PDF の構造検査による悪性 PDF ファイルの検知

大坪雄平 †　　三村守 ‡　　田中英彦 ‡

† 内閣官房情報セキュリティセンター
100-0014 東京都千代田区永田町 2-4-12
yuhei.otsubo@cas.go.jp

‡ 情報セキュリティ大学院大学
221-0835 神奈川県横浜市神奈川区鶴屋町 2-14-1

あらまし　実行ファイルを埋め込んだ PDF（Portable Document Fomat）ファイルを用いた標的型攻撃が発生している．このような悪性 PDF ファイルを検知する手法としては，PDF ファイルの中の JavaScript 等の不正なコードを検知する手法が主流である．これに対し，われわれは実行ファイルが埋め込まれた PDF ファイルには構文解釈ができない部分，表示内容と関係しない部分が含まれる等の特徴が表れることに着目する．本論文では，PDF ファイルにこれらの特徴がないか検査することにより，悪性 PDF ファイルを検知する手法を提案する．実行ファイルが埋め込まれた悪性 PDF ファイル 164 個に対し実験した結果，99.4%を検知することができた．

Methods to Detect Malicious PDF File using PDF Structure Inspection

Yuhei Otsubo†　　Mamoru Mimura‡　　Hidehiko Tanaka‡

†National Information Security Center
2-4-12 Nagata-cho, Chiyoda-ward, Tokyo 100-0014, JAPAN
yuhei.otsubo@cas.go.jp
‡Institute of Information Security
2-14-1 Tsuruya-cho, Kanagawa-ward, Yokohama 221-0835, JAPAN

Abstract　Targeted attacks using PDF (Portable Document Format) files that contain executable files are popular. Finding malicious code such as JavaScript from a PDF file is the most popular method to detect malicious PDF files. On the other hand, we focus on features of malicious PDF files that contain executable files. The features are , for example, the malicious PDF files contain parts that can not be parsed, or the malicious PDF files contain data that is not related to display contents of the PDF files. In this paper, we propose investigating these features to detect malicious PDF files. The experimental result using 164 PDF files that contain executable files shows the effectiveness of the methods. The methods could detect 99.4% of the malicious PDF files in the experiment.

1　はじめに

近年では，特定の組織や個人を狙って情報窃取等を行う標的型攻撃が顕在化している．標的型攻撃の防御策として最新のパターンファイルを適用したウイルス対策ソフトを導入したとしても，標的型攻撃に用いられるマルウェアを検知できないことがほとんどである．この原因として，攻撃者は標的型攻撃に用いるマルウェアが最新のウイルス対策ソフトで検知できないこ

とを確認しているということが考えられる．さらに，攻撃を秘匿するため，マルウェアの本体である実行ファイルが埋め込まれた悪性文書ファイルが送付されることもある．

標的型攻撃に用いられる実行ファイルが埋め込まれた悪性文書ファイルの典型的な動作を以下に示す．悪性文書ファイルを開くと，閲覧ソフトの脆弱性を攻撃する exploit と呼ばれる部分が動作し，shellcode（侵入した端末を制御できるようにするためのコード）が実行される．shellcode は文書ファイルに埋め込まれた実行ファイルやダミー表示用の文書ファイルを取り出し，実行ファイルを実行したりダミー表示用の文書ファイルを表示する．これによって悪性文書ファイルを閲覧した端末はマルウェアに感染する．一方，文書ファイルに埋め込まれた実行ファイルやダミー表示用の文書ファイルはウイルス対策ソフト等の検知を回避するため様々な方式でエンコード（符号化）されているほか，ダミー表示用の文書ファイルの表示内容は通常の文書ファイルと変わらないため，一般に，悪性文書ファイルの受信者には実行ファイルが埋め込まれた悪性文書ファイルと通常の文書ファイルとを区別することは困難である．

われわれが実行ファイルが埋め込まれた悪性 PDF（Portable Document Format）ファイルを分析したところ，多くの悪性 PDF ファイルで通常の PDF ファイルと構造に違いがあることが明らかになった．よって，PDF ファイルの構造を検査し，悪性 PDF ファイルの特徴を検知すれば，悪性 PDF ファイルの検知ができるものと考えられる．そこで本論文の目的を，PDF ファイルが悪性 PDF ファイルか否かを効率的に検知することとする．

2　関連研究

本論文では，PDF ファイルを対象に悪性 PDF ファイルであるか否かを検査する手法を提案する．以下，悪性 PDF ファイルの検知に関連する先行研究について述べる．

文献 [1] では，潜在的に危険なアクションを伴うフィルタを対象に機械学習を適用することで，不審な PDF ファイルを高速に検知する手法が提案されている．文献 [2] では，PDF ファイルの中に含まれる JavaScript を対象に機械学習を適用することで，不審な PDF ファイルを高速に検知する手法が提案されている．文献 [3] では，PDF のドキュメント階層構造を対象に機械学習を適用することで，不審な PDF ファイルを高速に検知する手法が提案されている．これらの手法では，教師あり学習モデルを用いているため，学習するためのサンプルを集める必要があるだけでなく，その精度は学習のサンプルに依存する．本論文では，学習を必要としないため，そのためのサンプルは不要である．

文献 [4] では，様々な形式の悪性文書ファイルに埋め込まれた実行ファイルを自動的に抽出するツールが提案されている．この手法では，実行ファイルを埋め込む際に使用される様々なエンコード方式を自動的に解読し，実行ファイルを抽出することができる．しかしながら，新たなエンコード方式が現れるたびに検知手法を検討しなければならないという課題がある．本論文では，PDF ファイルの構造のみを検査しており，PDF ファイルに埋め込まれたデータの分析は行わない．

3　PDF の基本構造

3.1　ファイル構造

PDF のファイル構造はコメント，本体，相互参照テーブルおよびトレーラの 4 つのセクションに分類される．単純な PDF ファイルの例を図 1 に示す．

コメントは，“%” キーワードで始まる 1 行のセクションである．ここに記述された情報はコメントとして扱われる．

本体は，ページコンテンツやグラフィックスコンテンツ等，多くの補助的な情報がオブジェクト一式としてエンコードされているセクションである．各オブジェクトにはオブジェクト番号が割り振られている．

相互参照テーブルは，本体中の各オブジェクトの位置を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オフセットおよびオブジェ

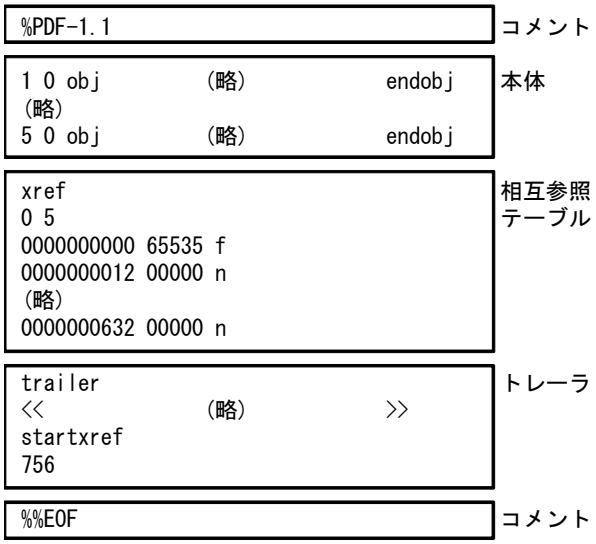

図 1: PDF ファイルの例

クトが閲覧ソフトの表示に使用されるものか否かが記述されている．

トレーラは，トレーラ辞書というオブジェクトが格納されているセクションである．この中には PDF ファイル中に格納された様々なメタデータの位置が記述されている．

3.2 ドキュメント構造

PDF のドキュメント構造はオブジェクトの階層構造となっている．ドキュメントカタログと名付けられたオブジェクトがドキュメント構造の頂点に位置するルートオブジェクトであり，その他すべてのオブジェクトはここからの間接参照を通じてアクセスされるようになっている．なお，ドキュメントカタログの位置はトレーラ辞書に格納されている．

3.3 オブジェクト型

PDF では数値，文字列，名前（後述する辞書のキーなどに使われる），ブーリアン値および null の 5 つの基本的なオブジェクト型がサポートされている．また，配列，辞書および Stream という 3 つの複合オブジェクト型もサポートされている．配列は，他のオブジェクトを順序を

表 1: よく使われるフィルタ

ASCII85 Decode	“!” から “Z” までの印字可能文字を使用して表現した文字列をバイナリデータに変換するフィルタ
ASCIIHex Decode	2 桁の 16 進数の文字列をバイナリデータに変換するフィルタ
DCT Decode	JPEG による不可逆圧縮されたデータを展開するフィルタ
Flate Decode	オープンソースの zlib ライブラリで用いられている Flate 圧縮されたデータを展開するフィルタ
JBIG2 Decode	JBIG2 による圧縮されたデータを展開するフィルタ

付けて複数格納できるもので，辞書は，名前とそれに関連付けられたオブジェクトをペアにしたものを複数格納できるものである．Stream はバイナリデータを格納するために用いられるものであり，圧縮等のエンコードをすることができる．Stream はバイナリデータとともにデータの長さや，デコードに使用するフィルタの種類といった各種の属性を格納した辞書とセットにしたものである．Stream に実行ファイルやダミー表示用の文書ファイルが埋め込まれたときに，よく宣言されているフィルタを表 1 に示す．さらに，オブジェクト間を関連付ける間接参照（あるオブジェクトから他のオブジェクトへのリンク）というオブジェクト型もサポートされている．PDF のオブジェクトはこれら 9 種類のオブジェクト型に分類される．

また，PDF1.5 以降では，多くのオブジェクトを単一の ObjStm（オブジェクト Stream）という Stream 内に格納し，その Stream を圧縮することで PDF ファイルをさらにコンパクトなものにするという仕組みも導入されている．

3.4 ドキュメントの暗号化

PDF1.1 から，ドキュメントの暗号化機能がサポートされている．暗号化されている PDF ファイルには暗号化辞書というオブジェクトが格納されており，これはトレーラ辞書から間接参照されている．暗号化辞書には暗号化の方式など

復号に必要な情報が格納されている．暗号化は基本的に，Stream と文字列に適用され，数値やその他のオブジェクト型には適用されず，ファイル全体を暗号化することはない．したがって，復号しなくてもドキュメント構造にアクセスすることは可能である．ただし，前述した ObjStm に格納されているオブジェクトは暗号化の対象となるため，ObjStm がある暗号化 PDF ファイルの場合は復号化しなければ，ドキュメント構造にアクセスすることはできない．

4　悪性 PDF の構造

exploit の多くは閲覧ソフトの脆弱性を利用しており，閲覧ソフトが通常読み込む部分に埋め込まれている．一方，実行ファイルやダミー表示用の文書ファイルを閲覧ソフトが通常読み込む部分に埋め込むと，閲覧ソフトが誤動作したり，表示される内容が文字化けする可能性がある．したがって，実行ファイルやダミー表示用の文書ファイルは閲覧ソフトが通常読み込まない部分に埋め込まれることが多い．その結果，PDF の構造に通常の PDF ファイルとは異なる特徴が表れる．

われわれが実行ファイルが埋め込まれた悪性 PDF ファイルの構造を分析し，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：分類できないセクション

PDF ファイル内のデータはすべてコメント，本体，相互参照テーブルおよびトレーラという 4 種類のセクションに分類される．

一方，実行ファイルが埋め込まれた PDF ファイルの中にはファイル構造を無視して実行ファイルを埋め込んだため，4 種類のセクションに分類できない部分が存在するものがあった．

4.2　特徴 2：参照されないオブジェクト

一般的な PDF ファイルでは，ドキュメントカタログからの間接参照を通じて，相互参照テーブルで未使用とされるオブジェクトや null オブジェクトを除いたすべてのオブジェクトに間接参照を通じてアクセスできるようになっている．

一方，実行ファイルが埋め込まれた PDF ファイルの中にはドキュメント構造を無視してオブジェクトの中に実行ファイルを埋め込んだため，どのオブジェクトからも参照されていないオブジェクトが存在するものがあった．

4.3　特徴 3：偽装された Stream

4.3.1　フィルタ偽装

FlateDecode などのフィルタを使用している Stream はエントロピー（情報のランダムさ．小さいほど規則性があり，大きいほどランダムに近い状態であることを表す.）が大きいという特徴がある．同様に実行ファイルが埋め込まれた Stream もエントロピーが大きいという特徴がある．そのため，FlateDecode などのフィルタを実行ファイルが埋め込まれた Stream に使用しているように偽装しているものがあった．この場合，Stream の中身とデコードに使用するフィルタが対応していないため，Stream のデコード処理は失敗する．仮に，この部分が閲覧ソフトに読み込まれると，誤動作したり表示内容が文字化けする可能性が高い．それを防ぐため，フィルタ偽装をして実行ファイルを埋め込んだ Stream は，どのオブジェクトからも参照されないようにされることが多い．したがって，特徴 2 の特徴を合わせて持つことが多い．

4.3.2　Stream の末端に追加

FlateDecode，DCTDecode および JBIG2Decode というフィルタでデコードするデータの末端には，データの終端を示す情報が記録されている．そのため，当該フィルタでデコードする Stream の末端に別のデータを追加しても，追加したデータがデコードに使用されない以外は問題なくデコードができる．この特性を利用して Stream の末端に実行ファイルを追記したものがあった．

5　試験プログラムの実装

これまでに示した 3 つの特徴を検知するプログラムを，オープンソースのプログラミング言語である Python を用いて実装した．

5.1　動作の概要

実装したプログラムの概要を図 2 に示す．試験プログラムは PDF ファイルを引数として受け取り，悪性 PDF ファイル特有の特徴を検知するコマンドラインプログラムである．まず，PDF ファイルを入力として受け付け，特徴 1 に該当するか否かを判定する．その後，PDF ファイルが暗号化されているか否かを判定する．PDF ファイルに使用されている暗号化方式が 40bitRC4，128bitRC4，128bitAES または 256bitAES の場合，空白のパスワード，トレーラ辞書の情報および暗号化辞書の情報を元に暗号鍵を生成する．生成した暗号鍵が正しい暗号鍵であるか否かを暗号化辞書の情報を用いて確認し，正しい暗号鍵であった場合には PDF ファイルを復号化する．平文の PDF ファイルの場合，または暗号化 PDF ファイルであっても暗号鍵の生成に成功した場合は，特徴 2 および特徴 3 に該当するか否かを独立して判定する．復号に失敗した場合，PDF ファイルの中に ObjStm がなければすべてのドキュメント構造にアクセスすることができるため，特徴 2 に該当するか否かを判定する．ObjStm がある場合は，特徴 2 および特徴 3 の判定は実施しない．すべての判定終了後，いずれかの特徴に合致すれば悪性 PDF ファイル検知とした．

5.2　特徴 1 の判定

PDF ファイルを 1 文字ずつ読み込み，4 種類のどのセクションに該当するか分類を行う．具体的には，“%” で始まる行はコメント，“[数字] [数字] obj” と “endobj” に囲まれた部分は本文，“xref” または “start xref” から始まる部分は相互参照テーブル，“trailer” から始まる部分はトレーラと分類した．その結果，どのセクションにも分類できなかった場合に特徴 1 の検知とした．

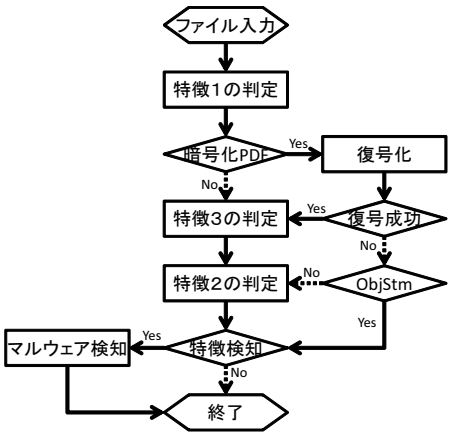

図 2: 試験プログラムの動作

5.3　特徴 2 の判定

PDF ファイルの中にあるすべてのオブジェクトを読み込む．その後，すべての間接参照のリンク先を一覧にし，各オブジェクトがどのオブジェクトからリンクされているかを調べる．どのオブジェクトからもリンクされていないオブジェクトがあった場合に特徴 2 の検知とした．一方，実行ファイルのヘッダは MS-DOS 用ヘッダ，MS-DOS 用スタブプログラムおよび NULL 領域で 256Byte，NT ヘッダおよび NULL 領域で 256Byte の合計 512 バイトの領域を使用する．したがって，物理的に実行ファイルを埋め込むことができない 512Byte 未満の大きさのオブジェクトは特徴 2 の検知対象から除いた．

5.4　特徴 3 の判定

PDF ファイルの中にあるすべての Stream を読み込む．FlateDecode，ASCIIHexDecode または ASCII85Decode が Stream のデコードに使用するフィルタに指定されている場合は，デコードを試みる．この時，Stream に格納されているデータが各種フィルタの形式に沿っていないと，デコードに失敗する．この場合を特徴 3 の検知とした．

表 2: 検体の概要

悪性 PDF ファイル		通常の PDF ファイル	
検体数	平均容量 (KB)	検体数	平均容量 (KB)
164	351.2	9,109	101.7

また，FlateDecode，DCTDecode または JBIG2Decode が Stream のデコードに使用するフィルタに指定されている場合は，データの終端を示すマーカーの位置と Stream の末端の位置を比較することにより，デコードに使用していないデータの有無について調べる．デコードに使用していないデータがあった場合についても特徴 3 の検知とした．

6 実験

6.1 実験内容

試験プログラムの性能を評価するため，悪性 PDF ファイルと通常の PDF ファイルを入力して結果を分析する．実験の対象となる PDF ファイルの概要を表 2 に示す．表 2 の左側の検体は，2009 年から 2012 年までに複数の組織において採取した固有のハッシュ値を持つ PDF ファイルで，分析により実行ファイルが埋め込まれていることをあらかじめ確認しているものである．これらの検体を試験プログラムに入力し，検知の成功率および平均実行時間を求める．また，試験プログラムの検知率と，採取した当時の最新パターンファイルを適用した大手ベンダのウイルス対策ソフトの検知率を比較する．

表 2 の右側の検体はマルウェアダンプサイト contagio でマルウェアではない（clean）とされ，研究用に公開された検体 [5] である．マルウェアではないとされた検体で悪性 PDF ファイルの特徴を検知した場合を誤検知とする．これらのマルウェアではないとされた検体を試験プログラムに入力し，誤検知率を確認する．

実験を実施する環境は表 3 に示すとおりであり，実験はすべて仮想マシン上で行った．

表 3: 実験環境

CPU	Core i5-3450 3.1GHz
Memory	8.0GB
OS	Windows 7 SP1
Memory(VM)	2.0GB
OS(VM)	Windows XP SP3
Interpreter(VM)	Python 2.7.3

表 4: 試験プログラムの検知率等

	検知数	検知率
特徴 1	81	49.4%
特徴 2	72	43.9%
特徴 3	104	63.4%
全体	163	99.4%

6.2 実験結果

検体の検知率および特徴ごとの検知状況を表 4 に示す．この表における検知率は 検知数/164 である．検知の成功率は全体で 99.4%であった．また，平均実行時間は約 0.69s であり，最も実行時間が長いもので 5.63s であった．

次に，試験プログラムの検知率と，大手ベンダのウイルス対策ソフトの検知率との比較結果を表 5 に示す．実験に使用した検体に対しては，採取した当時の最新のパターンファイルを適用した大手ベンダのウイルス対策ソフトでも 3.0%から 19.5%の低い確率でしかマルウェアを検知することができなかった．しかも，ウイルス対策ソフトで検知できるマルウェアの種類には重複があったため，3 種類のウイルス対策ソフトを組み合わせた場合（T，S，M 社 AV）でも，検知率は 23.8%であった．

マルウェアではないとされた検体 9,109 個のうち，試験プログラムは 19 個を悪性 PDF ファイルとして誤検知し，誤検知率は 0.2%であった．

7 考察

7.1 検知に失敗した原因

試験プログラムが検知に失敗した検体は 1 個であり，それを分析した結果，失敗の原因は試

表 5: ウイルス対策ソフトとの検知率の比較

	検知数	検知率
試験プログラム	163	99.4%
T 社 AV	32	19.5%
S 社 AV	16	9.8%
M 社 AV	5	3.0%
T，S，M 社 AV	39	23.8%

験プログラムが未対応の暗号鍵生成技術が使用されたためであった．検知に失敗した検体は Public-Key Security Handler（PDF 32000-1: 2008 7.6.4 を参照）を使い暗号鍵を生成し，この鍵を用い AES により暗号化されていた．この検体には ObjStm が含まれており，ObjStm に格納されているオブジェクトを復号することができなかったことから，特徴 2 および特徴 3 の判定は実施していなかった．試験プログラムがこの方式により暗号化された PDF ファイルの復号にも対応すれば，特徴 2 および特徴 3 の判定を実施することができる．

7.2 誤検知の原因

試験プログラムが誤検知した検体を分析した結果，誤検知の原因は以下の 3 点であった．

- 間接参照されない通常のオブジェクト
- ファイルの一部が破損しているもの
- ファイルの途中に不要なデータが付加されているもの

まず最初に誤検知の原因としてあげられるのは，間接参照されない通常のオブジェクトである．今回誤検知した検体 19 個のうち 12 個は特徴 2 のみに合致していた．どのオブジェクトからも間接参照でリンクされていないオブジェクトを調べたところ，ページの背景等を設定する管理情報に類似するデータであった．同種の管理情報でも間接参照でリンクされているものとされていないものがあり，間接参照されない原因については特定できなかった．当該オブジェクトのサイズは 4KByte 未満であり，悪性 PDF ファイルに埋め込まれていた実行ファイルの大きさの 10 分の 1 未満であることから，ある一定の大きさ未満のオブジェクトは検知の対象から外すことで当該誤検知を回避することは可能と考えられる．しかしながら，フィルタリングした大きさより小さなサイズのオブジェクトを用いた悪性 PDF ファイルがあった場合は，検知することができなくなってしまう．

次の原因としてあげられるのは，ファイルの一部が破損しているものである．誤検知した検体はファイルの末端に不要なデータが付加されているもの（特徴 1），ファイルが途中で切れているもの（特徴 1，特徴 2 および特徴 3）および Flate 圧縮されているデータが壊れているもの（特徴 3）であった．このような PDF ファイルは，通信回線の状況によっては発生しうるものであり，本論文の提案手法では誤検知してしまう．しかしながら，ファイルの末尾に不要なデータが付加されている PDF ファイルは，PDF の仕様に準拠した PDF 生成ソフトであれば作成することはないため，異常な PDF ファイルとして検知するという運用も考えられる．また，ファイルが途中で切れているものや Flate 圧縮されているデータが壊れているものは，閲覧ソフトで正しく内容を表示することができないため，一般的な PDF ファイルとして使用されることはほぼないと考えて良いだろう．

最後の原因としてあげられるのは，ファイルの途中に不要なデータが付加されているもの（特徴 1）である．誤検知した検体は，どのオブジェクトにも対応しない “`endstream endobj`” という文字列が挿入されているものであった．これは，明らかに PDF の構造に沿わない記述である．その他の構造に異常は見られなかったため，この構造は PDF ファイルを生成したソフトの仕様によるものであると考えられる．閲覧ソフトのエラー処理機能により問題が顕在化していないが，すべての PDF 生成ソフトが PDF の仕様に完璧に準拠しているわけではない．したがって，一部の PDF 生成ソフトで作成した PDF ファイルは，本論文の提案手法で常に誤検知する可能性がある．

7.3 試験プログラムの効果

試験プログラムは，検査処理に要する時間の平均値はわずか 0.69s で，99.4%という高い確率で悪性 PDF ファイルを検知することに成功した．さらに，誤検知率は 0.2%という低い確率であった．試験プログラムは高速に検査することが可能であることから，試験プログラムを組織内のメールサーバ等で自動実行させれば，組織内に到達するメールの簡易チェックを実施することが可能である．PDF ファイルがパスワードで暗号化されたものであった場合，exploit や shellcode を検索することは通常できない．しかしながら，試験プログラムは，特徴 1 の判定ができ，場合によっては特徴 2 の判定もできる．

試験プログラムはパターンファイルを用いず，悪性 PDF ファイルに埋め込まれていた exploit やマルウェアのエンコード方式を解析することなく，高い確率で悪性 PDF ファイルを検知することに成功した．文書ファイルの仕様は，攻撃者の意志で変更することが困難であり，その結果，悪性 PDF ファイルの構造はマルウェアやエンコード方式と比較して時間に対する変化が少なくなる．試験プログラムはその悪性 PDF ファイルの構造を検査対象としており，今後プログラムを更新しなくても高い検知率を維持することが可能であると考えられる．

一方，本論文の提案手法は，原理的に exploit や shellcode 部分は検知することはほぼできない．したがって，exploit や shellcode に連結する形でマルウェアが埋め込まれているものや exploit や shellcode のみが埋め込まれてるもの，例えば実行ファイルを外部のサーバ等からダウンロードするようなものは検知することができない．これらについては本論文の提案手法以外の手法で検知する必要がある．

8 おわりに

本論文では，実行ファイルが埋め込まれた PDF ファイルの構造を分析し，実行ファイルが埋め込まれた PDF ファイルの特徴を 3 つ明らかにした．さらに，悪性 PDF ファイルの検知手法として，PDF ファイルの構造検査により当該 3 つの特徴を検知することを提案し，提案手法の有効性を検証する実験を行った結果，平均実行時間 0.69s で 99.4%の悪性 PDF ファイルを検知することができた．

今後の課題としては，未対応のフィルタを利用した悪性 PDF ファイルの検知があげられる．試験プログラムは 5 種類のフィルタでエンコードされた Stream を検査することができる．これら以外の未対応のフィルタを利用した悪性 PDF ファイルが出現した場合には，対応するフィルタを増加させる必要がある．

参考文献

[1] Xu, W. Wang, X. Zhang, Y. and Xie, H.: A Fast and Precise Malicious PDF Filter, *Proceedings of the 22nd Virus Bulletin International Conference*, pp.14-19 (2012).

[2] Pavel, L. Nedim, S.: Static Detection of Malicious JavaScript-Bearing PDF Documents, *Proceedings of the 27th Annual Computer Security Applications Conference*, pp.373-382 (2011).

[3] Nedim, S. Pavel, L.: Detection of Malicious PDF Files Based on Hierarchical Document Structure, 20th Annual Network & Distributed System Security Symposium, (2013).

[4] 三村守，田中英彦：Handy Scissors：悪性文書ファイルに埋め込まれた実行ファイルの自動抽出ツール，情報処理学会論文誌, Vol.54, No.3, pp.1211-1219 (2013).

[5] Mila, P.：16,800 clean and 11,960 malicious files for signature testing and research(online), `http://contagiodump.blogspot.jp/2013/03/16800-clean-and-11960-malicious-files.html` (2013-05-21).