Computer Security Symposium 2016
11 - 13 October 2016

# オブジェクト間の参照関係に基づいた悪性PDF分類手法

今 健吾[1]　長瀬 智行[1]

**概要**：PDF (Portable Document Format) ファイルは電子文書のフォーマットとして広く定着しており，現在も多くの場面で使用されている．そのため，攻撃者の格好のターゲットであり，標的型攻撃や Drive-by Download 攻撃では依然として悪性 PDF ファイルが利用され続けている．本稿では，悪性 PDF ファイルに含まれるオブジェクト間の参照関係を辿った *Path* を抽出し，悪性 PDF ファイル特有の *Path* 構造に着目することで，悪性 PDF ファイルがいくつかのパターンに分類できることを示す．また，これらのパターンを利用することによる悪性 PDF の検知への応用や有効性について考察する．

**キーワード**：悪性 PDF，JavaScript，標的型攻撃，Drive-by Download 攻撃

# A Path Tracing Method for Classifying Malicious PDF Files

Kon Kengo[1]　Nagase Tomoyuki[1]

***Abstract:*** PDF (Portable Document Format) files are widely used as a format for electronic documents. A malicious code can be intentionally imbedded in PDF files during targeted and Drive-by Download attacks. In this paper, a path tracing method to define the characteristics of the relationship between objects in a PDF file is proposed. The method is scrutinized a file that may contain malicious codes and classified the types of the patterns. Accordingly, the effectiveness of the detected malicious PDF using these patterns is also considered.

***Keywords:*** Malicious PDF, JavaScript, Targeted Attack, Drive-by Download Attack

## 1. はじめに

近年，特定の組織や個人を対象に情報窃取などを目的に行われる様々なサイバー攻撃が大きな社会問題となっている．とりわけ，攻撃用のファイルを添付して巧妙に作成されたメールを特定の相手へ送信する標的型メール攻撃や，不正に改ざんされた Web サイトを閲覧したクライアント PC にマルウェアをダウンロードさせる Drive-by Download 攻撃がその典型である．これらの攻撃に悪用されるファイルの一つに PDF ファイルがある．PDF ファイルは電子文書のフォーマットとして広く定着しており，Adobe Reader に代表される閲覧ソフトが多くのクライアント PC にインストールされている．そのため，閲覧ソフトの脆弱性を狙った攻撃による影響範囲が大きい．また，PDF のような文書ファイルは実行形式のマルウェアとは異なり，受信者が自ら拡張子などの情報から不審なファイルであるかを判断することが難しいといった問題もある．このように PDF を悪用した攻撃が脅威となっている一方で，市販のウイルス対策ソフトの多くはシグネチャによるパターンマッチに頼っているのが現状である．シグネチャによるパターンマッチを使用した対策の問題点は，Adobe Reader や Adobe Acrobat Reader といった閲覧ソフトのゼロデイ脆弱性を利用した攻撃に対応できないことにある．

また，攻撃に利用するために悪意を持って作成された PDF ファイル (以下，悪性 PDF) では，不正な JavaScript コードをファイル内に埋め込んだものが多く利用されている [1]．悪性 PDF 内に埋め込まれる JavaScript コードは高度に難読化されており，さらには，PDF の機能を利用して，JavaScript コードをファイル上の複数箇所に

[1] 弘前大学大学院理工学研究科
Graduate School of Science and Technology, Hirosaki University

分散して格納させるというような解析妨害テクニックが使用されているものも存在する．例えば，PDF の注釈と呼ばれるページコンテンツとは無関係の，一種のメモ代わりに使用できる領域に JavaScript コードを配置した場合，`this.syncAnnotScan();` で注釈をスキャンした後に，`this.getAnnots(nPage:0);` というコードを実行することによって，ページ番号 0 の注釈に配置していた JavaScript コードを取得することができる．このような手法を使うことで，一見 JavaScript コードとは無関係の領域に JavaScript コードを格納させておくことができる．

現在も Adobe Reader や Adobe Acrobat Reader の脆弱性が多数報告されており，2015 年には 130 件 [2] もの脆弱性が発見されている．このような状況から，悪性 PDF による脅威は現在も続いており，その対策が求められる．そこで本研究では，PDF ファイルに含まれるオブジェクト間の参照関係を辿った *Path* を抽出し，悪性 PDF 特有の *Path* 構造に着目することで，悪性 PDF をいくつかのパターンに分類する手法を提案する．その結果，悪性 PDF に埋め込まれた JavaScript コードの動作条件を可視化することによる解析支援や，悪性 PDF 特有の *Path* 構造を発見することで，悪性 PDF 検知のための一つの特徴として利用することが期待できる．

## 2. 関連研究

悪性 PDF の検知に関する研究として，悪性 PDF に埋め込まれた JavaScript コードに着目した方法と，ファイル構造やメタデータに着目した方法が挙げられる．

JavaScript コードに着目した方法では，悪性 PDF から抽出した JavaScript コードを静的解析，動的解析した結果から得られた特徴を利用した検知を目指している．Laskov らは，PDF 内に含まれる JavaScript コードを静的解析して，そこから得られる情報を特徴とした機械学習による検知手法を提案している [3]．しかし，この手法はキーが `/JS` の辞書オブジェクトを対象にしているため，複数箇所に JavaScript コードを配置する解析妨害テクニックで回避されうる．神薗らの研究では，JavaScript for Acrobat API をエミュレートすることで悪性 PDF 内の難読化された JavaScript コードを解析するシステムを実装している [4]．これらの方法は，攻撃に使用される JavaScript for Acrobat API などの攻撃に直接関わる重要な特徴が得られるため，高精度な検知が可能であるというメリットの反面，JavaScript コードが高度に難読化されている場合や，ファイル上に JavaScript コードが分散配置されるというように，JavaScript コード自体の抽出が容易ではない場合には解析のコストが大きくなりやすいといった問題がある．

また，JavaScript コードの解析を必要としない方法として，ファイル構造やメタデータに着目した方法がある．ファイル構造やメタデータに着目した方法では，JavaScript コードに依らず，静的解析で比較的容易に得られる特徴を使用している．Smutz らは，メタデータと PDF の構成要素を特徴とした機械学習による検知手法を提案している [5]．Srndic らは，PDF オブジェクトの階層構造に着目して Root からの全ての *Path* を抽出し，それを特徴とした機械学習による検知手法 [6] を提案しており，オブジェクト間の参照関係を辿った *Path* が悪性 PDF 検知に有効であることを示している．大坪らは，悪性 PDF 特有のファイル構造の不整合性に着目することで高精度な検知を実現できることを示している [7]．しかし，この手法は実行ファイルが埋め込まれた悪性 PDF のみを対象にしているため，Drive-by Download 攻撃などで使用される，マルウェアを後からダウンロードするタイプの悪性 PDF の検知はできない．

ファイル構造やメタデータに着目した方法では，特徴の抽出は比較的容易である一方で，ファイル構造やメタデータを攻撃者の手によって偽装された場合，検知精度が落ちてしまうという問題がある．

## 3. PDF の基本構造

### 3.1 ファイルレイアウト

PDF (Portable Document Format) ファイルはヘッダ，本体，相互参照テーブル，トレーラといった 4 つのセクションで構成される [8]．PDF ファイルに含まれる多くのコンテンツは本体内にオブジェクトという形で格納されており，本研究において作成した簡易 PDF パーサーも本体から複数のオブジェクトを抽出する処理を行っている．これら 4 つのセクションの説明を以下に示す．また，それぞれのセクションの例を図 1 に示す．

**ヘッダ**

ドキュメントが使用している PDF のバージョン番号を指定するセクションであり，ファイルの先頭に位置する．

**本体**

オブジェクト群で構成されており，それぞれのオブジェクトの先頭にはオブジェクト番号と世代番号，`obj` キーワードが，終端には `endobj` キーワードが記述される．また，オブジェクト間には参照関係があり，例えば，図 1 の `２ ０ R` はオブジェクト 2 を参照していることを意味している．

**相互参照テーブル**

本体内に存在する各オブジェクトへのバイトオフセットを一覧化したセクションである．相互参照テーブルを利用することで，PDF リーダーは全てのオブジェ

クトをメモリにロードせずとも任意のオブジェクトへのランダムアクセスが可能となり，これにより，数十 MB といった巨大な PDF ファイルであっても効率的で高速なアクセスが実現できる．

トレーラ

トレーラ辞書と呼ばれる構造体が含まれており，ファイル中に存在する様々なメタデータの位置が記述されている．例えば，図 1 のトレーラ中に記述されている `/Root 1 0 R` は，オブジェクト間の関係を木構造で表現したときにトップ (Root) に位置するオブジェクトはオブジェクト 1 であることを示している．

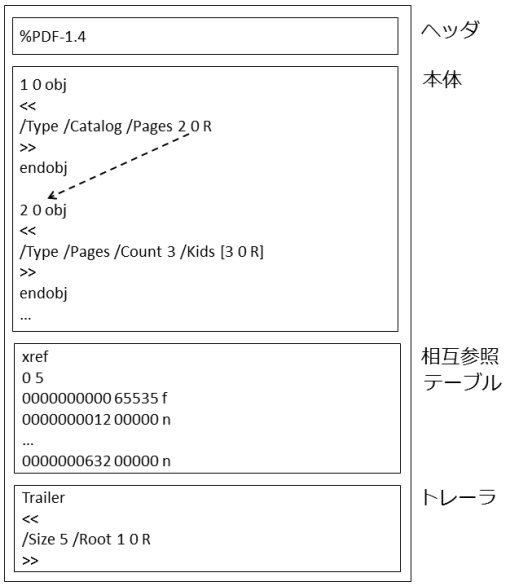


図 1　PDF ファイルの構造

### 3.2　オブジェクト

PDF では数値，文字列，名前，ブーリアン値，`null` の 5 つの基本オブジェクトと配列，辞書といった 2 つの複合オブジェクトがサポートされている．配列は，オブジェクトを順序付きの並びとして表現したものである．辞書は，キーとそのキーに対応した値のペアを順序のない並びとして表現したものである．

### 3.3　間接参照

PDF は複数のオブジェクトで構成されており，それぞれのオブジェクトを一つにまとめるためにオブジェクトから他のオブジェクトへのリンクを作成する仕組みが用意されている．その仕組みのことを間接参照といい，図 1 の `2 0 R` はオブジェクト 2 への間接参照を意味している．

## 4.　*Path* を利用した PDF 分類手法

### 4.1　概要

本稿では，1 章で述べた難読化 JavaScript コードを埋め込んだもののように，巧妙に作成された悪性 PDF の解析や分析を効率的に行うための悪性 PDF の分類手法を提案する．本手法は，PDF のファイル構造に着目した方法であるため，JavaScript コードの解析は必要としない．また，2 章で述べた関連研究では使用されていない新たな特徴点として提案手法により得られたパターンを利用することによって，悪性 PDF の検知に応用することが可能であるかを考察する．

本研究では，以下のような調査を行い，その結果からいくつかのパターンを定義する．調査の概略図を図 2 に示す．

(1) データセットに含まれる全ての PDF から *Path* の抽出を行う．*Path* の抽出には我々が作成した簡易 PDF パーサを使用する．

(2) (1) の結果から，`/JS` と `/JavaScript` が含まれている *Path* のみを抽出する．

(3) (2) で得られた *Path* を dot 言語に変換し，階層構造を Graphviz[9] で可視化する．

(4) (3) の結果から目視でいくつかのパターンを定義する．

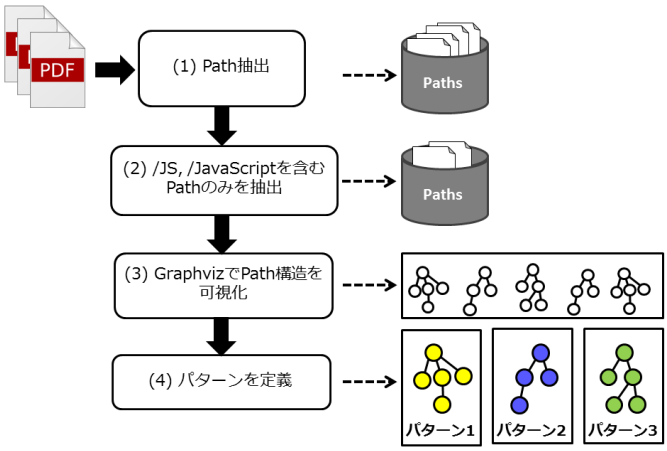


図 2　調査の概略図

### 4.2　*Path* の抽出

本手法ではまず，PDF オブジェクト間の参照関係を得るために，PDF サンプルから *Path* を抽出する必要がある．*Path* の抽出には，我々が作成した簡易 PDF パーサを使用した．

#### 4.2.1　PDF の階層構造

PDF は複数のオブジェクトで構成されており，それぞれのオブジェクトを一つにまとめるために，3 章で述べた間接参照と呼ばれる仕組みが利用されている．基本的には，トレーラ辞書の `/Root` キーから間接参照を辿ることで全てのオブジェクトを参照できる仕組みになっており，このような構造が PDF の階層構造と呼ばれる．PDF の階層構造

は図 3 に示すような木構造で表現でき，本稿では，この木構造を根から葉へ順に辿り辞書オブジェクトのキーを/区切りで連結させたものを *Path* と定義している．例えば，図 3 の PDF ファイルから得られる *Path* は図 4 のようになる．

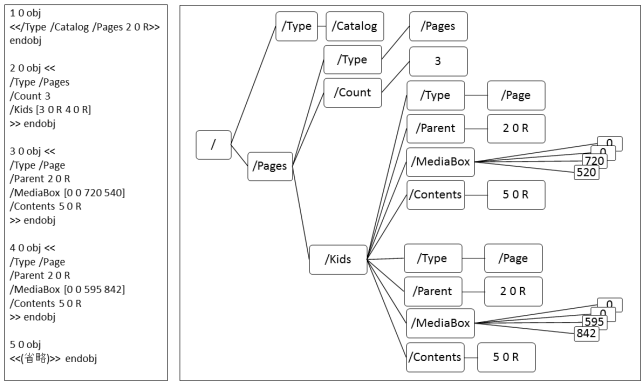


図 3　PDF ファイルとその論理構造

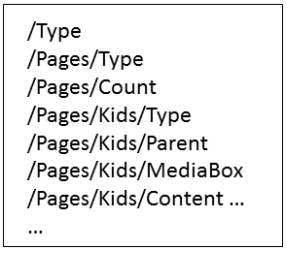


図 4　PDF ファイルから得られる *Path* の例

#### 4.2.2　使用する PDF サンプル

PDF サンプルは，マルウェアダンプサイト contagio[10] で研究用に公開されている検体を使用した．公開されている検体は，良性 PDF が 9000 検体，悪性 PDF が 10980 検体となっている．調査はこの中から JavaScript コードが埋め込まれた悪性 PDF(9808 検体) と良性 PDF(265 検体) を対象とした．

### 4.3　特定キーワードを含む *Path* の抽出

PDF サンプルはそれぞれページ数やオブジェクト数が違うため，得られる *Path* の数も様々である．例えば，オブジェクト数が少ないものは数十〜数百，オブジェクト数が多いものや一つのオブジェクトから複数のオブジェクトへの間接参照がある場合は数十万〜数百万という数になることもある．そこで本手法では，キーワードとして/JS と/JavaScript を含む *Path* のみを対象とすることで，潜在的に危険なコンテンツが含まれる *Path* に絞った調査を行った．/JS キーが参照するオブジェクトには実際の JavaScript コードが含まれているため，/JS を含む *Path* を調査することで JavaScript コードがどのようなオブジェクトを辿って参照されるかといった情報を得ることができる．

### 4.4　*Path* 構造の可視化

続いて，前段階までで得られた *Path* を対象に *Path* 構造を可視化することでいくつかのパターンに分類する．*Path* 構造の可視化には Graphviz[9] を利用した．

### 4.5　パターンの定義

最後に，それぞれの PDF の *Path* 構造を可視化した結果を目視で比較することで，いくつかのパターンを定義し，悪性 PDF サンプルと良性 PDF サンプルがそれぞれのパターンにどの程度属するかを調べる．

## 5.　調査結果

本章では，提案手法により悪性 PDF サンプルと良性 PDF サンプルの *Path* 構造を可視化した結果，分類できた特徴的な 5 つのパターンと，それ以外のパターンとして一つの例を示す．

### 5.1　パターン I

まず 1 つ目のパターンを図 5 に示す．図 5 のパターンは，/OpenAction キーに対応するオブジェクトとして，実際に JavaScript コードが格納される/JS キーが参照されている．/OpenAction は PDF ファイルを開いてすぐに動作させたいアクションを指定するキーであり，このパターンに分類された悪性 PDF は，ユーザによって開かれると同時に攻撃コードが動作するのが分かる．また，/OpenAction からのみ参照されているため，他のイベントは攻撃には直接関係しない部分であり，/OpenAction 以下の/JS のみを解析対象に絞ることができる．

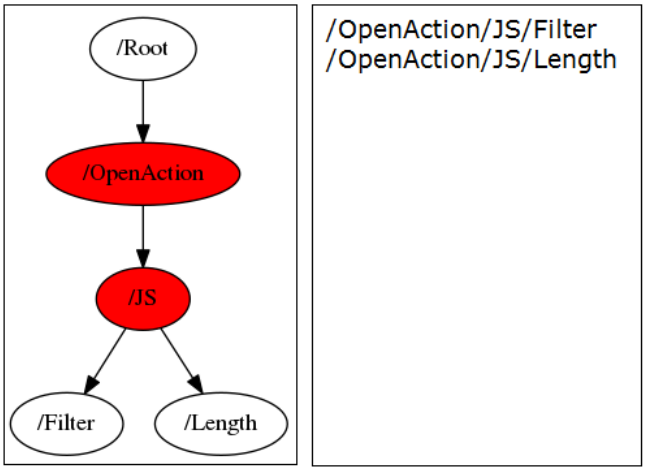


図 5　パターン I

### 5.2 パターン II

次に，2 つ目のパターンを図 6 に示す．図 6 は，/Names/JavaScript/Names の先に/JS キーが存在するパターンであり，これもパターン I と同様に，PDF ファイルが開かれると同時に JavaScript コードが動作する．したがって，このパターンに分類された悪性 PDF もパターン I と同様，ユーザによって開かれると同時に攻撃コードが動作するのが分かる．また，このパターンにおいても，/Names/JavaScript/Names 以下の/JS のみを解析対象に絞ることができる．

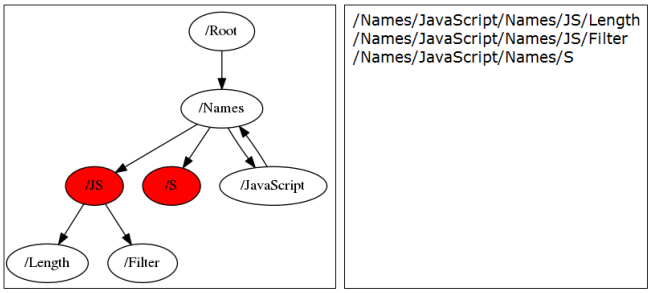


図 6　パターン II

### 5.3 パターン III

次に，3 つ目のパターンを図 7 に示す．図 7 は，図 5 と図 6 を合わせたような形になっているのが分かる．したがって，このパターンにおいても PDF ファイルが開かれると同時に JavaScript コードが動作する．しかし，このパターンの場合，/Names/JavaScript/Names と/OpenAction の二つから参照されているため，解析対象は/Names/JavaScript/Names から参照されている/JS と，/OpenAction から参照されている/JS の二つに絞られることになる．

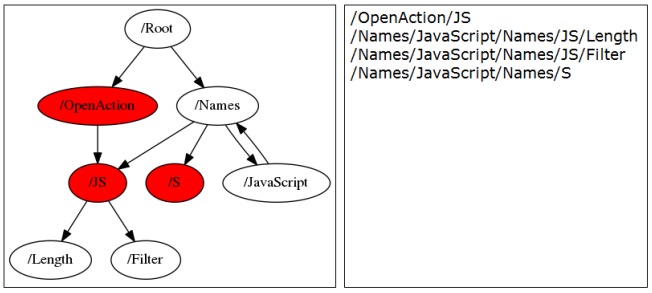


図 7　パターン III

### 5.4 パターン IV

次に，4 つ目のパターンを図 8 に示す．図 8 は，/Pages キーから始まっており，これはページレベルのアクションであることを示している．また，/AA は Additional-Actions を，/O は Open を意味している．すなわちパターン IV はページが開かれると同時に JavaScript コードが動作するパターンであり，上記のパターンとは文書レベルかページレベルかという点で異なる．また，このパターンにおいても，/Pages/Kids/AA/O 以下の/JS のみを解析対象に絞ることができる．

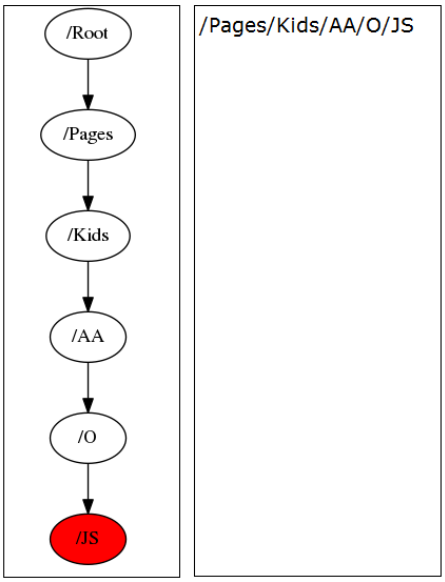


図 8　パターン IV

### 5.5 パターン V

最後の 5 つ目のパターンを図 9 に示す．図 9 は，/StructTreeRoot を含んでいるパターンである．PDF 文書には利便性のために，構造ツリーと呼ばれる論理構造が付与されている場合があり，/StructTreeRoot はこの構造ツリーの Root を表している．構造ツリーの構成上，このような PDF ファイルから得られた *Path* は複雑な構造になっているという特徴があり，これをパターン V と定義する．

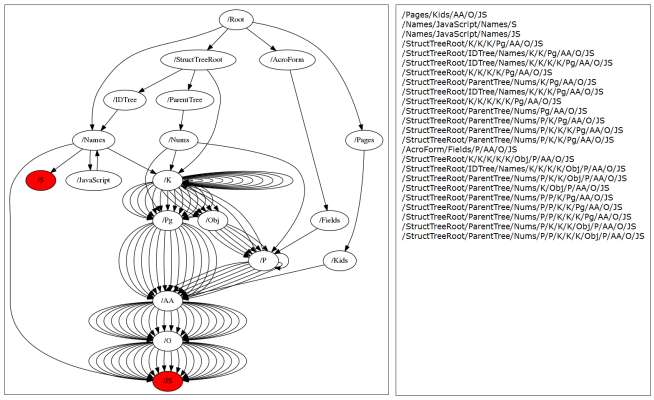


図 9　パターン V

### 5.6 その他のパターン

その他のパターンとして，図 10 に一例を示す．図 10 はパターン Ⅲ に文書アクションをトリガにした JavaScript コード実行を合わせたパターンである．図 10 は，/WS (Will Save) , /DS (Did Save) , /WP (Will Print) , /DP (Did Print) , WC (Will Close) の 5 つの文書アクションが JavaScript コードに対応付けられている．そのため，この悪性 PDF は 7 つの/JS が解析対象となる．

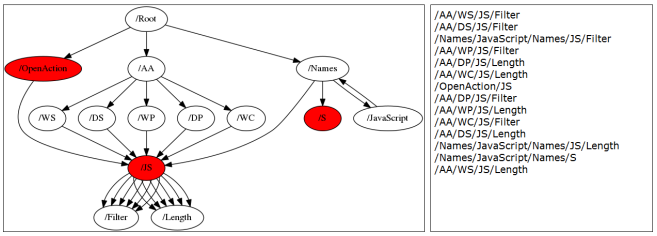


図 10　その他のパターン

## 6. 考察

### 6.1 PDF サンプルのパターン分布

提案手法を適用した結果，悪性 PDF サンプルから得られた *Path* 構造のパターン分布を表 1 に，良性 PDF サンプルから得られた *Path* 構造のパターン分布を表 2 に示す．

表 1 より，悪性 PDF の 72.12%はパターン I に分類されているのが分かる．これは，/OpenAction を使用した方法が最も簡単に JavaScript コードを PDF に埋め込むことができることに起因していると考えられる．また，パターン Ⅱ も 21.76%と高い割合で分類されている．悪性 PDF の場合，パターン I〜Ⅲ で 99%以上を占めていることから，PDF ファイルが開かれると同時に JavaScript コードが動作するタイプの悪性 PDF が 99%以上であることが分かる．これは，JavaScript コード実行のトリガがボタンクリックや，印刷開始時といったアクションに限定させると，攻撃が行われない可能性があり，感染動作を確実に行わせたい攻撃者にとってのデメリットが大きいためだと考えられる．また，PDF ファイルが開かれると同時に早急に JavaScript コードを実行させることから，感染動作をユーザに気づかれずに行わせたいという攻撃者の意図があると言える．

表 1　悪性 PDF のパターン分布

| | 検体数 | 割合 |
|---|---|---|
| パターン I | 7074 | 72.12% |
| パターン Ⅱ | 2135 | 21.76% |
| パターン Ⅲ | 552 | 5.63% |
| パターン Ⅳ | 8 | 0.08% |
| パターン V | 7 | 0.07% |
| その他 | 32 | 0.33% |
| 合計 | 9808 | 100.00% |

表 2 より，良性 PDF の 39.62%がパターン V に分類されているのが分かる．これは，良性 PDF の多くが構造ツリーを持っているからであると考えられる．構造ツリーによって PDF 文書に章や段落といった構造を持たせることができ，文書に含まれる文字のコピーを行いやすくするなどのメリットがある．そのため良性 PDF の多くは，利便性の向上のため構造ツリーを持っている場合が多く，JavaScript コードが使用される PDF 文書でも例外ではないため，このような結果になったと考えられる．また，良性 PDF には 5 つのパターンには分類されないその他のパターンが悪性 PDF に比べて多いのが分かる．これは悪性 PDF が攻撃目的で作成されるため，必要な機能や表示コンテンツが少なく同じような *Path* 構造が多くなるのに対して，良性 PDF は多様なコンテンツが文書内に使用されており，/JS も様々なオブジェクトから参照されているからだと考えられる．

表 2　良性 PDF のパターン分布

| | 検体数 | 割合 |
|---|---|---|
| パターン I | 0 | 0.00% |
| パターン Ⅱ | 70 | 26.41% |
| パターン Ⅲ | 0 | 0.00% |
| パターン Ⅳ | 17 | 6.42% |
| パターン V | 105 | 39.62% |
| その他 | 73 | 27.55% |
| 合計 | 265 | 100.00% |

### 6.2 パターンの解析支援としての利用

5 章の結果より，本手法を用いて PDF ファイルを潜在的に危険なコンテンツが含まれる *Path* 構造に基づいて分類することで，PDF ファイルに埋め込まれた JavaScript コードの動作条件を可視化できた．これにより，解析支援としての利用が期待できる．

しかし，本手法は 1 章で述べたような，JavaScript コードをファイル上の複数箇所に分散して格納させる解析妨害の対策にはならない．この解析妨害の対策をするためには，/JS や/JavaScript だけでなく，/Annots (注釈) などの JavaScript コードから読み込むことができる部分も対象に調査する必要がある．

### 6.3 パターンの悪性 PDF 検知への応用

表 1，表 2 より，悪性 PDF はパターン I が多くを占めているのに対して，良性 PDF はパターン I に分類されたものが一つもなかった．また，良性 PDF ではパターン V が多くを占めていたものの，悪性 PDF ではその数が極端に少なかった．そのため，パターン I とパターン V を特徴に使用することで悪性 PDF の検知がある程度可能であると言える．しかし，パターン V は攻撃者が意図して偽装

することが可能であるため，確実な方法とは言えない．したがって，このパターンのみを特徴とした検知は難しいため，複数の特徴の一つとして位置づける必要がある．

### 6.4 *Path* 構造の悪性 PDF 検知への応用

別の視点として一つの PDF ファイルから得られた *Path* 構造の完全一致を検査することによる悪性 PDF の検知手法も考えられる．本手法では，PDF から *Path* の抽出を行った後に/JS と/JavaScript を含む *Path* のみを抽出した．この手法では，PDF 全般に見られる汎用的な特徴が得られるため，この特徴単体で検知に利用するのは難しくなる．

そこで，*Path* 構造の完全一致を検査することによる悪性 PDF の検知手法の可能性について考えてみる．良性 PDF の場合は，オブジェクト数が多く *Path* 構造が複雑になるため，*Path* 構造が完全一致する可能性は限りなく低い．それに対して悪性 PDF の多くはファイルサイズやオブジェクト数が少ないため *Path* 構造がシンプルでがあり，特定キーワードを含む *Path* の抽出を行わなくとも *Path* 構造が完全一致するケースがいくつか存在した．例えば，図 11 は一つの悪性 PDF から得られた完全な *Path* であり，シンプルな構造となっている．この *Path* 構造と完全一致する悪性 PDF は多数存在しており，これらは同じツールによって生成されたものだと考えられる．したがって，図 11 のようにシンプルな *Path* 構造との完全一致を検査することで，同じツールによって生成された悪性 PDF を検知できる可能性がある．

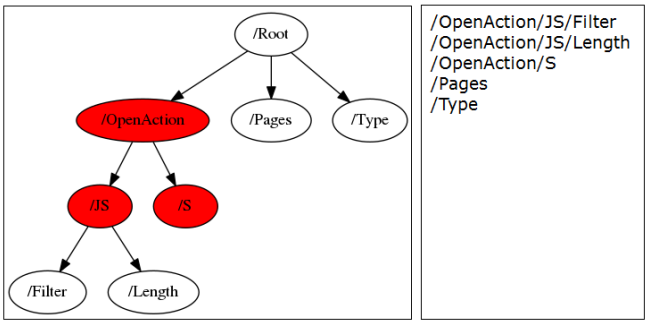


図 11　悪性 PDF の *Path* 例

## 7. まとめと今後の課題

本稿では悪性 PDF と良性 PDF の *Path* 構造を調査し，いくつかのパターンに分類できることを示した．また，悪性 PDF と良性 PDF それぞれのパターン分布を調べることで，どのような特徴が得られるかを調べ，悪性 PDF の検知へ応用できる可能性について考察した．

今後は，今回得られたパターンをはじめ，PDF の *Path* 構造を利用して悪性 PDF の検知を実現する手法を検討する．

### 参考文献


[1] K. Chang, Y. Lin, "Advanced Persistent Threat: Malicious Code Hidden in PDF Documents", 2014.

[2] "Adobe Acrobat Reader : Security Vulnerabilities Published In 2015", http://www.cvedetails.com/vulnerability-list/vendor_id-53/product_id-497/year-2015/Adobe-Acrobat-Reader.html

[3] P. Laskov, N. Srndic, "Static Detection of Malicious JavaScript-Bearing PDF Documents", ACM, in Annual Computer Security Applications Conference, pp. 373-382, 2011.

[4] 神薗雅紀，西田雅太，星澤裕二，"動的解析を利用した難読化 JavaScript コード解析システムの実装と評価", MWS2010, 2010.

[5] C. Smutz, A. Stavrou, "Malicious PDF Detection using Metadata and Structural Features", ACM, in Annual Computer Security Applications Conference, pp. 239-248, 2012.

[6] N. Srndic, P. Laskov, "Detection of Malicious PDF Files Based on Hierarchical Document Structure", Proceedings of the 20th Annual Network & Distributed System Security Symposium, 2013.

[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", Computer Security Symposium, 2013.

[8] John Whitington『PDF 構造解説』(村上雅章訳) オライリー・ジャパン, 2012.

[9] Graphviz - Graph Visualization Software, http://www.graphviz.org/

[10] P.Mila, 16,800 clean and 11,960 malicious files for signature testing and research., http://contagiodump.blogspot.jp/2013/03/16800-clean-and-11960-malicious-files.html.