Attività di R&D in ambito malware detection PDF per PagoPA S.p.A.

Analisi attacchi tramite documenti PDF

[Confidential]

Data: 18/04/2023
Progetto: Attività di R&D in ambito malware detection PDF per PagoPA S.p.A.
Versione: 1.0

Tabella dei contenuti

1 Introduzione

Il formato di file *PDF (Portable Document Format)* è uno dei formati di file più diffusi al mondo per la condivisione di documenti. La sua popolarità deriva dalla capacità di preservare l'aspetto originale del documento indipendentemente dal dispositivo o dal sistema operativo utilizzato per visualizzarlo. I PDF sono utilizzati ampiamente in tutto il mondo da organizzazioni di qualsiasi dimensione per archiviare e condividere informazioni in modo rapido e affidabile. Tuttavia, nonostante la loro popolarità, i documenti PDF possono rappresentare una minaccia per la sicurezza informatica, in quanto il loro contenuto può essere forgiato ad arte da attaccanti malintenzionati per infettare i sistemi con malware o rubare informazioni sensibili, mettendo a rischio utenti e organizzazioni. Per questo motivo, esiste un'urgente necessità di sviluppare soluzioni per rilevare documenti PDF malevoli prima che vengano elaborati da qualsiasi lettore, sia esso un lettore su un PC o un server, una stampante o qualsiasi altro dispositivo che possa aprire un documento PDF ed elaborarlo.
Il presente lavoro è stato commissionato a **SecSI S.r.l** da **PagoPA S.p.a.** con l'obiettivo di effettuare uno studio di ricerca finalizzato ad identificare i possibili vettori di attacco attuabili mediante documenti PDF.
Il documento è strutturato come di seguito. Il Capitolo 2 descrive le caratteristiche più importanti dello standard PDF e il processo di elaborazione di un documento PDF, il Capitolo 3 descrive lo standard PDF/A e le problematiche ad esso associate, il Capitolo 4 presenta uno studio sulle tecniche di attacco attuate tramite documenti PDF e una loro categorizzazione, il Capitolo 5 presenta invece uno studio sulle possibili contromisure attuabili per proteggersi da tali tipologie di attacchi, il Capitolo 6 fornisce infine delle analisi statistiche effettuate su un insieme di documenti PDF malevoli, il Capitolo 7 fornisce le considerazioni finali, il Capitolo 8 presenta i Next Steps

2 Lo standard PDF

In questo capitolo viene presentata una panoramica sulla struttura e l'elaborazione di un documento PDF, con particolare attenzione alle caratteristiche utili a capire le metodologie di attacco presentate nei capitoli successivi.

2.1 Panoramica

Il formato PDF è stato originariamente introdotto, con la sua versione 1.0, nel 1992 da Adobe. Successivamente, la versione 1.7 del formato PDF è stata resa uno standard in ISO 32000-1:2008. Attualmente, la versione più recente dello standard risulta essere ISO 32000-2:2020, che si riferisce alla versione 2.0 del formato PDF [1], [2], [3].
I PDF possono non solo contenere testo, ma anche elementi multimediali e codice JavaScript. Nel dettaglio, un file PDF può contenere:

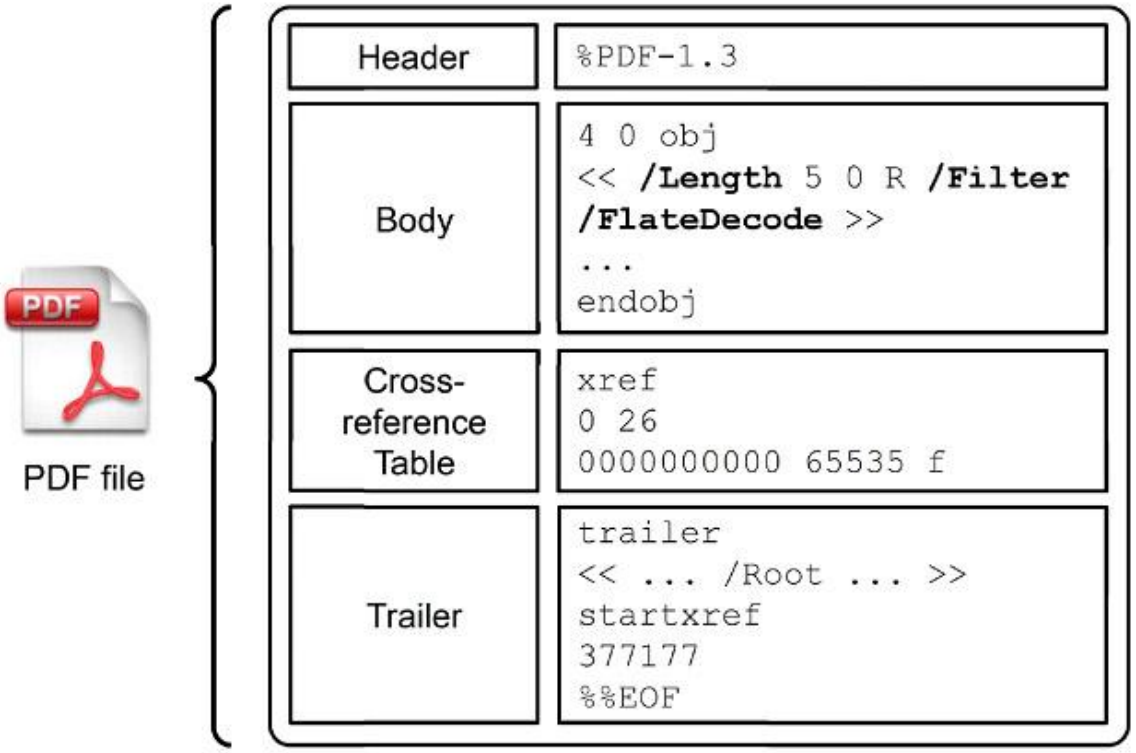
- Testo formattato memorizzato sotto forma di *content stream*
- Grafiche vettoriali
- Grafiche raster
- Oggetti multimediali
- Link (interni al documento o verso pagine web)
- Forms interattivi (*AcroForms* e *XML Forms Architecture*)
- Annotazioni (es. commenti, highlighting, ecc)
- Codice JavaScript
- Allegati
- Metadati
- Altri tipi di contenuto che possono essere visualizzati attraverso dei plug-ins

2.2 Struttura

Un file PDF contiene caratteri ASCII a 7 bit, ad eccezione di alcuni elementi che possono avere contenuto binario [4], [5], [15] .
La struttura base di un PDF prevede le seguenti 4 parti:

- *Header*: rappresentato dalla prima riga, contiene il *magic number* con la versione del formato PDF utilizzata
- *Body*: parte contenente gli oggetti del documento, tra cui ci sono quelli visibili e che ne compongono il contenuto
- *xref (cross-reference) table*: tabella che specifica l'*offset* degli oggetti all'interno del documento
- *Trailer*: parte finale del documento che contiene informazioni circa l'inizio del documento e il marcatore di *End of File (EOF)*.

Di seguito una rappresentazione di alto livello di tale struttura:



2.2.1 Header

L'header contiene il *magic number*, che identifica il file come file PDF, seguito dalla versione del formato PDF utilizzata dal file. Di seguito si riporta come appare l'header di un file PDF che usa la versione 1.7 dello standard:

```
%PDF-1.7
```

Solitamente sono anche presenti alcuni caratteri ASCII non stampabili, che di solito indicano ad alcuni prodotti software che il file contiene dati binari e non deve essere trattato come testo ASCII a 7 bit.

2.2.2 Body

Il body contiene gli oggetti che compongono il documento, che possono rappresentati in due modi:

- *Indirect objects*: definiscono un *top-level object* con un proprio identificativo
- *Direct objects*: sono oggetti inclusi in altri oggetti senza un proprio identificativo

Gli oggetti indiretti sono identificati da un *indirect object ID*, composto da:

1. *Object Number* (il primo numero)
2. *Generation Number* (il secondo numero)

Questi due numeri sono poi seguiti dalla parola chiave *obj* e poi, una volta andati a capo, inizia l'oggetto vero e proprio. Al termine del contenuto dell'oggetto, su una nuova riga c'è la parola chiave *endobj*. Grazie all'esistenza di tale ID, essi possono essere referenziati attraverso un *indirect reference*, che rappresentano un riferimento a un oggetto indiretto. Tale riferimento è rappresentato da tre caratteri del tipo `n r R`, in cui i primi due sono *Object Number* e *Revision Number*, mentre il terzo è la lettera *R*, che indica che tale oggetto è un riferimento.
Gli oggetti possono essere di vari tipi:

- Number: un qualsiasi numero è un oggetto
- *Name*: identificatori che specificano il nome dell'oggetto, iniziano con il forward slash `/`
- *Dictionary*: elenco non ordinato di coppie nome-oggetto comprese tra parentesi angolari
- *Array*: liste ordinate di oggetti racchiusi tra parentesi quadre
- *String*: testo racchiuso tra parentesi tonde
- *Stream*: solitamente contenenti grandi quantità di dati binari opzionalmente compressi, preceduti da un dizionario che descrive lo stream (ad esempio con la sua lunghezza e l'encoding) e racchiusi tra le parole chiave *stream* e *endstream*
- *Boolean*: valori `true` e `false`
- *Null*: valore `null`

Su tali tipi di oggetti di base vengono poi costruite delle strutture dati che vengono utilizzate per diversi scopi, come *date*, *name tree* o *rectangle*.
Gli oggetti nel documento sono organizzati secondo una struttura ad albero, dove l'oggetto radice è chiamato *document catalog*. Esso contiene riferimenti ad altri oggetti che definiscono il contenuto del documento e i suoi attributi.
Nelle sezioni successive verranno mostrate le informazioni più interessanti relativamente agli

oggetti presenti nei file PDF in modo da rendere più semplice la comprensione delle metodologie di attacco che verranno discusse nei prossimi capitoli.

2.2.2.1 Indirect e direct objects

Di seguito è mostrato un esempio di *Indirect Object*:

```
2 0 obj
<<
  /Type /Pages
  /MediaBox [ 0 0 200 200 ]
  /Count 1
  /Kids [ 3 0 R ]
>>
endobj
```

Esso è compreso tra le parole chiave *obj* e *endobj* e ha come *object number* il numero 2, mentre il suo *revision number* è 0. È inoltre un oggetto di tipo *dictionary*, in quanto è composto da coppie nome-oggetto comprese tra parentesi angolari. Tale dizionario rappresenta l'oggetto *Pages*, che contiene l'insieme delle pagine del documento. Esso infatti contiene 4 coppie nome-oggetto, dove il primo oggetto è un nome esso stesso (`/Pages`), il secondo è un array di numeri, il terzo è un numero e il quarto è un *indirect reference* a un altro oggetto indiretto. Tali oggetti sono tutti *direct objects*, in quanto non sono *top-level objects* e sono quindi inclusi in un altro oggetto.

2.2.2.2 Name dictionaries and name trees

Tra le varie chiavi contenute nell'oggetto *catalog* molto interessante è la chiave `/Names`, la quale referenzia un particolare dizionario chiamato *name dictionary*. Ogni chiave in tale dizionario referenzia la radice di un *name tree*, che è una struttura dati che permette di associare delle stringhe a determinati oggetti. Ad esempio, la chiave `/JavaScript` referenzia un *name tree* che associa stringhe ad *actions* JavaScript. Un altro esempio è invece la chiave `/EmbeddedFiles`, che referenzia un *name tree* che associa stringhe a *file specifications* che descrivono *embedded file streams*. Questi tipi di oggetti verranno esaminati nei paragrafi successivi.
Un *name tree* è un particolare tipo di *dictionary* che è composto da nodi. Se un nodo contiene la chiave `/Kids`, tale chiave ha come valore un array di oggetti che sono a loro volta dei *name tree*. Se, invece, un nodo contiene la chiave `/Names`, il suo valore sarà un *array* contenente una

serie di coppie chiave-valore, dove la chiave è una stringa e il valore è il riferimento all'oggetto associato a tale stringa.
Di seguito è mostrato un esempio di come sono collegati tra loro l'oggetto *catalog*, il *name dictionary* e un *name tree*:

```
1 0 obj
<<
  /Type /Catalog
  /Pages 2 0 R
  /Names 4 0 R
>>
endobj

4 0 obj
<<
 /JavaScript 5 0 R
>>
endobj


5 0 obj
<<
 /Names [ (test) 6 0 R ]
>>
endobj

6 0 obj
<<
  /S /JavaScript
  /JS (app.alert\(1\))
>>
endobj
```

In questo caso, l'oggetto identificato dall'*object number* 1 è il *catalog*, il quale referenzia tramite la chiave `/Names` l'oggetto con *object number* 4, che è il *name dictionary*. Esso contiene la chiave `/JavaScript`, che punta all'oggetto identificato dall'*object number* 5, che è il *name tree* delle *actions* JavaScript, il quale ha come valore della chiave `/Names` l'array che associa la stringa

`test`, che in questo caso rappresenta il nome dello script, all'oggetto identificato dall'*object number* 9. Tale ultimo oggetto rappresenta in questo caso una *action* che permette di inserire codice *JavaScript* all'interno del documento, come sarà spiegato nel dettaglio nei prossimi paragrafi.

2.2.2.3 Actions

Il formato PDF supporta anche delle *actions*, ossia è possibile specificare delle azioni effettuate in maniera automatica al verificarsi di un certo evento. Una *action* è un *dictionary* che può avere la chiave opzionale `/Type` con valore `/Action`, e specifica il tipo di azione da effettuare ed eventualmente referenzia la prossima *action* da eseguire dopo quella corrente attraverso la chiave `/Next`. Il tipo di *action* è specificato attraverso la chiave obbligatoria `/Subtype`, talvolta abbreviata a `/S`. A seconda del tipo di *action*, poi, il *dictionary* ad essa associato può contenere altri elementi [8]. Alcuni tipi di *action* sono i seguenti:

- *GoTo*: referenzia un'altra parte del documento PDF corrente
- *GoToR*: referenzia un documento PDF esterno
- *GoToE*: referenzia un PDF inserito nel documento PDF corrente
- *Launch*: esegue un'applicazione
- *URI*: apre una connessione verso un certo URI
- *SubmitForm*: invia i dati presenti in un *form* verso un dato URI
- *ImportData*: importa dati da un file esterno o da un URI
- *JavaScript*: esegue codice JavaScript

Le *actions* possono essere associate a diversi oggetti nel documento, ad esempio è possibile associare un'azione ad una pagina del documento, a una *form*, a una *annotation*, ecc. Di seguito un esempio di *action* di tipo *URI* specificata come *indirect object* che ha come effetto quello di contattare un determinato URI:

```
5 0 obj
<<
  /Type /Action
  /S /URI
  /URI (https://www.google.com)
>>
endobj
```

Un primo modo di associare una *action* ad un oggetto è quello di utilizzare la chiave `/A` nel *dictionary* che rappresenta un qualsiasi oggetto, che fa sì che l'azione venga eseguita quando

l'oggetto viene attivato (es. vi si clicca sopra). Di seguito è riportato un esempio di utilizzo della chiave `/A` per associare l'azione a un oggetto:

```
/A 5 0 R
```

Un altro metodo prevede invece l'utilizzo della chiave `/AA`. Tale chiave sta per *Additional Actions* e rende possibile specificare il particolare tipo di evento scatenante. Di seguito un esempio in cui viene utilizzato l'identificativo `/O` per specificare che l'evento scatenante è quello di *Open*, ossia la URI viene contattato quando l'oggetto in cui si trova l'identificativo `/AA` viene aperto:

```
/AA
<<
  /O 5 0 R
>>
```

Un ulteriore modo prevede l'utilizzo della chiave */OpenAction* che può essere inserita all'interno dell'oggetto *catalog*, In tal caso, l'evento scatenante per l'azione sarebbe l'apertura del documento. Di seguito un esempio in cui la *action* viene referenziata nel *catalog*:

```
1 0 obj
<<
  /Type /Catalog
  /Pages 2 0 R
  /OpenAction 5 0 R
>>
endobj
```

In questo modo, all'apertura del documento verrà immediatamente contattato l'URI specificato nella *action*.

2.2.2.4 Streams e Filters

Di seguito un altro esempio di *indirect object*, questa volta di tipo *stream*:

```
5 0 obj
```

```
<<
  /Length 44
>>
stream
BT
70 50 TD
/F1 12 Tf
(Hello, world!) Tj
ET
endstream
endobj
```

Esso è un *indirect object* che ha un *dictionary* al suo interno che specifica le caratteristiche dello *stream* (in questo caso solo la sua lunghezza) e poi ha tra le parole chiave *stream* e *endstream* il contenuto dello stream, che è in questo caso il testo formattato.

Spesso gli *stream* non sono rappresentati come *cleartext*. In realtà, la maggior parte di essi sono codificati attraverso dei *Filter* e rappresentati quindi da contenuto binario. Il tipo di codifica è specificato all'interno del *dictionary* associato allo *stream*. Di seguito è mostrato come appare lo *stream* dell'esempio precedente quando codificato attraverso il *Filter* di tipo *FlateDecode*, che è un tipo di codifica comunemente utilizzato basato sull'algoritmo DEFLATE o Zip.

```
5 0 obj  % page content
<<
  /Length 52
  /Filter /FlateDecode
>>
stream
xœs
á27P05P'qáÒw3T04R'IãÒðHÍÉÉ×Q(Ï/ÊIQÔT'Éâr
á'á·
Á
endstream
endobj
```

Un particolare *stream* può essere codificato più volte con più filtri, alcuni dei quali sono *FlateDecode*, *ASCII85Decode* e *ASCIIHexDecode*.

2.2.2.5 Object Streams

Gli *indirect objects* che non siano degli *stream* essi stessi possono anche essere collocati in *stream* speciali noti come *object stream* (contrassegnati da `/Type /ObjStm`). Gli *object stream* consentono di comprimere gli oggetti all'interno di un documento PDF. Funzionano raggruppando una serie di oggetti e comprimendoli come un unico flusso di dati, che può poi essere decompresso per accedere ai singoli oggetti. In questo modo è possibile ridurre in modo significativo le dimensioni del file di un documento PDF, soprattutto se contiene un gran numero di oggetti di piccole dimensioni. Tuttavia, gli *object stream* possono anche rendere un file PDF più complesso e difficile da manipolare. Un *object stream* è identificato da un *object number* e i singoli oggetti al suo interno sono accessibili in base al loro *offset* all'interno del flusso. Quando invece si utilizzano *indirect objects* diversi, ogni oggetto all'interno del documento PDF è identificato da un *object number* unico e non viene compresso come parte di uno stesso flusso di dati. Di seguito un esempio di come appare un *object stream*:

```
6 0 obj
<<
    /Type /ObjStm
    /Length <length of stream>
    /N <number of objects in stream>
    /First <offset to first object>
>>
stream
...compressed data...
endstream
endobj
```

L'offset di un oggetto all'interno di un *object stream* è necessario perché indica al lettore PDF dove trovare l'oggetto specifico al suo interno. Poiché gli oggetti all'interno del flusso sono compressi come un singolo blocco di dati, il visualizzatore deve conoscere la posizione di ogni singolo oggetto all'interno del blocco per potervi accedere. Quando il lettore PDF incontra un *object stream*, utilizza le informazioni sull'offset per trovare l'oggetto specifico di cui ha bisogno all'interno del flusso, decodifica i dati appropriati e quindi utilizza i dati decodificati per eseguire il rendering dell'oggetto. Inoltre, l'offset di un oggetto all'interno di un *object stream* viene utilizzato per determinare la dimensione del blocco di dati compressi per quell'oggetto. Ciò consente una decompressione più efficiente dei dati, in quanto il visualizzatore sa esattamente quanti dati decomprimere per ogni oggetto.

2.2.2.6 File Specifications e Embedded File Streams

Il formato PDF dispone di un oggetto chiamato *File Specification*, che è un dictionary che può sia contenere un riferimento a un file esterno, sia incorporare il contenuto dei file riferiti direttamente all'interno del corpo del PDF utilizzando degli oggetti denominati *Embedded File Streams*. Se viene referenziato un file esterno, questo può essere sia presente sulla stessa macchina in cui è aperto il documento PDF, oppure su una macchina remota. Nel caso in cui sia su una macchina remota, può sia essere referenziato tramite un URI, ad esempio tramite HTTP, sia essere identificato come risorsa in rete presente su un *file share*, ad esempio attraverso SMB. Se, invece, il file è direttamente all'interno del corpo del documento, allora ci sarà un riferimento a uno stream contenente il contenuto dello stesso. Questi file non sono parte della visualizzazione del documento, ma sono solitamente file allegati, che è possibile vedere dalla tab "allegati" del nostro lettore pdf, oppure vederli associati a una *annotation* nel documento (es. "Clicca qui per vedere l'allegato") [7].
Di seguito un esempio di *File Specification* che fa riferimento a un *Embedded File Stream* che rappresenta un'immagine in formato SVG:

```
31 0 obj
<</Type /Filespec /F (mysvg.svg) /EF <</F 32 0 R>> >>
endobj
```

In questo caso il dizionario è un *File Specification* e viene riportato sia il nome che verrà dato al file tramite la chiave `/F`, sia il riferimento allo stream contenente i dati del file tramite la chiave `/EF`. Questo è l'effettivo *Embedded File Stream*:

```
32 0 obj
<</Type /EmbeddedFile /Subtype /image#2Fsvg+xml /Length 72>>
stream
...SVG Data...
endstream
endobj
```

Si noti che la parte tra le parole chiave *stream* e *endstream* contiene i dati effettivi del file, in questo caso un'immagine SVG, ma potrebbe essere qualsiasi altra cosa.

2.2.2.7 JavaScript

Una tipologia di azioni che merita particolare attenzione è quella delle *JavaScript actions*. Un documento PDF può infatti contenere del codice JavaScript che viene attivato al verificarsi di determinati eventi. Viene spesso utilizzato per la validazione di campi di form, però può essere utilizzato nei modi più disparati. Le operazioni che è possibile implementare usando codice JavaScript dipendono da quali sono le API JavaScript esposte dal lettore PDF dal quale il documento è interpretato e tali API possono variare da lettore a lettore. In questo caso, la chiave `/S` che indica il tipo di *action* avrà valore `/JavaScript`, mentre il codice vero e proprio sarà inserito come valore per la chiave `/JS`.
Le azioni JavaScript possono essere sia a livello di documento che riferite in un determinato oggetto. Quelle a livello di documento sono riferite tramite l'utilizzo del *name dictionary* e dei *name trees*, come nell'esempio mostrato nei paragrafi precedenti. Tali azioni non sono solitamente referenziate da altri oggetti, ma contengono codice utilizzato dalle azioni JavaScript che sono poi referenziate dai singoli oggetti. Quelle riferite in un determinato oggetto sono infatti riferite da una chiave `/A` oppure `/AA` presente in tale oggetto, oppure dalla chiave `/OpenAction` se tale oggetto è il *catalog*.
Di seguito è mostrato un esempio di *action* JavaScript specificata come *indirect object*:

```
6 0 obj
<<
  /S /JavaScript
  /JS (app.alert\(1\))
>>
endobj
```

In questo caso, il codice JavaScript si trova direttamente all'interno del *dictionary* e utilizza la API JavaScript offerta dal lettore (in questo caso tramite l'oggetto *app*) per mostrare un *alert* all'utente contenente il valore "1". Ciò tuttavia non è l'unico modo per inserire il codice, in quanto esso potrebbe essere inserito in uno *stream* e codificato in modo da ridurre lo spazio da esso occupato. Di seguito è mostrato un esempio in cui l'azione JavaScript referenzia l'oggetto *stream* che contiene il codice vero e proprio, che è codificato utilizzando il filtro `/FlateDecode`:

```
7 0 obj
<<
  /S /JavaScript
```

```
  /JS 8 0 R
>>
endobj

8 0 obj
<<
  /Length < stream length >
  /Filter [/FlateDecode]
>>
stream
... encoded JavaScript code ...
endstream
endobj
```

Si noti che il codice in questo modo può essere codificato in diversi modi, anche più di uno, e che il codice decodificato potrebbe a sua volta essere minimizzato o offuscato a livello JavaScript. Inoltre, gli oggetti contenenti codice JavaScript potrebbero far parte di un *object stream* e quindi essere codificati e compressi insieme ad altri oggetti.

2.2.2.8 Forms

Il formato PDF supporta l'utilizzo di forms interattivi, anche detti *AcroForms*, i quali possono contenere dei campi nei quali inserire informazioni di qualsiasi tipo. Esistono tre modi attraverso i quali è possibile inserire dei form all'interno di documenti PDF:

- *Forms Data Format (FDF)*
- *XML Forms Data Format (xFDF)*
- *XML Forms Architecture (XFA)*

Il formato *FDF* è stato il primo ad essere introdotto e definisce un insieme di coppie chiave-valore che rappresentano i campi del form. In seguito è stato introdotto il formato *xFDF* che è la versione XML di *FDF*. Più recentemente è stato inoltre introdotto il formato *XFA*, che hanno la funzionalità aggiuntiva di ridisposizione dinamica del contenuto del PDF, però perdono in compatibilità e accessibilità. Per individuare dei form all'interno di un file PDF basta guardare alla chiave `/AcroForm` all'interno del *catalog*, che referenzia i diversi form interattivi del documento. Se un form inoltre è di tipo *XFA*, sarà presente la chiave /XFA nell'oggetto referenziato. Entrambi i tipi di form possono includere codice *JavaScript*, che è solitamente utilizzato per la validazione dei campi.

2.2.2.9 Contenuti multimediali

I documenti PDF possono contenere file multimediali, come ad esempio immagini, audio e video in diversi formati. Per lungo tempo, inoltre, potevano essere inclusi contenuti interattivi nel formato *Small Web Format (SWF)*, creati utilizzando il linguaggio *ActionScript* che è interpretato da *Adobe Flash Player*. Nel 2020, tuttavia, *Flash* è andato in *end of life*, quindi non è più supportato.

2.2.3 Cross-Reference Table

La *cross-reference table*, chiamata anche *xref table*, è una tabella che si trova in genere vicino alla fine del file e fornisce l'offset in byte di ogni *indirect object* rispetto all'inizio del file. Ciò consente un accesso casuale efficiente agli oggetti nel file e consente anche di apportare piccole modifiche senza riscrivere l'intero file. Essa inizia con la parola `xref`. Di seguito un esempio di *xref table*:

```
xref
0 6
0000000000 65535 f
0000000010 00000 n
0000000079 00000 n
0000000173 00000 n
0000000301 00000 n
0000000380 00000 n
```

Una *xref table* può essere composta da più sottosezioni. Nell'esempio precedente è presente una sola sottosezione e tutti i dati che seguono la parola `xref` fanno parte di essa. La prima riga di una sottosezione contiene due numeri, dove il primo numero rappresenta l'*object number* del primo oggetto della sottosezione (in questo caso l'oggetto 0), mentre il secondo numero rappresenta il numero di oggetti presenti nella sottosezione (in questo caso 6). Il resto della sottosezione è composto da una sequenza di righe, ciascuna associata a un *indirect object* del PDF e lunga 20 byte (compresi i caratteri di fine riga), con le seguenti informazioni da sinistra verso destra:

1. La posizione dell'oggetto specificata utilizzando l'offset in byte dell'oggetto dall'inizio del file PDF se l'oggetto è in uso, oppure il numero del prossimo oggetto libero nella tabella se l'oggetto è libero
2. Il *generation number* dell'oggetto

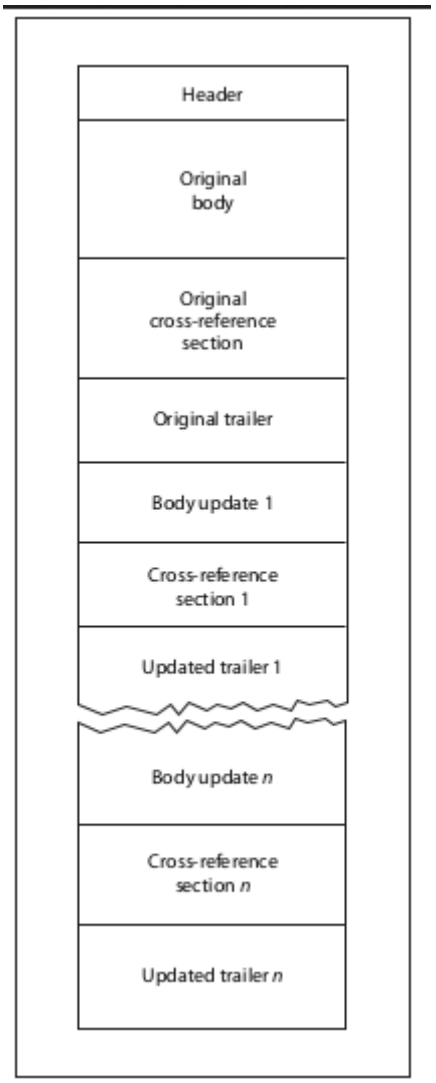
3. Un flag che definisce se lo specifico oggetto è in uso (valore del flag `n`) o libero (valore del flag `f`)

Il formato PDF consente la revisione e il rollback dei documenti, con la cronologia delle revisioni memorizzata all'interno del documento stesso. Per questo motivo, ogni oggetto del file include un numero di generazione che parte da 0 quando il documento viene creato per la prima volta e aumenta di uno ogni volta che viene effettuata una revisione dell'oggetto. Il primo oggetto è l'oggetto con *object number* pari a 0 e avrà sempre come *generation number* 65536 (il massimo possibile) e risulterà libero, mentre tutti gli altri oggetti partiranno come oggetto in uso e *generation number* pari a 0. Quindi, la *xref table* dell'esempio precedente identifica 6 oggetti e l'*object number* di un certo oggetto è dato dalla sua posizione nella tabella, quindi la riga 1 individua il primo oggetto, la riga 2 il secondo e così via. Si noti che, nonostante nell'esempio siano in ordine di offset crescente, ciò non è sempre vero.
Di seguito è mostrato un altro esempio che permette di mostrare il funzionamento della *xref table* più nel dettaglio:

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

In questo esempio sono presenti più sottosezioni, ognuna con un solo oggetto. La prima sottosezione contiene solo il primo oggetto, il quale è libero ed ha sempre *object number* pari a 0 e *generation number* pari a 65535. La seconda sottosezione ha come primo ed unico oggetto quello con *object number* pari a 3, il quale si trova ad un offset di 25324 byte dall'inizio del documento. La terza sottosezione ha quattro oggetti, il primo dei quali ha *object number* pari a 21 e si trova ad un offset di 25518 dall'inizio del file. Gli successivi nella sottosezione hanno *object number* rispettivamente pari a 22, 23 e 24. Tutti gli oggetti sono contrassegnati da un flag che indica se sono in uso o liberi. Un oggetto libero è un oggetto ancora presente nel file, ma non è attualmente utilizzato. Esso contiene un riferimento al prossimo oggetto libero al posto

dell'offset in byte di tale oggetto dall'inizio del file, inoltre il suo *generation number* rappresenta quello da utilizzare se esso torna nuovamente valido (è incrementato quando passa dall'essere in uso a libero). Si noti che l'oggetto zero punta all'oggetto libero successivo nella tabella, l'oggetto 23. Poiché anche l'oggetto 23 è libero, esso stesso punta al successivo oggetto libero della tabella, l'oggetto 24. Infine l'oggetto 24 è l'ultimo oggetto libero, che quindi punta di nuovo all'oggetto zero.
Una *xref table* può contenere più sottosezioni e i *generation number* possono essere diversi da 0 nel caso in cui il documento sia stato aggiornato in maniera incrementale. Tuttavia, i sistemi di modifica di documenti PDF odierni non modificano più una singola *xref table* del documento, bensì ne creano una nuova per ogni modifica apportata, tenendo conto dei soli oggetti attualmente utilizzati nel documento. In particolare, viene aggiunto un nuovo *body* contenente gli oggetti aggiornati, seguito dalla nuova *xref table* e dal nuovo *trailer*, questo per ogni modifica al documento. Un PDF quindi contiene le informazioni di tutte le sue modifiche sin dal momento in cui è stato creato. Di seguito è rappresentata la struttura di un documento aggiornato in maniera incrementale:



Si noti, inoltre, che esiste anche la possibilità di utilizzare uno *xref stream* invece di una *xref table*, in modo da ridurre le dimensioni del file e supportare quindi documenti di dimensioni maggiori. Essi conterranno le stesse informazioni, con la differenza che in un *xref stream* tali informazioni saranno compresse.

2.2.4 Trailer

Il *trailer* rappresenta la parte finale del documento ed inizia con la parola `trailer`. Esso è rappresentato da un *dictionary* contenente varie informazioni, seguito dalla parola `startxref`

che indica la posizione della *xref table* (o *xref stream*) in termini di offset in byte dall'inizio del documento. Termina poi con il marcatore di *End of File* rappresentato da `%%EOF`. Di seguito un esempio di *trailer*:

```
trailer
<<
  /Size 6
  /Root 1 0 R
>>
startxref
492
%%EOF
```

In questo esempio viene riportato il numero di oggetti presenti nel documento (tramite l'attributo `/Size`) e il riferimento all'oggetto radice del documento (tramite l'attributo `/Root`).

2.3 Processo di lettura di un documento PDF

Per mostrare come avviene la lettura e l'interpretazione di un documento PDF da parte di un lettore, viene di seguito proposto un *toy example* di documento PDF [6]:

```
%PDF-1.7

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

2 0 obj
<<
  /Type /Pages
  /MediaBox [ 0 0 200 200 ]
  /Count 1
  /Kids [ 3 0 R ]
>>
```

```
endobj

3 0 obj
<<
  /Type /Page
  /Parent 2 0 R
  /Resources <<
    /Font <<
      /F1 4 0 R
    >>
  >>
  /Contents 5 0 R
>>
endobj

4 0 obj
<<
  /Type /Font
  /Subtype /Type1
  /BaseFont /Times-Roman
>>
endobj

5 0 obj
<<
  /Length 44
>>
stream
BT
70 50 TD
/F1 12 Tf
(Hello, world!) Tj
ET
endstream
endobj

xref
0 6
```

```
0000000000 65535 f
0000000010 00000 n
0000000079 00000 n
0000000173 00000 n
0000000301 00000 n
0000000380 00000 n
trailer
<<
  /Size 6
  /Root 1 0 R
>>
startxref
492
%%EOF
```

Tale documento viene visualizzato nel seguente modo da un qualsiasi lettore PDF:

Un lettore PDF non analizza il documento in modo sequenziale (dall'alto verso il basso), ma accede al file in modo più complesso. Per leggere il documento presentato nell'esempio precedente, il lettore PDF:

1. legge la prima riga (*header*) per ottenere la versione PDF
2. salta alla fine del documento e controlla il marcatore %%EOF,
3. si sposta una riga sopra per leggere l'offset in bytes della *xref table* rispetto all'inizio del documento, pari a 492
4. salta alla *xref table* e costruisce un elenco di offset degli oggetti del documento
5. legge dal *dictionary* presente nel trailer la posizione del *Catalog*, che è l'oggetto radice di un documento PDF
6. salta all'oggetto *Catalog*, il quale in questo caso contiene solo un riferimento all'oggetto *Pages*, identificato dall'*object number* 2
7. salta all'oggetto *Pages* per leggere il numero di pagine del documento (tramite `/Count`) e i riferimenti a tali pagine (tramite l'array `/Kids`)
8. salta al primo oggetto *Page* dell'array (che in questo caso è anche l'unico) identificato dall'*object number* 3
9. legge dall'oggetto *Page* i riferimenti ai font utilizzati e agli oggetti contenuti nella pagina che esso rappresenta
10. salta all'oggetto con *object number* 4 per leggere la tipologia di font utilizzato
11. salta all'oggetto con *object number* 5, che è uno *stream* che contiene le istruzioni per la visualizzazione della pagina

In questo caso le istruzioni per la visualizzazione della pagina sono comprese tra BT ed ET (che rappresentano rispettivamente *Begin Text* e *End Text*) in quanto la pagina contiene solo del testo. In particolare, le istruzioni sono le seguenti:

- *operatore TD*: posiziona il cursore di testo sulla pagina alle coordinate (x, y), che in questo caso sono (50, 70)
- *operatore Tf*: imposta il nome e la dimensione del font, che in questo caso è il font F1, come definito nelle risorse, con dimensione 12
- *operatore Tj*: mostra il testo rappresentato in questo caso dalla stringa di testo "Hello, world!"

Si noti che talvolta è possibile incontrare dei documenti PDF che sono stati soggetti a un processo di *linearization*, che fa sì che il lettore trovi le informazioni per la lettura del documento già all'inizio dello stesso. Ciò è utile per i cosiddetti "PDF ottimizzati per il web", in quanto non è necessario attendere il download di tutto il documento per poterne leggere la prima parte o pagine singole.

2.4 Cifratura

Un documento PDF può essere cifrato, così che il suo contenuto possa essere visualizzato o modificato soltanto conoscendo una determinata password impostata in fase di cifratura. Lo standard PDF prevede due diversi metodi di cifratura. Il primo prevede l'utilizzo di una *user password*, che cifra il contenuto del file e ne impedisce l'apertura se non si conosce tale password. In questo caso il file non viene completamente cifrato, ma vengono cifrati soltanto gli *stream* e le stringhe presenti nel file, lasciando tutto il resto della struttura intatta, in modo da rendere comunque fattibile l'elaborazione del documento. Il secondo metodo prevede invece l'utilizzo di una *owner password*, che permette di specificare le azioni che possono essere compiute sul documento, ad esempio modifica, stampa, copia del testo o aggiunta di note. La protezione offerta da questo secondo metodo, tuttavia, si basa sul fatto che il lettore PDF con cui viene aperto il documento rispetti le restrizioni originariamente impostate, motivo per il quale è una protezione facilmente aggirabile.

3 Lo standard PDF/A

In questo capitolo viene presentata una panoramica sullo standard PDF/A, le versioni di tale standard e i livelli di conformità che un documento deve rispettare per essere aderente allo standard. Vengono inoltre evidenziate le problematiche relative alla conversione di un documento PDF in documento PDF/A e all'identificazione di un documento che rispetta lo standard PDF/A.

3.1 Panoramica

Il formato PDF/A è definito per la prima volta dallo standard ISO 19005-1:2005 ed è un formato appositamente pensato per l'archiviazione nel lungo periodo di documenti elettronici basato sulla versione 1.4 del formato PDF di Adobe. Esso è un sottoinsieme di PDF, ripulito delle funzioni non pensate per l'archiviazione di lungo periodo [9], [10], [11].
Lo standard PDF/A, nelle sue varie versioni, si basa sul fatto che ci sono alcune parti del documento che potrebbero essere visualizzate in maniera diversa nel tempo dai software di visualizzazione. Per evitare ciò, quindi, il requisito chiave di PDF/A è quello di imporre che tutte le informazioni necessarie per visualizzare il documento siano incorporate nel file. Ciò include tutti i contenuti (testo, immagini raster e grafica vettoriale), i font e le informazioni sui colori. Un documento PDF/A non può fare affidamento su informazioni provenienti da fonti esterne,

ma può includere annotazioni (ad esempio, collegamenti ipertestuali) che rimandano a documenti esterni.
Elementi chiave di PDF/A includono:

- Assenza di contenuti audio e video
- Assenza di codice JavaScript
- Assenza di invocazioni di file eseguibili
- Assenza di riferimenti a contenuti esterni
- Assenza di supporto alla cifratura
- Utilizzo di soli font incorporati nel documento
- Utilizzo di sole immagini incorporate nel documento
- Utilizzo di soli profili di colori incorporati nel documento
- Utilizzo di soli metadati standard

3.2 Standard di riferimento

Lo standard di riferimento è lo standard ISO 19005 di cui ci sono varie versioni:

- 19005-1 (PDF/A-1)
  - Basato sulla versione di PDF 1.4 (quando PDF non era ancora standardizzato ma proprietario Adobe) e pubblicato nel 2005
- 19005-2 (PDF/A-2)
  - Basato sulla versione di PDF 1.7 (ossia la prima versione standardizzata di PDF del 2008 in ISO 32000-1)
  - Aggiunge il supporto alle immagini in formato JPEG 2000
  - Aggiunge il supporto per effetti di trasparenza e contenuti opzionali (livelli)
  - Aggiunge il supporto per le firme digitali secondo lo standard *PAdES*
  - Aggiunge la possibilità di incorporare altri file PDF/A per facilitare l'archiviazione di insiemi di documenti in un singolo file
- 19005-3 (PDF/A-3)
  - Basato sulla versione di PDF 1.7 (ossia la prima versione standardizzata di PDF del 2008 in ISO 32000-1)
  - Aggiunge il supporto ad inclusione di molti tipi di file (es. XML, CSV, CAD, Word, Excel, ecc), come anche di file PDF (così da poter avere anche il file sorgente da cui è stato ricavato il PDF/A dentro un solo file)
    - Questo formato non è responsabile dei files al suo interno, ma solo di estrarli, quindi non si può sapere se tali files rimarranno leggibili in futuro
- 19005-4 (PDF/A-4)
  - Basato sulla versione di PDF 2.0 (ossia la versione standardizzata di PDF del 2020 in ISO 32000-2)

3.3 Livelli di conformità

Tra di essi esiste la *forward compatibility*, ossia un documento PDF/A-1 è anche PDF/A-2, ma non il viceversa (assenza di *backward compatibility*).
Per ognuno di questi standard (fino al PDF/A-3) ci sono poi dei livelli di *conformance*:

- *Level A (Accessible) conformance*: richiede il più alto livello di accessibilità. Richiede che la struttura del documento sia etichettata con metadati per aiutare la tecnologia assistiva, come i lettori di schermo, a interpretare accuratamente il contenuto. Richiede inoltre che tutti i font utilizzati nel documento siano incorporati e che le informazioni sul colore e le immagini abbiano descrizioni alternative per le persone con disabilità visive. Questo livello di *conformance* di solito può essere raggiunto solo convertendo documenti nati in formato digitale.
- *Level B (Base) conformance*: è meno rigoroso rispetto al livello *Accessible*. Richiede solo che i font utilizzati nel documento siano incorporati e che tutte le informazioni necessarie siano incluse nel file stesso. Esso garantisce comunque la riproducibilità univoca del contenuto del documento, però non garantisce l'estrazione o la ricerca del testo. I documenti cartacei scansionati di solito possono essere convertiti in questo livello di *conformance* senza molti problemi.
- *Level U (Unicode)*: è stato introdotto insieme a PDF/A-2. Espande il livello di conformità B per specificare che tutto il testo può essere mappato ai codici carattere Unicode standard ed è quindi estraibile.

PDF/A-4 abbandona i precedenti livelli di *conformance* e ne introduce di altri:

- *PDF/A-4f*: consente di incorporare tipi di file di qualsiasi altro formato, in maniera simile a come PDF/A-3 estende PDF/A-2
- *PDF/A-4e*: introduce il supporto per RichMedia e annotazioni di tipo 3D per creare una versione PDF/A compatibile con le moderne tecnologie utilizzate in ambito ingegneristico

3.4 Conversione in PDF/A

La conversione da PDF a PDF/A-1 spesso comporta dei problemi, in quanto alcuni elementi del PDF potrebbero essere non supportati. Esempi sono problemi di grafica vettoriale, perdita di link, caratteri illeggibili, testo mancante, parti del documento mancanti ed errori di ortografia. È richiesta quindi un'ispezione manuale del PDF a valle della conversione per evitare tali problemi. La conversione da PDF (fino alla versione 1.4) in PDF/A-2 invece solitamente presenta un minor quantitativo di errori, dovuti principalmente ai glifi e caratteri unicode. In

generale, gli errori di conversione sono il motivo per il quale, nei casi in cui venga richiesto un file in formato PDF/A, l'onere di convertire il file ricade sul mittente e non sul destinatario.

3.5 Identificazione e validazione di PDF/A

Un documento PDF/A può essere identificato come tale attraverso metadati specifici presenti nel namespace "http://www.aiim.org/pdfa/ns/id/". Questi metadati rappresentano tuttavia solo una dichiarazione di conformità e di per sé non garantiscono la conformità. Infatti, un documento PDF può essere conforme a PDF/A, tranne che per la mancanza di metadati PDF/A. Questo può accadere, ad esempio, con documenti generati prima della definizione dello standard PDF/A, da autori consapevoli di caratteristiche che presentano problemi di conservazione a lungo termine. Inoltre, un documento PDF può essere identificato come PDF/A, ma può contenere erroneamente caratteristiche PDF non consentite da PDF/A. Ciò vuol dire che i documenti che dichiarano di essere conformi a PDF/A dovrebbero essere verificati per la conformità a PDF/A. La validazione dei documenti PDF/A serve a verificare se un file è davvero un file PDF/A o meno. Sfortunatamente, i validatori PDF/A sono spesso in disaccordo, poiché l'interpretazione degli standard PDF/A non è sempre chiara.

4 Attacchi tramite PDF

In questo capitolo verrà analizzato lo stato dell'arte dei modi con cui un documento PDF può essere utilizzato per portare a termine degli attacchi. A tal scopo sono stati presi in considerazione sia pubblicazioni accademiche, sia analisi affrontate con un approccio più tecnico. Mentre l'approccio accademico si concentra sulle metodologie scientifiche (le quali implicano un certo rigore e precisione nello studio), l'approccio tecnico si concentra principalmente sul raggiungimento degli obiettivi e favorisce quindi i risultati rispetto ai metodi, producendo spesso risultati eccezionali con metodi nuovi, innovativi e talvolta non ortodossi. In particolare, sono stati analizzate 13 fonti, di cui:

- 4 fonti che utilizzano un approccio accademico
- 9 fonti che utilizzano un approccio tecnico

A valle della lettura di tali fonti, viene proposta una categorizzazione di alto livello con lo scopo di raggruppare le tipologie di attacco analizzate.

4.1 Stato dell'arte

Come evidenziato in precedenza, il formato PDF è un formato molto complesso che offre molteplici funzionalità. Purtroppo, maggiore è il numero di funzionalità offerte da una tecnologia, maggiore è la superficie di attacco a cui essa è esposta. Ciò è valido anche per i documenti PDF, che permettono, tra le altre cose, di inserire contenuti multimediali, di contattare URI, di contenere allegati e addirittura di contenere codice JavaScript. Di seguito vengono esposte le tipologie di attacco presentate nelle varie fonti analizzate.

4.1.1 A Curious Exploration of Malicious PDF Documents

Solitamente, le vulnerabilità vengono sfruttate dal documento all'atto dell'apertura dello stesso. Ciò è possibile grazie all'utilizzo delle *actions*. In [12] viene evidenziata l'importanza delle *actions* per lo sfruttamento di vulnerabilità e vengono identificate quelle più prone ad essere utilizzate con scopi maliziosi. Prima di introdurre tali azioni, viene anche evidenziata l'importanza delle chiavi *OpenAction* e *AA*, che, come già spiegato, permettono l'esecuzione di tali azioni in maniera automatica. Ad esempio, le prime azioni di cui si parla sono le azioni *GoToR* e *GoToE*, che rispettivamente permettono di fare riferimento a destinazioni remote (come *network shares* o URI) o a destinazioni *embedded*, ossia file incorporati nel documento. Utilizzando queste azioni è possibile manipolare i percorsi dei file a cui viene fatto riferimento in modo da far sì che il lettore si connetta automaticamente a delle destinazioni, solitamente controllate dall'attaccante, attraverso diversi protocolli. Tali protocolli possono essere ad esempio HTTP, ma anche SMB. Un'altra azione molto interessante è rappresentata dall'azione *URI*, che ha come effetto quello di aprire il browser di default verso un determinato URI. Ciò può poi esporre la vittima a tutte le vulnerabilità a cui si è normalmente esposti quando si naviga sul web, come ad esempio il download di un eseguibile malevolo. Ancora, un'azione interessante è *SubmitForm*, che permette di inviare i dati contenuti in una form del documento verso una determinata destinazione remota via HTTP. Ciò può essere sfruttato per esfiltrare dati sensibili inseriti nelle form dall'utente. L'azione più interessante è, infine, l'azione *Launch*. Essa permette di eseguire applicazioni presenti sul sistema della vittima, come anche di lanciare eseguibili incorporati nel file PDF o eseguibili che si trovano a una destinazione remota. Come spesso accade, tuttavia, malgrado lo standard PDF preveda la possibilità di utilizzare tali azioni, spesso sono i lettori PDF che pongono dei limiti al loro utilizzo, in modo da scongiurare eventuali utilizzi maliziosi. Esempi sono alcuni lettori che bloccano totalmente l'esecuzione di eseguibili tramite l'azione *Launch*, oppure lo permettono ma chiedono conferma all'utente, avvisandolo del pericolo. La richiesta di conferma è infatti un meccanismo

implementato dai lettori per molte delle azioni analizzate, anche per l'apertura di un semplice URI, in modo da mettere al corrente l'utente che il documento sta tentando di effettuare delle azioni in maniera automatica. Il lettore PDF viene quindi considerato vulnerabile a un determinato attacco solo se tale attacco può essere portato a termine senza interazione dell'utente, oppure nei casi limite in cui può essere sfruttata una vulnerabilità che induce l'utente a consentire l'attacco con più facilità. Da non trascurare, infine, è il supporto a codice JavaScript contenuto nel documento. Viene presa come esempio la specifica della API JavaScript offerta da Adobe e viene evidenziato il fatto che le funzioni sono suddivise in *privileged* e *unprivileged*, con le prime che richiedono l'autorizzazione dell'utente per essere utilizzate. Ciononostante, si nota che molte funzioni tra quelle *unprivileged* possono comunque essere utilizzate con scopi maliziosi, e quindi sono gli stessi lettori PDF a limitarne l'utilizzo.

4.1.2 Processing Dangerous Paths – On Security and Privacy of the Portable Document Format

Una trattazione molto interessante delle funzionalità dello standard PDF giudicate insicure è presente in [13]. Vengono infatti evidenziati i modi attraverso i quali è possibile sfruttare le *actions* e il codice JavaScript a scopi maliziosi. Sono presi in considerazione diversi lettori PDF, i quali sono giudicati vulnerabili a una determinata categoria di attacco se e solo se tale attacco può essere portato a termine senza richiedere alcuna interazione utente. Ciò vuol dire che, malgrado lo standard PDF sia vulnerabile a determinate categorie di attacchi, sono i lettori PDF a implementare delle misure di sicurezza al di sopra di esso per essere resi sicuri. In questa trattazione gli attacchi vengono suddivisi in quattro categorie.
La prima categoria è quella degli attacchi di tipo *Denial of Service (DoS)*, in cui è possibile far sì che il lettore PDF rimanga intrappolato in un ciclo infinito tramite ad esempio riferimenti ciclici tra oggetti del PDF, oppure utilizzando il linguaggio JavaScript. Inoltre, è anche possibile causare una *data expansion* tramite l'utilizzo di *filters* che applicano decodifiche e decompressioni successive, il che può saturare la memoria associata al lettore PDF affinché smetta di funzionare, provocando eventualmente problemi anche al sistema operativo sottostante.
La seconda categoria è quella degli attacchi che mirano a causare *information disclosure*, ossia la connessione verso un server controllato da un attaccante per ottenere informazioni sulla vittima, dati inseriti nelle form, file dalla macchina della vittima o credenziali. Alcune azioni che permettono di fare ciò sono ad esempio *URI*, *GoToR* e *SubmitForm*, oppure è possibile usare codice JavaScript.

La terza categoria è quella relativa alla *data manipulation*, dove l'obiettivo dell'attaccante è quello di creare un documento che si modifichi da solo, ad esempio cambiando il proprio contenuto quando viene stampato, o quando viene chiuso. Ciò è ottenibile sia utilizzando codice JavaScript, sia utilizzando azioni come *ImportData*. Inoltre, è possibile creare dei documenti che vengono visualizzati in maniera differente da diversi lettori, utilizzando tecniche che cercano di confondere il lettore PDF creando situazioni ambigue all'interno del documento, come ad esempio l'inserimento di oggetti duplicati o errori di sintassi.. Questa categoria di attacchi mira a far sì che l'utente creda di visualizzare un PDF diverso da quello che l'attaccante intendeva.
La quarta categoria è quella più dannosa in assoluto, ossia quella degli attacchi di *code execution*. In questo caso viene presa in considerazione soltanto l'esecuzione di codice che è possibile ottenere sfruttando l'azione *Launch*, quindi l'esecuzione di un eseguibile incorporato nel file oppure situato all'esterno del file, sia esso presente sul sistema della vittima o in maniera remota.
Questo lavoro presenta anche un'ottima componente di *evaluation*. Sono stati infatti testati 28 diversi lettori (compresi lettori incorporati nei web browser) con settaggi di default e si nota che 26 di questi sono vulnerabili ad almeno un attacco senza richiedere interazioni utente. Si nota inoltre che le applicazioni per sistemi Linux e Mac OS risultano in generale meno vulnerabili rispetto a quelle per sistemi Windows, così come i lettori PDF presenti nei browser web, che implementano maggiori funzionalità di *sandboxing* per prevenire i classici attacchi web. È possibile infine notare che la maggior parte delle applicazioni sono vulnerabili ad attacchi di tipo *DoS* e che molte sono vulnerabili ad attacchi di tipo *information disclosure*.

4.1.3 NTLM Credentials Theft via PDF Files

Un esempio di attacco che sfrutta funzionalità legittime del PDF è documentato in [14], che permette di ottenere le credenziali NTLM di un utente Windows che apre il documento PDF incriminato con un lettore PDF, come ad esempio Adobe Reader. L'attacco sfrutta la possibilità di referenziare un documento PDF esterno mediante una *file specification* contenuta all'interno dell'azione *GoToR*. Se viene infatti referenziato un documento PDF che è presente su un server SMB remoto controllato dall'attaccante, viene automaticamente tentata l'autenticazione NTLM verso tale server, mandando le credenziali NTLM all'attaccante in termini di nome utente e hash della password. Utilizzando tale hash NTLM si può risalire alla password dell'utente tramite un processo di *cracking*, oppure si possono utilizzare attacchi quali *Pass The Hash*, che permettono di effettuare l'accesso a un sistema Windows conoscendo soltanto l'hash in questione. A tale vulnerabilità è stato poi assegnato l'identificativo CVE-2018-4993 e poi

risolto dai lettori affetti con degli aggiornamenti di sicurezza. Ciò conferma come funzionalità legittime del PDF possano essere sfruttate e che sono poi i lettori PDF a dover evitare il loro utilizzo a scopi maliziosi.

4.1.4 Towards Adversarial Malware Detection: Lessons Learned from PDF-based Attacks

Gli attacchi, tuttavia, non avvengono soltanto attraverso l'utilizzo intelligente di *actions* e funzionalità legittime supportate dallo standard PDF non adeguatamente protette dai lettori PDF. Essi, infatti, avvengono anche e soprattutto sfruttando delle vulnerabilità nelle implementazioni di alcune funzionalità esposte dal lettore PDF o dal motore JavaScript offerto da tale lettore. La trattazione proposta in [15] analizza le vulnerabilità del lettore Adobe Reader più sfruttate tra il 2008 e il 2018, suddividendole in tre categorie principali. La prima categoria è quella degli attacchi che sfruttano vulnerabilità nell'implementazione dell'API JavaScript del lettore PDF, ad esempio utilizzando delle funzioni vulnerabili che, chiamate con determinati parametri, causano l'esecuzione di codice sul sistema vittima. La seconda categoria è quella degli attacchi che sfruttano vulnerabilità nell'implementazione dell'interprete ActionScript, utilizzando ad esempio dei file multimediali corrotti che vanno ad eseguire codice in aree di memoria controllate dall'attaccante. La terza categoria è quella degli attacchi che sfruttano vulnerabilità nei parser che il lettore PDF utilizza per processare determinati tipi di file, ad esempio includendo nel documento delle immagini del formato del parser vulnerabile che portano infine all'esecuzione di codice.

4.1.5 Anatomy of a malicious PDF file

Alcuni esempi di vulnerabilità dovute al lettore PDF sono riportati in [16], dove viene evidenziato che le vulnerabilità sfruttate sono principalmente nell'interprete JavaScript e nelle primitive PDF, e che l'obiettivo dell'attacco è solitamente l'esecuzione di codice sulla macchina vittima. Un esempio di vulnerabilità dell'interprete JavaScript è identificato da CVE-2009-0927, dove una chiamata al metodo *Collab.getIcon* causa un *buffer overflow* in Adobe Reader 9.0. Un esempio di vulnerabilità che sfrutta le primitive PDF è identificato da CVE-2009-3459, che causa un *heap overflow* in Adobe Reader 9.3 quando viene chiamata la primitiva *Colors* con un argomento troppo grande. Viene inoltre evidenziato come, nonostante

la presenza di JavaScript non sia necessaria nel caso in cui la vulnerabilità non sia relativa all'interprete JavaScript, solitamente esso è comunque presente in quanto affinché l'esecuzione di codice arbitrario possa avvenire con successo, l'attaccante dovrebbe conoscere l'esatto layout della memoria del lettore PDF, il che è molto difficile. Ciò significa che spesso è comunque presente una porzione di codice JavaScript che effettua un cosiddetto *heap spray*, ossia una tecnica che tenta di saturare la memoria heap con puntatori verso il codice dell'attaccante, così da aumentare la probabilità che esso venga eseguito senza conoscere esattamente la struttura della memoria del processo.

3.1.5 Analysis of CVE-2009-0658 (Adobe Reader 0day)

Un esempio in cui viene utilizzato codice JavaScript per effettuare *heap spray* nonostante la vulnerabilità in sé non lo richieda è fornito dalla vulnerabilità identificata da CVE-2009-0658, che riguarda l'algoritmo di decompressione immagini *JBIG2Decode* presente in Adobe Reader 9.0 e versioni precedenti. In tal caso, inserendo un'immagine malformata e creata in un particolare modo in un documento è possibile ottenere l'esecuzione di comandi arbitrari sulla macchina vittima. Tale vulnerabilità risulta molto interessante anche perché affligge anche l'estensione di Adobe Reader che permette di visualizzare un'anteprima del documento dalla cartella in cui si trova, per cui non è nemmeno necessaria l'apertura del documento per il suo sfruttamento. In [17] è proposta un'analisi di tale vulnerabilità e viene anche fatto notare che nonostante in linea di principio non sia richiesta la presenza di codice JavaScript, tutte le *Proof of Concept (PoC)* presenti in rete utilizzano codice JavaScript avviato tramite una *action* all'apertura del documento per facilitare lo sfruttamento della vulnerabilità, effettuando un *heap spray* preliminare. Da ciò, si evince che la maggior parte dei PDF malevoli contiene codice JavaScript.

4.1.6 Analyzing Malicious PDF Files

Le vulnerabilità che sfruttano la API JavaScript del lettore sono moltissime ed è molto facile trovare in rete analisi di documenti PDF che le sfruttano. Spesso, per l'analisi, vengono utilizzati strumenti a riga di comando che effettuano il parsing della struttura del documento, i cui più utilizzati sono *peepdf* [18], *pdfid* [19] e *pdf-parser* [20].
Un esempio di analisi è fornito in [21], dove viene analizzato un documento che sfrutta la vulnerabilità identificata da CVE-2009-1492, la quale permette di eseguire codice arbitrario in alcune versioni di Adobe Reader utilizzando una combinazione di delle *annotations* e una

OpenAction. In tal caso il documento è processato con *peepdf*, che immediatamente rileva la presenza della vulnerabilità sulla base di firme contenute al suo interno, però è comunque interessante l'analisi manuale proposta, in cui vengono individuate le sezioni contenenti codice JavaScript e vengono esplorati tutti i riferimenti tra gli oggetti che compongono l'exploit finale. Infatti, il codice JavaScript risulta offuscato, ossia reso difficile da comprendere per un lettore umano, e utilizza la API JavaScript del lettore per ottenere degli oggetti *annotations*, i quali conterranno degli *stream* codificati e compressi attraverso dei *filters*. Al loro interno è infine possibile trovare il codice JavaScript che rappresenta lo *shellcode* che andrà a sfruttare la vulnerabilità per ottenere l'esecuzione di comandi. L'offuscamento del codice JavaScript, la codifica degli stream e i riferimenti a catena tra diversi oggetti sono classiche tecniche di offuscamento che hanno il solo obiettivo di rendere più complessa l'analisi e il rilevamento di tali documenti malevoli. Un'altra cosa interessante da notare è che gli *shellcode* sono più di uno e viene scelto quale attivare sulla base della versione del lettore PDF in uso. Tale ultima tecnica è anch'essa molto diffusa e permette di evitare che il documento tenti di sfruttare versioni non vulnerabili del lettore PDF, permettendo così di nascondere ancora di più la presenza del malware al suo interno.

4.1.7 PDF malware analysis

Un altro esempio di malware che agisce in maniera differente sulla base della versione del lettore PDF utilizzato è proposto in [22], il quale esegue uno *shellcode* diverso in base alla versione di Adobe Reader utilizzata. Essi hanno poi come scopo finale quello di scaricare altro malware, in questo caso sotto forma di eseguibile, da un server controllato dall'attaccante. Ciò in quanto lo *shellcode* deve essere di piccole dimensioni, quindi quel che fa è limitarsi a scaricare il vero e proprio malware sulla macchina vittima.

4.1.8 A Guided Tour of a Classic PDF-Based Malware Dropper

Un ulteriore esempio di analisi manuale di un documento PDF è proposto in [23], dove viene ribadito che solo per il lettore PDF Adobe Reader vengono scoperte ogni anno molte vulnerabilità, in particolare nel 2016 ne sono state scoperte 20, di cui 17 relative a esecuzione di comandi. Anche in questo caso viene proposta un'analisi in cui il documento sfrutta la vulnerabilità identificata da CVE-2007-5659, che sfrutta ancora una volta una vulnerabilità nella API JavaScript di alcune versioni di Adobe Reader. Di nuovo, il codice JavaScript risulta

offuscato e sparpagliato tra *stream* differenti che sono essi stessi codificati e compressi, tutto con il solo scopo di rendere il documento difficile da analizzare. Di nuovo, scopo ultimo dello *shellcode* è quello di scaricare un eseguibile malevolo ed eseguirlo.

4.1.9 Malicious Documents - PDF Analysis in 5 Steps

Una tecnica di offuscamento interessante è quella di utilizzare forms XFA per nascondere il codice JavaScript, come mostrato in [24]. In questo esempio viene sfruttata la vulnerabilità della API JavaScript offerta da alcune versioni di Adobe Reader identificata da CVE-2013-2729, ma il codice JavaScript incriminato non si trova banalmente all'interno di una *action* JavaScript del documento. Infatti, gli strumenti di analisi riportano che il numero di sezioni JavaScript presenti è pari a 0. Ciò, tuttavia, non vuol dire che il codice JavaScript sia assente, in quanto il form XFA presente nel documento è uno *stream* che, una volta decodificato, si rivela essere un XML contenente un tag che ha al suo interno il codice JavaScript che rappresenta lo *shellcode*, che ha l'obiettivo di scaricare un eseguibile malevolo da un sito controllato da un attaccante.

4.1.10 PDF-Malware Detection: A Survey and Taxonomy of Current Techniques

Risulta quindi evidente che un ruolo chiave negli attacchi che utilizzano documenti PDF malevoli, allo stesso modo di come accade in generale con qualsiasi tipo di attacco, è svolto dalle tecniche di offuscamento. Esse rendono più difficile la rilevazione del malware e anche la sua analisi. In [25] viene proposta una interessante disamina delle tecniche di offuscamento più utilizzate nei documenti PDF, sia relative al codice JavaScript contenuto nel documento, sia agli oggetti del documento stesso. Molte di esse sono state utilizzate negli esempi esaminati prima, come ad esempio la separazione del codice in più oggetti differenti e l'applicazione di codifiche e compressioni mediante *filters* per quanto riguarda gli oggetti del documento, ma esistono altre tecniche che sfruttano la possibilità di inserire spazi bianchi e ritorni a capo nel file con lo scopo di neutralizzare semplici scanner che si basano sull'hash del documento per la sua rilevazione. Molte di tali tecniche sono anche esemplificate in [26], che aggiunge anche la possibilità di utilizzare la cifratura di un file PDF senza alcuna password per cambiare la rappresentazione delle stringhe al suo interno. Altre tecniche, relative invece al codice

JavaScript, sono la codifica del codice in *base64* o esadecimale, l'inserimento di commenti e blocchi di codice inutili che possono sviare l'analisi, oppure la trasformazione dei nomi di funzioni e variabili in modo da renderlo incomprensibile ad un umano.

4.1.11 Escape From PDF

Oltre alle vulnerabilità dello standard PDF e del lettore PDF che permettono di eseguire attacchi in automatico, ci sono anche, come già detto, quelle che facilitano l'applicazione di tecniche di *social engineering* da parte degli attaccanti, affinché una vittima venga convinta ad autorizzare un'azione malevola che altrimenti non avrebbe mai autorizzato, come ad esempio l'apertura di un eseguibile contenente malware. Un esempio di tale tattica è documentato in [27], in cui viene inserito un eseguibile all'interno del file PDF ed eseguito con l'azione *Launch*, la quale chiede conferma all'utente. La vulnerabilità sta non solo nel fatto che la versione del lettore PDF in questione permetta di lanciare eseguibili, cosa perfettamente lecita secondo lo standard PDF, ma anche nel fatto che sia possibile controllare parzialmente il messaggio mostrato all'utente, così da trarlo in inganno e fargli aprire l'eseguibile.

4.1.12 PDF Malware Is Not Yet Dead

Un altro esempio di utilizzo di *social engineering* è riportato in [28], dove viene inserito un documento in formato *docx* all'interno del documento PDF, per la cui apertura è richiesta conferma all'utente. Il documento all'interno viene tuttavia salvato con un nome tale che fa sì che il messaggio per la richiesta di conferma risulti rassicurante per l'utente, che è invogliato ad aprire il documento. Tale documento in formato *docx* si rivela poi essere un malware.

4.2 Categorizzazione degli attacchi

A valle della lettura delle diverse fonti esposte in precedenza, si è giunti alla conclusione che gli attacchi perpetrati attraverso l'utilizzo di documenti PDF possono ricadere in quattro macro categorie:

- Sfruttamento di funzionalità rischiose offerte dallo standard PDF e dalla API JavaScript non adeguatamente limitate dal lettore
- Sfruttamento di vulnerabilità nell'implementazione dello standard PDF del lettore

- Sfruttamento di vulnerabilità nell'implementazione della API JavaScript offerta dal lettore
- Sfruttamento di tecniche di *social engineering* rese possibili da vulnerabilità del lettore

Tale categorizzazione è giustificata dal fatto che il documento PDF non può arrecare danni alla vittima in sé per sé, come potrebbe ad esempio fare un file eseguibile. Ciò infatti può accadere solo nel momento in cui un determinato lettore PDF, o un qualsiasi software che interpreta i documenti PDF, tenta di leggerne il contenuto. Sono infatti presenti diverse funzionalità legittime offerte dallo standard PDF e dalle API JavaScript che, se non adeguatamente limitate dai lettori PDF (utilizzando ad esempio delle richieste di conferma o proibendo del tutto il loro utilizzo), possono essere sfruttate per degli attacchi. Inoltre, sono numerosi i casi in cui l'implementazione di una particolare parte dello standard PDF da parte di una certa versione di un determinato lettore PDF risulta essere affetta da una vulnerabilità. Ciò è valido anche per l'implementazione della API JavaScript offerta da tale lettore PDF, il che vuol dire che alcune funzioni JavaScript che è possibile richiamare da script presenti in un documento PDF hanno delle implementazioni che le rendono vulnerabili a vari tipi di attacchi. Inoltre, è degno di nota il fatto che il codice JavaScript è spesso presente anche per facilitare il compito ad un attaccante nello sfruttare una vulnerabilità che non è relativa alla API JavaScript. Infine, un lettore PDF potrebbe avere delle vulnerabilità nelle funzionalità che offre, che però non sono gravi come le precedenti, nel senso che non è permesso il loro sfruttamento in maniera automatica e trasparente alla vittima. Solitamente in questi casi la vulnerabilità permette di facilitare lo sfruttamento di tecniche di *social engineering* per far sì che la vittima accetti di compiere le azioni volute dall'attaccante.
Di seguito è proposta una mappatura in forma tabellare tra le categorie di attacco individuate e le fonti analizzate. Si noti, inoltre, che una fonte può ricadere in più di una categoria, il che vuol dire che essa parla in maniera trasversale di diverse tipologie di attacco.

ID attacco	Descrizione	Fonti
#1	Sfruttamento di funzionalità rischiose offerte dallo standard PDF e dalla API JavaScript non adeguatamente limitate dal lettore	[12], [13], [14]
#2	Sfruttamento di vulnerabilità nell'implementazione dello standard PDF del lettore	[15], [16], [17]

#3	Sfruttamento di vulnerabilità nell'implementazione della API JavaScript offerta dal lettore	[15], [16], [21], [22], [23], [24], [25]
#4	Sfruttamento di tecniche di social engineering rese possibili da vulnerabilità del lettore	[27], [28]

5 Contromisure per attacchi tramite PDF

In questo capitolo vengono discussi i vantaggi e gli svantaggi dei diversi modi con cui è possibile proteggersi da attacchi che utilizzano documenti PDF. Non esistono purtroppo, infatti, contromisure per proteggersi totalmente da attacchi basati su documenti PDF. Secondo [16] i migliori metodi per proteggersi da tali attacchi sono quelli di **mantenere costantemente aggiornato il proprio lettore PDF** e di **disabilitare l'esecuzione di codice JavaScript al suo interno**. Sulla base del fatto che, come già detto, la maggior parte degli attacchi di questo tipo contiene codice JavaScript, tali metodi risultano essere molto efficaci, ma non permettono di proteggersi da attacchi che sfruttano vulnerabilità non relative a codice JavaScript e non ancora sanate, come ad esempio quelle di tipo *0-day*. Un'ulteriore linea di difesa è, infine, quella di avere un buon livello di *security awareness*, il che vuol dire non aprire documenti PDF della cui provenienza non si è certi.

5.1 Disarmo del documento

Alcune metodologie per proteggersi da questi tipi di attacchi prevedono di effettuare il cosiddetto *disarmo* del documento. Ciò vuol dire trasformare il documento in un nuovo documento, rimuovendo tutto ciò che può essere dannoso, come ad esempio il codice JavaScript. Ci sono diversi metodi per fare ciò, come ad esempio l'approccio *Content Disarm and Reconstruction (CDR)* [41], che esegue la scansione del contenuto del documento e crea un nuovo file con solo il contenuto, il che però non garantisce la preservazione della visualizzazione del documento. Un'altra opzione invece è quella di convertire tutte le pagine del PDF in immagini, preservando la visualizzazione del contenuto ma perdendo in capacità di selezione testo e aumentando di molto la dimensione del file. Tutti questi metodi, però,

potrebbero non essere applicabili in tutte le situazioni, in quanto vanno a cambiare la struttura del documento, che risulterebbe diverso da quello originale.

5.2 Utilizzo di PDF/A

Un metodo efficace per ridurre la probabilità che un documento PDF contenga malware è quello di utilizzare dei documenti che rispettano lo standard PDF/A. Come già detto, un documento che rispetta tale standard non conterrà codice JavaScript, inoltre potrebbe non contenere file al suo interno in base alla versione di PDF/A utilizzata. Di seguito è proposta una rappresentazione tabellare che illustra gli attacchi vanificati da PDF/A in base alla versione dello standard utilizzato:

ID attacco	PDF/A-1	PDF/A-2	PDF/A-3	PDF/A-4
#1	Parzialmente vulnerabile	Parzialmente vulnerabile	Parzialmente vulnerabile	Parzialmente vulnerabile
#2	Parzialmente vulnerabile	Parzialmente vulnerabile	Parzialmente vulnerabile	Parzialmente vulnerabile
#3	Non vulnerabile	Non vulnerabile	Non vulnerabile	Non vulnerabile
#4	Parzialmente vulnerabile	Parzialmente vulnerabile	Vulnerabile	Vulnerabile

Dalla tabella si nota che PDF/A previene totalmente gli attacchi che sfruttano codice JavaScript, tuttavia non riesce a prevenire totalmente le altre categorie di attacco. Per quanto riguarda gli attacchi che sfruttano una incorretta protezione di funzionalità dello standard PDF, lo standard PDF/A offre protezione solo verso lo sfruttamento di funzionalità della API JavaScript, mentre non è in grado di offrire alcuna protezione per quelle dello standard PDF in sé, in quanto tali funzionalità possono essere limitate solo dal lettore PDF utilizzato. Gli attacchi che invece sfruttano le vulnerabilità nelle implementazioni dei lettori PDF, come ad esempio quelle relative ai parser di determinate categorie di oggetti del documento, non sono, in linea di principio, intaccate dal passaggio da PDF a PDF/A. Tuttavia, si ha una parziale vulnerabilità se si considera che, come già detto, la maggior parte degli attacchi che sfruttano tali vulnerabilità utilizzano anche codice JavaScript per facilitare l'operazione. Infine, relativamente alle tecniche di *social engineering*, gli standard PDF/A-1 e PDF/A-2 sono marcati

come parzialmente vulnerabili in quanto non consentono l'inserimento di files arbitrari all'interno del documento, quindi non possono essere attuate tecniche di attacco che tentano di far aprire all'utente ad esempio un file eseguibile o un documento in formato *docx*. Ciò, ovviamente, non preclude l'utilizzo di altre tecniche di attacco basate su *social engineering*. La conversione in PDF/A presenta inoltre non pochi problemi e potrebbe avere gli stessi svantaggi della procedura di disarmo di un documento PDF. Il vantaggio rispetto al disarmo è, però, che esistono software, anche gratuiti, che permettono ad un qualsiasi utente di convertire il documento PDF in un documento PDF/A e assicurarsi che non ci siano artefatti legati alla conversione nel documento finale, prima di inviarlo o conservarlo. Esistono, inoltre, strumenti gratuiti come *veraPDF* [33] che permettono di verificare l'aderenza di un dato documento PDF allo standard PDF/A. Come detto in precedenza, tuttavia, il processo di conversione e verifica di un documento PDF/A non sempre risulta affidabile. Un esempio notevole è il lettore PDF *Anteprima* presente su MacOS, attualmente alla versione 11, che è in grado di creare un documento PDF/A a partire da un normale documento PDF, ma tale documento non è riconosciuto né da *veraPDF*, né da altri validatori PDF/A presenti in rete. Ciò significa che un qualsiasi documento PDF/A generato in buona fede tramite tale strumento da un utente *MacOS* potrebbe essere ingiustamente rigettato se i soli documenti accettati fossero solo quelli aderenti allo standard PDF/A.

5.3 Rilevamento dei documenti PDF malevoli

In base a quanto detto finora, non esiste un modo per evitare totalmente gli attacchi tramite documenti PDF. Il miglior modo per proteggersi risulta quindi quello di rilevare la presenza di un documento PDF malevolo prima della sua apertura o elaborazione. Esistono diversi approcci per effettuare tale rilevamento.

5.3.1 Analisi statica

L'analisi statica si basa su firme o caratteristiche di malware noti, analizzando quindi la struttura del documento PDF. Essa può essere fatta manualmente, in maniera simile agli esempi mostrati nei precedenti capitoli, però tale pratica non è applicabile in scenari in cui la rilevazione debba essere effettuata in maniera automatica. In tal caso, ci si può affidare a strumenti automatici che esaminano la struttura del documento in cerca di tracce di malware noti, cosa che però può facilmente essere aggirata dalle diverse tecniche di offuscamento applicate dagli attaccanti. Inoltre, questo approccio non è in grado di rilevare attacchi ancora sconosciuti, come nel caso di *0-day*.

5.3.2 Analisi dinamica

L'analisi dinamica si basa sull'apertura di un documento PDF tramite un lettore PDF in un ambiente virtuale isolato ed osservare il comportamento del sistema. In questo modo è possibile rilevare anche attacchi non noti, in quanto si osservano comportamenti indesiderati a valle dell'apertura del documento. Ciò però risulta **solitamente impraticabile** nel caso dei documenti PDF, in quanto bisognerebbe idealmente tentare di aprire il documento su tutte le possibili versioni di tutti i lettori PDF esistenti, su sistemi operativi diversi. Inoltre, anche se fosse possibile utilizzarla, spesso l'analisi dinamica è dispendiosa in termini di tempo e risorse.

5.3.3 Analisi tramite algoritmi di intelligenza artificiale

L'utilizzo dell'intelligenza artificiale si basa sulla creazione di modelli di *machine learning* che, a partire da un insieme di documenti identificati come malevoli o benigni raccolti in ciò che viene chiamato *dataset*, permettono di rilevare documenti malevoli anche precedentemente sconosciuti al modello stesso. Tramite tale approccio è quindi possibile individuare anche la presenza di documenti contenenti codice malevolo che non hanno delle firme appartenenti ad indicatori di compromissione noti, a patto che vengano utilizzati modelli e dataset adatti. Nello specifico, le tecniche più comunemente utilizzate in letteratura per assolvere a questo compito sono descritte di seguito.

- *Stacking Learning:* consiste nell'addestrare molti modelli su sottoinsiemi di dati diversi, utilizzando le loro previsioni come input per un meta-modello che effettua le predizioni finali. Questa tecnica permette di combinare le previsioni di molti modelli per creare un modello più accurato e robusto rispetto a qualsiasi modello individuale. Tale approccio può essere utilizzato con qualsiasi tipo di modello e si è dimostrata efficace in una vasta gamma di applicazioni, come mostrato in [37] e [38]. Affinché possano essere utilizzati per l'addestramento, i documenti PDF devono essere prima convertiti in una serie di *features*, ossia caratteristiche estrapolate dai documenti stessi. Una volta addestrati, a tali modelli possono essere fornite in input le *features* di documenti PDF ancora sconosciuti alla rete, ottenendo spesso grandi risultati in termini di precisione e accuratezza del rilevamento.
- *Natural Language Processing (NLP):* si considera il contenuto testuale dei documenti PDF e si effettuano analisi testuali basate su motori che analizzano il flusso testuale e le

eventuali interconnessioni tra le parti semantiche del testo con l'obiettivo di identificare la presenza di anomalie. In questo tipo di approccio non si effettua un'estrazione di *features*, ma si effettua l'addestramento direttamente sul contenuto del testo. Un esempio di applicazione di NLP è fornito in [40], in cui si mostra che un approccio basato su trasformatori [39] permette di avere un buon livello di detection. Quest'ultimo è un progetto sperimentale, che sicuramente però può essere una buona base per valutare l'implementazione di un modello basato su questo approccio.

Tutti questi modelli hanno però uno scopo comune, ossia quello di risolvere un problema di **classificazione**, che consiste nell'individuare delle caratteristiche dei PDF in maniera da suddividerli in due sottoinsiemi:

- Il sottoinsieme di documenti PDF malevoli
- Il sottoinsieme di documenti PDF benigni

I documenti PDF classificati come malevoli saranno quindi i documenti da rigettare, mentre quelli benigni saranno quelli da accettare.

6 Analisi statistiche su PDF malevoli e benigni

In questo capitolo viene presentata un'analisi effettuata su un dataset contenente documenti PDF etichettati come malevoli o benigni. A tal scopo, vengono utilizzati degli strumenti da riga di comando per estrarre informazioni sul loro contenuto JavaScript e sull'eventuale standard PDF/A a cui sono conformi. L'obiettivo è quello di determinare:

- Il numero di documenti PDF malevoli e benigni contenenti codice JavaScript
- Il numero di documenti PDF malevoli e benigni conformi ad una delle possibili versioni dello standard PDF/A

Tali informazioni sono di estremo interesse. Conoscere, infatti, la percentuale di documenti PDF malevoli che contengono JavaScript può essere utile per determinare quanto è possibile limitare la superficie di attacco soltanto evitando di accettare documenti contenenti JavaScript. Allo stesso modo, conoscere quanti documenti PDF sono conformi ad una versione dello standard PDF/A può essere utile per determinare di quanto l'accettazione di soli documenti PDF/A può ridurre la superficie di attacco.

6.1 Informazioni sul dataset

Il dataset utilizzato per l'analisi è *CIC-Evasive-PDFMal2022* [30], il quale è stato creato con l'obiettivo di realizzare un dataset di documenti PDF le cui caratteristiche siano tali che li rendano difficili da classificare come benigni o malevoli. In particolare, sono stati raccolti 11173 file malevoli e 9109 file benigni da Contagio Malware Dump [31], oltre a 20000 file malevoli da *VirusTotal* [32]. Contagio Malware Dump è un sito web che raccoglie campioni di malware e li mette a disposizione della comunità di ricercatori per analisi e studio, tra cui figurano anche documenti PDF malevoli. *VirusTotal* è un sito web che fornisce un servizio online gratuito per l'analisi di file e URL per la rilevazione di virus, worm, trojan e altri tipi di contenuti malevoli. *VirusTotal* consente agli utenti di caricare file o inserire URL da analizzare tramite oltre 70 motori antivirus e altri strumenti di sicurezza. Il servizio fornisce inoltre un report sui risultati delle scansioni, incluso l'elenco delle minacce rilevate e dei motori che le hanno rilevate. I file analizzati sono condivisi pubblicamente ed è possibile effettuare una sottoscrizione che permette il download di tali file.
La creazione del dataset viene trattata nel dettaglio in [37], dove viene inoltre evidenziato che l'insieme di file ottenuti da Contagio Malware Dump, nonostante sia stato utilizzato da tanti lavori in letteratura, è abbastanza vecchio ed è soggetto a diversi problemi, come ad esempio la presenza di duplicati. In tale lavoro viene infatti evidenziato che il dataset di Contagio non è sufficientemente generico, in quanto si focalizza solo su alcune caratteristiche. Inoltre, gli autori identificano le features più rappresentative che permettono di identificare un PDF malevolo. Infine, viene mostrato che con un approccio basato sullo *stacking learning* [38] è possibile avere un elevato livello di detection.
Ai fini della creazione del dataset finale a partire da Contagio e *VirusTotal*, è stato creato un record per ciascun file PDF attraverso l'estrazione di 37 features, tra cui 12 generali e 25 strutturali. Le features generali descrivono il file PDF in termini di dimensioni, presenza di crittografia, numero di pagine, presenza di testo o immagini e altre caratteristiche. Le features strutturali forniscono una descrizione più approfondita del file PDF e includono features come il numero di oggetti, flussi e filtri utilizzati, nonché la presenza di parole chiave come Javascript, URI e launch. Tali features sono state estratte mediante l'ausilio dello strumento a riga di comando *pdfid*. Per creare l'insieme evasivo di record malevoli, è stato utilizzato un algoritmo di *unsupervised machine learning* chiamato *K-means* per raggruppare i record in due gruppi in base alla loro somiglianza. I record relativi a campioni malevoli che sono finiti nel gruppo sbagliato sono stati identificati come malware evasivi. La stessa logica è stata applicata ai record relativi a campioni benigni. In questo modo, sono stati selezionati soltanto i record relativi a campioni malevoli che sono simili a quelli benigni e i record relativi a campioni benigni che sono simili a quelli malevoli. I risultati sono stati combinati per creare il dataset finale, che è composto da 10025 record, di cui 5557 malevoli e 4468 benigni, con associati documenti PDF.

6.2 Utilizzo dei documenti PDF del dataset

Dalla pagina ufficiale del dataset è possibile scaricare gli archivi contenenti i documenti PDF utilizzati per la sua creazione. Dopo aver estratto i documenti dagli archivi, sono stati rimossi i duplicati, ottenendo un totale di 9094 documenti benigni e 12945 documenti malevoli.

6.3 Strumenti utilizzati per l'analisi

Per l'analisi sono stati utilizzati due strumenti a riga di comando con l'obiettivo di determinare il contenuto in termini di codice JavaScript dei documenti PDF e l'eventuale standard PDF/A a cui aderiscono.

6.3.1 peepdf

Lo strumento utilizzato per la verifica della presenza di codice JavaScript all'interno dei documenti PDF analizzati è *peepdf*. Esso è scritto in Python ed offre varie funzionalità, tra cui quella di vedere tutti gli oggetti presenti nel documento, evidenziando quelli contenenti codice JavaScript. Inoltre, se il documento tenta di sfruttare una vulnerabilità nota, essa viene riportata dallo strumento. Oltre alla modalità da riga di comando classica, offre anche una comoda modalità interattiva a riga di comando, che permette di analizzare il documento in maniera approfondita. L'utilizzo di *peepdf* è stato preferito a quello di *pdfid* in quanto, malgrado anche il secondo permetta di ottenere una panoramica degli oggetti presenti nel documento, il primo permette di effettuare un'analisi più approfondita del documento e della sua struttura. Un esempio a favore di questa scelta è riportato in [24], dove *pdfid* non rileva la presenza di codice JavaScript nel documento in quanto inserito all'interno di un form *XFA*, mentre *peepdf* riesce a rilevare la sua presenza.

6.3.2 veraPDF

Lo strumento utilizzato per la validazione dei documenti PDF secondo le varie versioni dello standard PDF/A è *veraPDF*. Esso è uno strumento open source che permette di verificare che un documento PDF sia conforme a una delle varie versioni dello standard PDF/A. Una delle caratteristiche principali di veraPDF è la sua capacità di validare i file PDF/A in modo accurato e completo. Il software utilizza un'ampia gamma di regole e controlli per verificare che il file sia conforme allo standard PDF/A, compresi controlli sulla struttura del documento, sui font, sulle

immagini, sulle annotazioni, sui metadati e su altri elementi. Inoltre, *veraPDF* supporta tutti i profili PDF/A e può essere utilizzato per verificare la conformità dei file in qualsiasi di questi formati. Lo strumento offre sia una versione con interfaccia grafica, sia una utilizzabile da riga di comando. Può inoltre essere sia utilizzato come strumento autonomo, sia integrato in altre applicazioni o processi. Ad esempio, il software può essere utilizzato per verificare la conformità dei documenti PDF/A prima dell'archiviazione a lungo termine, o come parte di un processo di validazione automatizzato.

6.4 Risultati dell'analisi

Per l'analisi sono stati utilizzati i documenti PDF del dataset precedentemente descritto, a valle della fase di rimozione dei duplicati. L'analisi è stata condotta sia sui documenti malevoli che su quelli benigni. Il numero totale di documenti benigni analizzati è 9094 mentre il numero totale di documenti malevoli analizzati è 12945.

6.4.1 Analisi sul contenuto di codice JavaScript

I risultati dell'analisi sono presentati nella seguente tabella.

Categoria	Documenti totali	Documenti con JS	Percentuale di documenti con JS
Benigni	9094	4100	45.08%
Malevoli	12945	12051	93.09%

Come è possibile notare dalla tabella, la quasi totalità dei documenti PDF malevoli presenta un contenuto di codice JavaScript, il che avvalora lo studio effettuato in precedenza, in quanto il codice JavaScript può essere sia utilizzato per sfruttare vulnerabilità della API JavaScript del lettore, sia per facilitare lo sfruttamento di altri tipi di vulnerabilità. Ciò, inoltre, si verifica nonostante i documenti analizzati rispettassero le caratteristiche di evasività descritte nel dataset di cui fanno parte. Sempre dalla tabella è possibile inoltre notare che quasi la metà dei

documenti benigni presenta codice JavaScript. Questo, di nuovo, potrebbe essere dovuto alle caratteristiche di evasività di tali documenti, che sarebbero tali da potersi confondere con documenti malevoli. Visto che la caratteristica principale dei documenti malevoli sembra essere la presenza di codice JavaScript, ciò giustificherebbe la così alta percentuale di documenti benigni che lo contengono. In base allo studio effettuato in precedenza, infatti, la presenza di codice JavaScript nei documenti benigni si limita spesso alla verifica di campi delle form e poco più, il che può tranquillamente essere evitato. Da ciò si evince che la non accettazione di documenti contenenti codice JavaScript può essere una già forte linea di difesa contro attacchi tramite documenti PDF, senza compromettere troppo l'usabilità dei documenti stessi.

6.4.2 Analisi sul formato PDF/A

I risultati dell'analisi sono presentati nella seguente tabella.

Categoria	Documenti totali	Documenti PDF/A	Percentuale di documenti PDF/A
Benigni	9094	1	0.01%
Malevoli	12945	0	0%

Come è possibile notare dalla tabella, nessuno dei documenti PDF malevoli risulta essere aderente allo standard PDF/A, però ciò è praticamente valido anche per i documenti PDF benigni, dove un solo documento risulta essere aderente a tale standard. Tale risultato è, però, probabilmente dovuto al fatto che i campioni presenti nel dataset analizzato non prevedevano quasi nessun documento aderente allo standard PDF/A. Inoltre, lo strumento *veraPDF* utilizzato per la validazione dei documenti rispetto allo standard PDF/A potrebbe essere in disaccordo con la struttura del documento PDF/A generata da alcuni strumenti.

6.5 Limiti dell'analisi

L'analisi è stata effettuata sul dataset *CIC-Evasive-PDFMal2022*, il quale raccoglie sia documenti PDF selezionati dalla nota collezione di malware Contagio, sia documenti PDF ottenuti da

sottomissioni degli utenti su *VirusTotal*. In particolare, da questi sono filtrati soltanto quei documenti PDF che risultano avere caratteristiche cosiddette evasive, il che vuol dire che i documenti presi in considerazione sono soltanto quelli benigni che hanno caratteristiche di documenti malevoli e viceversa. Tuttavia, nonostante tale caratteristica può essere molto utile ai fini di un'analisi a sé stante, potrebbe non rispecchiare la realtà dei documenti PDF scambiati nell'ambito delle Pubbliche Amministrazioni in Italia, che è per l'appunto l'ambito in cui opera il cliente PagoPA.
Per superare tali limiti, emerge la necessità di effettuare l'analisi su un dataset più rappresentativo della realtà in esame. Tale attività è attualmente in corso e consiste nell'ottenimento di documenti PDF sottomessi dagli utenti a diverse piattaforme di *threat intelligence*, in maniera simile a come avviene con VirusTotal. Tali documenti risulteranno non solo più aggiornati, ma anche non filtrati rispetto alle loro caratteristiche di evasività. In particolare, sono tutt'ora in corso le estrazioni da:

- *MalwareBazaar*, che è un progetto di Abuse.ch con l'obiettivo di condividere campioni di malware con la comunità della sicurezza informatica, i fornitori di antivirus e i fornitori di informazioni sulle minacce [34]
- *VirusShare*, che è un repository di campioni di malware con l'obiettivo di fornire a ricercatori di sicurezza e analisti l'accesso a campioni di malware in tempo reale [35]
- *Hybrid Analysis*, che è un servizio gratuito di analisi malware che rileva e analizza le minacce sconosciute utilizzando un'esclusiva tecnologia di analisi ibrida, basata su *CrowdStrike Falcon Sandbox* [36]

Mentre per *MalwareBazaar* non è necessaria alcuna registrazione, per gli altri due servizi è necessario un processo di verifica dell'utente per poter scaricare i campioni di malware. Per l'estrazione da *MalwareBazaar* e *VirusShare*, sono stati creati degli script Python che, in maniera automatica, effettuano il download degli archivi di campioni presenti sul sito e ne estraggono solo i documenti PDF. Per *Hybrid Analysis*, invece, è presente un limite di download di 200 campioni al giorno, motivo per cui lo script realizzato effettua il download una volta al giorno di 200 campioni sottomessi per ogni mese.
Tali dataset possono risultare più vicini alla realtà in quanto frutto di sottomissioni degli utenti di tutto il mondo, tuttavia il miglior modo per effettuare un'analisi è quello di utilizzare i documenti PDF forniti dal cliente PagoPA stesso.

7 Considerazioni finali

Le analisi effettuate nel presente documento evidenziano le seguenti considerazioni finali.

7.1 PDF malevoli e codice JavaScript

La maggior parte dei documenti PDF malevoli contiene codice JavaScript, sia che esso sia utilizzato per sfruttare le vulnerabilità dell'implementazione della API JavaScript del lettore, sia per rendere più facile lo sfruttamento di altri tipi di vulnerabilità del lettore. Tuttavia, anche molti documenti PDF benigni risultano avere la presenza di codice JavaScript ma, come già detto, tali documenti potrebbero non rappresentare la realtà dei fatti, in quanto esibiscono caratteristiche di evasività che gli hanno permesso di far parte del dataset analizzato. Dallo studio precedentemente effettuato risulta, infatti, che il codice JavaScript nei documenti PDF non malevoli viene solitamente utilizzato solo per il controllo dei campi delle form e poco più, il che può in molti casi essere tranquillamente evitato. Da ciò, risulta che la disabilitazione del JavaScript e/o la non accettazione di documenti PDF contenenti JavaScript può diminuire il rischio di molto.

7.2 Utilizzo di PDF/A come contromisura

L'approccio di accettare solo documenti PDF/A non solo include il precedente approccio di non permettere codice JavaScript nei documenti, ma permette anche di evitare che alcune tipologie di file possano essere inclusi nel documento e utilizzati per sfruttare vulnerabilità del lettore o trarre in inganno l'utente. Tuttavia, tale ultimo approccio si rivela spesso non attuabile, soprattutto a causa del fatto che gli strumenti di creazione di file PDF/A e i validatori PDF/A risultano spesso in disaccordo, causando potenzialmente il rigetto di documenti che potrebbero essere invece processati tranquillamente.

7.3 Utilizzo dell'intelligenza artificiale

Nonostante la presenza di tali misure di protezione, soprattutto valide per i documenti contenenti codice JavaScript, emerge comunque la necessità di utilizzare anche una metodologia di rilevamento dei documenti PDF basata su algoritmi di intelligenza artificiale che riesca a rilevare con un alto livello di confidenza un PDF malevolo, cercando di ridurre al minimo i falsi positivi (documenti benigni identificati come malevoli) e i falsi negativi (documenti malevoli identificati come benigni).

8 Next steps

Come suggerimento per le prossime attività, si consiglia di:

- Effettuare una valutazione sulla possibilità di **non accettare documenti contenenti JavaScript**
- Valutare la possibilità di **accettare solo documenti conformi allo standard PDF/A**

Tali valutazioni devono essere effettuate **analizzando le statistiche dei documenti PDF che PagoPA quotidianamente utilizza.** Si noti inoltre che, nonostante la seconda sia la soluzione più sicura, andrebbe valutata attentamente in quanto molti documenti esportati da software (come ad esempio *Anteprima* presente in macOS), non effettuano un'esportazione completamente conforme.

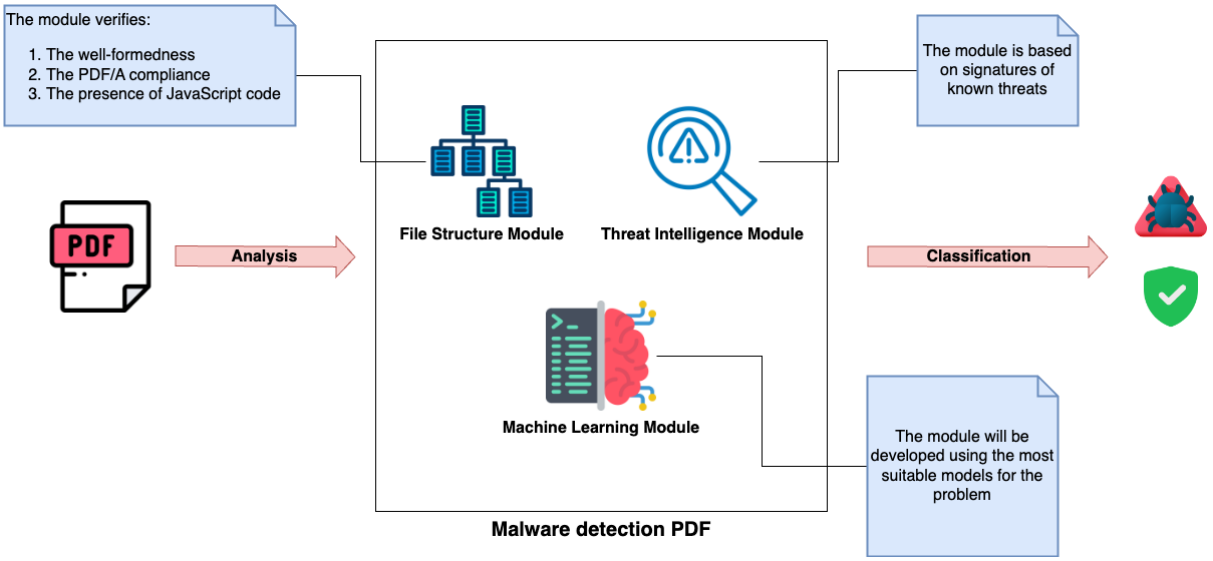
Attraverso le tecniche evidenziate in questo lavoro, inoltre, è possibile individuare quali sono le caratteristiche peculiari dei documenti PDF gestiti da PagoPA, ed eventualmente ottimizzare i modelli di machine learning per operare sullo specifico insieme di documenti gestiti da PagoPA. A valle di queste analisi sarà possibile implementare il **"PagoPA PDF Malware Detection Module"**, il quale sarà composto da:

- Un **"File Structure" module,** che effettua la validazione dei documenti in base alla loro struttura. In particolare, il modulo potrebbe implementare i seguenti sottomoduli:
    - **PDF/A validator:** modulo di validazione PDF/A che rigetta i documenti ritenuti non idonei.
    - **JavaScript validator**: modulo di validazione JS che rigetta i documenti contenenti codice JavaScript.

  E' da sottolineare che, qualora si implementi il modulo di validazione PDF/A, questo modulo non sarà necessario, in quanto un documento PDF/A per sua natura non conterrà codice JavaScript.
- Un **"Threat Intelligence" module**, che implementa una logica di controllo basata sugli indicatori di compromissione (IOC). In particolare, l'obiettivo è tenere aggiornata una lista completa di hash associati a documenti malevoli ed effettuare un controllo sulle anomalie presenti. Questo approccio protegge da molte minacce ma ha i limiti di tutti i sistemi di detection *signature-based*, tra cui il fatto che basta una piccola modifica al malware per evadere la detection e il fatto che non permette di rilevare attacchi per i quali non è già disponibile una firma.

- Un **"Machine Learning Module"**, che sfrutta le tecniche definite nel paragrafo 5.3.3 per implementare il miglior sistema di machine learning per il caso d'uso in esame. Si partirà effettuando una valutazione su dataset esistenti per addestrare un classificatore e si effettueranno delle valutazioni per individuare l'architettura avente la maggiore accuratezza dei risultati di detection. In particolare, gli approcci che verranno presi maggiormente in considerazione sono i seguenti:
    - L'approccio basato su *stacking learning* presentato in [37]
    - L'approccio basato su *trasformatori* presentato in [40]

Il seguente diagramma illustra l'ipotesi di architettura del modulo di detection proposto.



Bibliografia

[1] International Organization for Standardization. (2008). Document management — Portable document format — Part 1: PDF 1.7 (ISO Standard No. 32000-1:2008). https://www.iso.org/standard/51502.html

[2] PDF. (2023). *Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=PDF

[3] History_of_PDF. (2023). *Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=History_of_PDF

[4] Lukan, D. (2020, September 26). *PDF file format: Basic structure [updated 2020]*. Infosec Resources. https://resources.infosecinstitute.com/topic/pdf-file-format-basic-structure/

[5] Gavin, M. (2010, August 26). *PDF Cross Reference Table*. Appligent Labs. https://labs.appligent.com/pdfblog/pdf_cross_reference_table/

[6] *Introduction to PDF*. (2010, May 27). GNUpdf. Retrieved October 10, 2014, from https://web.archive.org/web/20141010035745/http://gnupdf.org/Introduction_to_PDF

[7] johan. (2013, January 9). *What do we mean by "embedded" files in PDF?* Open Preservation Foundation. https://openpreservation.org/blogs/what-do-we-mean-embedded-files-pdf/

[8] Esparza, J. M., [jesparza]. *Actions in the Portable Document Format (PDF)*. (2008, November 17). eternal-todo.com. https://eternal-todo.com/blog/pdf-actions

[9] PDF/A. (2023). *Wikipedia, L'enciclopedia libera*. https://it.wikipedia.org/w/index.php?title=PDF/A

[10] PDF/A. (2023). *Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=PDF/A

[11] *The PDF/A Family of Archiving Standards*. (n.d.). PDFlib. https://www.pdflib.com/pdf-knowledge-base/pdfa/the-pdfa-standards/

[12] Lindenhofer, J., Offenthaler, R., & Pirker, M. (2020). A Curious Exploration of Malicious PDF Documents. *International Conference on Information Systems Security*. https://doi.org/10.5220/0008992305770584

[13] Müller, J., Noss, D., Mainka, C., Mladenov, V., & Schwenk, J. (2021). Processing Dangerous Paths – On Security and Privacy of the Portable Document Format. *Network and Distributed System Security Symposium*. https://doi.org/10.14722/ndss.2021.23109

[14] *NTLM Credentials Theft via PDF Files*. (2018, April 26). Check Point Research. https://research.checkpoint.com/2018/ntlm-credentials-theft-via-pdf-files/

[15] Maiorca, D., Biggio, B., & Giacinto, G. (2018). Towards Adversarial Malware Detection: Lessons Learned from PDF-based Attacks. *ArXiv (Cornell University)*. http://arxiv.org/pdf/1811.00830

[16] *Computer Emergency Response Team - Industrie Services et Tertiaire*. (2010, February 9). Cert-IST. https://www.cert-ist.com/public/en/SO_detail?code=malicious_pdf

[17] Sineath, B. (2009, March 9). *Analysis of CVE-2009-0658 (Adobe Reader 0day)*. Secureworks. https://www.secureworks.com/blog/research-20947

[18] Esparza, J. M., [jesparza]. (n.d.). *peepdf* [Software]. https://github.com/jesparza/peepdf

[19] Stevens, D., [DidierStevens]. (n.d.). *pdfid* [Software]. https://github.com/DidierStevens/DidierStevensSuite/blob/master/pdfid.py

[20] Stevens, D., [DidierStevens]. (n.d.). *pdf-parser* [Software]. https://github.com/DidierStevens/DidierStevensSuite/blob/master/pdf-parser.py

[21] Saifullah, K. (2022, March 10). *Analyzing Malicious PDF Files - GISPP - Global InfoSec Pakistani Professionals*. GISPP - Global InfoSec Pakistani Professionals. https://www.gispp.org/2022/03/10/analyzing-malicious-pdf-files/

[22] Guðjónsson, K. (2009, December 14). *SANS Digital Forensics and Incident Response Blog | PDF malware analysis | SANS Institute*. https://www.sans.org/blog/pdf-malware-analysis/

[23] Douglas, K. (2017, April 4). *A Guided Tour of a Classic PDF-Based Malware Dropper*. LinkedIn.

https://www.linkedin.com/pulse/guided-tour-classic-pdf-based-malware-dropper-kevin-douglas-1/

[24] Rocha, L. (2014, September 22). *Malicious Documents – PDF Analysis in 5 steps*. Count Upon Security. https://countuponsecurity.com/2014/09/22/malicious-documents-pdf-analysis-in-5-steps/

[25] Elingiusti, M., Aniello, L., Querzoni, L., & Baldoni, R. (2018). PDF-Malware Detection: A Survey and Taxonomy of Current Techniques. *Advances in Information Security*, 169–191. https://doi.org/10.1007/978-3-319-73951-9_9

[26] Stevens, D. (2008, April 29). *PDF, Let Me Count the Ways*. Didier Stevens Blog. https://blog.didierstevens.com/2008/04/29/pdf-let-me-count-the-ways/

[27] Stevens, D. (2010, March 29). *Escape From PDF*. Didier Stevens Blog. https://blog.didierstevens.com/2010/03/29/escape-from-pdf/

[28] Schläpfer, P. (2022, May 20). *PDF Malware Is Not Yet Dead*. HP Threat Research Blog. https://threatresearch.ext.hp.com/pdf-malware-is-not-yet-dead/

[29] Fleury, N., Dubrunquez, T., & Alouani, I. (2021). PDF-Malware: An Overview on Threats, Detection and Evasion Attacks. *ArXiv*. https://doi.org/10.48550/arXiv.2107.12873

[30] Issakhani, M., Victor, P., Tekeoglu, A., & Lashkari, A. H. (2022). *CIC-Evasive-PDFMal2022* [Dataset]. Canadian Institute for Cybersecurity (CIC). https://www.unb.ca/cic/datasets/pdfmal-2022.html

[31] Parkour, M. (2013, March 24). *16,800 clean and 11,960 malicious files for signature testing and research.* Contagio Malware Dump. https://contagiodump.blogspot.com/2013/03/16800-clean-and-11960-malicious-files.html

[32] *VirusTotal*. (n.d.). https://www.virustotal.com/

[33] *veraPDF*. (n.d.). [Software]. https://verapdf.org/

[34] abuse.ch. (n.d.). *MalwareBazaar - Malware sample exchange*. https://bazaar.abuse.ch/

[35] Corvus Forensics. (n.d.). *VirusShare.com*. https://virusshare.com/

[36] CrowdStrike. (n.d.). *Hybrid Analysis*. https://www.hybrid-analysis.com/

[37] Issakhani, M., Victor, P., Tekeoglu, A., & Lashkari, A. H. (2022, February). *PDF Malware Detection based on Stacking Learning*. In ICISSP (pp. 562-570).

[38] Pavlyshenko, B. (2018, August). *Using stacking approaches for machine learning models*. In 2018 IEEE second international conference on data stream mining & processing (DSMP) (pp. 255-258). IEEE.

[39] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). *Attention is all you need*. Advances in neural information processing systems, 30.

[40] Liu, B., Hukill, C., Zhang, J., & Alandary, S. (2022, December 3). *ECE 188: Computer Security. Repository for "NLP-based Malware Detection on PDFs". Utilizing NLP techniques & transformer models to perform malware detection in PDFs*. GitHub. https://github.com/bliutech/nlp-pdf-malware-detection

[41] Content Disarm & Reconstruction. (2023). *Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Content_Disarm_%26_Reconstruction