情報処理学会論文誌 Vol.55 No.5 1530–1540 (May 2014)

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]

受付日 2013年9月13日, 採録日 2014年2月14日

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，実行ファイルが文書ファイルに埋め込まれた場合，一般に，受信者には通常の文書ファイルと区別する手段がない．我々が実行ファイル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，幾種かのファイル構造検査をすることを提案する．提案手法の有効性を検証する実験を行った結果，98.5%の悪性 MS 文書ファイル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，提案する Rich Text および CFB 形式の悪性文書ファイルの検知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，MS 文書ファイル，静的解析，検知

Methods to Detect Malicious MS Document File Using File Structure Inspection

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

Received: September 13, 2013, Accepted: February 14, 2014

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 an executable file 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 an executable file. 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. The experimental result shows the effectiveness of the methods. The methods could detect 98.5% of the malicious MS document files in the experiment. The methods are effective in the detection of malicious Rich Text files and malicious CFB files over a long time. Because the attacker is almost not able to alter a file structure.

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] する等，大きな脅威となっている．

標的型攻撃の概要を以下に示す．攻撃者は受信者が不審に思わないような件名および本文の電子メールにマルウェアを添付して送付する．受信者がマルウェアと気づかず

[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

に添付ファイルを開封すると端末がマルウェアに感染する．マルウェアに感染した端末は攻撃者が準備したコマンド&コントロールサーバ等に接続し，当該端末は遠隔操作される．攻撃者はシステム内にマルウェアの感染を拡大させ，システム内の端末やサーバの管理者権限を奪取し，情報を窃取する．

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．この理由として，攻撃者は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た後，特定の組織や個人に送付していることが考えられる．ウイルス対策ソフトのベンダがパターンファイルを作成するためには，特定の組織や個人が自らマルウェアに気づき，ウイルス対策ソフトのベンダに検体を提供する必要がある．しかしながら，特に個人の場合，ウイルス対策ソフトに頼らずに自らマルウェアに気づくことは困難である．

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 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（exe や dll）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したりする．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．

悪性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．さらに，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，受信者には実行ファイル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ことは困難である．

我々が Microsoft 社の開発した Rich Text [2]（rtf 拡張子．ただし，ほとんどの場合 doc 拡張子に偽装されている）や CFB（Compound File Binary）[3]（doc，xls，ppt，jtd，jtdc 拡張子）の MS 文書ファイルに実行ファイルが埋め込まれた悪性 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 MS 文書ファイル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高い精度で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検知する手法としては，文書ファイルを分析する手法およびそれ以外のファイルも分析できる手法に分類される．本論文の研究内容は MS 文書ファイルを検査対象としているため，文書ファイルを分析する手法に分類される．以下，本論文の研究内容に関連する先行研究について述べる．

文献 [4] では，バイナリデータの値を統計的に分析することによって，文書ファイルに隠された不正なコードを検出する手法が提案されている．文献 [5] では，バイナリデータの統計分析に加え，動的解析を組み合わせて文書ファイルに隠された不正なコードを検出する手法が提案されている．これらの手法では，ファイルに隠された exploit を含む不正なコードを検出することに主眼をおいている．しかしながら，統計分析では，統計的学習を用いているため学習するサンプルに依存するほか，不正なコードの大きさがある程度必要である．また動的解析では，ある特定の条件でのみ動作する不正なコードを検知することは難しい．本論文では，学習は必要としないため，そのためのサンプルは不要である．また，本論文では，MS 文書ファイルのファイル構造のみを検査しており，不正なコードの中身に検知率は依存しない．

文献 [6] 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，実行ファイルに頻出する文字列を検索することで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たり実行ファイルに頻出する文字列が改変されたりす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 [7]．MS 文書ファイル専用の解析ツールである OfficeMalScanner [8] 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ダミー表示用の文書ファイルのヘッダに使われる文字列を検索したりすることにより抽出することができる．しかしながら，不正なコードや実行ファイル等がエンコードされた場合は検知ができないという課題がある．本論文では，MS 文書ファイルのファイル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ため，検知率はエンコード方式に依存しないことに加え高速に動作する．

文献 [9] 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 種類の場所を表示する．この 4 種類の中で本論

文の提案手法で検査する場所と類似するものが 1 種類あった．表示内容と関係ないデータがある文書ファイルは，不審である可能性が考えられる．しかしながら，通常の MS 文書ファイルにも表示内容と関係のないデータが存在する．たとえば，CFB のファイル構造はある固定のサイズ単位で分割して管理されており，論理的に割り当てる領域と実際に使用する領域に差分がある場合，未使用の領域が発生する．これは表示内容と関係のないデータである．このことから，仮にこのツールを悪性 MS 文書ファイル検知に活用した場合，悪性 MS 文書ファイルだけでなく通常の MS 文書ファイルも検知してしまうという課題がある．本論文では，表示内容と関係のないデータではなく，実行ファイルが埋め込まれた悪性 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．

Microsoft 社が Microsoft Office 2010 から導入した Office ファイル検証機能 [10] は，Office ファイル（xls，doc，ppt，pub 拡張子）を開こうとした場合に，正常なファイル構造の Office ファイルか否かを検証し，正常なファイル構造でない場合に閲覧するか否か確認するポップアップウインドウが表示される機能である．この Office ファイル検証機能がどのような仕組みでファイル構造を検証しているかは不明であるものの，ファイル構造を検証するという，我々の提案と同じ目的の機能であるので，本論文では両者の検知率を比較することで間接的に差異を明らかにする．

3. 悪性 MS 文書ファイルのファイル構造

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，表示される内容がいわゆる文字化け状態になったりしてしまう．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，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．

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るが，多くの場合には，一般的なファイル構造を無視して文書ファイルの末尾に追加する形で埋め込まれる．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，インターネット上に公開されている文献 [2] や文献 [3] 等を参照して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．加えて，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る．また，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）が複雑になってしまう．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と，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である．

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

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

我々が 2009 年から 2012 年の間に複数の組織において採取した実行ファイルが埋め込まれた悪性 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 は Microsoft 社により開発された文書ファイルフォーマットの 1 つである．Rich Text のデータは通常，7 bit 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．閲覧ソフトは “}”（EOF）より後のデータについて通常は何もしない．

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，実行ファイルが埋め込まれた 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

CFB は Microsoft 社により開発された複合ファイルフォーマットの 1 つである．Microsoft Word，Microsoft

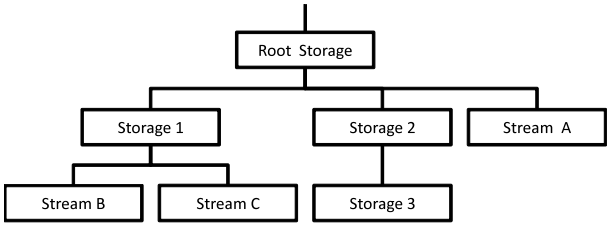

図 2 CFB の階層構造
Fig. 2 CFB hierarchy.

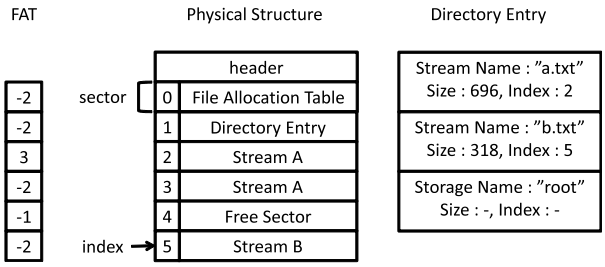

図 3 CFB のファイル構造
Fig. 3 CFB structure.

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 を利用しており，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．また，JustSystems 社が開発した日本語ワープロソフトである一太郎で使用される jtd および jtdc という拡張子のファイルも CFB を利用している．CFB の階層構造を**図 2** に示す．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．この階層構造を実現する CFB のファイル構造を**図 3** に示す．CFB のファイル構造は 512 Byte 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 は sector に格納されるが，Stream に格納するデータが sector サイズより大きい場合，Stream は複数の sector に分割して格納される．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．その内容の例を図 3 の左側に示す．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，親子関係等の情報は DE（Directory Entry）という領域で管理されている．FAT および DE が格納されている sector の index 番号はヘッダに記録されている．

閲覧ソフトが CFB ファイル内の a.txt という名前の Stream を読み込む場合の典型的な動作を図 3 を使って示す．まず，ヘッダの情報を元に FAT および DE の格納されている sector を特定する．次に，DE を読み込み a.txt というファイルを探す．DE には a.txt のデータが格納されている Stream の先頭 sector の index 番号が記録されており，当該 sector のデータを読み込む．読み込んでいる sector に対応する FAT 部分を読み込むと次に連結する sector があるか否か，ある場合はその index 番号が分かり，連結する sector がなくなるまでデータの読み込みを続ける．このように，閲覧ソフトはヘッダ，FAT および DE の情報をもとに文書ファイルの表示に必要な情報の読み込みを行う．

3.2.2 特徴 2：ファイルサイズ違反

一般的な CFB ファイル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 サイズで割ったときの余りは 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．

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

一方，実行ファイルが埋め込まれた CFB ファイルの中には，CFB ファイルが sector 単位で区切られているというファイル構造を無視してマルウェアを埋め込んでいるものがあり，上記式 (1) 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 4 Byte である．したがって，FAT に割り当てられた sector1 個あたり $\mathrm{Size}_{\mathrm{sector}} \div 4$ 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，この場合における FAT が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．

$$\mathrm{Size}_{\mathrm{FAT}} = \mathrm{Count}_{\mathrm{FAT}} \times \mathrm{Size}_{\mathrm{sector}} \div 4 \tag{2}$$

一般的な CFB ファイルはすべての sector の情報が FAT に記録されている．したがって，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，以下の数式がつねに成り立っていた．

$$\mathrm{Size}_{\mathrm{file}} - 512 \leq \mathrm{Size}_{\mathrm{FAT}} \tag{3}$$

一方，実行ファイルが埋め込まれた CFB ファイルの中には，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，上記式 (3) 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 [9] ではデータを秘匿する可能性のある場所の 1 つとして Free Sector が取り上げられている．一般的な CFB ファイルについて Free Sector が存在するか調べると，Free Sector は一般的な CFB ファイルにも存在することが判明した．そこで，ファイル末尾に該当する sector に絞って Free Sector か否かを調べたところ，Free Sector は見つからなかった．この原因としては，MS 文書ファイルのファイルサイズを小さくするため，文書作成ソフトで末尾の Free Sector を削るように実装されていることが考えられる．

一方，実行ファイルが埋め込まれた CFB ファイルの中には，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，実行ファイルが埋め込まれた CFB ファイルの中には，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図 4 に示す．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，特徴に合致すれば悪性 MS 文書ファイル検知とする．CFB 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 1 Byte 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．

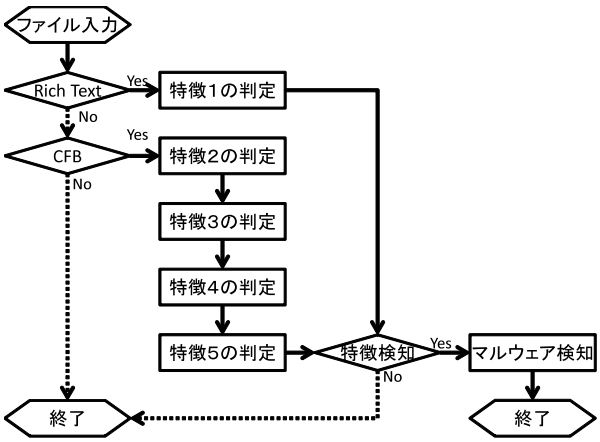

図 4 試験プログラムの動作の概要
Fig. 4 The algorithm of the test program.

4.3 特徴 2 の判定

CFB のヘッダの 30 Byte 目に sector サイズに関する情報が 2 Byte 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 項の式 (1) 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 44 Byte 目に FAT に使用している sector 数が 4 Byte の数値で格納されている．この値を $\mathrm{Count}_{\mathrm{FAT}}$ とし，3.2.3 項の式 (2) から $\mathrm{Size}_{\mathrm{FAT}}$ を計算し，3.2.3 項の式 (3) 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 ファイルのファイル末尾に該当する sector の index 番号を n とすると，CFB ファイルの中には 512 Byte 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．

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

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

使途不明の sector を検知する方法としては，先頭の sector から 1 個ずつ順番に使途を明らかにする方法がある．一方，本論文では，実装がより単純で高速となるヘッダ等の情報を活用する方法を採用した．具体的には，CFB のヘッダには FAT 等に使用している sector 数が格納されており，この情報を活用して sector の種類ごとに数を数え，使途を明らかにできた sector 数とファイルサイズから求めた実際の sector 数を比較し，差異があった場合に特徴 5 の検知とした．以下，その詳細について述べる．

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．

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

FAT に使用している sector 数 $\mathrm{Count}_{\mathrm{FAT}}$ は，CFB のヘッダの 44 Byte 目に 4 Byte 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 64 Byte 目に 4 Byte 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 72 Byte 目に 4 Byte 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 48 Byte 目に 4 Byte 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 $\mathrm{Count}_{\mathrm{Stream}}$ は DE の情報から計算する．DE は 1 エントリあたり 128 Byte のデータとなっており，120 Byte 目に Stream のサイズを示す 4 Byte 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 56 Byte 目に 4 Byte の数値で格納されており $\mathrm{Size}_{\mathrm{mini}}$ とする．DE に n 番目に登録されているエントリの Stream のサイズを Size_n，当該 Stream の使用している sector 数を Count_n とする．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，Count_n は 0 となる．Size_n が $\mathrm{Size}_{\mathrm{mini}}$ 以上の場合，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 Count_n となる．まとめると，以下の数式で表される．

$$\mathrm{Count}_n = \begin{cases} 0 & (\mathrm{Size}_n < \mathrm{Size}_{\mathrm{mini}}) \\ \lceil \mathrm{Size}_n \div \mathrm{Size}_{\mathrm{sector}} \rceil & (\mathrm{Size}_n \geq \mathrm{Size}_{\mathrm{mini}}) \end{cases} \tag{6}$$

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 数の合計となる．

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 ファイルでは実際の 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．

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 文書ファイルおよび通常の 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，複数の組織において採取した MS 文書ファイルで，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，特定の拡張子のものを機械的に選定した．そのうえで，同一のハッシュ値を持つものは取り除き，拡張子は doc であるものの実際の中身が Rich Text であるものの拡張子を rtf とした．このうち 2012 年に採取した検体に悪用された脆弱性を**表 2** に示す．最も悪用された脆弱性は Windows コモンコントロールの脆弱性（MS12-027:CVE-2012-0158）であり，その次に悪用された脆弱性は Microsoft Office の脆弱性（MS10-087:CVE-2010-3333）であり，その次に悪用された脆弱性は Adobe Flash Player の脆弱性（APSB12-18:CVE-2012-1535）であった．これらの脆弱性は 2012 年に発生した標的型攻撃に悪用された脆弱性のほとんどを占めている [11]．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，OfficeMalScanner の検知率および Office ファイル検証機能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 [12] 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 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 1 検体の概要
Table 1 A summary of the specimens.

	悪性 MS 文書ファイル		通常の MS 文書ファイル	
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	1,195	106.1
xls	49	180.4	298	191.7
jtd/jtdc	17	268.5	-	-
合計	200	243.0	1,692	169.4

表 2 2012 年の検体が利用する脆弱性
Table 2 Vulnerabilities used by the specimens of 2012.

拡張子	脆弱性	個数	割合
rtf	MS12-027	39 / 70	55.7%
	MS10-087	31 / 70	44.3%
doc	MS12-027	6 / 24	25.0%
	APSB12-22	5 / 24	20.8%
	APSB12-18	10 / 24	41.7%
	APSB12-03	1 / 24	4.2%
	APSB11-07	1 / 24	4.2%
	-	1 / 24	4.2%
xls	MS12-027	6 / 19	31.6%
	MS11-021	2 / 19	10.5%
	MS09-067	8 / 19	42.1%
	APSB12-03	1 / 19	5.3%
	APSB11-07	1 / 19	5.3%
	-	1 / 19	5.3%
jtd/jtdc	-	-	-

表 3 実験環境
Table 3 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 4 試験プログラムの検知率
Table 4 Detection rates of the test program.

拡張子	検知数	検知率	平均実行時間
rtf	97 / 98	99.0%	0.021 s
doc	35 / 36	97.2%	0.062 s
xls	48 / 49	98.0%	0.051 s
jtd/jtdc	17 / 17	100.0%	0.201 s
合計	197 / 200	98.5%	0.051 s

表 5 検体の特徴ごとの検知状況
Table 5 Detection rates of the features.

	検知数	検知率
特徴 1	97 / 98	99.0%
特徴 2	79 / 102	77.5%
特徴 3	92 / 102	90.2%
特徴 4	99 / 102	97.1%
特徴 5	98 / 102	96.1%

5.2 実験結果

検体の拡張子ごとの検知率を**表 4** に示す．検知の成功率は全体で 98.5%であった．また，平均実行時間は約 0.051 s であり，最も実行時間が長いもので 0.391 s であった．

検知に成功した 197 体の検体の特徴ごとの検知状況は**表 5** 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，xls 拡張子のファイルの検知数および jtd/jtdc 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，OfficeMalScanner の検知率およびファイル検証機能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．実験に用いた検体に対しては，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 20.0%から 21.0%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て，どれか 1 つでも検知した場合（T，S，M 社 AV）をとっても，検知率は 43.0%であった．

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

	検知数	検知率
試験プログラム	197 / 200	98.5%
T 社 AV	42 / 200	21.0%
S 社 AV	40 / 200	20.0%
M 社 AV	42 / 200	21.0%
T，S，M 社 AV	86 / 200	43.0%
OfficeMalScanner	182 / 200	91.0%
Office ファイル検証機能	33 / 85	38.8%

表 7 試験プログラムの誤検知率
Table 7 False positive rate of the test program.

拡張子	誤検知数	誤検知率
rtf	0 / 199	0.0%
doc	2 / 1,195	0.2%
xls	14 / 298	4.7%
合計	16 / 1,692	0.9%

実験に使用した OfficeMalScanner のバージョンは v0.58 であり，CFB ファイルについては，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 ファイルについては，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 91.0%であった．

Office ファイル検証機能の実験には，Microsoft Office 2007 にファイル検証機能を有効にするために必要な更新プログラム（KB2464583（MS11-021），KB2464605 および KB2509488（MS11-023））を適用し，Office ファイル検証機能のアドイン（KB2501584）をインストールしたものを使用した．doc 拡張子および xls 拡張子の悪性文書ファイルを閲覧ソフトで読み込んだときに，Office ファイル検証機能が正常なファイル構造でないと判定し，ポップアップウインドウが表示され閲覧処理が開始されなかった場合に Office ファイル検証機能による検知とした．実験に用いた検体に対する Office ファイル検証機能の検知率は 38.8%であった．また，Office ファイル検証機能で検知できて，試験プログラムで検知できなかったものはなかった．

マルウェアではないとされた検体 1,692 体に関する試験プログラムの誤検知率を**表 7** 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の概要を**表 8** に

表 8 検知に失敗した検体の概要
Table 8 A summary of the specimens not detected by the test program.

拡張子	脆弱性	実行ファイルの埋め込まれ方
rtf	MS10-087	shellcode の中
doc	APSB12-18	文書ファイルに埋め込まれた flash の中
xls	-	文書ファイルに埋め込まれた VBA の中

示す．検知に失敗した検体はすべて 2012 年に採取した検体であり，検知に失敗した検体に悪用された脆弱性は，他の検知に成功している検体にも悪用された脆弱性であったことから，特定の脆弱性を悪用した検体が必ず検知に失敗するということではなかった．分析の結果，失敗の原因は exploit および shellcode に連結する形で実行ファイルが埋め込まれているためがすべてであった．exploit および shellcode は閲覧ソフトが通常読み込む部分に埋め込まれることが多いことから，exploit および shellcode が埋め込まれた部分には本論文で論じたような特徴は現れないことが多い．exploit および shellcode と実行ファイルやダミー表示用の文書ファイルが別々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 および shellcode と実行ファイルやダミー表示用の文書ファイルが連結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．一方，検知に失敗した 3 個について，OfficeMalScanner では rtf 拡張子および xls 拡張子の 2 個が検知することができ，Handy Scissors では rtf 拡張子の 1 個が検知することができたが，いずれのツールでも検知することができなかったものは doc 拡張子の 1 個であった．

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 4 KByte 弱の同一の html データであった．これは，実験に使用した検体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，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オブジェクトを用いた悪性 MS 文書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．一方，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，一般的な文書作成ソフトが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），

等の特徴により，悪性 MS 文書ファイル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よいだろう．

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った．このようなファイルが実験で用いた検体に混在した理由の 1 つとして，ダウンロードの失敗が考えられる．しかしながら，今回の実験では contagio で配布されている zip ファイルを展開した際に作成される文書ファイルを使用しており，zip には CRC を用いたデータ破損を検査する仕組みがあるため，我々がダウンロードに失敗したとは考えにくい．つまり，contagio で配布されている zip ファイルの中に通常使用しないファイルが元々格納されていたと考えられるが，その原因については contagio でどのようにファイルを収集しているか明らかでないため推察することはできない．

6.3 Office ファイル検証機能との差異

ファイル構造を検証するという，我々の提案と同じ目的の機能を持つ Office ファイル検証機能の検知率は 38.8%であり，試験プログラムの検知率の 98.5%と比較して低い結果となった．検体の特徴ごとの検知状況と Office ファイル検証機能の検知状況を比較したが，今回の実験で使用した検体では関連性を見出すことができなかった．加えて，各特徴の検知率のうち最も検知率の低い特徴 2 の検知率は Office ファイル検証機能の検知率の値の 2 倍以上となっており，Office ファイル検証機能の機能に特徴 2 から特徴 5 までの判定がすべて包含されていたとは考えにくい．したがって，我々の提案手法は，Office ファイル検証機能とは異なる手法である可能性が高いと考えられる．

6.4 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 0.051 s で，98.5%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一般的な文書作成ソフトが作成しないファイルまたは閲覧ソフトで正しく内容を表示することのできないファイルであり，すべて通常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．トレンドマイクロ株式会社によると，わが国

表 9 2012 年上半期に攻撃に用いられたファイルの拡張子の割合
Table 9 Rate of extension of files that were used to attack in 2012.

拡張子	2012 年上半期	2012 年下半期
doc（rtf）	37.0%	17.0%
xls	8.0%	8.5%
jtd	1.0%	0.0%
pdf	19.0%	2.5%
exe	30.0%	61.0%
etc	5.0%	11.0%

で 2012 年に標的型攻撃に用いられたファイルの拡張子の割合は**表 9**（文献 [13] をもとに作成）に示すとおりであり，悪性 MS 文書ファイルは doc（rtf），xls および jtd の拡張子のものであった．試験プログラムの検知対象の rtf，doc，xls および jtd/jtdc 拡張子のファイルは，2012 年に標的型攻撃に用いられた悪性 MS 文書ファイルのほとんどを占めている．したがって，試験プログラムは大半の悪性 MS 文書ファイルに対応することが可能であると考えられる．

また，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．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．一方，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．これは，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 文書ファイルにも特徴 2 の判定が適用可能であることを示している．

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だったりする不正なコードは検知できない．Handy Scissors 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．実験に用いた検体は，少なくとも採取した時点では大手ベンダの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，ほとんど検知することができない未知のマルウェアであった．それにもかかわらず，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 や実行ファイル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．また，本論文の提案手法と同様にファイル構造を検証する Office ファイル検証機能と検知率を比較した結果，ファイル検証機能の検知率 38.8%に対し試験プログラムの検知率は 98.5%であり，試験プログラムが検知できないものでファイル検証機能で検知できるものは確認できなかったことから，本論文の提案手法の有効性が確認できた．

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 MS 文書ファイルのファイル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 MS 文書ファイルのファイル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．また，いわゆるゼロデイ攻撃にも本論文の提案手法は有効である．たとえば，実験に用いた jtd/jtdc 拡張子の検体 17 個中 14 個は検体入手時点で対策のとられていない脆弱性を利用したものであったが，試験プログラムはすべて検知することができた．doc 拡張子，rtf 拡張子等のファイルでも同様で，未知の脆弱性を利用した検体であっても，実行ファイルを埋め込む場所が変わらなければ検知することができると考えられる．

一方，本論文の提案手法は，原理的に exploit および shellcode はほぼ検知することはできない．したがって，exploit および 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 および 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．これらについては，本論文の提案手法以外の方法で検知する必要がある．

7. おわりに

本論文では，実行ファイルが埋め込まれた MS 文書ファイルのファイル構造上の特徴を 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 0.051 s で 98.5%の悪性文書ファイル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仕様を変更することが困難であることから，提案する Rich Text および CFB 形式の悪性文書ファイルの検知は長期にわたり有効である．

今後の課題としては，実行ファイルが MS 文書ファイルの末尾以外に埋め込まれた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，検知に失敗した 3 個の検体を除きすべての検体において，実行ファイルは MS 文書ファイルの末尾に追記する形で埋め込まれていた．そのため，ほとんどの文書ファイルに本論文で論じた 5 つの特徴のうち少なくとも 1 つが現れ，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．今回の実験では確認できなかったものの，理論的には実行ファイルを Stream に偽装して埋め込むという方式も考えられる．このように実行ファイルを Stream に偽装

して埋め込む悪性 MS 文書ファイルが現れた場合には，より詳細なファイル構造検査を検討する必要がある．

また，試験プログラムは OOXML（Office Open XML）[14]（docx，xlsx，pptx 拡張子）の MS 文書ファイルは検知対象としていない．OOXML の MS 文書ファイルは ZIP 形式で圧縮されており，マクロが埋め込まれた OOXML の文書ファイルか否か容易に区別できるだけでなく，Office の規定の設定では，マクロが有効な OOXML の MS 文書ファイルは作成されず，OOXML の MS 文書ファイルを開いた際にマクロは実行されない．さらに，OOXML の MS 文書ファイルに格納されている各ファイル相互の関連性を指定するリレーションシップという仕組みがあり，OOXML の MS 文書ファイルに埋め込まれた不正なファイルの検知が容易になっている．このような仕組みから，OOXML の悪性 MS 文書ファイルの作成は困難であり，2012 年までは OOXML の悪性 MS 文書ファイルを用いた標的型攻撃はほとんど発生していない．しかしながら，今後，OOXML の悪性 MS 文書ファイルを用いた標的型攻撃が増加した場合には，これへの対応についても検討する必要がある．

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⟩（参照 2013-05-08）.

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

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

[4] Stolfo, S.J., Wang, K. and Li, W.J.: Towards Stealthy Malware Detection, *Advances in Information Security*, Vol.27, pp.231–249 (2007).

[5] Li, W.J., Stolfo, S.J., Stavrou, A., Androulaki, E. and Keromytis, A.D.: A Study of Malcode-Bearing Documents, *Detection of Intrusions and Malware and Vulnerability Assessment* (*DIMVA*), Vol.4 of Lecture Notes in Computer Science, pp.231–250 (2007).

[6] 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).

[7] 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).

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

[9] 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, Proc. 2008 International Conference on Information Science and Security*, pp.141–146 (2008).

[10] Microsoft: Microsoft Office 向けの Microsoft Office ファイル検証機能の公開 (online)，入手先 ⟨http://technet.microsoft.com/ja-jp/security/advisory/2501584⟩（参照 2013-06-18）.

[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).

[13] トレンドマイクロ株式会社：2012 年国内における持続的標的型攻撃の分析，A Trend Micro White Paper (2013).

[14] ISO/IEC 29500: 2012: Information technology – Document description and processing languages – Office Open XML File Formats (2012).

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．電子情報通信学会，人工知能学会，各フェロー，IEEE ライフフェロー．