L-005

構成要素の関係性を利用した悪性 PDF ファイルの検知
Malicious PDF Detection using Relation of Components

ドゥアンパチャン　カンボリスット † Khambolisouth Duangphachan
今泉　貴史 †‡ Takashi Imaizumi

1. はじめに

近年、特定の企業や個人の情報を狙う標的型攻撃が急増しいる。攻撃者は標的の脆弱性を調べ、それに特化したマルウェアを作成するため、現在、既存の汎用的な対策手法では対処しきれなくなっている。

最も典型的な手法では、ターゲットの信頼する相手になりすまし、マルウェアを偽装メールに添付してターゲットに送信する。このとき、添付ファイルはアイコンや拡張子を偽装した実行可能ファイルである。さらに、ターゲットが見分けられないように、文書ファイルに埋め込んで送付されることも多く見られる。文書ファイルに埋め込む場合、マルウェア本体を特有のカギでエンコードしたり、文書ファイルの圧縮機能を用いて難読化したりすることで、ターゲット側の検知をくぐり抜けることができる [1]。

文書ファイルの中でも、特に PDF(Portable Document Format) ファイルから不正コード（exploit : 脆弱性を利用した悪意のあるコード）や、マルウェア本体が多く検出されている [2]。本論文では、不正コードやマルウェア本体が埋め込まれる PDF ファイルを悪性 PDF ファイルと呼ぶ。悪性 PDF ファイルは、文書を表示させるためのダミーファイルの部分、閲覧ソフトウェアの脆弱性を突く不正コードの部分と、マルウェア本体の部分から構成される [3]。

不正コードやマルウェア本体は難読化されているため、コンテンツのみの調査におけるマルウェア本体を抽出する静的解析手法では、こういった難読化の問題がある [4][5]。

本論文のアプローチとして、PDF ファイルの構成要素とそれらの関係性の観点から、悪性 PDF ファイルを効率的に検知することを提案する。しかし、良性と悪性 PDF ファイルを区別するための特徴が必要となっており、実際に実行ファイルが埋め込まれた悪性 PDF ファイルの構成要素を手動で分析し、特徴を分類する。そこで本論文の目的を、悪性 PDF ファイルを検知することとする。

2. 関連研究

従来のパターンマッチングという対策手法では、事前にマルウェアをコードレベルで手動分析をし、個々のマルウェアの特徴をパターン（シグネチャ）としてデータベースに登録しておく。以降、同じパターンを持つファイルが現れた場合、データベースのパターンと比較することで、マルウェアであるかどうかを判断する。しかし、この手法は個々のパターンを抽出するのに時間と労力がかかるため、近年急増するマルウェアの種類に対し、効率的な手法ではない。さらに、前述したように、標的型攻撃は事前にターゲットの情報や脆弱性を収集しているため、まだパターンが登録されていないマルウェアを送り込んで、攻撃することも多数存在する。

この問題を解決するものとして、実際に被疑ファイルを仮想環境で処理し、動的解析手法 [6] が提案されている。これは、コードレベルを解析するより、実際に挙動を観測することで検知作業の効率化を図る目的である。しかし、マルウェアが不審な行動を起動する条件は複雑になっており、仮想環境側にも様々な条件を用意する必要がある。例えば閲覧ソフトウェアの複数のバージョンで被疑ファイルを実行するなど。さらに、マルウェアに仮想環境の検知機能を追加し、仮想環境内では正常に振る舞うことで、解析から逃れるものも出現している。

動的解析手法に対し、よりマルウェアの検知率を上げるために、従来のパターンマッチングのような静的解析手法が再び注目されている。静的解析ではコードレベルでの調査を行うが、埋め込みや難読化が行われている場合に、ファイルの構成要素を用いる検知手法が提案されている [5]。この検知手法は、ファイルの構成を分割した各項目の情報を元に良性と悪性のファイルを識別する。これを実現するために、大量のサンプルデータセットを機械学習で分析し、識別処理を行う。しかし、この手法の検知精度は機械学習やサンプルの偏りに左右されるほか、大量の学習サンプルファイルが必要となるという問題がある。

類似のアプローチとして、機械学習を必要としない PDF ファイルの構成要素を検査する手法が提案されている [7]。この手法では、PDF ファイルの構成に従って、定義されている仕様と異なる構成要素に注目する。これを実行ファイルが埋め込まれた PDF ファイルとみなし、悪性 PDF ファイルの検知が行われる。しかし、最近の悪性 PDF ファイルは巧妙化してきており、良性と異なる構成要素を持たない悪性 PDF ファイルも存在している。このような悪性 PDF の種類には対応できないという問題がある [8]。本論文では、PDF ファイルの構成要素とそれらの関係性を考慮し、悪性 PDF ファイルを検知する。

†千葉大学大学院融合科学研究科
‡千葉大学統合情報センター

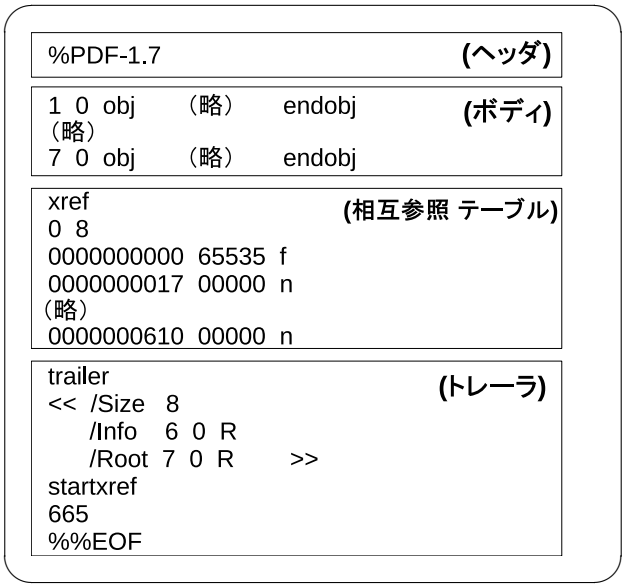

図 1: PDF ファイルの基本要素の形式

3.PDF ファイルの構成要素

3.1.PDF ファイルの基本要素

PDF ファイルは、ヘッダ、ボディ（文書内容の本体）、相互参照テーブル、およびトレーラの 4 つのセクションから構成される。単純な PDF ファイルの基本要素の例を図 1 に示す。

ヘッダは、コメントを意味する「%」文字で始まる 1 行のセクションである。ここに記載された情報は PDF ファイルのバージョンを表す。

ボディは、各ページの内容（コンテンツとリソース）、画像データ等といった複数のオブジェクトが記載される。オブジェクトとは、ファイルを構成する様々な要素である。各オブジェクトには番号が割り振られており、「obj」 の前の数字はそれぞれ、オブジェクト番号と世代番号である。他のオブジェクトから参照されるときに、この番号が用いられる。また、オブジェクトの内容は「obj」と「endobj」のタグで囲まれる。

相互参照テーブルは、ボディの中の各オブジェクトの位置情報を番号順に一覧化したセクションである。各オブジェクトごとの開始位置情報、世代番号および使用状況（f：未使用、n：使用中）の情報を表す。

トレーラは、文書内容を示す Catalog オブジェクト (図 1 では Root) と文書情報を示す Info オブジェクトが格納されているセクションである。文書内容のオブジェクトは階層的な構造となっており、階層のルート（最上位の）オブジェクトは Catalog である。Info オブジェクトは、PDF ファイルの作成者や作成日など文書に関する情報のオブジェクトである。他に、相互参照テーブルの位置情報が記述されている。

表 1: 主なオブジェクトの属性

属性	役割
Catalog	文書内容のルートオブジェクトである
OpenAction	文書を開くときの動作を指定する
Outlines	文書の概要を階層で示す（しおり）
JavaScript	JavaScript のインタプリタで 記述されたスクリプトのコードを実行する
Pages	ページ一覧のオブジェクトであり、 Page の親（Parent) である
Page	単体のページオブジェクトであり、 Pages の子（Kids) である
Fonts	使用するフォントを指定する
Filter	圧縮の機能を指定する
Length	圧縮されたデータのバイト数である
Info	文書情報のオブジェクトである

3.2. 主なオブジェクトの属性

PDF ファイルには、様々な属性を持つオブジェクトが存在しており、表 1 は主な属性の役割をまとめたものである。

Catalog は、文書内容のオブジェクト階層の最上位に位置する属性であり、すべての文書内容の情報を格納している。OpenAction を指定すると、文書を開くときに指定された動作が自動的に実行される。Pages と Page は、お互いに親と子の関係を持っており、Page は PDF ファイルの各ページの内容を格納する。Filter とフィルタ名を指定することで、圧縮されたストリームがオブジェクトに格納される。Length は、圧縮されたデータのバイト数を格納する。

3.3. 良性 PDF の構成要素

良性 PDF ファイルの構成要素を分析し、図 2 のような構成図にまとめる。なお、本論文では、PDF ファイルの構成要素からなる構成図は有向非循環グラフ (DAG:Directed Acyclic Graph) として考える。

図内の 1 つの長方形は 1 つのオブジェクトを表しており、最初のオブジェクトは必ず「Trailer」から始まる。長方形の上部はオブジェクトの番号を表しており（「Trailer」の場合はオブジェクトの数）、下部はオブジェクトのタイプを表している。矢印は参照する方向を表している。矢印の元の文字列は、参照するオブジェクトの属性を表しており、矢印の先の文字列は、参照されるオブジェクトの属性を表している。

3 番の Pages オブジェクトと 4 番の Page オブジェクト間を考えると、次に Pages が参照するオブジェクトの属性は「Kids」であるのに対し、Page が参照されるオブジェクトの属性は「Parent」である。このように、参照する側と参照される側が示されている場合は、お互いの宣言されている情報が一致するかどうかも加えて考慮する。

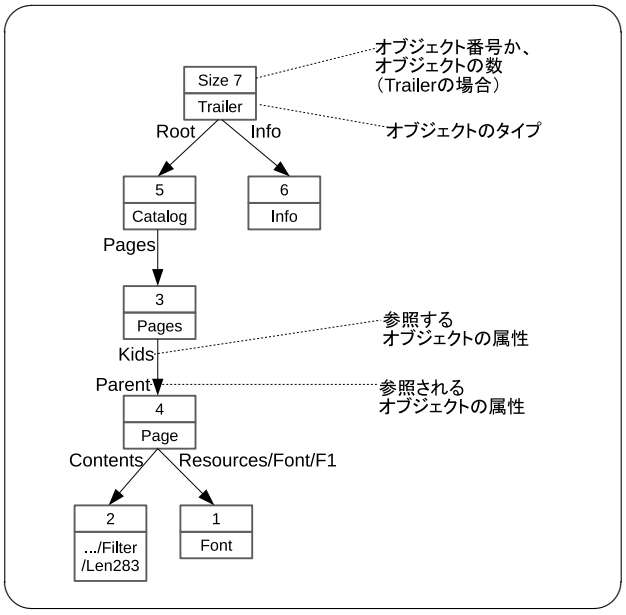

図 2: 簡単な良性 PDF ファイルの構成図

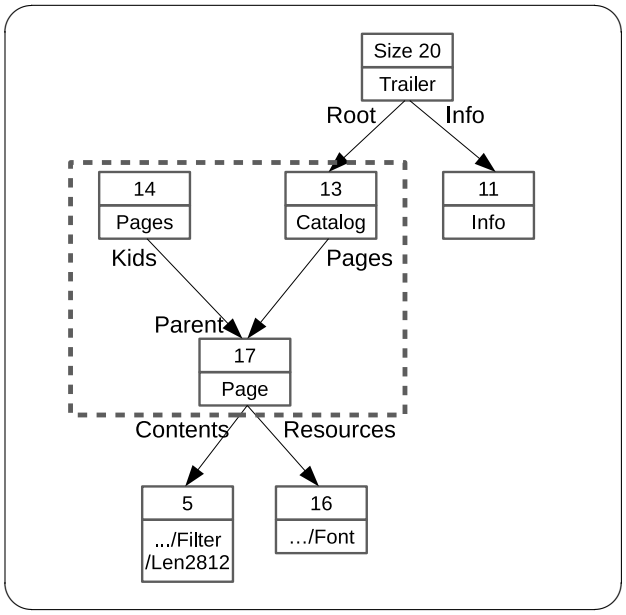

図 3: オブジェクト参照の矛盾の例

最後に、2 番のオブジェクトのタイプは省略されており、図 2 では「...」となっている。さらに、後ろの「/Filter」はオブジェクトが圧縮されていることを示し、「/Len283」は圧縮されたデータのバイト数が 283 であると示している。

4. 悪性 PDF の構成要素

前述したように、悪性 PDF は、ダミーファイル、不正コードとマルウェア本体から構成される [3]。不正コードは、閲覧ソフトウェアの脆弱性を突くものであり、PDF ファイル自身の構成要素の一部である。このとき、構成要素の観点からみると、良性 PDF ファイルと異なる要素の特徴がみられないと考えられる。一方、実行ファイルであるマルウェア本体は、PDF ファイルにとって異物であるため、良性 PDF ファイルと異なる要素の特徴が存在すると考えられる。

悪性 PDF ファイルの構成要素を分析し、判明した悪性 PDF ファイルの特徴を以下に示す。

4.1. 特徴 1：基本要素の欠落

基本要素の一つである相互参照テーブルが欠落しているファイルが存在する。これは、PDF ファイルを先頭から最後まで読み込んで、相互参照テーブルセクションを示す文字列の「xref」と、それ以降に続く 10 桁と 5 桁の数字の羅列がどこにも存在しないことを意味する。

4.2. 特徴 2：オブジェクト参照の矛盾

通常の PDF ファイルでは、あるオブジェクトが他のオブジェクトを参照するといった一連の情報を元に、アクセスすれば PDF ファイルの文書内容をページに表示させることができる。しかし、悪性 PDF ファイルでは、この一連の参照が一元に構成されない特徴がある。オブジェクト参照の矛盾の例を図 3 に示す。

図 3 の点線の枠について考える。13 番の Catalog オブジェクトが 17 番の Pages オブジェクトを参照すると示しているのに対し、17 番が自分は Page オブジェクトであり、14 番の Pages オブジェクトに参照されると示している。このように、各オブジェクトの宣言が矛盾になっている特徴をオブジェクト参照の矛盾に分類する。

また、オブジェクト A がオブジェクト B を参照すると示しているにもかかわらず、参照先のオブジェクト B が存在しない場合も、オブジェクト参照の矛盾とする。

4.3. 特徴 3：不自然な構成要素

図 4 の全体では、良性 PDF ファイルの特徴と比較すると、良性 PDF ファイルと異なる特徴的な構成図が見られない。しかし、図 4 の点線の枠について注目すると、文書情報を格納する Info オブジェクトから Creator オブジェクトに参照する流れを確認できる。さらに、8 番の Creator オブジェクトには、不自然な要素を確認できる。

Creator オブジェクトとは、この PDF ファイルに変換される前の元のアプリケーション名であり、あるいは、PDF ファイルの作成ツール名のことを示すオブジェクトである。図 4 では、作成ツール名の情報を格納するにもかかわらず、容量が 1346 の圧縮したストリームとなっている。このように、構成図には異常が見られないものの、属性の特性に違反するものを不自然な構成要素に分類する。

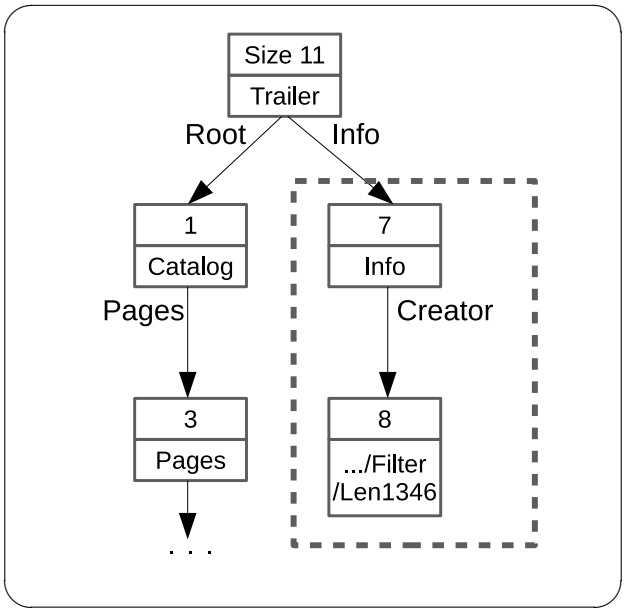

図 4: 不自然な構成要素の例

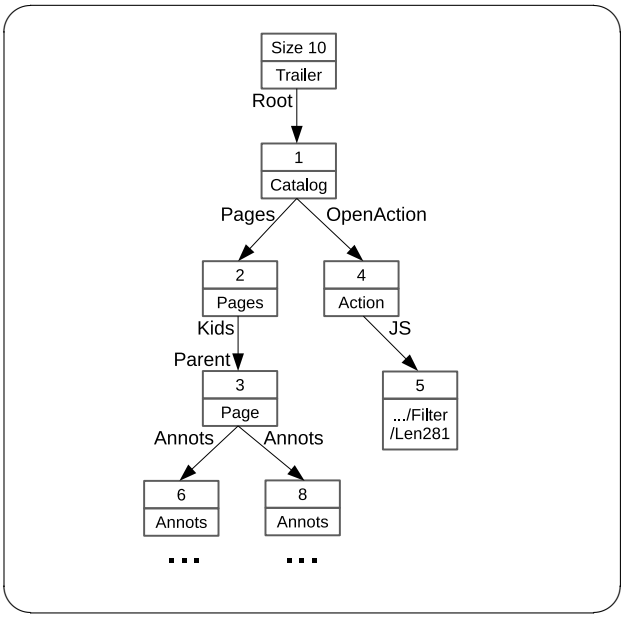

図 5: 最低限の構成要素の例

4.4. 特徴 4：最低限の構成要素

通常の簡易な PDF ファイルは、Trailer から始まって、ファイルの文書内容を記載する Catalog オブジェクトと、ページを構成するオブジェクトの要素から構成される。そして、図 5 の構成図も、その簡易な PDF ファイルと同様に、最低限の要素からしか構成されていないことが確認できる。このように、文書内容を格納する Catalog オブジェクトから、仕様上の定義と一致し、かつ最低限の構成要素からなるファイルも存在する。

5. 考察

5.1. 特徴 1：基本要素の欠落

PDF ファイルの 4 つの基本要素は、仕様上では必要不可欠となっているため、良性 PDF のファイルでも、基本要素のどれかが欠落していれば、閲覧不可能（エラー）となっている。

PDF の仕様では、ファイルが編集されるたびに、追加されたオブジェクトの相互参照テーブルが新規に加わっていくというルールとなっている。したがって、攻撃者が既存のダミーファイルを編集し、不正コードを挿入する過程で、相互参照テーブルが追加されたデータで上書きされるか、削除される可能性として考えられる。

5.2. 特徴 2：オブジェクト参照の矛盾

オブジェクト参照の矛盾の場合も、ダミーファイルが出来上がった後に、不正コードやマルウェアを挿入するといった過程で発生した特徴として考えられる。そのときに、完成した構成要素の参照を無理やり参照先を、埋め込まれたマルウェアに変えることで、こういった参照の矛盾の不具合が生じると考えられる。

5.3. 特徴 3：構成要素の不自然

通常の PDF ファイルは、Info オブジェクトに記載される項目（Title、Author、Creator など）は、文字列の属性である。そして、仕様上では文字列を圧縮しても問題なく処理される。しかし、通常では文書情報を管理する Info オブジェクトと Creator などの関連項目は、圧縮するほどのデータの容量があるのは、不自然であると考える。したがって、こういった圧縮されたマルウェアを、どこかに埋め込もうとすることも、悪性 PDF の一つの特徴として分類することができる。なお、Info オブジェクトとその関連項目は、すべてオプションの項目である。

5.4. 特徴 4：最低限の構成要素

これまで、悪性 PDF ファイルの 3 つの特徴を判明したが、特徴 1 と 2 は PDF ファイルの仕様を完全に理解していないことから、こういった分かりやすい特徴を残してしまったと考えられる。そして、特徴 3 は PDF ファイルの文書情報の特性を考慮していないことから、こういった特徴が表れると考えられる。

もし、PDF ファイルの仕様を完全に理解し、文書情報の特性も考慮したうえで、特徴 3 のような問題は避けることができる。つまり、Info オブジェクトにマルウェアを埋め込むより、通常の Page オブジェクトや Page の下に格納されるオブジェクト（Annots）に埋め込む

方が、通常の PDF ファイルの特徴に見せかけることができる。攻撃者が特徴 3 の不自然な要素をなくし、作成した悪性 PDF ファイルはこの特徴 4 となる。

現段階で、この特徴 4 を悪性 PDF ファイルとして検知することがまだできていない。原因としては、PDF ファイルの構成要素から得られる情報は少ないため、特徴 4 のような少ないオブジェクトから構成されるパターンは、正確に判別することができない。特徴 4 を悪性 PDF として検知するには、オブジェクトごとの属性も考慮し、さらに、それぞれの属性の制約を設けることが考えられる。したがって、以上で述べた要素を意識し、改めて悪性 PDF ファイルの構成要素を再検知する必要がある。

6. おわりに

6.1. まとめ

悪性 PDF ファイルの難読化により、PDF ファイルの中のコンテンツを把握できないため、埋め込まれたマルウェアの本体を正確に検知することができない問題がある。この問題を解消する PDF ファイルの構成要素を検知する手法に着目した。事前に入手した悪性 PDF ファイルの構成要素を分析し、マルウェアが埋め込まれた PDF ファイルの特徴を 4 つに分類した。このように、検知する PDF ファイルがいずれの特徴に当てはまれば、悪性 PDF ファイルとして判別することができる。ただし、現段階では特徴 4 を除く。

特徴 4 のように、構成要素からの情報が少ない場合は、構成要素の関係性のみで、悪性 PDF ファイルの特徴として捉えることが困難であり、正しく判別することができない問題が発生する。したがって、構成要素の関係性を加え、それぞれのオブジェクトの特性も考慮するなど、別のアプローチにも着目すると考えられる。

6.2. 今後の課題

本論文では、分類した 4 つの特徴のうち、特徴 1，2，3 の 3 つは構成的な要素が明らかに良性 PDF と異なるため、悪性 PDF ファイルであることを簡単に検知することができる。しかし、特徴 4 は最低限の構成要素からなる良性 PDF ファイルの特徴とほどんど一致しているため、現段階では、まだ判別することができない。したがって、実際に出回っている悪性 PDF ファイルを完全に検知し、マルウェアの感染から防ぐことができない。

今後の課題としては、改めてこの特徴 4 の構成要素とオブジェクトの特性を考慮し、悪性 PDF ファイルとして検知できるように分析していきたい。また、悪性 PDF を効率的に検知するために、分類した 4 つの特徴に基づいて、悪性 PDF ファイルを自動検知するツールを構築する必要がある。さらに、そのツールの有効性を検討するために、実際のデータセットを用いて検証する。

参考文献

[1] 標的型サイバー攻撃の脅威と対策. Technical report, 独立行政法人情報処理推進機構, 2013.

[2] CHECK POINT SECURITY REPORT 2014. Technical report, Check Point SOFTWARE TECHNOLOGIES LTD., 2014.

[3] 三村守, 大坪雄平, 田中英彦. 悪性文書ファイルに埋め込まれた RAT の検知手法. 情報処理学会論文誌, Vol. 55, No. 2, pp. 1089–1099, 2 2014.

[4] Igino Corona, Davide Maiorca, Davide Ariu, and Giorgio Giacinto. Lux0R: Detection of Malicious PDF-embedded JavaScript code through Discriminant Analysis of API References. *AISec '14 Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*, pp. 47–57, 11 2014.

[5] Charles Smutz and Angelos Stavrou. Malicious PDF Detection using Metadata and Structural Features. *ACSAC '12 Proceedings of the 28th Annual Computer Security Applications Conference*, pp. 239–248, 2012.

[6] 神薗雅紀, 西田雅太, 星澤裕二. 動的解析を利用した PDF マルウェア解析システムの実装と評価. ICSS 情報通信システムセキュリティ, No. 475, pp. 47–52, 3 2011.

[7] 大坪雄平, 三村守, 田中英彦. PDF の構造検査による悪性 PDF ファイルの検知. コンピュータセキュリティシンポジウム, Vol. 2013, No. 4, pp. 649–656, 10 2013.

[8] Davide Maiorca, Igino Corona, and Giorgio Giacinto. Looking at the bag is not enough to find the bomb: an evasion of structural methods for malicious PDF files detection. *ASIA CCS '13 Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pp. 119–130, 5 2013.