# ファイル構造検査による悪性MS文書ファイルの検知

大坪 雄平[1,a)]　三村 守[2,b)]　田中 英彦[2]

**概要**：今日，標的型攻撃は増加傾向にあり，多くの組織にとって真の脅威となってきている．標的型攻撃には様々な手法があるが，受信者の興味を引くメールにマルウェアを添付する方式が最も一般的である．攻撃を秘匿するため，マルウェアが文書ファイルに埋め込まれた場合，一般に，受信者にはマルウェアを見抜く手段がない．われわれが実行ファイル形式のマルウェアが埋め込まれた悪性 MS 文書ファイル（Rich Text または Compound File Binary）を分析したところ，多くの悪性 MS 文書ファイルで通常の MS 文書ファイルとファイル構造に違いがあることが分かった．本論文では，悪性 MS 文書ファイルの検知手法として，ファイル構造検査をすることを提案する．具体的には，5 種類の新しいマルウェア検知法を提案する．提案の有効性を検証する実験を行った結果，98.4%の悪性 MS 文書ファイルを検知することができた．

**キーワード**：標的型攻撃，マルウェア，MS 文書ファイル，静的解析，検知

# Methods to Detect Malicious MS Document File using File Structure Inspection

***Abstract:*** Today, the number of targeted attacks is increasing, and targeted attacks are becoming a serious threat for many organizations. There are various kinds of targeted attacks. Above all, a method to attach malware to interesting e-mail for the recipient is the most popular. In general, there is no way to distinguish a malicious document file from a normal one, because malware is embedded in a document file to hide oneself during an attack. We analyzed malicious MS document (Rich Text or Compound File Binary) files containing malware. Then, we found that there are differences in file structure between normal MS document files and malicious ones. In this paper, we propose detection methods of malicious MS document files using file structure inspection. Specifically, we propose five novel malware detection methods. The experimental result shows the effectiveness of the methods. The methods could detect 98.4% of the malicious MS document files in the experiment.

***Keywords:*** targeted attack, malware, MS document 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] するなど，大きな脅威となっている．

標的型攻撃に用いられるマルウェアが，特定の組織や個人に送付された場合，マルウェアに気づいた組織や個人がウイルス対策ソフトのベンダに検体を提供しなければ，パターンファイルを作成することが難しい．したがって，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でもマルウェアを検知できないことがほとんどである．

標的型攻撃に用いられる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 が動作し，shellcode（侵入したマシン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マルウェアやダミー表示用の文書ファイルを取り出し，実行する．これによってマルウェアに感染する．一方，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，受信者には

[1] 内閣官房情報セキュリティセンター
NISC, Chiyoda, Tokyo 100–0014, Japan
[2] 情報セキュリティ大学院大学
IISEC, Yokohama, Kanagawa 221–0835, Japan
[a)] yuhei.otsubo@cas.go.jp
[b)] dgs104101@iisec.ac.jp

マルウェアの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手段がない．

閲覧ソフトに読み込ませて脆弱性を攻撃する exploit と違い，文書ファイルに埋め込まれたマルウェア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場所に埋め込まれることが多い．われわれが CFB（Compound File Binary）[2]（doc，xls，ppt 拡張子）や Rich Text[3]（rtf 拡張子）の悪性 MS 文書ファイルを分析したところ，多くの悪性 MS 文書ファイルで通常の MS 文書ファイルとファイル構造に違いがあることが明らかになった．よって，悪性 MS 文書ファイルのファイル構造を検査し，悪性 MS 文書ファイル特有の特徴を検知すれば，悪性 MS 文書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，MS 文書ファイルが悪性 MS 文書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，MS 文書ファイルが悪性 MS 文書ファイルであるか否かを，exploit 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．静的解析によってマルウェアを検知する手法としては，実行ファイルを分析する手法，それ以外のファイルも分析できる手法および文書ファイルを分析する手法に分類される．MS 文書ファイルを検査対象としているため，本論文の研究内容は文書ファイルを分析する手法に分類される．

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．MS 文書ファイル専用の解析ツールである OfficeMalScanner[5] は，MS 文書ファイルから不正なコードによく利用されるコードを検索したり，文書ファイルに埋め込まれた実行ファイルや別の文書ファイルをヘッダに使われる文字列を検索することにより抽出することができる．

本論文では，MS 文書ファイルの構造のみを検査しており，不正なコードやヘッダに使われる文字列の検索は行わない．

文献 [6] では，MS 文書ファイルの構造を検査することにより，MS 文書ファイルに埋め込まれた，表示内容と関係のないデータを解析するツールが提案されている．このツールは MS 文書ファイル内でデータが秘匿される可能性がある 4 種類の場所を検索する．この 4 種類の中で本論文の提案手法と類似するものが 1 種類あった．

本論文では，MS 文書ファイルの構造を検査することにより悪性 MS 文書ファイルを検知することを目的としており，悪性 MS 文書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 MS 文書ファイルを検知する確率を高くしている．

```
{\rtf
Hello,\par
{\b world}!\par
}
```

**図 1** Rich Text の例
**Fig. 1** An example of a Rich Text format file.

## 3. 悪性 MS 文書ファイルのファイル構造

exploit は閲覧ソフトの誤動作を目的としており，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，閲覧ソフトの誤動作を防いだり，表示される内容がいわゆる文字化け状態になることを防ぐため，マルウェア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場所に埋め込まれることが多い．その結果，ファイル構造に通常の文書ファイルとは異なる特徴が表れる．

われわれが悪性 MS 文書ファイルのファイル構造を分析し，判明した悪性 MS 文書ファイルの特徴を以下に示す．

### 3.1 Rich Text の場合

#### 3.1.1 基本構造

Rich Text のデータは通常，7bit の ASCII 文字列で記述されており，プレーンテキストに装飾やレイアウトのための制御用の文字列を付加した形式となっている．単純な Rich Text の例を**図 1** に示す．ファイルの最初の文字は，“{” である．Rich Text は入れ子構造となっており，ファイルの最後の文字は，ファイルの最初の “{” に対応する “}”（EOF）となっている．

#### 3.1.2 特徴 1：EOF 違反

一般的な Rich Text では “}”（EOF）がファイルの末尾となっていたが，マルウェアを埋め込まれた Rich Text では EOF の後にデータが追加されているものがほとんどであった．

### 3.2 CFB の場合

#### 3.2.1 基本構造

Microsoft Word，Microsoft Excel や Microsoft PowerPoint 等で保存されるときに使用される doc，xls，ppt という拡張子のファイルは CFB と呼ばれるファイルの一種であり，文書ファイルに利用される様々なデータを 1 つのファイルに集約して保存している．CFB の階層構造はファイルシステムによく似た構造となっており，ファイルに相当する Stream とディレクトリに相当する Storage の集合体となっている．CFB のファイル構造は 512Byte のヘッダと sector と呼ばれる一連の index 番号が振られた小

さなブロックの集合で構成されている．Stream に格納するデータが sector サイズより大きい場合，複数の sector に分割される．各 sector がどう連結しているかという情報は FAT（File Allocation Table）という領域で管理されている．$n$ 番目の sector の次に連結する sector の番号が FAT の $n \times 4$Byte 目のデータに格納されている．ただし，次に連結する sector がない場合は”-2“が，該当 sector が Free Sector（未使用の sector）の場合は”-1“が格納されている．各 Stream，Storage の名称，サイズ，親子関係などの情報は DE（Directory Entry）という領域で管理されている．

#### 3.2.2 特徴 2：ファイルサイズ違反

一般的な CFB のファイルサイズはヘッダサイズを除くと sector サイズの倍数であり，ファイルサイズからヘッダサイズを除いたものを sector サイズで割った時の余りは 0 となる．ファイルサイズを $\mathrm{Size}_{\mathrm{file}}$，sector サイズを $\mathrm{Size}_{\mathrm{sector}}$ とすると，以下の数式が成り立つ．

$$(\mathrm{Size}_{\mathrm{file}} - 512) \bmod \mathrm{Size}_{\mathrm{sector}} = 0 \quad (1)$$

一方，マルウェアを埋め込まれた CFB の中には，CFB が sector 単位で区切られているという特徴を無視してマルウェアを埋め込んでいるものがあり，上記 (1) 式が成り立たないものがあった．

#### 3.2.3 特徴 3：FAT 参照不可能領域

FAT において sector1 個分を管理するために必要な領域は 4Byte である．したがって，FAT の sector1 個で $\mathrm{Size}_{\mathrm{sector}} \div 4$ 個の sector を管理できる．FAT に割り当てられている sector の数を $\mathrm{Count}_{\mathrm{FAT}}$ とすると，FAT で理論的に参照可能な領域の大きさ $\mathrm{Size}_{\mathrm{FAT}}$ は以下の数式で表される．

$$\mathrm{Size}_{\mathrm{FAT}} = \mathrm{Count}_{\mathrm{FAT}} \times \mathrm{Size}_{\mathrm{sector}} \div 4 \quad (2)$$

一般的な CFB のファイルサイズは，ヘッダサイズを除くと，FAT で理論的に参照可能な領域の大きさに収まっており，以下の数式が常に成り立っていた．

$$\mathrm{Size}_{\mathrm{file}} - 512 \leq \mathrm{Size}_{\mathrm{FAT}} \quad (3)$$

一方，マルウェアを埋め込まれた CFB の中には，FAT で参照可能な領域の上限を超えたファイルサイズのものがあり，上記 (3) 式が成り立たないものがあった．

#### 3.2.4 特徴 4：Free Sector 位置違反

文献 [6] で検索している Free Sector についてファイル末尾に該当する sector に絞って一般的な CFB を調べると，すべて Free Sector ではなかった．これは，MS 文書ファイルのファイルサイズを小さくするため，文書編集ソフトで末尾の Free Sector を削るように実装されているものと思われる．

一方，マルウェアを埋め込まれた CFB の中には，ファイル末尾に該当する sector で Free Sector であるものがみられた．

#### 3.2.5 特徴 5：使途不明の sector

CFB では，sector は，DIFAT（Double-Indirect FAT），FAT，miniFAT，DE，Stream または Free Sector の 6 種類に分類される．ここでいう DIFAT は，FAT に使用されている sector を管理するための領域であり，miniFAT はある一定サイズ未満の Stream をまとめて管理するための領域である．

一方，マルウェアを埋め込まれた CFB の中には，上記 6 種類に分類できない sector を持つものがあった．

## 4. 試験プログラムの実装

これまでに示した 5 つのファイル構造上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．

### 4.1 動作の概要

実装したプログラムの概要を示す．試験プログラムは文書ファイルを引数として受け取り，悪性 MS 文書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，文書ファイルを入力として受け付け，ヘッダの文字列から Rich Text か，CFB かを判定する．Rich Text であれば特徴 1 に該当するか判定し，特徴に合致すればマルウェア検知とする．CFB 形式であれば特徴 2 から特徴 5 に該当するか否かを独立して判定し，判定終了後，いずれかの特徴に合致すれば悪性 MS 文書ファイル検知とした．

### 4.2 特徴 1 の判定

文書ファイルを 1Byte ずつ読み込み，EOF に該当する “}” を読み込んだ時点で，まだ読み込まれていないデータがある場合に特徴 1 の検知とした．

### 4.3 特徴 2 の判定

CFB のヘッダの 30Byte 目に sector サイズに関する情報が 2Byte の数値で格納されている．この値を SectorShift とすると sector サイズ $\mathrm{Size}_{\mathrm{sector}}$ は $2^{\mathrm{SectorShift}}$ で表される．この値を用いて 3.2.2 の (1) 式が成り立たない場合に特徴 2 の検知とした．

### 4.4 特徴 3 の判定

CFB のヘッダの 44Byte 目に FAT に使用している sector 数が 4Byte の数値で格納されている．この値を $\mathrm{Count}_{\mathrm{FAT}}$ とし，3.2.3 の (2) 式から $\mathrm{Size}_{\mathrm{FAT}}$ を計算し，3.2.3 の (3) 式が成り立たない場合に特徴 3 の検知とした．

### 4.5 特徴 4 の判定

CFB のファイル末尾に該当する sector の index 番号を $n$ とすると，CFB の中には 512Byte のヘッダと $n+1$ 個の sector があるため，$\mathrm{Size}_{\mathrm{file}}$ は以下の数式で表される．

$$\mathrm{Size}_{\mathrm{file}} = 512 + (n+1) \times \mathrm{Size}_{\mathrm{sector}} \quad (4)$$

この式を $n$ について解き，$n$ 番目の FAT の値が”-1“（Free Sector）であった場合に特徴４の検知とした．

### 4.6 特徴５の判定

CFB のヘッダには FAT 等に使用している sector 数が格納されている．sector1 個ずつ種類を調べることはせずにヘッダの情報を活用すると実装が単純で高速となる．したがって，CFB のファイルサイズから求めた実際の sector 数とヘッダ，FAT および DE から導き出せる理論上の sector 数を比較することにより，特徴５の判定を行う．

実際の sector 数 $\mathrm{Count}_{\mathrm{real}}$ は，ヘッダサイズを除いたファイルサイズを sector サイズで割ったものであり，以下の数式で表される．

$$\mathrm{Count}_{\mathrm{real}} = (\mathrm{Size}_{\mathrm{file}} - 512) \div \mathrm{Size}_{\mathrm{sector}} \quad (5)$$

FAT に使用している sector 数 $\mathrm{Count}_{\mathrm{FAT}}$ は，CFB のヘッダの 44Byte 目に 4Byte の数値で格納されている．

miniFAT に使用している sector 数 $\mathrm{Count}_{\mathrm{miniFAT}}$ は，CFB のヘッダの 64Byte 目に 4Byte の数値で格納されている．

DIFAT に使用している sector 数 $\mathrm{Count}_{\mathrm{DIFAT}}$ は，CFB の 72Byte 目に 4Byte の数値で格納されている．

DE が格納されている Stream の最初の sector の index 番号が CFB のヘッダの 48Byte 目に 4Byte の数値で格納されている．この index 番号をもとに FAT の情報を参照し，DE に使用している sector 数 $\mathrm{Count}_{\mathrm{DE}}$ を数えることで求める．

Stream に使用している sector 数は DE の情報から計算する．DE は 1 エントリあたり 128Byte のデータとなっており，120Byte 目に Stream のサイズを示す 4Byte の数値が格納されている．Stream のサイズが一定サイズ未満の場合，当該 Stream は Root Entry という Stream にまとめて格納される．Root Entry に格納される Stream のサイズの上限は，CFB のヘッダの 56Byte 目に 4Byte の数値で格納されており $\mathrm{Size}_{\mathrm{mini}}$ とする．Directory Entry に $n$ 番目に登録されているエントリの Stream のサイズを $\mathrm{Size}_n$，当該 Stream の使用している sector 数を $\mathrm{Count}_n$ とする．$\mathrm{Size}_n$ が $\mathrm{Size}_{\mathrm{mini}}$ より小さい場合，当該 Stream は Root Entry に格納されるため，$\mathrm{Count}_n$ は 0 となる．$\mathrm{Size}_n$ が $\mathrm{Size}_{\mathrm{mini}}$ 以上の場合，$\mathrm{Size}_n$ を sector サイズで割った値の小数点以下を切り上げた値が $\mathrm{Count}_n$ となる．Stream に使用している sector 数 $\mathrm{Count}_{\mathrm{Stream}}$ は，すべてのエントリの Stream に使用している sector 数の合計である．

Free Sector の数 $\mathrm{Count}_{\mathrm{free}}$ は，FAT の値が”-1“となっている sector を数えることで求める．

理論上の sector 数 $\mathrm{Count}_{\mathrm{theoretical}}$ は，$\mathrm{Count}_{\mathrm{FAT}}$，$\mathrm{Count}_{\mathrm{miniFAT}}$，$\mathrm{Count}_{\mathrm{DIFAT}}$，$\mathrm{Count}_{\mathrm{DE}}$，$\mathrm{Count}_{\mathrm{Stream}}$ および $\mathrm{Count}_{\mathrm{free}}$ の合計となり，一般的な CFB では実際の sector 数と理論上の sector 数は同じ値であり，異なる値をとった場合に特徴５の検知とした．

表 1 検体の概要
**Table 1** A summary of the speciments.

| | マルウェア | | マルウェアではない | |
|---|---|---|---|---|
| 拡張子 | 検体数 | 平均容量 (KB) | 検体数 | 平均容量 (KB) |
| rtf | 98 | 266.5 | 199 | 516.2 |
| doc | 36 | 252.2 | 1195 | 106.1 |
| xls | 49 | 180.4 | 298 | 191.7 |

表 2 実験環境
**Table 2** An experimental environment.

| | |
|---|---|
| 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 |

## 5. 実験

### 5.1 実験内容

試験プログラムの性能を評価するため，悪性 MS 文書ファイルを入力して結果を分析する．実験の対象となる MS 文書ファイルの概要を**表 1** に示す．表 1 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 MS 文書ファイルで，分析により実行ファイル（マルウェア）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．ただし，拡張子は doc であるものの実際の中身は Rich Text であるものは rtf としている．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及び OfficeMalScanner の検知率を比較する．

表 1 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 [7] である．ただし，ファイルの先頭に html が付加され，文書ファイルとして認識できない状態のものがあったことから，拡張子とヘッダの中身が一致しない検体を除外している．マルウェアではないとされた検体で悪性 MS 文書ファイル特有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 2** 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．

### 5.2 実験結果

検体の拡張子ごとの検知率を**表 3** に示す．検知の成功率

表 3　試験プログラムの検知率
**Table 3**　Detection rates of the test program.

| 拡張子 | 検知数 | 検知率 | 平均実行時間 |
|---|---|---|---|
| rtf | 97 | 99.0% | 0.021s |
| doc | 35 | 97.2% | 0.062s |
| xls | 48 | 98.0% | 0.051s |
| 合計 | 180 | 98.4% | 0.037s |

表 4　検体の特徴ごとの検知状況
**Table 4**　Detection rates of the features.

| | 検知数 | 検知率 |
|---|---|---|
| 特徴 1 | 97 / 98 | 99.0% |
| 特徴 2 | 65 / 85 | 76.4% |
| 特徴 3 | 77 / 85 | 90.6% |
| 特徴 4 | 82 / 85 | 96.5% |
| 特徴 5 | 81 / 85 | 95.3% |

は全体で 98.4%であった．また，平均実行時間は約 0.037s であり，最も実行時間が長いもので 0.125s であった．

検知に成功した 180 体の検体の特徴ごとの検知状況は**表 4** に示すとおりである．表中の特徴 1 は rtf 拡張子のファイルの検知数であり，特徴 2 から特徴 5 までは doc 拡張子のファイルの検知数と xls 拡張子のファイルの検知数を合算した値である．

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および OfficeMalScanner の検知率との比較結果を**表 5** に示す．標的型攻撃に対しては，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 20.2%から 23.0%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でも，検知率は 44.8%であった．

OfficeMalScanner は，一般的な shellcode のパターンを検索する SCAN オプションおよび，総当たりで実行ファイル等を検索する BRUTE オプションを使用して実行した．また，Rich Text については，OfficeMalScanner に同封されている RTFScan を，SCAN オプションを使用して実行した．表中の OfficeMalScanner の検知数は OfficeMalScanner，RTFScan いずれかで検知した数を示す．OfficeMalScanner の検知率は 91.3%であった．

マルウェアではないとされた検体 1,692 体に関する誤検知率を**表 6** に示す．誤検知率は全体で 0.9%であったが，特に xls 拡張子において 4.7%という高い誤検知率となった．

## 6. 考察

### 6.1 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を分析した結果，失敗の原因は以下の 1 点に集約された．

表 5　ウイルス対策ソフト等との検知率の比較
**Table 5**　Comparing detection rates with antivirus softwares, etc.

| | 検知数 | 検知率 |
|---|---|---|
| 試験プログラム | 180 | 98.4% |
| T 社 AV | 41 | 22.4% |
| S 社 AV | 37 | 20.2% |
| M 社 AV | 42 | 23.0% |
| T,S,M 社 AV | 82 | 44.8% |
| OfficeMalScanner | 167 | 91.3% |

表 6　試験プログラムの誤検知率
**Table 6**　False positive rate of the test program.

| 拡張子 | 誤検知数 | 誤検知率 |
|---|---|---|
| rtf | 0 | 0.0% |
| doc | 2 | 0.2% |
| xls | 14 | 4.7% |
| 合計 | 16 | 0.9% |

- exploit の中にマルウェアが埋め込まれている．

exploit のみが埋め込まれた悪性 MS 文書ファイルには本論文で論じたような特徴は現れない．exploit とマルウェアやダミー表示用の文書ファイルが別の場所に埋め込まれている場合は本論文の提案手法で検知することができるが，exploit とマルウェアやダミー表示用の文書ファイルが一体化している場合は本論文の提案手法で検知することはできない．検知に失敗した 3 個のうち 2 個は，文献 [4] または OfficeMalScanner で検知をすることができた．

### 6.2 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 2 点に集約された．

- ファイルの末尾に不要な html が付加されている．
- ファイルが途中で切れている．

まず最初に誤検知の原因としてあげられるのは，ファイルの末尾に不要な html が付加されている場合である．今回検知したデータは，すべて 4KByte 弱の同一の html データであった．この中にマルウェアが埋め込まれている可能性は低いと考えられるため，付加されたデータのサイズでフィルタリングすることで当該誤検知を回避することは可能ではある．しかしながら，ファイルの末尾に不要な html が付加されている MS 文書ファイルは，一般的な文書編集ソフトは作成することはないため，異常な MS 文書ファイルとして検知するという運用も考えられる．

次の原因としては，ファイルが途中で切れている場合である．ファイルが途中で切れているため，

- ヘッダを除いたファイルサイズが sector サイズ単位になっていない．（特徴 2 ）
- ファイルサイズから求めた sector 数とヘッダ情報等から計算した sector 数が一致しない．（特徴 5 ）

などの特徴により，悪性 MS 文書ファイル特有の特徴として誤検知していた．しかしながら，ファイルが途中で切れている MS 文書ファイルは，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 MS 文書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．

したがって，今回誤検知した検体はいずれも，通常使用しないファイルであり，contagio において検体を収集する際にダウンロードに失敗したもの等が混在した可能性が考えられる．

### 6.3 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 0.037s で，98.4%という高い確率で悪性 MS 文書ファイルを検知することに成功した．さらに，誤検知率が 0.9%であったが，誤検知したファイルはすべて通常は使用しないファイルであったことから, 試験プログラムで検知したものはほぼ間違いなく不審なものと考えて問題ないであろう．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．添付ファイルがパスワードで暗号化された zip ファイルであった場合，ウイルス対策ソフトでは通常中身を検査することができない．しかしながら，パスワードで暗号化された zip ファイルであっても，格納されているファイルの名称とサイズは復号しなくても判明する．一方，特徴 2 の判定に関しては，拡張子とファイルサイズだけで 76.4%の悪性 MS 文書ファイルを検知することができる．したがって，特徴 2 の判定についてはパスワードで暗号化された圧縮ファイルに格納された悪性 MS 文書ファイルにも適用可能である．

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いる．OfficeMalScanner は不正なコードによく利用されるコードを検知するが，エンコードされたり未知の不正なコードは検知できない．文献 [4] 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に高い確率で悪性 MS 文書ファイルを検知することに成功した．さらに，試験プログラムは悪性 MS 文書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 MS 文書ファイルを検知することに成功した．試験プログラムは MS 文書ファイルの構造という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いものを検査対象としている．したがって，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ろう．

一方，本論文の提案手法は，exploit 部分は検知することはできない．したがって，exploit の中にマルウェアが埋め込まれているものや exploit のみが埋め込まれてるもの，例えばマルウェア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．

## 7. おわりに

本論文では，マルウェアが埋め込まれた MS 文書ファイルのファイル構造を分析し，マルウェアが埋め込まれた MS 文書ファイル特有の特徴を 5 つ明らかにした．さらに，悪性 MS 文書ファイルの検知手法として，ファイル構造検査により当該 5 つの特徴を検知することを提案し，提案の有効性を検証する実験を行った結果，平均実行時間 0.037s で 98.4%の悪性文書ファイルを検知することができた．

今後の課題としては，悪性 MS 文書ファイル特有の 5 つの特徴がない場合への対策があげられる．2009 年から 2012 年に発見された悪性 MS 文書ファイルでは，ほとんどの文書ファイルに悪性 MS 文書ファイル特有の特徴があったため，高い確率で悪性 MS 文書ファイルを検知することができた．しかしながら，マルウェアを Stream に偽装して埋め込むという方式も考えられる．PDF ファイルでは，マルウェアを正規のオブジェクトに偽装して埋め込んだ悪性文書ファイルも確認されている．このようにマルウェアを正規のオブジェクトに偽装して埋め込む手法が主流になった場合には，より詳細なファイル構造検査を実施する必要がある．

### 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 (online)，入手先 ⟨http://www.meti.go.jp/press/2011/05/20110527004/20110527004.html⟩ (2013-05-08).

[2] Microsoft : [MS-CFB]: Compound File Binary File Format(online)，入手先 ⟨http://msdn.microsoft.com/en-us/library/dd942138.aspx⟩ (2013-05-22).

[3] Microsoft : Rich Text Format (RTF) Specification, version 1.9.1(online)，入手先 ⟨http://www.microsoft.com/en-us/download/details.aspx?id=10725⟩ (2013-05-22).

[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] Boldewin, F. : Analyzing MSOffice malware with OfficeMalScanner(online)，入手先 ⟨http://www.reconstructer.org/papers/Analyzing%20MSOffice%20malware%20with%20OfficeMalScanner.zip⟩ (2013-05-08).

[6] Hyukdon, K. Yeog, K. Sangjin, L. and Jongin, L. : A Tool for the Detection of Hidden Data in Microsoft Compound Document File Format，*ICISS '08 Proceedings of the 2008 International Conference on Information Science and Security*, pp.141-146 (2008).

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