情報処理学会論文誌 Vol.55 No.2 1089–1099 (Feb. 2014)

推薦論文

悪性文書ファイルに埋め込まれたRATの検知手法

三村 守[1,a)] 大坪 雄平[2] 田中 英彦[1]

受付日 2013年6月3日, 採録日 2013年11月1日

概要：今日，機密情報や個人情報の搾取を目的とする標的型攻撃は，多くの組織にとって脅威である．標的型攻撃の初期段階では，攻撃者はRAT（Remote Access Trojan またはRemote Administration Tool）と呼ばれる実行ファイルをメールで送付し，コンピュータの遠隔操作を試みることが多い．近年ではRATが文書ファイルに埋め込まれることが多くなっており，検知はより困難となってきている．よって，標的型攻撃を防ぐためには，悪性文書ファイルに埋め込まれたRATを検知する必要がある．本論文では，RATがどのように悪性文書ファイルに埋め込まれているのかを調査し，その方式を体系化する．さらに，悪性文書ファイルへのRATの埋め込み方式を解読し，RATを検知する手法を提案するとともに，実験により提案手法の有効性を定量的に示す．

キーワード：標的型攻撃，RAT，マルウェア，悪性文書ファイル，静的解析

How to Detect an Embedded RAT in a Malicious Document File

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

Received: June 3, 2013, Accepted: November 1, 2013

Abstract: Today, targeted attacks that exploit confidential information or personal information are serious threats for many organizations. At an early phase of a targeted attack, most attackers send a executable file called RAT (Remote Access Trojan or Remote Administration Tool) by e-mails, and attempt to control the computer. Recently a document file in which embedded a RAT is increasing, and it is difficult to reveal it. Thus, to defeat targeted attacks, it is necessary to detect RATs in malicious document files. In this paper, we investigate how to embed a RAT in a malicious document file, and classify the methods. Moreover, we consider how to break the embedding methods and propose how to detect an embedded RAT in a malicious document file. The experimental results quantitatively show the validity and effectiveness of the methods.

Keywords: targeted attack, RAT, malware, malicious document file, static analysis

1. はじめに

近年，組織が保有する機密情報や個人情報の搾取を目的とするサイバー攻撃の脅威が顕在化している．2011年には国会，政府関係機関，民間企業等において大規模なサイバー攻撃が相次いで発生し，2012年には遠隔操作ウイルスにともなう誤認逮捕が発覚する等，大きな社会問題となっている．サイバー攻撃の中でも特に目立つのは，主に機密情報や個人情報の搾取を目的とし，ある組織や個人に標的を絞って実施される標的型攻撃である．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%に拡大している [1]．標的型攻撃の中でも，ある組織に特化した，時間および手法を問わずに継続的に行われる一連の攻撃はAPT（Advanced Persistent Threat）や新しいタイプの攻撃 [2] と呼ばれることもあり，大きな脅威となっている．

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

本論文の内容は2013年3月のコンピュータセキュリティ研究発表会にて報告され，同研究会主査により情報処理学会論文誌ジャーナルへの掲載が推薦された論文である．

典型的な標的型攻撃の概要を以下に示す．まず攻撃者は，業務等を装った件名やファイル名をつけたメールで不正プログラム（マルウェア）を送信する．そのメールを受信したユーザが，業務等を装った件名やファイル名を不審に思わず，添付ファイルを開封した場合，その端末で不正な命令が実行され，攻撃者の管理下に入る．攻撃者の管理下に入った端末は，遠隔操作によって踏み台とされ，不正な情報の搾取に利用される．搾取された情報は，コマンド&コントロールサーバ等に送信され，攻撃者によって回収される．攻撃者はまた，発信元を秘匿するために，Tor [4] 等の匿名性を向上させるソフトウェアを利用したり，すでに管理下にある遠隔操作が可能な端末を利用したりしてマルウェアを送信している可能性も考えられる．さらに近年では，不正に搾取した情報を用い，新たに業務等を装った件名，ファイル名，あるいは搾取したファイルそのものを付与した巧妙な手口も確認されている．標的型攻撃に用いられるマルウェアは特定の組織に特化したものも多く，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，検知できないケースがほとんどである．

標的型攻撃に用いられるマルウェアの本体は，RAT（Remote Access Trojan または Remote Administration Tool）と呼ばれる実行ファイル（EXE および DLL）である場合が多く，メールの添付ファイルで直接送付される場合と，メールのリンクや添付ファイルからネットワーク経由でダウンロードされる場合がある．メールの添付ファイルで直接送付される場合はさらに，ほかのファイルには埋め込まれずに拡張子やアイコンが偽装される場合と，DOC 形式，PDF 形式等の文書ファイルに埋め込まれて偽装される場合が考えられる．実行ファイルがほかのファイルに埋め込まれていない場合には，利用者は拡張子やアイコンの偽装等に注意すれば不審なメールを見抜くことができる．しかしながら，実行ファイルが文書ファイルに埋め込まれて偽装された場合には，利用者は一般的な手法で不審なメールを見抜くのは困難である．実際に，実行ファイルと埋め込み対象となる文書ファイルを指定するだけで，容易にマルウェアを作成するツールの存在が確認されている [3]．実行ファイルが埋め込まれた悪性文書ファイルの一般的な構成を**図 1** に示す．文書ファイルを開くと，まず閲覧ソフトウェアの脆弱性を突くコードである Exploit が動作し，コンピュータの制御を奪って Shellcode を実行する．Shellcode は，文書ファイルに隠されたマルウェア本体である RAT（実行ファイル）をデコードし，ファイルに出力して実行する．通常の文書ファイルには，このように実行ファイルが偽装されて埋め込まれているとは考え難い．よって，文書ファイルに実行ファイルが埋め込まれていれば，マルウェアと判定して差し支えないものと考えられる．そのように考えると，文書ファイルに実行ファイルが埋め込まれているか否かを判定できれば，マルウェアを検知することができるものと考えられる．そこで本論文の目的を，文書ファイルに RAT が埋め込まれているか否かを自動的に判定することとする．

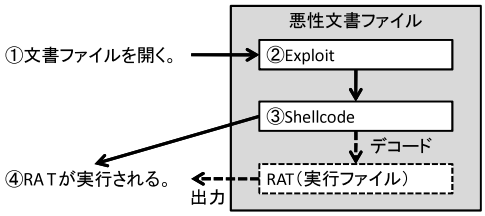

図 1 悪性文書ファイルの構成
Fig. 1 Structure of a malicious document file.

2. 関連研究

ファイルに埋め込まれたマルウェアを検知する手法としては，実行ファイルを対象としたものが提案されている．文献 [5] では，静的解析と動的解析を組み合わせ，実行ファイルに隠されたコードを自動的に展開する手法が提案されている．文献 [6] では，マルウェアを逆アセンブルして特定の機械語命令の類似性を検査することで，自身を書き換えるマルウェアを検知する手法が提案されている．文献 [7] では，難読化された実行ファイルの様々な統計データから難読化の方式を特定し，マルウェアの亜種を検知する手法を提案している．これらの手法は，実行ファイル形式のマルウェアを対象としている．しかしながら，文書ファイル形式のマルウェアの場合には，実行ファイルは何らかの方式でエンコードされて隠されることが多い．したがって，これらの手法をそのまま文書ファイル形式のマルウェアに適用することは困難である．

ほかに関連する研究としては，難読化されたマルウェアの通信内容を検知するための研究があげられる．文献 [8] では，仮想環境で機械語命令を実行し，XOR 等でエンコードされた不正なコードを検知する手法が提案されている．文献 [9] では，XOR 等でエンコードされたマルウェアの通信内容の特徴を学習することで，通信内容を動的に変化させるマルウェアを検知する手法が提案されている．これらの手法はマルウェアの通信内容を対象としたものであるが，文書ファイル形式のマルウェアを検知するために応用することも可能である．本論文で提案する手法は，機械語命令の実行を必要とせず，事前に学習を必要としない単一の文書ファイルを対象とした方式である．

本論文では，文書ファイルに RAT（実行ファイル）が埋め込まれているか否かを，脆弱性を突くコード（Exploit）や，RAT を展開・実行するコード（Shellcode）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文書ファイルを対象とする静的解析の一種といえる．文献 [10] 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．この手法では，ファイルに隠された脆弱性を突く不正なコードを検出することにより，マルウェアか否かの判定を試みている．文献 [11] では，DOC 形式等の文書ファイルを対象とし，主として RAT を展開・実行する際の典型的な機械語のパターンを検索することにより，マルウェアか否かを判定するツールが提案されている．本論文では，脆弱性を突くコードや RAT を展開・実行するコード等の不正なコードは検査せず，マルウェア本体である RAT を直接検出することによってマルウェアか否かの判定を試みる．文献 [12] では，DOC 形式等の文書ファイルの構造を検査することにより，文書ファイルの表示内容と関係がないデータを抽出するツールが提案されている．このツールで表示内容と関係がないデータが検出された文書ファイルは，不審である可能性が考えられるが，その文書ファイルがマルウェアであるか否かの判定は実施していない．また，このツールでは PDF 形式の文書ファイルを分析することができない．文献 [13] では，マルウェアが用いる潜在的に危険な活動をともなうフィルタ（データに適用する変換方法）を学習させることで，PDF 形式のファイルをマルウェアか否か判断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．また，PDF 形式のファイル以外の文書ファイルを分析することができない．本論文では，DOC 形式および PDF 形式を含む，様々な形式の文書ファイルを分析の対象としている．文献 [14] では，様々な形式の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，DOC 形式および PDF 形式を含む様々な文書ファイルを分析することが可能であり，学習するためのサンプルを集める必要もない．通常の文書ファイルには，実行ファイルが偽装されて埋め込まれているとは考え難いことから，このツールで文書ファイルから実行ファイルを抽出することができた場合には，その文書ファイルをマルウェアとして検知するという利用法が考えられる．しかしながら，このツールには，PDF 形式の悪性文書ファイルの検知率が低いという課題がある．本論文では，文書ファイルに埋め込まれた実行ファイルの検知率を向上させるため，実際に標的型攻撃に用いられた検体を分析し，RAT がどのように文書ファイルに埋め込まれているかを明らかにする．さらに，RAT の埋め込み方式を体系化して整理するとともに，効率的な解読手法を検討する．

3. RAT の埋め込み方式

実行ファイルである RAT は，そのまま文書ファイルに埋め込まれる場合もあるが，多くの場合には何らかの方式でエンコードされて文書ファイルに埋め込まれる．我々は，2009 年から 2012 年の間に複数の組織の標的型攻撃に用いられたマルウェアから，RAT を展開・実行するコードを抽出して逆アセンブラで解読した．その結果，判明したエンコード方式に用いられていた主な演算を**表 1** に示す．エンコード方式は，主としてこれらの基本的な演算を組み合わせて構成されており，換字による方式，転置による方式およびファイル形式固有の方式に分類することができる．換字による方式は，実行ファイルをエンコードする場合に必ず使用される基本的なエンコード方式である．転置による方式およびファイル形式固有の方式は，換字による方式に加えて使用される付加的なエンコード方式であり，複数の方式を組み合わせている場合もある．

表 1 エンコードに用いられる主な演算
Table 1 Main instructions to encode executable files.

演算	説明
XOR	排他的論理和
ADD, SUB	算術加算または算術減算
ROL, ROR	左論理シフトまたは右論理シフト

3.1 換字による方式

換字による方式は，何らかの方式で作成した鍵との排他的論理和（XOR）を計算することで，実行ファイルを別の文字列に変換する方式である．換字による方式では，元の実行ファイルのオフセット（先頭から所定の位置までのバイト数）はエンコード後にも変化しない．

3.1.1 XOR

実行ファイルと任意の 1 byte の鍵を XOR 演算するエンコード方式であり，最も基本的かつ頻繁に用いられている．通常の XOR による方式では，実行ファイルと鍵の XOR 演算を実施すると，実行ファイルの NULL の部分は鍵と同一の値となる．そのため，実行ファイルの NULL の部分に着目することで，容易にエンコードに用いる鍵を特定することが可能である．

3.1.2 Multibyte XOR

実行ファイルと任意の 2 byte 以上の鍵を XOR 演算するエンコード方式である．この方式もやはり，実行ファイルの NULL の領域に 2 byte 以上の鍵の値がそのまま表れる傾向がある．したがって，実行ファイルの NULL の部分に着目し，2 byte 以上の繰返し文字列を検出することで，エンコードに用いる鍵を推定することが可能である．ただし，繰返し文字列を検出することで鍵を特定するためには，実行ファイルに鍵の長さの 2 倍以上の NULL の領域が存在する必要がある．

3.1.3 NULL-Preserving XOR

NULL-Preserving XOR は，エンコードに用いる任意の 1 byte の鍵の値と一致する場合，または NULL の場合は演算を実施せず，それ以外の場合のみ XOR 演算を実施する

方式である．NULL-Preserving XOR では実行ファイルの NULL の部分は鍵の値とならず，NULL のままとなっているため，一見しただけでは鍵を特定することができない．したがって，鍵を特定するためには，すべての考えうる鍵を総当たりで試す必要がある．

3.1.4 Multibyte NULL-Preserving XOR

Multibyte NULL-Preserving XOR は，鍵と同じ長さの領域がすべて NULL の場合には演算を実施せず，それ以外の場合には任意の 2 byte 以上の鍵を XOR 演算するエンコード方式である．Multibyte NULL-Preserving XOR では，Multibyte XOR のように実行ファイルの NULL の領域に 2 byte 以上の鍵の値がそのまま表れることがない．したがって，2 byte 以上の繰返し文字列を検出することで，エンコードに用いる鍵を推定することができない．ただし，Multibyte NULL-Preserving XOR では鍵と同じ長さの領域に 1 byte でも NULL 以外の値が含まれている場合には XOR 演算を実施するため，文字コードの頻度分析によって鍵を推定できる可能性がある．

3.1.5 Rolling XOR

Rolling XOR は 1 byte 単位の鍵による XOR 演算と考えた場合，鍵の値がある周期で 1 byte ごとに変化するエンコード方式である．最も基本的な鍵の値の変化の方式は，1 ずつ加算するインクリメント方式および 1 ずつ減算するデクリメント方式である．これらの方式の場合，256 byte ごとに鍵の値が元の値となるため，その周期は 256 となる．2 周目以降は，1 周目と同じ 256 byte の値が鍵としてエンコードに用いられる．そのエンコードの結果は，256 byte の鍵を用いた Multibyte XOR と同じである．つまり，Rolling XOR を 2 byte 以上の鍵による XOR 演算と考えた場合には，Multibyte XOR と実質的に同義である．この場合，Rolling XOR の鍵の周期が Multibyte XOR の鍵の長さに相当する．したがって，Multibyte XOR と同様に実行ファイルの NULL の部分に着目し，2 byte 以上の繰返し文字列を検出することで，エンコードに用いる鍵を推定することが可能である．Rolling XOR や Multibyte XOR によるエンコード結果は，Rolling XOR と XOR による多重エンコードの結果と同義となる．なぜならば，Rolling XOR と XOR による多重エンコード演算には結合法則が成り立つため，結果として生成される鍵の長さは元の Rolling XOR の周期となるからである．よって多重にエンコード方式が用いられている場合にも，2 byte 以上の繰返し文字列を検出することで，エンコードに用いる鍵を推定できる可能性がある．

3.1.6 そのほかの方式

そのほかの換字によるエンコード方式としては，特定の周期のオフセットの場合にのみ演算を実施する方式や，独自のストリーム暗号を用いる方式等があげられる．また，少し変わった方式としては，ファイルの先頭から 1 byte ずつ順番に読み込み，次の 1 byte の値を鍵として XOR 演算を実施する方式も確認されている．

3.2 転置による方式

転置による方式は，何らかの方式で実行ファイルあるいは鍵の順序を並べ替える方式である．並べ替えの単位は，方式によって byte 単位の場合と bit 単位の場合がある．byte 単位の転置による方式では，元の実行ファイルのオフセットはエンコード後に変化する．

3.2.1 論理シフト

対象とする 1 byte の値を，1〜4 bit 単位で左論理シフトまたは右論理シフトする方式である．4 bit 左論理シフトした値と 4 bit 右論理シフトした値は同一となる．論理シフトは，実行ファイルに対して実施される場合と，Rolling XOR の鍵に対して実施される場合がある．周期が 256 の Rolling XOR の鍵に対して論理シフトが実施された場合にも，結果として生成される鍵の周期は変わらない．なぜならば，論理シフトの周期である 8 は，256 の約数となるからである．よって，Rolling XOR に論理シフトが併用された場合にも，2 byte 以上の繰返し文字列を検出することで，エンコードに用いる鍵を推定できる可能性がある．

3.2.2 SWAP

対象とする 2 byte の値を，1 byte 単位で順序を入れ換える方式である．SWAP 演算を実施した文字列に対し，オフセットを 1 byte ずらして再度 SWAP 演算を実施する Double SWAP 方式も確認されている．また，1 byte 単位ではなく，対象とする 1 byte を half byte 単位で順序を入れ換える Nibble SWAP 方式もある．なお，Nibble SWAP 方式は，4 bit 左論理シフトおよび 4 bit 右論理シフトと同義である．

3.3 ファイル形式固有の方式

ファイル形式固有の方式は，対象とする文書ファイルの形式に依存しており，特定のファイル形式のみで用いられているエンコード方式である．

3.3.1 RTF ファイル

ほとんどの RTF（Rich Text Format）形式のファイルにおいては，実行ファイルの 1 byte を 2 byte の 0〜f の 16 進数にエンコードする方式が用いられている．この方式は，他の方式と組み合わせて使用され，単独で用いられることはほとんどない．この方式でエンコードされた実行ファイルは，元の実行ファイルの長さの 2 倍の長さの 0〜f の 16 進数にエンコードされているため，容易に発見することが可能である．

3.3.2 PDF ファイル

PDF（Portable Document Format）形式のファイルにおいては，換字による方式および転置による方式で直接実行ファイルを埋め込む以外にも，Java Script を用いたエ

ンコード方式が確認されている．Java Script には，最終的に実行ファイルの 1 byte を 2 byte の 0〜f の 16 進数にエンコードして埋め込まれていた．ただし，PDF 形式の場合には，実行ファイルが埋め込まれた Java Script が，さらにこれまでに示した方式等でエンコードされることになるため，それらの方式も組み合わせて検索する必要がある．

3.3.3 そのほかの方式

これまでに示した方式以外にも，VBScript や，Flash に使用されるプログラム言語である Action Script を用いて実行ファイルをエンコードする方式が確認されている．これらの方式では，文書ファイル内で用いるプログラム言語に命令として実行ファイルと同等の内容を記述する．そのプログラム言語をサポートしている文書ファイルであれば，文書ファイルを開く際にプログラム言語で記述された命令が実行される．そのため，文書ファイルがサポートするプログラム言語を用いて，実行ファイルをエンコードすることが可能である．VBScript や Action Script を用いた方式は，Microsoft Word や Excel で用いられていることが確認されている．

4. 埋め込み方式の解読手法

悪性文書ファイルに埋め込まれた RAT を検知するためには，その埋め込み方式を解読する必要がある．RAT の埋め込み方式を解読する手法としては，手法 1：総当たり，手法 2：繰返し文字列の検出および，手法 3：文字コードの頻度分析が考えられる．

4.1 手法 1：総当たり

何らかの方式で作成した鍵と XOR を計算する換字による方式は，RAT を埋め込む場合に必ずといっていいほどつねに用いられる方式である．換字による方式の最も単純な解読手法は，すべての考えられうる鍵を総当たりで試す手法である．任意の方式である転置による方式やファイル形式固有の方式は，換字による方式の解読と並行して検索すれば効率が良い．総当たりによる手法は，XOR や NULL-Preserving XOR 等の鍵長が短い場合に有効である．特に，NULL-Preserving XOR は他の手法では解読できないため，唯一の解読手法であると考えられる．しかしながら，鍵長が長い Multibyte XOR や Multibyte NULL-Preserving XOR に対しては，現在のコンピュータでは実用的な時間内での解読は困難である．Rolling XOR については，単純なインクリメント方式やデクリメント方式であれば，実用的な時間内での解読は可能である．

4.2 手法 2：繰返し文字列の検出

鍵長が長いエンコード方式に対しては，総当たりでは実用的な時間内で解読することは困難である．そこで，実行ファイルの NULL の領域に着目し，2 byte 以上の繰返し文字列を検出することで，エンコードに用いる鍵を推定する手法が考えられる．この手法が機能するためには，実行ファイルに鍵の長さの 2 倍以上の NULL の領域が存在し，鍵がそのままの状態で悪性文書ファイルに含まれている必要がある．この手法は，XOR，Multibyte XOR，Rolling XOR および多重エンコードに対して有効であると考えられる．同様に，転置による方式やファイル形式固有の方式についても，並行して検索することで実用的な時間内での解読は可能であると考えられる．しかしながら，鍵がそのままの状態で悪性文書ファイルに含まれない NULL-Preserving XOR や Multibyte NULL-Preserving XOR に対しては，原理的に解読は不可能である．

4.3 手法 3：文字コードの頻度分析

鍵長が長い Multibyte NULL-Preserving XOR に対しては，総当たりでも繰返し文字列の検出でも実用的な時間内で解読することは困難である．Multibyte NULL-Preserving XOR では，鍵と同じ長さの領域に 1 byte でも NULL 以外の値が含まれている場合には XOR 演算を実施する．ゆえに，実行ファイルのある程度以上の NULL の領域が含まれていれば，鍵はそのままの状態では含まれていないが，文字コードの頻度分析によって鍵を推定できる可能性が考えられる．この手法は，XOR，Multibyte XOR および Rolling XOR に対してもある程度有効であると考えられる．同様に，転置による方式やファイル形式固有の方式についても，並行して検索することで実用的な時間内での解読は可能であると考えられる．しかしながら，ファイル形式固有で使用頻度が高い文字コードがある場合には，文字コードの頻度に偏りが生じてしまい，正しい鍵が推定できない可能性が考えられる．たとえば，RTF 形式や PDF 形式では，データに表示可能な文字コードが多く含まれるため，その影響を緩和する必要がある．

5. RAT の検知手法

本論文では，通常の文書ファイルには実行ファイルが偽装されて埋め込まれているとは考え難いことから，文書ファイルに埋め込まれている実行ファイルを RAT と判断する．悪性文書ファイルに埋め込まれた RAT を検知するためには，解読した埋め込み方式を用いてデコードしたデータから，実行ファイルが含まれていることを確認する必要がある．実行ファイルが含まれていることを確認する手法は，MS-DOS 用スタブプログラムを検索する方式と，実行ファイルのヘッダを検索する方式が考えられる．

5.1 MS-DOS 用スタブプログラムを検索する方式

RAT を検知するためには，実行ファイルの MS-DOS 用スタブプログラムに含まれる文字列を検索するのが基本である．MS-DOS スタブプログラムは，実行ファイルが

MS-DOS 上で実行された場合に実行されるプログラムであり，一般的には “This program cannot be run in DOS mode” や “This program must be run under Win32” という文字列を表示して終了するプログラムである．しかしながら，MS-DOS 用スタブプログラムに含まれる文字列は任意に設定することが可能であるため，この文字列のみで RAT を検知することはできない．我々が分析したマルウェアには，この方式による検知を回避するため，これらの文字列を 1 byte のみ変更した実行ファイルも含まれていた．また，“PE for Win32” や “Win32 only!” のようなまったく異なる文字列が含まれている場合や，あるいは空白となっている実行ファイルも確認された．

5.2 ヘッダを検索する方式

MS-DOS 用スタブプログラムを検索するよりも確実なのは，実行ファイルのヘッダを検索する方式である．実行ファイルの先頭には，MZ ヘッダおよび PE ヘッダがついているため，“MZ” や “PE” といったこれらのヘッダに含まれる固有のパターンを検索することで，実行ファイルである RAT を検知することができる．しかしながら，この方式による検知を回避するため，“MZ” や “PE” の文字列を削除した実行ファイルを埋め込んでいるマルウェアも確認された．“MZ” や “PE” の文字列は，Shellcode が実行された場合には復元されるため，実行ファイルは正常に動作する．このような場合には，MS-DOS 用スタブプログラムを検索する方式と合わせて総合的に検索する手法が有効である．実行ファイルの NULL が含まれるオフセット等も，RAT を検知するための有効な手がかりとなるものと考えられる．

6. 試験プログラムの実装

これまでに示した 3 つの RAT の埋め込み方式の解読手法および 2 つの RAT の検知方式を組み合わせた手法を，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．実装した試験プログラムの動作の概要を**図 2** 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，埋め込まれた RAT を検知するコマンドラインプログラムである．STEP1 では，RTF ファイル固有である 2 byte の 0〜f の 16 進数にエンコードする方式を解読するため，指定長以上の連続する 0〜f の 16 進数の文字列を抽出し，バイナリコードに変換する．次に，変換したバイナリコードから，提案する手法 1〜3 を用いて埋め込み方式を解読し，RAT の検知を試みる．STEP1 で RAT が検知できなかった場合には STEP2 に進む．STEP2 では，入力したファイル全体から，手法 1〜3 を用いて埋め込み方式を解読し，RAT の検知を試みる．ファイルの検索を終了するか，RAT を検知した場合にはマルウェアを検知したものと判定し，動作を終了する．各 STEP における埋め込み方式の解読（手法 1〜3）および RAT 検知のアルゴリズムは同一である．以下，その詳細について説明する．

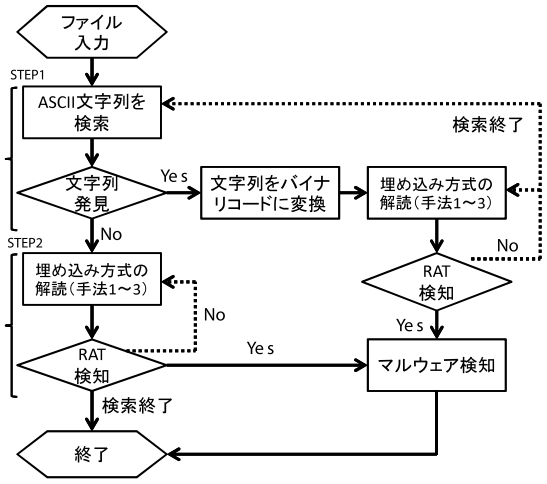

図 2 試験プログラムの動作の概要
Fig. 2 The algorithm of the build test program.

6.1 埋め込み方式の解読手法

試験プログラムは，以下に示す 3 つの手法で埋め込み方式の解読を試みる．実用性を考慮すると，いずれかの手法で RAT の埋め込み方式を解読した場合には，他の手法による解読は不要となる．しかしながら今回は，これらの 3 つの手法による解読の成功率を比較するため，最後まですべての手法により解読を試みるように実装した．

6.1.1 手法 1：総当たり

0x00〜0xFF までの 256 通りの 1 byte の鍵候補を作成し，各鍵候補によりエンコードした RAT の検知のための固有のパターンを検索する．各鍵候補を用いて検索するエンコード方式を以下に示す．

必須である換字による方式については，XOR，NULL-Preserving XOR および Rolling XOR の一部であるインクリメント方式およびデクリメント方式を検索する．Multi-byte XOR, NULL-Preserving XOR およびその他の Rolling XOR 方式については，実用的な時間内での解読は困難であるため検索しない．任意である転置による方式については，実行ファイルまたは鍵の論理シフト，SWAP および Double SWAP 方式を同時に検索する．また，PDF ファイル固有である 2 byte の 0〜f の 16 進数にエンコードする方式を同時に検索する．

6.1.2 手法 2：繰返し文字列の検出

総当たりによる鍵の検索では，多バイト長の鍵の検索に時間を要してしまい，RAT を短時間で検知することができない．したがってそれを緩和するために，文書ファイルから繰返し文字列を検出し，多バイト長の鍵の推定を試みる．以下，具体的な手順を示す．

まず，文書ファイルを 256 byte の長さで分割し，その各バイナリデータを順番に並べたリストを作成する．この

256 byte は，推定する鍵候補の長さである．次に，連続するリストのバイナリデータを比較し，一致するリストのバイナリデータを鍵候補として蓄積する．以後，分割する文書ファイルの長さ（鍵候補の長さ）を半分とし，その長さが 2 byte になるまで同手順を繰り返す．この手順により抽出された鍵候補は，手法 1 における鍵候補に換字による方式を組み合わせた出力結果に相当するものである．任意である転置による方式については，実行ファイルまたは鍵の論理シフト，SWAP および Double SWAP 方式を同時に検索する．また，PDF ファイル固有である 2 byte の 0〜f の 16 進数にエンコードする方式を同時に検索する．

6.1.3 手法 3：文字コードの頻度分析

鍵長が長い Multibyte NULL-Preserving XOR に対しては，総当たりでも繰返し文字列の検出でも実用的な時間内で解読することは困難である．そこで，文字コードの頻度分析によって鍵の推定を試みる．以下，具体的な手順を示す．

まず，文書ファイルの文字コードを，そのオフセットの 256 の剰余ごとに配分する．ここで，256 は推定する鍵候補の長さを示す．次に，256 通りの各オフセットの剰余ごとに文字コードを集計し，最も使用頻度が高い文字コードを求める．この際に，文書ファイル固有の使用頻度が高い文字コードの影響を緩和するため，表示可能な文字コードが多く含まれる領域を集計の対象から除外する．次に，各オフセットごとの最頻出文字コードを順に連結し，鍵候補とする．以後，鍵候補の長さを半分とし，その長さが 2 byte になるまで同手順を繰り返す．任意である転置による方式については，実行ファイルまたは鍵の論理シフト，SWAP および Double SWAP 方式を同時に検索する．また，PDF ファイル固有である 2 byte の 0〜f の 16 進数にエンコードする方式を同時に検索する．

6.1.4 そのほかの方式への対応

試験プログラムには，3 つの埋め込み方式の解読手法のほかに，ファイルの先頭から 1 byte ずつ順番に読み込み，次の 1 byte の値を鍵として XOR 演算を実施する方式の検索機能を実装した．

6.2 RAT の検知手法

各手法による埋め込み方式の解読の成否の判定は，以下に示す RAT の検知手法によって実施する．

まず，PE ヘッダのシグネチャおよび MS-DOS スタブプログラムの文字列の一部を検索する．各々の文字列の一部を発見した場合には，PE ヘッダのシグネチャまたは MS-DOS スタブプログラムの文字列のすべてを検索する．PE ヘッダのシグネチャまたは MS-DOS スタブプログラムの文字列が見つかった場合には，そのオフセットの直前の MZ ヘッダのシグネチャを検索する．発見した MZ ヘッダのシグネチャのオフセットが，PE ヘッダのシグネチャまたは MS-DOS スタブプログラムの文字列のオフセットから一定の範囲内であれば，実行ファイル（RAT）を検知したものと判定する．このオフセット間の距離の制約は，PE ヘッダのシグネチャまたは MS-DOS スタブプログラムの文字列と，MZ ヘッダのシグネチャが同一の RAT のものであることを確認するために設定している．なお，今回の実験では，MS-DOS スタブプログラムの文字列には “This program” を用いることとする．

7. 実験

提案する埋め込み方式の解読手法および RAT の検知手法の効果を確認するため，実装した試験プログラムを用いて実験を実施する．

7.1 実験内容

実験の対象となる文書ファイルの概要を**表 2** に示す．これらの文書ファイルは，複数の組織において採取した固有のハッシュ値を持つ検体であり，あらかじめ不審な挙動を示すマルウェアであることが確認されている．我々が逆アセンブラで解読した検体も，これらの文書ファイルに含まれている．検体に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 月とし，特定の拡張子のものを機械的に選定した．doc 拡張子の検体には，拡張子を偽装した RTF 形式の検体も含まれている．これらのマルウェアを試験プログラムに入力し，RAT の検知率と，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の検知率を比較する．また，各埋め込み方式の解読手法ごとの成功率を比較する．実験を実施する環境は**表 3** に示すとおりである．

7.2 実験結果

7.2.1 検知率の比較

検体の拡張子ごとの RAT の検知率を**表 4** に示す．354

表 2 検体の概要
Table 2 A summary of the specimens.

拡張子	検体数	平均容量（KB）
doc（rtf）	135	262.7
xls	49	180.4
pdf	170	351.2
合計	354	293.8

表 3 実験環境
Table 3 An experimental environment.

CPU	Core2 Duo 2.4 GHz
Memory	DDR2 SDRAM 3.0 GB
OS	Windows XP SP3
Interpreter	Python 2.7.3

表 4 RAT の検知率
Table 4 Detection rates of RAT.

拡張子	成功数	成功率（%）
doc（rtf）	131	98.5
xls	45	91.8
pdf	160	98.2
合計	336	97.4

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

	検知数	検知率（%）
試験プログラム	336	94.9
T 社 AV	57	16.1
S 社 AV	69	19.5
M 社 AV	47	13.3
T, S, M 社 AV	128	36.2

体の検体のうち，実行ファイル（RAT）が含まれていた検体は 345 体であり，このうちの 97.4%にあたる 336 体の検体を検知することができた．さらに，検知した検体には RAT が含まれており，実行ファイルとして動作することを確認した．この結果から，実行ファイルの判定は正しく動作したものと判断できる．RAT が含まれていなかった 9 体の検体は，Shellcode が RAT をネットワーク経由でダウンロードするタイプの検体であった．

次に，ウイルス対策ソフトとの検知率の比較結果を**表 5** に示す．試験プログラムの RAT の検知数をマルウェアの検知数と見なすと，試験プログラムで検知した 336 体の検体は，354 体の検体のうちの 94.4%にあたる．これに対し，その検体を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は，13.3%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．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 4 割未満にとどまった．

7.2.2 エンコード方式ごとの検知率

エンコード方式ごとの各手法の検知率を**表 6** に示す．RAT の検知に成功した 336 体の検体のうち，304 体の検体を本論文で体系化した主要なエンコード方式に分類することができた．RAT の検知には成功したが，RAT がそのまま埋め込まれていたり，体系化されないそのほかの方式でエンコードされていた 32 体の検体は表 6 から除外している．全体としては，手法 2 の成功率が 85.5%で最も高く，次いで手法 1 の 83.9%，手法 3 の 73.0%という結果となった．各手法に着目すると，手法 1 では，XOR，NULL-Preserving XOR，インクリメント方式およびデクリメント方式の Rolling XOR でエンコードされた検体は，100%検知することができた．鍵長が長い Multibyte XOR，Multibyte NULL-Preserving XOR，インクリメント方式およびデクリメント方式以外の Rolling XOR でエンコードされた検体については，まったく検知することができなかった．手法 2 では，XOR，Multibyte XOR，Rolling XOR でエンコードされた検体は，100%に近い確率で検知することができた．鍵のパターンがバイナリデータに表れない NULL-Preserving XOR および Multibyte NULL-Preserving XOR でエンコードされた検体については，まったく検知することができなかった．手法 3 では，Multibyte NULL-Preserving XOR でエンコードされた検体は，100%検知することができた．XOR，Multibyte XOR，Rolling XOR でエンコードされた検体については，51.9%から 84.4%と検知率にばらつきがでる結果となった．NULL-Preserving XOR でエンコードされた検体については，まったく検知することができなかった．

表 6 エンコード方式ごとの検知率
Table 6 Detection rates that correspond to each encoding method.

エンコード方式	検体数	手法 1	手法 2	手法 3
	150	150	150	124
XOR		100%	100%	82.7%
Multibyte	7	0	7	4
XOR		0%	100%	57.1%
NULL-Preserving	28	28	0	0
XOR		100%	0%	0%
Multibyte NULL-	15	0	0	15
Preserving XOR		0%	0%	100%
Rolling XOR	77	77	76	65
（Inc or Dec）		100%	98.7%	84.4%
Rolling XOR	27	0	27	14
（Other）		0%	100%	51.9%
	304	266	260	222
計		83.9%	85.5%	73.0%

8. 考察

試験プログラムを用いた実験の結果をふまえ，RAT の検知に失敗した原因について考察する．さらに，エンコード方式ごとの検知率から，効率的な RAT のエンコード方式の解読手法を導出し，最後に提案手法の効果と限界について考察する．

8.1 検知に失敗した原因

まず，試験プログラムが RAT の検知に失敗した原因について考察する．

試験プログラムでは，354 体の検体のうち，RAT が含まれていなかった 9 体の悪性文書ファイルを検知することができなかった．これらの悪性文書ファイルは，Shellcode がネットワーク経由で RAT をダウンロードするタイプのマルウェアであった．RAT を悪性文書ファイルに埋め込まず，ネットワーク経由でダウンロードするマルウェアに対

しては，OfficeMalScanner [11] のように Shellcode を検知する手法が有効である．

RAT が含まれていた 345 体の検体うち，9 体の悪性文書ファイルの RAT を検知することができなかった．これらの悪性文書ファイルには，VBScript，Action Script，独自のストリーム暗号等を用いて RAT がエンコードされていた．VBScript，Action Script によるエンコード方式に対しては，RAT の検知に用いるシグネチャを追加することで，対応することは可能である．しかしながら，独自のストリーム暗号を用いた方式に対しては，Shellcode から初期化ベクトル（同じ暗号鍵で異なるストリームを生成するために用いるビット列）を特定し，キーストリームを生成してエンコードする必要があるため，対応は困難である．このような複雑なエンコード方式に対しては，固有のシグネチャを作成して対応せざるをえないものと考える．

8.2 効率的な解読手法

エンコード方式ごとの検知率から，効率的な RAT のエンコード方式の解読手法を導出する．

総当たりは，鍵長が 1 byte の場合には，確実にエンコード方式を解読できることが確認できた．特に，NULL-Preserving XOR によりエンコードされた RAT を検知するためには，総当たりしかないことが確認できた．しかしながら，2 byte 以上の場合には，実用的な時間内での解読は困難である．

繰返し文字列の検出は，最も多くのエンコード方式に対して有効であり，ほぼ確実に解読できることが確認できた．しかしながら，NULL-Preserving XOR のように鍵のパターンがバイナリデータに表れない場合には，まったく解読できないことが確認できた．この結果から，ほとんどの RAT には，鍵長より長い NULL の領域が含まれているものと考えられる．解読に失敗した悪性文書ファイルには，鍵長より長い NULL の領域が含まれていなかったため，鍵が完全な形で含まれていなかったものと考えられる．

文字コードの頻度分析は，多くのエンコード方式に対してある程度有効であるが，解読の成否にばらつきがあることが確認できた．この原因は，文字コードの頻度にファイル形式固有の偏りがあったため，正しい鍵が推定できなかったためであると考えられる．文書ファイルの表示内容や属性情報等はファイル形式固有の領域に記録されており，これらの領域も文字コードの頻度を分析するための集計の対象となる．これら領域は，利用者が文書ファイルの表示内容や属性情報等を編集することによって変化してしまう．ゆえに，ファイル形式固有の偏りを緩和するのは容易ではないものと考えられる．この結果から，文字コードの頻度分析は，RAT を検知するために単体で用いるべきではないものと考えられる．しかしながら，Multibyte NULL-Preserving XOR によりエンコードされた RAT を検知するためには，唯一の手法であることが確認できた．

表 7 試験プログラムが検知できるエンコード方式
Table 7 Encoding methods that the build test program can detect.

対応可能	対応不可能
XOR	VBScript
Multibyte XOR	Action Script
NULL-Preserving XOR	独自のストリーム暗号
Multibyte NULL-Preserving XOR	その他
Rolling XOR	
論理シフト，SWAP	
RTF, PDF 固有	

以上の考察から，図 2 中の手法 1〜3 の部分においては，主として繰返し文字列の検出を行い，補助的に総当たりで NULL-Preserving XOR を，文字コードの頻度分析で Multibyte NULL-Preserving XOR を解読することで，最も効率良く RAT を検知することができるものと考える．

8.3 提案手法の効果と限界

試験プログラムが対応することができる RAT のエンコード方式を**表 7** に示す．試験プログラムでは，論理シフトや SWAP を組み合わせた様々な XOR を基本とするエンコード方式や，RTF 形式および PDF 形式固有のエンコード方式に対応することが可能である．しかしながら，VBScript，Action Script，独自のストリーム暗号等を用いたエンコード方式や，その他の変則的なエンコード方式には対応することはできない．これらの方式に対応するためには，固有のシグネチャ等を導入して試験プログラムを改良する必要がある．

提案手法では，標的型攻撃に用いられるマルウェアのうち，RAT が埋め込まれた悪性文書ファイルを検知することを目的としている．提案手法では，実行ファイルそのものや RAT が埋め込まれていない悪性文書ファイルは検知することができない．しかしながら，今回検証した doc, xls および pdf 拡張子のファイルは，2012 年に発生した標的型攻撃のメールに添付された悪性文書ファイルのほとんどを占めている [15], [16]．また，2009 年から 2012 年の間に発生した標的型攻撃では，97.5%の悪性文書ファイルに RAT が埋め込まれていた．したがって，RAT の検知率を向上させることができれば，実行ファイルをそのまま用いたものを除く，多くの標的型攻撃を検知することが可能になるものと考えられる．

9. おわりに

本論文では，実行ファイルである RAT がどのように悪性文書ファイルに埋め込まれているのかを調査し，その方式を体系化して整理するとともに，各々の方式の特徴に

ついて考察した．そして，各々の特徴を考慮して悪性文書ファイルからのRATの埋め込み方式を解読し，RATを検知する手法を複数提案した．さらに，各々の提案手法を試験プログラムに実装し，実験によりその効果を定量的に評価した．最後に，その実験結果から，まず繰返し文字列を検出し，次に総当たりでNULL-Preserving XORを，文字コードの頻度分析でMultibyte NULL-Preserving XORを解読する手法を導出した．

今後の課題としては，鍵のパターンがバイナリデータに表れ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文書ファイルでは，ほとんどのRATに鍵長より長いNULLの領域が含まれていたため，高い確率でRATを検知することができた．しかしながら，最近では実行ファイルであるRATを圧縮したり，より複雑なエンコード方式を用いた悪性文書ファイルも確認されている．RATが圧縮された場合には，NULLの領域がほとんどなくなってしまうため，鍵のパターンがバイナリデータに表れにくくなってしまう．しかも，MS-DOS用スタブプログラムやヘッダのパターンも変化してしまうため，検知はより困難となる．今回実装した試験プログラムでは，このようにRATを圧縮したり，本研究で検討した方式とは異なるエンコード方式を用いたマルウェアは検知することができない．これらに対しては，その圧縮方式やエンコード方式に対応した解読手法を実装する必要がある．さらに，近年ではRATを動的に生成するような，より高度なマルウェアも出現しており，そのようなものへの対応は今後の課題である．また，バイナリデータに含まれる不完全な鍵のパターンから，完全な鍵を復元する手法についても検討する価値があるものと考える．メールサーバやネットワークセンサへの実装も今後の課題である．メールサーバやネットワークセンサでリアルタイムに悪性文書ファイルを検知することができれば，より強固な標的型攻撃に対する防御システムの構築に寄与するものと考える．

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 2013-08-26）．

[2] 情報処理推進機構：『新しいタイプの攻撃』に関するレポート—Stuxnet（スタックスネット）等の新しいサイバー攻撃手法の出現（オンライン），入手先 ⟨http://www.ipa.go.jp/about/technicalwatch/20101217.html⟩（参照 2013-08-26）．

[3] 情報処理推進機構：脆弱性を利用した新たなる脅威の監視・分析による調査（オンライン），入手先 ⟨http://www.ipa.go.jp/security/vuln/report/newthreat200907.html⟩（参照 2013-08-26）．

[4] Tor Project: Anonymity Online (online), available from ⟨https://www.torproject.org/⟩ (accessed 2013-08-26).

[5] Royal, P., Halpin, M., Dagon, D., Edmonds, R. and Lee, W.: PolyUnpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware, *Proc. 22nd Annual Computer Security Applications Conference*, pp.289–300 (2006).

[6] Leder, F., Steinbock, B. and Martini, P.: Classification and Detection of Metamorphic Malware Using Value Set Analysis, *Proc. 4th International Conference on Malicious and Unwanted Software*, pp.39–46 (2009).

[7] Jacob, G., Comparetti, P.M., Neugschwandtner, M., Kruegel, C. and Vigna, G.: A Static, Packer-agnostic Filter to Detect Similar Malware Samples, *Proc. 9th Conference on Detection of Intrusions and Malware & Vulnerability Assessment*, pp.102–122 (2012).

[8] Huang, H.L., Liu, T.J., Chen, K.H., Dow, C.R. and Wuu, L.C.: A Polymorphic Shellcode Detection Mechanism in the Network, *Proc. 2nd International Conference on Scalable Information Systems*, No.64 (2007).

[9] Rossow, C. and Dietrich, C.J.: PROVEX: Detecting Botnets with Encrypted Command and Control Channels, *Proc. 10th Conference on Detection of Intrusions and Malware & Vulnerability Assessment*, pp.21–40 (2013).

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

[11] Boldewin, F.: Analyzing MSOffice malware with OfficeMalScanner (online), available from ⟨http://www.reconstructer.org/papers/Analyzing%20MSOffice%20malware%20with%20OfficeMalScanner.zip⟩ (accessed 2013-08-26).

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

[13] 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).

[14] 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).

[15] 日本アイ・ビー・エム株式会社：2012年上半期 Tokyo SOC情報分析レポート，IBM - Japan（オンライン），入手先 ⟨http://www-935.ibm.com/services/jp/its/pdf/tokyo_soc_report2012_h1.pdf⟩（参照 2013-08-26）．

[16] 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（オンライン），入手先 ⟨http://www-935.ibm.com/services/jp/its/pdf/tokyo_soc_report2012_h2.pdf⟩（参照 2013-08-26）．

推薦文

本論文は，文書ファイルに埋め込まれたRATに関する実際の検体を多数分析して，その埋め込み方法を整理分類している．その分析結果に基づいて，文書ファイル埋め込み型RATの検知手法を実装し，数百の検体を用いて評価を行っている．今後の同研究分野の発展に有用であるため，推薦論文として推薦したい．

（コンピュータセキュリティ研究会主査　西垣正勝）

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．

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．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．同年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，ディ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．