情報処理学会論文誌 Vol.55 No.10 2281–2289 (Oct. 2014)

ファイル構造検査の悪性PDFファイル検知への応用

大坪 雄平[1,a)] 三村 守[2,b)] 田中 英彦[2]

受付日 2013年11月18日, 採録日 2014年7月11日

概要：標的型攻撃に用いられる悪性文書ファイルの多くには実行ファイルが埋め込まれており，我々はMS文書ファイル（Rich Textまたは Compound File Binary）の構造を検査することで悪性文書ファイルを検知する手法を提案した．この手法はファイルフォーマットに依存しており，悪性PDFファイルの検知に対応していなかった．本論文では，PDFファイルに構文解釈できない部分，表示内容と関係しない部分が含まれる等の特徴がないか構造を検査することにより，実行ファイルが埋め込まれた悪性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．ファイル構造は攻撃者の意志で変更させることが困難であることから，提案手法は長期にわたり有効である．

キーワード：標的型攻撃，マルウェア，PDFファイル，静的解析，検知

Applying File Structure Inspection to Detecting Malicious PDF Files

Yuhei Otsubo[1,a)] Mamoru Mimura[2,b)] Hidehiko Tanaka[2]

Received: November 18, 2013, Accepted: July 11, 2014

Abstract: Targeted attacks using document files that contain executable files are popular. We had proposed methods to detect malicious MS document files (Rich Text or Compound File Binary) using file structure inspection. The methods depend on file format, and couldn't detect malicious PDF files. In this paper, focus on features of malicious PDF files that contain executable files to detect them. 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. 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. The methods are effective over a long time. Because an attacker is almost not able to alter a file structure.

Keywords: targeted attack, malware, PDF file, static analysis, detection

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．経済産業省が実施した調査によると，2007年には標的型攻撃を受けた経験がある企業は5.4%にとどまっていたが，2011年には約6倍の33%に拡大[1]する等，大きな脅威となっている．特に，標的型攻撃にMS文書（Rich Textまたは Compound File Binary）ファイル，PDF（Portable Document Format）ファイル等の文書ファイルが用いられた場合，受信者が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．

一方，標的型メール攻撃に用いられる悪性文書ファイルの多くには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ている．そこで，我々はMS文書ファイルの構造を検査してその特徴を把握することで実行ファイルが埋め込まれた悪性文書ファイルを検知する手法を提案した[2]．この手法では，悪性文書ファイルに埋め込まれたexploit（閲覧ソフトの脆弱性を攻撃するコード）や実行ファイル（exeやdll）等の不正なコードの分析はしない．

[1] 警察庁
NPA, Chiyoda, Tokyo 100–8974, Japan
[2] 情報セキュリティ大学院大学
IISEC, Yokohama, Kanagawa 221–0835, Japan
[a)] mjp11001@grips.ac.jp
[b)] dgs104101@iisec.ac.jp

その代わり，文書ファイルフォーマットの整合性を検査するファイル構造検査のみを実施し，実行ファイルが埋め込まれた悪性 MS 文書ファイルの多くを検知することができる．しかしながら，ファイル構造検査は対象文書ファイルのファイルフォーマットに依存しており，悪性 PDF ファイルの検知には対応していなかった．

我々が実行ファイルが埋め込まれた悪性 PDF 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 ファイルについても，構造を検査してその特徴を把握し，それを用いて実行ファイルが埋め込まれた悪性 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，実行ファイルが埋め込まれた悪性 PDF ファイルを高い精度で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，exploit を含む不正なコードを動作させずに検査する．この検査では実際にマルウェアは動作しないため，本論文の研究内容は静的解析の一種といえる．静的解析によって悪性文書ファイルを分析する手法としては，文書ファイルを分析対象とする手法およびそれ以外のファイルも分析対象とできる手法に分類される．本研究内容は PDF ファイルのみを検査対象としているため，文書ファイルを分析対象とする手法に分類される．さらに，文書ファイルを分析対象とする手法は exploit に着目した手法，文書ファイルに埋め込まれた実行ファイルに着目した手法，不正なコード以外の文書ファイルの構造に着目した手法等がある．本研究は不正なコード以外の文書ファイルの構造に着目した手法に分類される．以下，文書ファイルを分析対象とする手法のうち，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．

2.1 exploit に着目した手法

文献 [3] 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．この手法では，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．我々の提案手法では学習を必要としないため，サンプルは不要である．文献 [4] では，不正な JavaScript の分類および検知に利用可能な特徴点を抽象構文解析木を用いて抽出する手法が提案されている．この手法は HTML ファイルに埋め込まれた JavaScript を対象としているが，exploit に着目しており，悪性 PDF ファイルの exploit に利用された JavaScript の分類および検知にも適用可能であると考えられる．これらの手法は実行ファイルが埋め込まれていない悪性 PDF ファイルも検知することが可能である．しかしながら，悪性 PDF ファイルが exploit に利用するものは JavaScript 以外にもたとえば Flash，フォントや画像等があり，それぞれに対応した検知手法を検討する必要がある．本論文では，標的型メール攻撃によく用いられる実行ファイルが埋め込まれた悪性 PDF ファイルに対象を限定することで，exploit に依存することなく悪性 PDF ファイルを検知することができる．

2.2 実行ファイルに着目した手法

文献 [5] 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 Handy Scissors という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 [6]．この手法は，実行ファイルが埋め込まれた PDF ファイルを対象としている点が本論文と共通する．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．

2.3 文書ファイルの構造に着目した手法

文献 [7] では，潜在的に危険なアクションをともな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 [8] 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．これらの手法は，悪性 PDF ファイル全般を対象としており，実行ファイルが埋め込まれていない PDF ファイルも検知することが可能である．しかしながら，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．我々の提案手法では学習を必要としないため，サンプルは不要である．また，標的型メール攻撃によく用いられる実行ファイルが埋め込まれた悪性 PDF ファイル特有のファイル構造に絞ってファイル構造を検査することで，悪性 PDF ファイルを検知する精度を高めている．

3. PDF の基本構造

PDF の仕様は，2008 年 7 月に ISO（国際標準化機構）によって標準化されている [9]．加えて，Adobe 社独自のバージョンアップも “Adobe Extensions” という形で行われている [10]．

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

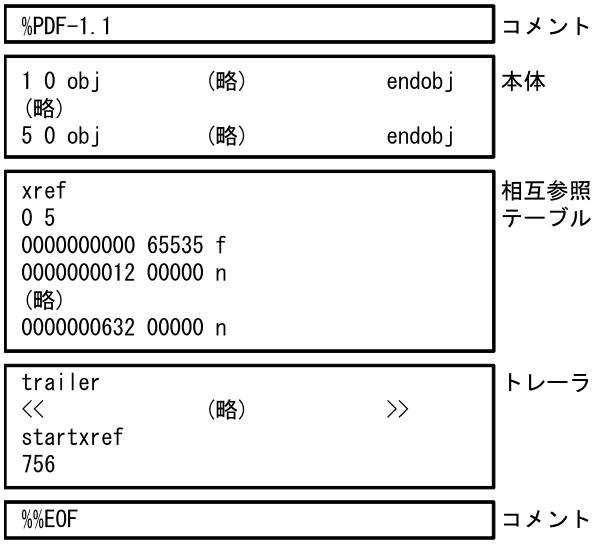

図 1 PDF ファイルの例
Fig. 1 A sample PDF file.

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，閲覧ソフトは特に処理を実施しない．ただし “%PDF-” は PDF のバージョンを示すヘッダとして使用され，“%%EOF” は PDF ファイルの終端を示すマーカとして使用される．

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と世代番号（たいていの場合は 0）が割り振られている．オブジェクトのコンテンツは “[オブジェクト番号] [世代番号] obj” および “endobj” という文字列に囲まれている．

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．本体に含まれるオブジェクトの数や各オブジェクトごとの開始オフセットおよびオブジェ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．

相互参照オブジェクト，トレーラおよび EOF マーカ（コメント）は一連で続いていることがほとんどであり，その構造は以下に述べるとおりである．“xref” 文字列のあとに相互参照テーブルのコンテンツが続く．その後，“trailer” 文字列のあとにトレーラ辞書オブジェクト，“startxref” 文字列，相互参照テーブルのオフセットを示す数字，EOF マーカの順に続く．

```
4 0 obj
<</Length 24 /Filter /ASCIIHexDecode>>
stream
48656C6C6F2C576F726C6421
endstream
endobj
```

図 2 オブジェクトの例
Fig. 2 An example of an object.

表 1 悪性 PDF ファイルによく使われるフィルタ
Table 1 Filters that used in malicious PDF files.

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

3.2 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等に使われる），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 オブジェクトの例を**図 2** に示す．Stream のコンテンツは，辞書オブジェクトのあとに続く “stream” 文字列と “endstream” 文字列に囲まれている．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．

3.3 ドキュメント構造

一般的な PDF ファイルのドキュメント構造を**図 3** に示す．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

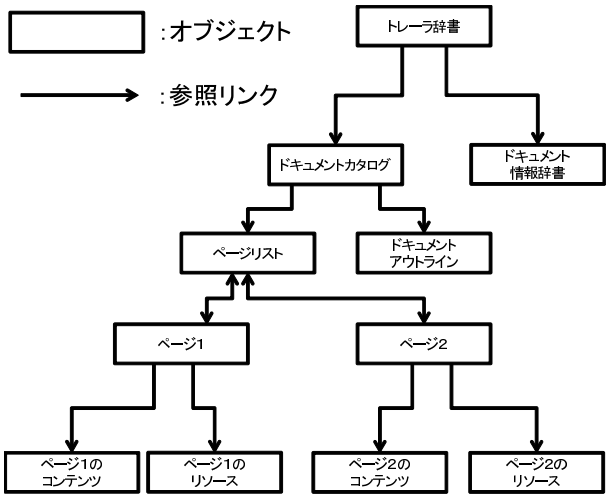

図 3 PDF ファイルのドキュメント構造の例
Fig. 3 A sample document structure of a PDF file.

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．

図 3 中のドキュメント情報辞書には，ファイルの作成日，更新日，著者名等が格納されている．また，ドキュメントカタログにはページツリーのルートオブジェクトへの間接参照，ドキュメントアウトライン（しおり）への間接参照等が格納されている．

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，このオブジェクトへの間接参照がトレーラ辞書の “/Encrypt” キーに格納されている．暗号化辞書には暗号化の方式等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．

しかしながら，暗号化 PDF ファイルの中には，復号しなければドキュメント構造にアクセスすることができないものもある．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が導入されている．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 ファイルの構造

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．

また，実行ファイルは，文書ファイルの一般的なファイル構造に沿う形で文書ファイルの途中に埋め込まれる場合もあり得るが，多くの場合には，一般的なファイル構造を無視する形で埋め込まれる．何故なら，一般的なファイル構造に沿って実行ファイルを埋め込む場合，ファイル構造規約を理解するだけでなく，文書作成ソフトが生成する構造規約に規定されていない文書作成ソフト固有の実装についても熟知する必要があるし，さらに，一般的なファイル構造に沿おうとすると埋め込むファイルのエンコード方式や大きさに一定の制約を受ける場合があり，一般的なファイル構造に沿うために実行ファイルを複数に分割して埋め込む等，実行ファイルの埋め込み方を複雑にするとデコーダ（文書ファイルに埋め込まれたファイルを取り出すコード）が複雑になるからである．たとえば，プログラムが確保しているバッファ領域を超える大きさのデータを入力することにより，メモリ破壊を起こして当該プログラムの誤動作を引き起こす脆弱性を利用して攻撃する場合，攻撃手法によっては，プログラムの制御を奪うための入力データに NULL を含めざるをえない．その場合，バッファへのコピーがそこで終了してしまうため，入力するデータに含まれる shellcode のサイズも制限されてしまう．このように，攻撃対象の脆弱性によっては攻撃手法が限定されるため，一般に shellcode に複雑なデコーダを実装することは困難である．したがって，一般的なファイル構造に沿わない形で実行ファイルを文書ファイルに埋め込んだ悪性文書ファイルがほとんどと考えられる．

我々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

一般的な PDF ファイルの Stream は，Stream とセットとなっている辞書に定義されたフィルタで問題なくデコードできる．また，デコードに使用するデータの大きさは Stream 内のデータの大きさと等しい．

一方，実行ファイルを埋め込まれた PDF ファイルの中には，異なる特徴を持つものがあった．具体的には，以下のとおりである．

4.3.1 フィルタ偽装

FlateDecode 等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 等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閲覧ソフトは動作する．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．

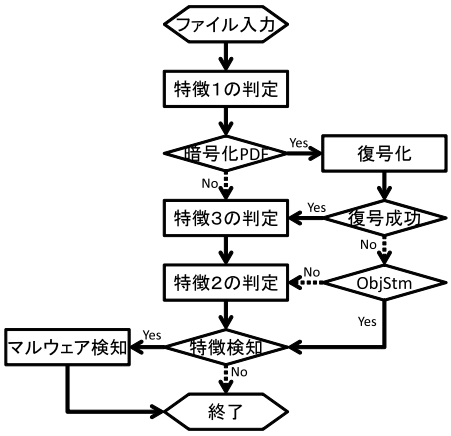

図 4 試験プログラムの動作の概要
Fig. 4 The algorithm of the test program.

5.1 動作の概要

実装したプログラムの概要を図 4 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，トレーラ辞書の情報から PDF ファイルが暗号化されているか否かを判定する．

PDF ファイルが暗号化されている場合，暗号化辞書の情報から暗号化方式を判定する．PDF ファイルに使用されている暗号化方式が 40 bitRC4，128 bitRC4，128 bitAES または 256 bitAES 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復号化に成功した場合は，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，悪性 PDF ファイルの特徴のうち 1 つでも合致すれば悪性 PDF ファイル検知とし，当該特徴を検知したデータの PDF ファイルに格納されている位置等を表示する．

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 “startxref” 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データがあった場合に特徴 1 の検知とした．

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．その結果，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．ただし，物理的に実行ファイルを埋め込むことができない大きさのオブジェクトは検知対象から除いた．これは，相互参照テーブルで表示に使用するオブジェクトとなっているものの，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，いくつかの一般的な PDF ファイルに見られたためである．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 256 Byte，NT ヘッダおよび NULL 領域で 256 Byte の合計 512 バイトの領域を使用することから，検知対象から除くオブジェクトのサイズは 512 Byte とした．

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．このとき，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．

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，複数の組織において採取した 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．特定の脆弱性の種類，検知名，RAT の種類等について，同一のものが多数含まれるといった検体の偏りが生じるのを防ぐため，検体の採取期間は 2009 年 1 月から 2012 年 12 月までとし，その間に標的型攻撃に用いられたメールとして提供を受けたものすべてから添付ファイルを取り出し，拡張子が pdf のものを機械的に選定した．その上で，同一のハッシュ値を持つものは取り除いた．このうち 2012 年に採取した検体に悪用された脆弱性を**表 3** に示す．最も悪用された脆弱性は APSB11-08（CVE-2011-0610 等）であり，その次に悪用された脆弱性は APSB10-21（CVE-2010-2883 等）であった．これらの脆弱性は 2012 年に発生した PDF ファイルを用いた標的型攻撃で悪用された脆弱性のほとんどを占めている [11]．なお，1 つの検体で複数の脆弱性を悪用するものがあるため，表 3 の左側の数字の合計は 100%にならない．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 検体の概要
Table 2 A summary of the speciments.

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

表 3 2012 年の検体が利用する脆弱性
Table 3 Vulnerabilities used by the specimens of 2012.

脆弱性	個数	割合
APSB09-04	2/17	11.8%
APSB10-02	1/17	5.9%
APSB10-07	3/17	17.6%
APSB10-21	8/17	47.1%
APSB11-08	10/17	58.8%
APSB11-30	1/17	5.9%

表 4 実験環境
Table 4 An experimental environment.

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

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 [12] 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．

実験を実施する環境は**表 4** 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．

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 5** 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

表 5 試験プログラムの検知率等
Table 5 Detection rates of the test program.

	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%

表 6 ウイルス対策ソフトとの検知率の比較
Table 6 Comparing detection rates with antivirus softwares.

	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%

は全体で 99.4%であった．また，平均実行時間は約 0.69 s であり，最も実行時間が長いもので 5.63 s 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 6** に示す．実験に用いたウイルス対策ソフトのパターンファイルは毎日最新のものに更新しており，我々が検体を入手した時点でマルウェアを検知するか否かを確認した．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 という技術が使われていた．これは，受信者の公開鍵で暗号化された PDF ファイル復号用のパスワードをリスト化して PDF ファイルに格納することで，受信者ごとに PDF ファイルのアクセス権を設定することができる技術である（文献 [9] 7.6.4 項参照）．この技術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 4 KByte 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は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 0.69 s 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．exploit や shellcode は JavaScript や Flash を利用したものが多く，PDF ファイルの Stream に埋め込まれる．PDF ファイルが暗号化された場合，Stream は暗号化の対象となる．したがって，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 をパターンマッチングで検索することは困難である．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．

ウイルス対策ソフトはマルウェアに対応するパターンファイルを作成して検知するが，マルウェアは日々新たなものが出現しており，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．Handy Scissors [5] はエンコード方式を解析し埋め込まれた実行ファイルを検知するが，未知のエンコード方式を利用したものは検知することができない．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．

マルウェアのエンコード方式は，攻撃者の意志で変更することが可能である．これに対し，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る．さらに，PDF の国際標準化は過去に 1 年半を要しており，ある程度の期間が必要となっている．また，PDF の国際規格として新たに ISO 32000-2 が策定中であるが，提案手法で検査している PDF ファイルの構造にはほとんど変更がないと想定される．よって，PDF ファイルの構造に変化をともなう仕様の変更の頻度は，マルウェアのエンコード方式の変更の頻度よりもかなり低いものと考えられる．提案手法では悪性 PDF ファイルの構造を検査対象としているため，たとえプログラムを更新しなかったとしても，ある程度の高い検知率を維持することが可能であると考えられる．

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 に連結する形で実行ファイル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，たと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．また，文字化けしたり，ファイル保存用のダイアログウインドウが表示される等の不審な挙動を示すことがあるものの，PDF 標準の機能を用いて実行ファイルをオブジェクトとして埋め込んだ場合についても提案手法は検知することはできない．これらについては，exploit で悪用される JavaScript のコードを検索する [3], [4] 等本論文の提案手法以外の手法で検知する必要がある．

試験プログラムの活用法としては，たとえば次のものが考えられる．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．ほかにも，マルウェアを解析している部署では，悪性 PDF ファイルの特徴として検知した位置から悪性 PDF ファイルに埋め込まれている実行ファイルの位置の特定に活用したり，インシデント対応では，標的型攻撃によりマルウェアに感染している可能性のあるネットワークで初期侵入に利用された端末を特定するため，当該ネットワークに属する端末に保存されている PDF ファイルを試験プログラムで検査する等，様々な活用法が考えられる．試験プログラムは Python というスクリプト言語で書かれており，Windows や Linux 等様々な環境で動作するため，動作させる環境によるプログラムの変更もほとんど必要がない．

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．これらの特徴は PDF ファイルに埋め込まれた実行ファイルや exploit または shellcode の中身に依存しない．そこで，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 0.69 s で 99.4%の悪性 PDF ファイルを検知することができた．実行ファイルや exploit または shellcode と異なり，PDF ファイルの構造は攻撃者の意志で変更することが困難であることから，提案手法は長期にわたり有効である．また，提案手法はメールサーバへの実装や，インシデント対応への活用等様々な活用法が期待できる．一方，ドライブバイダウンロードに用いられる PDF ファイルは標的型メール攻撃に用いられる PDF ファイルと異なり，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ていることがほとんどない．したがって，本論文の提案手法ではほとんど検知することができないため，本論文の提案手法以外の手法で検知する必要がある．

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．また，PDF の仕様に準拠しない PDF 生成ソフトで作成した PDF ファイルを誤検知する可能性があるため，これについても対応する必要がある．加えて，他の標的型メール攻撃対策手法と比較した有効性の評価については引き続き調査する必要がある．

参考文献

[1] 経済産業省：最近の動向を踏まえた情報セキュリティ対策の提示と徹底（オンライン），入手先 ⟨http://www.meti.go.jp/press/2011/05/20110527004/20110527004.html⟩（参照 2013-05-08）．

[2] 大坪雄平，三村　守，田中英彦：ファイル構造検査による悪性 MS 文書ファイルの検知，情報処理学会論文誌，Vol.55, No.5, pp.1530–1540 (2014).

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

[4] 神薗雅紀，西田雅太，小島恵美，星澤裕二：抽象構文解析木による不正な JavaScript の特徴点抽出法の提案，情報処理学会論文誌，Vol.54, No.1, pp.349–356 (2013).

[5] 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).

[6] 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 (2014).

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

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

[9] ISO 32000-1: Document management - Portable document format - Part 1: PDF1.7, International Organization for Standardization (2008).

[10] Adobe: PDF Reference and Adobe Extensions to the PDF Specification (online), available from ⟨http://www.adobe.com/devnet/pdf/pdf_reference.html⟩ (accessed 2013-09-13).

[11] 日本アイ・ビー・エム株式会社：2012 年下半期 Tokyo SOC 情報分析レポート，IBM - Japan (2013).

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

大坪 雄平

1981 年生．1987 年頃からプログラム作成に興味を持つ．2005 年東京大学工学部マテリアル工学科卒業．ナノサイズの物性解析に用いるシミュレーション手法の開発・研究に従事．同年警察庁入庁．2007 年警察庁生活安全局情報技術犯罪対策課．2010 年警察庁情報通信局情報技術解析課．2012 年政策研究大学院大学公共政策プログラム（修士課程）修了．2012 年から 2014 年の間内閣官房情報セキュリティセンター出向．

三村 守　（正会員）

2001 年防衛大学校情報工学科卒業．同年海上自衛隊入隊．2008 年防衛大学校理工学研究科前期課程修了．同年海上自衛隊保全監査隊勤務．2011 年情報セキュリティ大学院大学博士後期課程修了．博士（情報学）．同年情報セキュリティ大学院大学客員研究員．マルウェア解析，標的型攻撃の相関分析に関する研究に従事．2011 年から 2013 年の間内閣官房情報セキュリティセンター出向．

田中 英彦　（名誉会員，フェロー）

1970 年東京大学大学院博士課程修了，工学博士．東京大学工学部教授，同情報理工学系研究科教授・研究科長を経て，2004 年情報セキュリティ大学院大学教授，研究科長．2012 年同大学学長．計算機アーキテクチャ，分散処理，知識処理，デペンダブル情報システム等に興味を持つ．著書に『非ノイマンコンピュータ』，『計算機アーキテクチャ』，『Parallel Inference Engine』等がある．人工知能学会名誉会員，電子情報通信学会フェロー，IEEE ライフフェロー．