XFA stream information or the array of references (referential integrity).

2 **Validate:** Validate the XDP packets against the appropriate schema.

3 **Remove**: Remove the entire /XFA item and its referenced data, and references to the XFA object throughout the document. This may result in an empty PDF file.

4 **Remove**: Remove any actions associated with an XFA form.

5 **Replace:** N/A

6 **External Filtering Required**: Pass the `/XFA` item to an external filter. This includes remote data connections.

7 **Review:** N/A

PDF 5.67: END

5.10 Document and Page Navigation

Links and navigation features in PDF files have the ability to reference other parts of the same document or possible locations outside of the document in the form of a URL or a collection pointing to attached files. Some examples of these are link annotations, bookmarks, and thumbnail images. While they may provide some benefit in large PDF documents, their contents should be inspected, particularly if they have the ability to invoke actions by the PDF viewer and access data external to the PDF file.

PDF 5.68: LINK ANNOTATIONS

OVERVIEW:

Link annotations are a special type of annotation that creates a hyperlink to another part of the PDF document, or references an action to be performed. In a hyperlink, a URI can be referenced to access a website. Internal references to sections, chapters, tables are often used in larger PDF documents. Link annotations may reference an object outside of the PDF document, another PDF file or a hyperlink are examples.

This can be useful for linking in the same document; however there are actions that can be associated with this annotation which can lead it to link outside the document. In the ideal case, a link annotation should point to a destination that is within the current file. If the annotation supports an action that references another file or a URI, then it becomes more of a concern.

CONCERNS:

Links in PDF files cause concern due to the destination of the link, which could be an external file. This could present a data attack risk. Various fields within the dictionary may contain reference information, which can present the risk of data disclosure. A Link Annotation can also reference an action object which may be malicious or present a data attack risk; the action object along with the annotation that references it should be examined.

PRODUCT: PDF-1.0

LOCATION:

Link annotations are identified by an object with the optional field `/Type /Annot` and the required `/Subtype /Link`. It also requires the `/Dest` entry in the dictionary to identify the destination that is to be displayed by the PDF viewer software when the link is invoked.

EXAMPLE:

```
4 0 obj
<<
    % This is optional
    /Type /Annot
    % This is required
    /Subtype /Link
    % Optional contents
    /Contents (Link to Table 5)
    %Destination is in object 3 0
    /Dest [3 0 R /FitR -4 399 199 533]
```

```
    % Alternatively as opposed to destination above, cannot be included when
Dest
    % is defined.
    %This could point to internal or external link
    % /A <<reference to action object>>
>>
endobj
```

RECOMMENDATIONS (INTERNAL LINKS):

1 **Validate**: Examine each link destination to check that they reference contents within the same PDF file.

2 **Validate**: Check for referential integrity and make sure that the action object associated to the correct annotation.

3 **Remove**: Remove the annotation dictionary and references to the annotation throughout the document.

4 **Replace**: N/A

5 **External Filtering Required**: N/A

6 **Review**: N/A

RECOMMENDATIONS (EXTERNAL LINKS):

1 **Validate**: Check for referential integrity and make sure that the action object associated to the correct annotation.

2 **Remove**: Remove the annotation dictionary and references to the annotation throughout the document.

3 **Replace**: Replace the Link annotation that points to an existing location in the file or an explicit destination

4 **External Filtering Required**: Pass all path text or fully qualified pathname information to an external filter.

5 **Review**: N/A

PDF 5.68: END

PDF 5.69: DOCUMENT OUTLINE

OVERVIEW:

A PDF document has the capability to present an outline structure of that document that allows a user to navigate through various sections and chapters of the document. In reader software the hierarchy of the document can be shown by clicking on the Bookmarks tab. Each element in the outline is considered a bookmark object or an outline item.

CONCERNS:

Document outline objects or bookmarks can be an issue because they can contain a link to an external action or source of data. This could potentially be a data attack risk or data disclosure risk if it identifies an external source of data. Bookmarks can be useful when they are used in large PDF documents to reference internal sections of the document purely for navigation. If they are used in any other manner, they should be carefully inspected.

PRODUCT: PDF-1.0

LOCATION:

Bookmark objects are denoted by a dictionary in the Document Catalog. The Outline dictionary which is references in the `/Outlines` entry in the Document Catalog, the root of the PDF document. The outline dictionary can contain entries for the `/First`, `/Last`, and `/Count` properties. `/First` and `/Last` refer to an outline object in the PDF document that contains entries in its dictionary such as `/Parent`, `/Prev`, `/Next` which describe the hierarchy of outline objects throughout the document. Then in an individual outline item, which can have a `/Dest` or `/A`, which represents a destination or action. These require caution because they may be used maliciously or reference information outside of the PDF document.

EXAMPLE (EXTERNAL):

```
21 0 obj
<<
     /Count 6
     /First 22 0 R
     /Last 29 0 R
>>
endobj

22 0 obj
<<
     /Title (Chapter 1)
     /Parent 21 0 R
     /Next 26 0 R
     /First 23 0 R
     /Last 25 0 R
     /Count 3
     /A
     <<
        /Type /Action
        /S /URI
        /URI (http://www.google.com)
     >>
>>
endobj
```

EXAMPLE (INTERNAL):

```
9950 0 obj
<<
     /Type/Catalog
     /Outlines 9951 0 R
     /Pages 9889 0 R
```

```
          /Names 9953 0 R
          /PageMode/UseOutlines
     >>
     endobj

     9951 0 obj
     <<
          /Count 822
          /Last 9971 0 R
          /First 9972 0 R
     >>
     endobj

     9972 0 obj
     <<
          /Dest(name1)  %Name tree for name destinations
          /Parent 9951 0 R
          /Title(title1)
          /Next 9973 0 R
     >>
     endobj
```

RECOMMENDATIONS:

1 **Validate**: Check for referential integrity and that each indirect reference is valid.

2 **Validate:** Check that destinations are within the current file.

3 **Remove**: Remove outline item, fix the remaining tree hierarchy.

4 **Remove**: Remove the outline dictionary entirely.

5 **Replace:** Remove action if present, replace with Destination to an internal location in the file.

6 **External Filtering Required**: Pass title information (free text) to an external filter.

7 **Review**: N/A

PDF 5.69: END

PDF 5.70: THUMBNAIL IMAGES

OVERVIEW:

PDF documents can contain small images, known as thumbnail images, to represents pages within the document in smaller format. They can be included in large document where a user wishes to see a miniaturized version of the page before clicking on it to view the full contents.

Thumbnail images are deprecated in ISO 32000-1 and should not be used in documents.

CONCERNS:

A thumbnail image just as any other image format can be used to hide data from the user. Thumbnail images also may not necessarily represent the actual content of the pages, which is a hidden data risk.

PRODUCT: PDF-1.0

LOCATION:
A thumbnail image is a XObject image defined by the Thumb entry in the page object. It provides an image dictionary with the parameters of `Width`, `Height`, `ColorSpace`, `BitsPerComponent`, and `Decode` entries being relevant to this object.

EXAMPLE:

```
%% Page object
20 0 obj
<<
    /Type /Page
    /Parent 15 0 R
    ...
    /Thumb 12 0 R
>>
endobj
%% Typical image dictionary followed by image stream
12 0 obj
<<
    /Type /XObject
    /SubType /Image
    /Width 76
    /Height 99
     ...
>>
stream
%Image contents, not readable
endstream
endobj
```

RECOMMENDATIONS:

1 **Validate**: Ensure referential integrity and validate the stream object and how it is referenced by the page object.

2 **Validate**: Ensure consistency and validate the parameters in a similar manner to the XObject validation method presented in this document.

3 **Remove**: Remove the `/Thumb` entries in page objects, along with the XObject image streams.

4 **Replace**: N/A

5 **External Filtering Required**: Extract and decode the image and have an external filter examine it.

6 **Review**: Extract and decode the image and present for manual review.

PDF 5.70: END

PDF 5.71: NAMED DESTINATIONS

OVERVIEW:

Destinations are used often in action objects (to GoTo) or other `/Dest` fields in many dictionaries that reference a position or location in the PDF document. Destinations can be specified by explicit syntax defining the page number, the view of the document. This syntax is called Destination Syntax by the PDF specification. As opposed to listing this information for destination, name objects and byte strings can be used to represent a location within the PDF file.

CONCERNS:

Destinations are commonly used with actions, particularly GoTo actions, which cause document level navigation. It has similar risks presented along with those actions. A data attack risk, data hiding, and data disclosure risks are all possible.

PRODUCT: PDF-1.1

LOCATION:

Named destinations exist ultimately through the Document catalog through the /Dests field. There are other named objects that can reside from this as defined by the standard.

EXAMPLE:

```
%Document catalog, defines Names object in 63 0
61 0 obj
<<
    /Names 63 0 R
    /Pages 58 0 R
    /Type/Catalog
>>endobj
%Names Dictionary
63 0 obj
<<
    /Dests 28 0 R  %Named Destinations
>>endobj

%Name tree for Destinations in the document.
28 0 obj
<<
    /Kids[32 0 R]
>>endobj

%Example continues with Name Destions, which will be covered in the next
section.
%Names, array maps objects to name strings ()
32 0 obj
<<
    /Limits[(F)(P.5)]
    /Names[(F)33 0 R(G1002871)34 0 R(G1013045)35 0 R(G1013343)36 0
    R(G1013390)37 0 R(G1013741)38 0 R(G1014217)39 0 R(G1014219)40 0
    R(G1014327)41 0 R(I1.1013412)42 0 R(I1.1013424)43 0 R(I1.1013590)44 0
    R(I1.1013752)45 0 R(I1.1014342)46 0 R(L)47 0
    R(M9.17908.Tablelargetitle.Table.1.Common.entries.in.all.build.properties.d
```

```
        ictionary)48 0
        R(M9.30470.Tablelargetitle.Table.2.Entries.in.the.build.properties.dictiona
        ry)49 0
        R(M9.51987.Tablelargetitle.Table.3.Common.entries.in.handler.build.data.dic
        tionaries)50 0
        R(M9.64792.Tablelargetitle.Table.4.Entries.in.build.data.dictionary)51 0
        R(P.1)52 0 R(P.2)53 0 R(P.3)54 0 R(P.4)55 0 R(P.5)56 0 R]
    >>endobj

    %Name Destination through the name defined in 32 0
    104 0 obj
    <<
        /Parent 103 0 R
        /Dest(G1013045)  %In Name dictionary, this is 35 0 R
        /Title(3.1 Build Data Dictionary)
    >>endobj

    %Destination in File that is referenced through a Name
    35 0 obj
    <<
        %This is the Destination Syntax, [page /XYZ left top zoom]
        %This destination is page 1 0 , positioned 72,724 from upper left corner,
    no
        %zoom
        /D[1 0 R/XYZ 72 724 null]
    >>endobj
```

RECOMMENDATIONS:

1 **Validate**: Check that all destinations are within the current PDF file and that both coordinate values and zoom values are acceptable.

2 **Remove**: Remove the destination, from the name tree, the actual dictionary, and Dests entry in the catalog.

3 **Replace**: Replace destinations with a harmless and default location, such as the beginning of the document.

4 **External Filtering Required**: Pass all text strings to external filter.

5 **Review**: N/A

PDF 5.71: END

5.11 Digital Signatures and Legal Attestation

PDF 5.72: DIGITAL SIGNATURES

OVERVIEW:

Digital signatures are used to sign PDF documents and attest the identity of the author such that they cannot deny the signature. They are used to authenticate the content of the document to

prevent undetected changes, while still granting a certain level of permissions on the content of the document. There are two actions that are taken with digital signatures: adding a digital signature to a document and the second is checking to ensure the validity of the signature. Signatures are created by computing a digest on the entire data, and then the digest is stored within the file. Signature verification is performed when the digest is recomputed using the same algorithm and compared against the one in the document.

PDF files can contain different types of digital signatures [1]:

- Approval Signatures: These appear in signature form fields.
- Modification Detection and Prevention (`MDP`) signatures. These signatures allow detection of disallowed changes to the document as indicated by the author.
- Usage Rights Signatures that enable additional interactive features that is not available in a PDF viewer application.
- The `/Sig` entry in the catalog of a Forms Data Format (`FDF`) file, which is the file format used for interactive form data.

Multiple signatures can exist in the PDF. This feature is supported in writers through the use of incremental updates (PDF 4.16:), as the second signature covers the entire file (excluding the actual signature), including the previous signature. The first signature should cover the entire original file, excluding the signature in the `/Contents` field. This is identified in the `/ByteRange` field.

CONCERNS:

A digital signature can be the source of concern since they contain information and fields in their dictionaries that are not viewable in the actual document, which might introduce a hidden data risk or data disclosure risk. They may also reveal identities of persons who generated the document which might be undesirable, which would present a data disclosure risk. Due to the complexity of the algorithm implemented to check digital signatures over a region of supplied data, a data attack risk may also be present.

In a cleansing application, the contents of the document will most likely be modified, thus breaking a digital signature since the document is no longer the original as it was signed. If the contents of the entire PDF file can be validated and the contents are not modified, a validated digital signature can remain in the document.

PRODUCT: PDF-1.3

LOCATION:

Signature information in a PDF is contained in a Signature Dictionary, which is defined by the optional `/Type /Sig` field. The dictionary provides a required `/ByteRange` entry that specifies the byte range for the message digest calculation, a required `/Filter` entry that contains the signature handler (`Adobe.PPKLite, Entrust.PPKEF, CICI.SignIt, VeriSign.PPKVS`, or other approved handlers), and a required `/Contents` entry that contains the signature value. Depending on the type of signature, optional entries may be used (e.g., use of `/Reference` for MDP and usage rights signatures to specify Signature Reference dictionaries).

The dictionary contains optional fields such as `/Name`, `/M (date)`, `/Location`, `/Reason`, and `/ContactInfo` that provides information about the signing and signer of the document. This information is not directly present in the document's view.

In the Signature dictionary, there is an important field called `/ByteRange`. This is an array of four numbers that indicate which bytes are to be used in the calculation. The first number if always 0 (for the beginning of the file). The second number is the length of the first stream of bytes. The third number is the byte location of the next starting location of the bytes to be used. The final number is the size of the next block of data. This range should cover the entire file, and leave a small byte range in between the blocks of data where this actual signature value should reside. In an incremental update file, existing signatures may exist in the document. When signed again, the byte range should cover the entire previous version of the document and leave only a gap for the latest value of the signature.

The Document Catalog of the PDF file specifies a Permissions Entry, which then contains a DocMDP entry. This entry points to the Signature dictionary for the document, also referenced by a Field Dictionary in the document as well. This is only true for Certified Documents, not documents that are simply signed.

EXAMPLE:

```
73 0 obj %% Signature form field
<<
    /TM(form1[0].#subform[0].SignatureField1[0])
    /FT/Sig
    /Subtype/Widget
    /TU(Sign Here)
    ...
    /V 150 0 R %% Specifies location of signature dictionary
>> endobj

%Signature dictionary
150 0 obj
<<
    %Optional but indicates a Signature Dictionary
    /Type /Sig

    %Required, signature handler
    /Filter /Adobe.PPKLite

    /SubFilter /adbe.pkcs7.detached

    %Required
    /Contents <signature>

    %Optional, indicates person who signed the document. (Identity)
    /Name (Signing Username)
    /Location (Office)
    /Reason (I agreed to sign this)
    /ContactInfo (Phone: 555-1234)

    %Required, specifies the starting byte offset and length in bytes to
    %compute the digest, each pair can represent a different range that should
be
    %used in the computation. In this case, there are two byte ranges.
    /ByteRange [ 0 356503 358995 15198 ]

    %Required if /SubFilter=adbe.x509.rsa_sha1
    %Byte strings that represent the X.509 certificate chain
```

```
    /Cert <cert array of byte string>
>>
endobj
```

RECOMMENDATIONS:

1 **Validate**: Check for consistency and that the filter is from a list of approved signature handlers.

2 **Validate**: Check that the byte range is accurate and covers the correct content. Check that there is no overlap and it does not contain the signature value. In the case of incremental updates, ensure that it does not cover the most recent signature value. Check that it covers the entire file, except for the signature value located in `/Contents`.

3 **Validate**: Check that all required fields are implemented properly.

4 **Remove**: If the document's contents have not been altered, optional fields such as `/Name`, `/Location`, `/M`, `/Reason`, and `/ContactInfo` should be removed since they contain information that could be hidden. This may require re-signing since the contents have now changed.

5 **Remove**: Remove the entire signature dictionary and references to it from a Form Field and Document Catalog permissions.

6 **Replace**: N/A

7 **External Filtering Required**: Pass all text strings to an external filter.

8 **Review**: N/A

9 **Reject**: Fail documents that contain digital signatures.

PDF 5.72: END

PDF 5.73: LEGAL ATTESTATION

OVERVIEW:

A legal attestation states the content and intent of a PDF document so that the recipient can determine whether to trust it. Since PDF documents and its syntax can allow for a PDF document with a differently rendered view from what its internal content implements, it is important having a feature that allows a user to trust the document.

The Legal Attestation dictionary contains various entries that provide a count for each type of action, text state parameter that may hide information, optional content, embedded versus non-embedded fonts.

CONCERNS:

Legal attestation information can provide useful information if it can be trusted. While the information listed here is for reference and is usually associated with a digital signature that has risks discussed in the earlier section. It provides a `/Attestation` field which is a text field which could introduce a data disclosure risk.

PRODUCT: PDF-1.3

LOCATION:
Legal attestations are identified by a legal attestation dictionary that is specified in the `/Legal` entry of the Document Catalog. The attestation dictionary consists of a series of optional entries that specify features such as the number of annotations, number of external streams, number of various action types, and number of font types of various types in the document. The dictionary may also contain an optional `/Attestation` entry which is a statement from the document author explaining the presence of any of the other entries in the legal attestation dictionary or the presence of any other content affecting the legal integrity of the document.

EXAMPLE:

```
1 0 obj
<<
    /Type /Catalog
    /Pages 2 0 R
    /Legal %Legal Attestation Dictionary
    <<
       /JavaScriptActions 10 %There are 10 JavaScript actions in this PDF
       /OptionalContent true %Optional content exists.
       /Attestation (This can be anything)
    >>
>>
endobj
```

RECOMMENDATIONS:

1 **Validate**: Ensure consistency and validate that each entry in the dictionary and its value is accurate with the remainder of the document.

2 **Remove**: Remove the Legal Attestation dictionary and reference from the Document Catalog.

3 **Replace**: N/A

4 **External Filtering Required**: Pass all string data to an external filter.

5 **Review**: N/A

PDF 5.73: END

5.12 PDF Structure

There are numerous elements in the PDF structure that this report has documented. However, there are several uncategorized objects that are presented in this chapter that may not fit in well with the remainder of the objects.

PDF 5.74: PAGE-PIECE DICTIONARIES

OVERVIEW:

Page-Piece dictionaries are used to hold private product data, which may be related to a Form XObject or a page. This is normally application-specific data that PDF workflow applications may add for internal use in the processing of a PDF document. The standard indicates that this information may be ignored by a conforming reader. This information can be created by a PDF writer or some type of markup application that modifies changes to the PDF file, such as PDFAnnotator [17].

CONCERNS:

This information is not represented in the viewable context of the PDF file and could be a source of a hidden data risk.

PRODUCT: PDF-1.3

LOCATION:

Page-piece dictionary objects can be inserted into PDF files at multiple objects. A `/PieceInfo` entry can be added to the Document Catalog, a page object, and in a XObject (typically a Form XObject).

A page-piece dictionary contains a key and value pair. The Key is a produce name and the value is a data dictionary. The data dictionary contains a LastModified date and a private data field which can be of any data type.

EXAMPLE:

```
1 0 obj
<<
    /Type /Catalog
    /Pages 2 0 R
    /PieceInfo 3 0 R
>>
endobj

3 0 obj
<<
   %Page-Piece dictionary.
   /Key <<
              /LastModified (date)
              /Private (Name of Application and Version)
        >>
>>
endobj
```

RECOMMENDATIONS:

1 **Validate**: Validate the contents of a page-piece dictionary to ensure required fields and correct data types are present. Validate the data dictionary entries that exist in the Page-piece dictionary.

2 **Remove**: Remove all page-piece dictionaries and references throughout the document.

3 **Replace**: N/A

4 **External Filtering Required**: N/A

5 **Review**: N/A

PDF 5.74: END

PDF 5.75: WEB CAPTURE

OVERVIEW:

The PDF specification allows for the creation of PDF document from an Internet-based or local HTML, PDF, GIF, JPEG, or ASCII file through a method called Web Capture. This allows information from a website to be retrieved once and possibly added to an existing PDF file, update a PDF file with information that was gathered from an online source, and to save locally and preserve information that was pulled from the Internet. This is simply additional data that explains how the content was captured and imported into the PDF file when it was written. It does not imply that an action will occur based upon user activity with the document. When a PDF creator converts the content into PDF, it uses this information similar to Metadata such that the source is documented.

CONCERNS:

This introduces a concern since the information that has been retrieved from another file may not immediately visible within the syntax of the PDF document. It also presents a data disclosure risk because it can reveal information about the source of the content (Source URL).

PRODUCT: PDF-1.3

LOCATION:

The document catalog contains a field called `/SpiderInfo` which contains a reference to the Web Capture information dictionary. The Web Capture information dictionary is an indirect object within the PDF document that contains the mandatory field `/V` (Web capture version) and optional `/C` entry (array of references to Web Capture command dictionaries). Most of the content is in the web capture command dictionaries which represent a command to retrieve pieces of source data to create new or modify content in a PDF file.

A web capture command dictionary requires the field `/URL` which identifies the source data, this can be online or a local file. Additional parameters include the number of levels of pages to retrieve from the source, the field `/P` for data that was posted to the URL, and optional additional HTTP request headers.

EXAMPLE [1]:

```
%Document Catalog, pointing to Spider Info Dictionary
9950 0 obj
<<
   /Type/Catalog
   /Pages 9889 0 R
   /SpiderInfo 9937 0 R
   /Names 9953 0 R
```

```
>>
endobj

%Web Capture Information Dictionary
9937 0 obj
<<
     /V 1.25  %Web Capture version# 1.25
     /C 9938 0 R    %Web capture command dictionaries
>>
endobj

%Web capture command dictionaries array, only one, in 9942
9938 0 obj
    [9942 0 R]
endobj

%Web Capture Command Dictionary
9942 0 obj
<<
     /S 9943 0 R    %Command settings
     /URL(http://www.adobe.com/)  %URL from which data was requested

     %/P optional string or stream data that was posted to URL
     %/H (optional additional header information)

>>
endobj

%Command settings dictionary
9943 0 obj
<<
    /G 9944 0 R  %Global conversion engine settings
>>
endobj

%Global conversion Engine Settings
9944 0 obj
<<
     /AL 0
     /S 0
     /M[10.080002 10.080002 26.0 36.0]
     /SU 1
     /PO 0
     /AS 1
     /CB 0
     /AH 0
     /AT 70
     /PS[792.0 612.0]
>>
endobj
```

RECOMMENDATIONS:

1 **Validate**: Ensure the remote content is sanitized before it can be used in conjunction with a PDF document.

2 **Validate**: Check that the contents of SpiderContentSet dictionary are properly implemented.

2 **Remove**: Remove all SpiderInfo dictionaries, web capture command dictionaries, web capture content sets, and references to them such as those that exist in the Document catalog shall be removed.

3 **Replace**: N/A

4 **External Filtering Required**: Extract the URL information and the pathname and pass to external filter.

5 **Review**: N/A

PDF 5.75: END

6. ACRONYMS

Table 6-1. Acronyms

Acronym	Denotation
AA	Additional Action
ADBC	Acrobat Database Connectivity
AES	Advanced Encryption Standard
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BMP	Bitmap
CCITT	Committee Consultatif International Telephonique et Telegraphique
CID	Character Identifier
CMap	Character Map
CSS	Cascading Style Sheet
CTM	Coordinate Transformation Matrix
DCT	Discrete Cosine Transform
DOM	Domain Object Model
DTG	Data Transfer Guidance
EOF	End of File
EOL	End of Line
EPS	Encapsulated PostScript
FDF	Forms Data Format
GIF	Graphics Interchange Format
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
ICC	International Color Consortium
ISG	Inspection and Sanitization Guidance
ISO	International Organization for Standardization
JPEG	Joint Photographic Experts Group
JS	JavaScript
MCD	Media Clip Data
MCS	Media Clip Section
MDP	Modification Detection and Prevention
OBJ	Object (PDF Object)
OCG	Optional Content Group
OCMD	Optional Content Membership Dictionary
OGC	Open Geospatial Consortium
PDF	Portable Document Format
PDL	Page Description Language
PKCS	Public Key Cryptographic Standard
PNG	Portable Network Graphics
PRC	Produce Representation Compact
PS	PostScript
RC4	Rivest Cipher 4
RNG	RELAX: Next Generation language
TIFF	TIFF
U3D	Universal 3D

URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XFA	XML Forms Architecture
XFDF	XML Forms Data Format
XHTML	eXtensible HyperText Markup Language
XML	eXtensible Markup Language
XOBJECT	External Object
xref	Cross Reference Table

7. REFERENCED DOCUMENTS

[1] International Standard ISO 32000-1. Document management – Portable Document Format – Part 1: PDF 1.7. First Edition, 2008-07-01.

[2] Onley, Dawn S. "PDF user slip-up gives DOD lesson in protecting classified information." Government Computer News. May 16, 2005. http://www.gcn.com/print/24_11/35808-1.html.

[3] Wait, Patience. "White House accidentally exposes data in PDF file". Post-Newsweek Media, Inc. December 5, 2005. http://www.gcn.com/online/vol1_no1/37688-1.html.

[4] Naraine, Ryan. "Hacker Discovers Adobe PDF Back Doors." eWEEK.com. September 15, 2006. http://www.eweek.com/article2/0,1759,2016606,00.asp

[5] "Hidden Data and Metadata in Adobe PDF Files: Publication Risks and Countermeasures." Enterprise Applications Division of the Systems and Network Analysis Center (SNAC), Information Assurance Directorate. 7/27/2008. http://www.nsa.gov/ia/_files/app/pdf_risks.pdf

[6] Adobe Extensions to ISO 32000-1. http://www.adobe.com/devnet/pdf/pdf_reference.html

[7] Open Geospatial Consortium Inc, OGC 08-139r2. GeoPDF Encoding Best Practice Version 2.2.

[8] JavaScript for Acrobat API Reference, Version 8.1. http://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/js_api_reference.pdf

[9] XML Forms Architecture (XFA) Specification, Version 3.1. http://partners.adobe.com/public/developer/en/xml/xfa_spec_3_1.pdf

[10] Map Projections – A Working Manual. USGS.

[11] Rosenthal, Leonard. "Refrying PDFs – the good, the bad and the ugly. October 25, 2008.

[12] Wolf, Julia. "OMG-WTF-PDF", 27th Chaos Computer Congress, Berlin, Germany. http://blog.fireeye.com/files/27c3_julia_wolf_omg-wtf-pdf.pdf

[13] Wolf, Julia. "World's Smallest PDF". June 21, 2010. http://blog.fireeye.com/research/2010/06/that-pdf-thing.html

[14] Adobe Security Bulletin. Security Updates available for Adobe Reader and Acrobat version 9 and earlier. March 24, 2009. http://www.adobe.com/support/security/advisories/apsa09-01.html

[15] "PDF Reference, fourth edition: Adobe Portable Document Format version 1.5." Adobe Systems Incorporated. 1985-2003. http://partners.adobe.com/public/developer/en/pdf/PDFReference15_v6.pdf.

[16] Aronshtam, Boris. "PDF/VT Transparency Guide" http://www.pdfvt.com/PDFVT_TransparencyGuide.html

[17] Grahl Software Design. PDF Annotator. http://www.ograhl.com/en/pdfannotator/

[18] GeoPDF Blog. Online reference available at: http://geopdf.blogspot.com/

Appendix A

GeoPDF

APPENDIX A: GEOPDF

GeoPDF is an extension which allows association of an arbitrary coordinate reference system to a PDF illustration or graphic [7]. These graphics are typically maps which overlay a coordinate system on top of the area as a point of reference. Since using a 2D coordinate system on a 3D representation can create distortion, there are methods to minimize this distortion such that the map and coordinate system is accurate. A Map projection is used to convert an ellipsoidal coordinate system to a 2D space. PDF 2.0 (ISO 32000-2) will contain additional information regarding Geo-related constructs and functionality; however the specification is unavailable at the time of this writing.

GeoPDF also supports three different coordinate systems. They are known as PDF, Projected, and Geodetic. For full definitions, the GeoPDF specification and Map Projections – A Working Manual should be consulted.

GeoPDF extensions in a PDF document are first introduced in the Page Object (PDF 4.8:). GeoPDF also introduces an object called a map frame and each map frame is associated with a page object in the PDF document. This allows the coordinate system in PDF for the entire page to map to a known coordinate reference system that is being specified. There can be multiple map frame dictionaries per page, as map frames can overlap each other or exist on separate areas of the page. Content or the map illustrations must be placed in the proper location in PDF space on the page and then associated with a map frame. The Page Dictionary introduces a new entry called `/LGIDict` which specifies an array of indirect references (for multiple map frames per page) or a dictionary object that implements the map frame dictionary.

The Map Frame dictionary is presented in this section since it is the main object that is introduced with GeoPDF. Map Frame dictionaries includes Projection and Display dictionaries which describe the algorithm and parameters that are required to define the coordinate reference system used in each mapframe on the page [7 These dictionaries Map Frame and Projection/Display are further analyzed in this Appendix and summarize how a coordinate system can be mapped onto a page of a PDF document.

PDF APP.A.1: GEOPDF – MAP FRAME DICTIONARIES

OVERVIEW:

GeoPDF allows multiple map frames, which contain a coordinate reference system, to associate itself with a PDF page. Map frames define how the coordinate system relates with the specific region of a PDF page, to overlay on a map image. GeoPDF Map Frames support three different types of coordinate systems: projected, geodetic, and unregistered Cartesian. All of these are transformed into the PDF coordinate system using a transformational matrix. Transformations between coordinate spaces have been discussed in PDF 5.2:. The map frame dictionary implements information about the transformation matrix and parameters required for a map projection [7].

CONCERNS:

GeoPDF presents a method to represent map data within an area on a PDF page. With this feature, additional overhead and dictionaries have been defined to support this functionality. Within these dictionaries are several optional fields (Description) and some fields that duplicate information (CTM/Registration) that could possibly provide data that is never actually represented in the PDF document. This may introduce a hidden data risk.

Map representations can also be further enhanced with the use of Annotations (5.7.1, PDF 5.39:) such that further map data can be shown through user interaction. There are several concerns with annotations starting in 5.7.1.

A data disclosure risk is also present if the coordinate information reveals location or county information, such as the Datum field, which is discussed in the next section PDF APP.A.2:.

PRODUCT: GEOPDF, PDF-1.7

LOCATION:

Supporting the addition on content per page, the page dictionary in PDF has been augmented. The entry `LGIDict` has been added to each page, and it represents an array (multiple map frames) or dictionary (single map frame). Each one references a map frame dictionary.

The map frame dictionary contains the required `/Type /LGIDict`, indicating the type of dictionary. It also contains an optional `/Version` number (which should equal 2.1). The dictionary also defines either the Coordinate Transformation Matrix (`/CTM`), or an array for `/Registration` point pairs. If both are defined, the CTM takes precedence. The CTM defines how to map from the PDF coordinate system of the projected coordinate system. While the Registration entry provides an array of four values: PDF x, PDF y, map x, and map y, for each point pair listed in the array.

The map frame dictionary defines the dictionary called `/Projection`, which is required if the Map Frame is a geodetic or projected coordinate system. An optional `/Display` dictionary can also be defined in the map frame dictionary. These dictionaries define the aspects of the coordinate reference system. This information should be examined in addition to map frame data since it includes relevant information about the entire map frame.

Lastly, an array entry entitled `/Neatline` is required when there are multiple maps per the page. This field defines the boundaries of the map frame, indicated by x and y points in the PDF page.

EXAMPLE (PAGE DICTIONARY): INAUGURATION PARADE EXAMPLE [18]

```
46 0 obj<<
    /Annots 91 0 R
    /Contents 136 0 R
    /CropBox[0 0 1560 948]
```

```
        /LGIDict[
        <<
            %The CTM takes precedence over the Registration defined below.
            /CTM[(4.043025)(0.115448)(0.085453)(4.082230)(319995.438565)
                 (4304721.730650)]
            /Description(My Map Frame)
            /Display
            <<
               /Datum(WE)
               /ProjectionType(GEODETIC)
               /Type/Projection
               /Units(M)
            >>
            /LGI(s4wnanjc7c4qf7yh)

    /Neatline[(0)(948)(1557.20166015625)(949.9636840820313)(1557.20166015625)
                      (1.504150390625)(0)(3.46783447265625)(-1.96368408203125)
                      (946.0363159179688)(0)(948)(0)(948)(0)(948)]

            /Projection
            <<
               /Datum(WE)
               /Hemisphere(N)
               /ProjectionType(UT)
               /Type/Projection
               /Units(M)
               /Zone 18
            >>
           %PDF x, PDF y, Map x, Map y
           /Registration[[(827.625732421875)(623.243408203125)(323388.7111085421)
                          (4307386.431240459)]
                         [(851.5419311523438)(415.55029296875)
                          (323484.4251785617) (4306472.773233708)]
                         [(534.3378295898438)(417.4383850097656)
                          (322192.3211218527)(4306483.939804825)]
                         [(827.625732421875)(183.3115692138672)
                          (323351.769059125)(4305587.864468223)]]

           /Type/LGIDict
           /Units(M)
           /Version 2
         >>
         ] %end of LGIDict array
         %remainder of the Page dictionary
         /MediaBox[0 0 1560 948]
         /Parent 38 0 R
         /Resources 135 0 R
         /Rotate 0
         /Type/Page
    >>
    endobj
```

EXAMPLE (LGIDICT): ST. MARTIN EXAMPLE [18]

```
    136 0 obj<<
         /CTM[(3.5278261913)(0.0000000000)(0.0000000000)
              (3.5278261913)(489354.9292752144)(1996337.2214303040)]

         /Description(Layers)
         /Display
         <<
```

```
                /Datum(WE)
                /Hemisphere(N)
                /ProjectionType(UT)
                /Type/Projection
                /Units(M)
                /Zone(20)
           >>
           /LGI(qzav2uhupka66jnw)
           /Neatline[(28.9575000000)(121.5347708363)(28.9575000000)(602.7627291637)

(835.0312500000)(602.7627291637)(835.0312500000)(121.5347708363)]
           /Projection
           <<
                /CentralMeridian(-63.00000)
                /Datum(WE)
                /FalseEasting(500000.00000)
                /FalseNorthing(0.00000)
                /OriginLatitude(0.00000)
                /ProjectionType(TC)
                /ScaleFactor(0.99960)
                /Type/Projection
           >>
          /Registration[[(28.9575000000)(121.5347708363)
                         (489457.0863021494)(1996765.9749780153)]
                        [(835.0312500000)(602.7627291637)
                         (492300.7743895258)(1998463.6635733924)]]
          /Type/LGIDict
          /Version(2.1)
     >>     endobj
```

RECOMMENDATIONS (MAP DICTIONARY):

1 **Validate**: Check that the required fields are present in the LGI dictionary.

2 **Validate**: Check that only fields that take precedence are present (CTM versus Registration).

3 **Validate**: Check that a Projection dictionary is referenced if it is either a geodetic or projected coordinate system.

4 **Validate**: If there are multiple map frames per page, validate that a Neatline array exists.

5 **Validate**: Check that they are on independent areas on the PDF page.

6 **Remove**: Remove additional information, such as CTM or registration if either field is redundant.

7 **Replace**: N/A

8 **External Filtering Required**:N/A

9 **Review**: N/A

PDF APP.A.1: END

PDF APP.A.2: GEOPDF – PROJECTION AND DISPLAY DICTIONARIES

OVERVIEW:

Within the Map Frame dictionary defined in the earlier section, Projection and Display dictionaries are defined to describe parameters and algorithm information about the coordinate reference system.

A projection dictionary is used to define parameters such as linear (unit in meters) or angular (in degrees). Latitude or longitude parameters are specified in the angular parameters and there are several fields in the Projection dictionary which indicate the scope of these values. These values indicate the range and properties of the map frame in the parent object.

A Display dictionary is used when an application utilizing the coordinate system displays non-PDF coordinate values.

CONCERNS:

These dictionaries are nested within the map frames, but provide additional information useful for GeoPDF. There are numerous optional parameters that can be used in either of these dictionaries. Based upon the projection type that is deployed, some may not be necessary, which could allow a location for hidden data. A data disclosure risk is also present if the coordinate information reveals location or county information, such as the Datum field.

PRODUCT: GEOPDF, PDF-1.7

LOCATION:

Project or Display dictionaries are defined within Map Frames (LGIDICT) dictionaries. Project and display dictionaries are very similar and have similar entries.

Both implement a `/Type` entry indicating `/Projection` or `/Display` indicating the type of dictionary. Although the examples in published GeoPDF examples both indicate Projection, despite being referenced as both (Project/Display) in the parent object.

Another required field in either dictionary is the `/ProjectionType`, which is a string that indicates the projection algorithm used to project geodetic coordinates into Cartesian coordinates [7]. The example below indicates a value of "TC" which means "Transverse Mercator". The GeoPDF standard lists a number of these string values. The table also indicates required parameters that must be included in the dictionary based upon the Projection Type. In this example, the fields `OriginLatitude, CentralMeridian, ScaleFactor, FalseEasting, and FalseNorthing` must all be defined.

Another optional field is the Datum field which can either be a string or a dictionary. This represents the source geodetic coordinate system. The value string is a GeoPDF code or a custom dictionary, which is defined in the GeoPDF specification. There is also a field called /Units which is an optional string that display linear measurements in the coordinate system, which may not make sense when used with angular coordinates.

EXAMPLE (PROJECTION):

```
%LGIDict
43 0 obj
<<
```

```
…
    /Projection
    <<
        /CentralMeridian(-63.00000)
        /Datum(WE)
        /FalseEasting(500000.00000)
        /FalseNorthing(0.00000)
        /OriginLatitude(0.00000)
        /ProjectionType(TC)
        /ScaleFactor(0.99960)
        /Type/Projection
    >>
…
>>
endobj
```

RECOMMENDATIONS:

1 **Validate**: Validate that the two required parameters Type and ProjectionType are present and valid. Based upon the ProjectionType, ensure that the other required parameters are also defined and are acceptable values for each parameter. The appendix of the GeoPDF specification defines all of these values.

2 **Validate**: Check that the correct datum is used as well, either a valid string or a dictionary.

3 **Remove**: Remove extra parameters that are not necessary (based upon ProjectionType).

4 **Replace**: N/A

5 **External Filtering Required**:N/A

6 **Review**: N/A

PDF APP.A.2: END

Appendix B

Summary of Risks per Feature

APPENDIX B: SUMMARY TABLE

Table B-1. Summary of Risks

PDF Feature	ISO Section	ISG Section	Attack	Hiding	Disclosure
Header	7.5.2	PDF 4.1:	1	1	0
Traditional Xref	7.5.4	PDF 4.2:	1	1	0
Trailer	7.5.5	PDF 4.3:	1	1	0
Basic Objects	7.3	PDF 4.4:	0	1	0
Unused Objects	N/A	PDF 4.5:	0	1	1
Indirect References	7.3.10	PDF 4.6:	0	1	0
Document Catalog	7.7.2	PDF 4.7:	1	0	1
Page Tree/Page Object	7.7.3	PDF 4.8:	1	1	1
Name Dictionary/Tree	7.7.4	PDF 4.9:	1	1	1
Streams	7.3.8	PDF 4.10:	1	1	1
Stream Filters	7.4	PDF 4.11:	1	1	1
XObjects	8.9.5	PDF 4.12:	1	1	0
Object Streams	7.5.7	PDF 4.13:	1	1	0
Cross-Reference Streams	7.5.8	PDF 4.14:	1	1	0
Object Revisions	N/A	PDF 4.15:	0	1	1
Incremental Updates	7.5.6	PDF 4.16:	1	1	1
Linearized PDF	F.3	PDF 4.17:	0	1	0
Encryption	7.6	PDF 4.18:	1	0	1
Comment Strings	7.2.3	PDF 4.19:	1	1	0
Illegal Content	N/A	PDF 4.20:	1	1	0
Malformed PDF	N/A	PDF 4.21:	1	0	0
Metadata	14.3.2	PDF 5.1:	0	1	1
Transformations	8.3	PDF 5.2:	1	1	0
Content Streams	7.8	PDF 5.3:	0	1	0
Embedded File Streams	7.11.4	PDF 5.4:	1	1	1
External File Spec.	7.11.3	PDF 5.5:	1	0	1
Embedded Data Streams	7.9.3	PDF 5.6:	1	1	0
Collections	12.3.5	PDF 5.7:	1	1	1
Text Content Streams	7.8	PDF 5.8:	0	1	0
Simple Fonts	9.6	PDF 5.9:	1	1	1
Type 3 Fonts	9.6.5	PDF 5.10:	1	1	0
Character Encoding	9.6.6	PDF 5.11:	0	1	0
Composite Fonts	9.7	PDF 5.12:	1	1	1
Font Descriptors	9.8	PDF 5.13:	1	1	1
Graphics Objects	8.2	PDF 5.14:	1	1	0
Image XObjects	8.9.5	PDF 5.15:	1	1	0
PostScript XObjects	8.8.2	PDF 5.16:	1	1	0
Form XObjects	8.10	PDF 5.17:	1	1	0
Group XObjects	8.10.3	PDF 5.18:	1	1	1
Reference XObjects	8.10.4	PDF 5.19:	1	1	0
Inline Images	8.9.7	PDF 5.20:	1	1	0

Text Coloring	8.6	PDF 5.21:	0	1	0
Image Coloring	8.6	PDF 5.22:	0	1	0
Transparency	11	PDF 5.23:	1	1	1
Rendering	10	PDF 5.24:	0	1	0
Positioning	7.7.3.3	PDF 5.25:	1	1	0
Masking	8.9.6	PDF 5.26:	1	1	1
Optional Content Groups	8.11	PDF 5.27:	0	1	1
Set OCG State Actions	12.6.4.12	PDF 5.28:	0	1	1
Sound/Movie Objects	13.3/13.4	PDF 5.29:	1	1	1
Sound Actions	12.6.4.8	PDF 5.30:	1	1	1
Movie Actions	12.6.4.9	PDF 5.31:	1	1	1
Rendition/Media Clips	13.2.3	PDF 5.32:	1	1	1
Rendition Actions	12.6.4.13	PDF 5.33:	1	0	0
3D Artwork/Presentation	13.6	PDF 5.34:	1	1	1
GoTo 3D View Action	12.6.4.15	PDF 5.35:	1	1	0
Alternate Presentations	13.5	PDF 5.36:	0	1	0
Rich Media	Supplement	PDF 5.37:	1	1	1
Rich Media Execute Actions	Supplement	PDF 5.38:	1	0	0
Text/Freetext Annotations	12.5.6.4/12.5.6.6	PDF 5.39:	0	1	1
Line Markup Annotations	12.5.6.7	PDF 5.40:	0	1	1
Text Markup Annotations	12.5.6.10	PDF 5.41:	0	1	1
Graphical Markup Annotations	12.5.6.8-12.5.6.9	PDF 5.42:	0	1	1
Pop-up Annotations	12.5.6.14	PDF 5.43:	0	1	1
File Attachment Annotations	12.5.6.15	PDF 5.44:	0	1	1
Screen Annotations	12.5.6.18	PDF 5.45:	1	1	0
Printer's Mark Annotations	12.5.6.20	PDF 5.46:	1	1	1
Watermark Annotations	12.5.6.22	PDF 5.47:	0	1	1
Redaction Annotations	12.5.6.23	PDF 5.48:	0	1	1
Projection Annotations	Supplement	PDF 5.49:	0	1	1
Appearance Streams	12.5.5	PDF 5.50:	0	1	0
Rich Text Strings	12.7.3.4	PDF 5.51:	1	0	0
Triggers	12.6.3	PDF 5.52:	1	1	1
GoTo Actions	12.6.4.2	PDF 5.53:	1	0	1
Launch Actions	12.6.4.5	PDF 5.54:	1	0	1
Thread Actions	12.6.4.6	PDF 5.55:	1	0	1
URI Actions	12.6.4.7	PDF 5.56:	1	0	1
Hide Actions	12.6.4.10	PDF 5.57:	0	1	0
Named Actions	12.6.4.11	PDF 5.58:	1	0	0
Transition Actions	12.6.4.14	PDF 5.59:	0	1	1

JavaScript Actions	12.6.4.16	PDF 5.60:	1	0	0	
Functions	7.10	PDF 5.61:	1	0	0	
Forms Data Format	12.7.7	PDF 5.62:	1	1	1	
AcroForms	12.7	PDF 5.63:	1	1	1	
Submit-Form Actions	12.7.5.2	PDF 5.64:	1	0	1	
Import-Data Actions	12.7.5.4	PDF 5.65:	1	0	1	
Reset-Form Actions	12.7.5.3	PDF 5.66:	0	0	1	
XML Forms Architecture	12.7.8	PDF 5.67:	1	1	1	
Link Annotations	12.5.6.5	PDF 5.68:	1	0	1	
Document Outline	12.3.3	PDF 5.69:	1	0	1	
Thumbnail Images	12.3.4	PDF 5.70:	0	1	0	
Named Destinations	12.3.2.3	PDF 5.71:	1	1	1	
Digital Signatures	12.8	PDF 5.72:	1	1	1	
Legal Attestation	12.8.5	PDF 5.73:	0	0	0	
Page-Piece Dictionaries	14.5	PDF 5.74:	0	1	0	
Web Capture	14.10	PDF 5.75:	0	0	1	
Map Frame Dictionary	GeoPDF	PDF APP.A.1:	0	1	1	
Projection/Display Dictionary	GeoPDF	PDF APP.A.2:	0	1	1	
Summary			**Attack**	**Hiding**	**Disclosure**	**Total**
Total			**64**	**77**	**57**	**198**