L-022

multiple-ESA 法の IDS への適用可能性に関する検討
Study of a multiple-ESA method's applicability to IDS

岡　大貴 †
Daiki Oka

今泉　貴史 ‡†
Takashi Imaizumi

1. 序論

近年、ネットワーク上のコンピュータを対象に、データの改ざんや漏洩、遠隔操作などといった、不正アクセスが増えている。その対策の 1 つとして、IDS(Intrusion Detection System: 侵入検知システム) がある。IDS では、ネットワーク上のトラフィックを監視することで、外部の通信者からの不正アクセスなどの攻撃を検知する。攻撃の検知方法によって、既知の攻撃の検知を対象としたシグネチャ型と、未知の攻撃にもある程度対応できるアノマリ型とに分類される。

シグネチャ型によって検知する場合、準備として、既知の攻撃トラフィックに現れる特徴をパターンとして定義しておく。検知の際には、パターンがトラフィックに含まれるかを照合し、攻撃か否かを判定する。過去の攻撃時のデータを元にパターンを定義するため、検知の精度は高い。代わりに、既知の攻撃にしか対応できないため、常に攻撃者とのいたちごっこになるという問題もある。なお、シグネチャ型・アノマリ型ともに攻撃の定義を完璧にはできないため、検知率は 100%とはならず、誤検知が発生してしまう。

IDS が抱える問題として、パターン数の増大や、IDS の処理負荷の増大などがある。シグネチャ型では、既知の攻撃に対処するため、パターン数が日々増大し続けている。さらに、IDS の処理負荷が増大してしまう原因としては、ネットワークを流れるトラフィックの増大などが挙げられる。トラフィックが増えることにより、IDS での処理が間に合わなくなり、パケットの取りこぼしが発生してしまう。また、意図的に IDS へ高負荷をかけることで、パケットの取りこぼしを発生させ、攻撃の検知を見逃す確率を上げるといった、DoS(サービス妨害) 攻撃的な不正アクセスの手法もある。こういった攻撃に対処するためにも、照合速度の向上が望まれている。

こういった事情から新しいパターンマッチングアルゴリズムとして **multiple-ESA 法** [1] を開発された。これは IDS を対象とした高速な照合が可能な複数パターン照合用のアルゴリズムである。文字列よりも表現力の高いパターン表現:**拡張文字列**を使っている。しかし、正規表現よりは低い表現力のため照合誤りが発生するが、もともと IDS での利用を想定しているため multiple-ESA 法では多少の照合誤りを許容してする。

本研究では multiple-ESA 法の IDS への適用可能性を調査する。multiple-ESA 法では拡張文字列は、正規表現と比べて繰返しや選言の演算の扱いが異なっている。正規表現では選言・連接・繰返しが文字種・文字列の両方に作用させられる。それに対して拡張文字列では文字種に対しては正規表現と同様だが、文字列に対しては連接演算しか行うことが出来ない。したがって、シグネチャの正規表現パターンすべてを表現するのは難しいと考えられる。よって、本研究ではパターンの表現力という観点から適用可能性を探っていく。

†千葉大学大学院融合科学研究科
‡千葉大学統合情報センター

2.multiple-ESA 法

multiple-ESA 法は、IDS を対象に高速な照合が可能な複数パターン照合用のアルゴリズムである。

multiple-ESA 法以前のパターンマッチング・アルゴリズムとしては、文字列で定義された複数パターン照合用のアルゴリズム、正規表現パターン照合用のアルゴリズムがある。しかし、文字列パターンを使う場合は、パターンの表現能力が低いという問題がある。また、正規表現パターンを使う場合は照合のために有限オートマトンを作成するが、オートマトンを表現するために使用する記憶領域はパターンの数や長さにつれて大きなものとなってしまう。

multiple-ESA 法では、文字列よりも表現能力の高い拡張文字列というパターン表現が使える、複数パターン照合用のアルゴリズムを提案することを目的とする。このアルゴリズムは、IDS での利用を想定しているため、低確率で発生する照合誤りを許容することで要求を実現している。

2.1. 拡張文字列

拡張文字列とは、正規表現の上で定義される、 正規表現よりも制限された言語クラスである。これを使うと、文字列よりも多彩なパターン表現が可能になる。

拡張文字列は、以下に示す、1.〜6. のいずれかにあてはまる。それぞれ、1.〜3. は文字種の定義、 4.〜6. は文字種の量化についての演算である。

1. 定数文字
   $a \in \Sigma.\ L(a) = a$
2. ワイルドカード文字
   “.”、 $L(.) = \Sigma$
3. 文字クラス
   $\beta = [abc...] \subseteq \Sigma.\ L(\beta) = a \cup b \cup c \cup \cdots$

4. オプション文字
   $\beta? = (\beta|\epsilon)$. $L(\beta?) = L(\beta) \cup \{\epsilon\}$.
5. 0 個以上の非限定繰り返し
   β^*. $L(\beta^*) = L(\beta)^*$
6. 1 個以上の非限定繰り返し
   β^+. $L(\beta^+) = L(\beta)L(\beta)^*$

以下に、拡張文字列の文法定義として、BNF を示す。

```
ES ← ES EC | EC
EC ← CC | CC "+" | CC "?" | CC "*"
CC ← C | "." | "[" CL "]"
CL ← CL C | C
C ← a ∈ Σ
```

ES、 EC、 CC、 CL、 C は非終端記号を表す。a は終端記号である。また、量化演算子について、+は 1 回以上の繰り返し、 ?はオプション文字、 *は 0 回以上の繰り返しをそれぞれ表す。さらに、文字種について、[ab$\cdots$z] のような [] の中にアルファベットの列を並べたものを**文字クラス**、「.」を**ワイルドカード文字**と呼ぶ。文字種とは、文字列中の任意の位置の文字についてどのアルファベットが出現し得るかを定義したものであり、文字は 1 つ、 文字クラスは [] 内のアルファベット、 ワイルドカード文字は全アルファベットが候補となる。正規表現と違う点は、**繰り返し** (閉包)、 **選言** (集合和) の演算の扱いである。正規表現では、3 つの基底演算 (繰り返し・**連接**・選言) が、文字種・文字列の両方に作用させられる。それに対して拡張文字列では、文字種に対しては 3 つの規定演算全てが作用させられるものの、文字列に対しては連接演算を作用させられるだけである。拡張文字列を処理するためのアルゴリズムは、Navarro と Raffinot によって、Shift-And 法 [2] を拡張した **Extended-Shift-And 法** [3] などが提案されている。

2.2. アプローチ

multiple-ESA 法では、先行研究である Extended-Shift-And 法と、SOG-like 法 [4] を基礎としている。multiple-ESA 法は、Extended-Shift-And 法に対し、SOG-like 法を参考に変更を加えた。この手法は、拡張文字列[§]で定義された複数のパターンを同時照合して照合候補を絞り込むアルゴリズムである。パターンの厳密な照合を検査しないが、処理速度は高速である。

なお、パターン数が増えても、照合時に消費する領域の量は変化しない。代わりに、全てのパターンの情報を一括管理しているため、照合誤りの確率が増える。

multiple-ESA 法では、Extended-Shift-And 法と SOG-like 法に対して以下の変更を加えた。

- 各パターンごとに、パターン長に差がある。そのため、どこで、どの文字が末尾に来るかという情報を保持するために、ビットテーブル End を持たせる。
- オプション文字の位置情報をアルファベットごとに記録するために、ビットテーブル E を持たせる。照合処理の際、$Emask$ の代わりに使用する。

multiple-ESA 法による処理は Extended-Shift-And 法同様、大きく前処理と照合処理の 2 段階に分けられる。

2.3. 前処理

拡張文字列パターンの集合を $P = \{P_1P_2\cdots\}$ とする。

multiple-ESA 法では、パターン P に対する、2 種類の演算を定義する。

- Len(P): 各演算子 (?+*) を含み、 文字種を 1 文字とカウントしない場合のパターン長。
- CS(P): 演算器号を含まず、 文字種を 1 文字とカウントした場合のパターン長。

multiple-ESA 法ではそれぞれ、m = Len(P)、 L = CS(P) とする。パターン P_i が存在するとき、m_i は Len(P_i)、L_i は CS(P_i) を表す。

ここで、ビットテーブルにおいて、パターン P_i の $pos(0 \leq pos \leq L_i - 1)$ 番目の文字照合中に、文字 c を読んだ状況を、以下の数式で表す (例として、B の場合を示す)。

- $B[c]$ & $(1 \ll pos) \neq 0 \Rightarrow$ パターン P_i について、pos 番目の文字が文字 c である可能性がある
- $B[c]$ & $(1 \ll pos) = 0 \Rightarrow$ パターン P_i について、pos 番目の文字が文字 c とはなりえない

パターン P_i の pos 番目の文字照合中の状況は、次のようになる。

- $Emask$ & $(1 \ll pos) \neq 0 \Rightarrow$ パターン P_i について、pos 番目の文字はオプション文字である可能性がある
- $Emask$ & $(1 \ll pos) = 0 \Rightarrow$ パターン P_i について、pos 番目の文字はオプション文字とはなりえない

以下に、前処理の概要を書く。

1) $Emask$ の初期化 ($Emask := 0$)
2) 全ての拡張文字列パターン $P_i \in P$ を走査し、以下の情報を取得
   - m_i:Len(P_i)
   - L_i: CS(P_i)

[§]正確には、提案手法ではワイルドカードを含まない。ただし、文字クラスで同じものを表現することは可能である。

- B: パターン中の文字について、出現する可能性のある位置を記憶
- End: パターン中の末尾文字について、出現する可能性のある位置を記憶 (パターンが L_{max} 文字以上なら、L_{max} ビット目にフラグ (1) をセット)
- S: パターン中の繰り返し文字について、出現する可能性のある位置を記憶
- E: パターン中の文字について、一つ先以降の文字 (遷移先) にオプション文字が連続して出現する可能性のある場合、その範囲 (ϵ 遷移で移動できる状態の集合) を記憶
- $Emask$: パターンにおいて、オプション文字 (ϵ 遷移) が出現する可能性のある位置を記憶 (照合処理では使わない)

3) マスク $Emask$ を走査、以下の情報 (全て、L_{max} ビットのマスク) を取得

- $Ebeg$: オプション文字かもしれない文字について、ブロックが始まる直前の位置を記憶
- $Eend$: オプション文字かもしれない文字について、ブロックが終わる位置を記憶

2.4. 照合処理

multiple-ESA 法の照合処理の工程を次に示す。pos は現在照合中の入力テキストの位置を表し、$T[pos]$ は入力テキストの現在照合している文字を表す。

1. 実行時状態マスク D (L ビット) を、0 で初期化
2. $0 \leq pos \leq n$ (テキスト長) -1 の間、以下を繰り返す
   (a) $D_1 \leftarrow \{\ (D \ll 1) \mid 1\}\ \&\ B[T[pos]]$
   (b) $D_2 \leftarrow D\ \&\ S[T[pos]]$
   (c) $D \leftarrow D_1 \mid D_2$
   (d) $Df \leftarrow D \mid Eend$
   (e) $D \leftarrow D \mid \{\ E[T[pos]]\ \&\ (\sim (Df - Ebeg) \bigoplus Df)\ \}$
   (f) もし、$D\ \&\ End[T[pos]] \neq 0$ ならマッチング成功
3. これまでに成功していなければマッチング失敗

Extended Shift-And 法における照合処理と異なるのは、オプション文字の位置判定に E を利用することおよび終了判定と報告内容である。muiltiple-ESA 法では、状態マスク D と末尾文字用ビットテーブル End との論理積の値によって、照合の完了を確認する。その後、どのパターンがテキストのどこで出現したかを識別するための情報として、それぞれ、パターンの末尾文字テキスト中における出現位置を報告する。

例として、パターン P_a = “PANI” とパターン P_b=“[IY]N?Y*NI?P” を誤り含みで照合する器械のモデルを、図 1 に示す。この照合器械モデルの動作を、multiple-ESA 法の照合では、前処理で準備したビットテーブル・ビットマスクをもとにシミュレートしている。

3. 調査

3.1. 概要

multiple-ESA 法の IDS への適用可能性を調査する。

本調査では調査対象に代表的な IDS である snort[5] を用いる。第 2 章で述べたように multiple-ESA 法では拡張文字列を利用しているが Snort のシグネチャは正規表現で定義されている。よって、拡張文字列でどの程度シグネチャを表現できるかを調査する。

また、同時に表現できない原因となる文字を調査する。

3.2. 調査対象

本調査の対象とするのは「Snortrules-snapshot-2946」とする。拡張子 rules ファイルに含まれるすべてのシグネチャを調査する。

3.2.1.snort

snort の簡単なシグネチャの例を図 2 に示す。(app-detect.rules より引用)

Snort のシグネチャは、ルールヘッダとルールボディの二つから構成される。例では 1 行目がルールヘッダ、“()”に囲まれた 2〜6 行目がルールボディである。ルールヘッダのフォーマットは以下に示すとおりである。

[ルールアクション] [プロトコル] [送信元 IP アドレス] [送信元ポート番号] [方向指示子] [送信先 IP アドレス] [送信先ポート番号]

ルールヘッダでは以下の条件を記述する。ルールヘッダで記述できる条件はパケットのヘッダ情報に一部に過ぎず、より詳細な条件はルールボディで指定する必要がある。

- ルールアクション
  パケットがルールに合致したときに行う処理。
  例では alert を指す。alert は管理者に通知を行い、パケットをログに記録することを意味する。

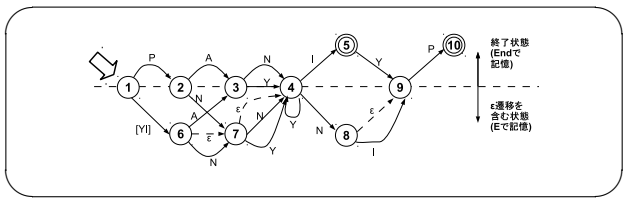

図 1: multiple-ESA 法の実行例で使用するパターン集合を照合する器械のモデル (照合誤りを含む)

```
# alert tcp $EXTERNAL_NET !22 -> $HOME_NET any
(msg:"APP-DETECT SSH server detected on non-standard port";
flow:to_client,established; content:"SSH-"; depth:4; nocase;
 pcre:"/^SSH-[12]¥.¥d+/smi"; reference:url,www.ietf.org/rfc/rfc4251.txt;
classtype:protocol-command-decode; sid:13586; rev:4;)
```

図 2: Snort のシグネチャの例

- 対象のプロトコル
  例では TCP を指す。TCP のほか、UDP や IP、ICMP も解析することが出来る。
- 送信元および送信先の IP アドレス
  $HOME_NET や$EXTERNAL_NET は、管理者が変数としてあらかじめ定義しておいたものをすべてのシグネチャで利用する。
- 方向指示子
  例では “->”を指す。通信の方向を指示している “->”の左側が送信元、右側が送信先を表す。方向指示子としては一方向を示す “->”に加え、双方向の通信を示す “<>”がある。
- 送信元および送信先のポート番号
  !は否定の意味であり、!22 は 22 番ポート以外を対象とする。 any はワイルドカードを指定するキーワードであり、0 番から 65535 番までのすべてのポートを対象とする。

ルールボディの記述は任意である。ルールボディはルールヘッダの情報に加えて、さらに詳細な条件を記述できるため、複雑な手法の不正アクセスを検知するうえで重要となる。以下に、例に示したシグネチャで使用されているオプションの機能を示す。

- msg
  アラートによって出力されるメッセージであり、何についての通知であるかを管理者に知らせる。ルールボディを記述する際にもっともよく用いられるオプションである。
- flow
  クライアント-サーバ間の通信の向きを指定する。また、TCP セッション確立の有無を指定することもできる。例では to_server オプションでサーバに向けて送られるパケットを指定し、さらに estabished オプションで確立された TCP セッションのパケットを対象とする。
- content
  パケットのペイロードを直接検査する。ASCII 文字に加えてバイナリデータを記述することもできる。content オプションで記述された文字列は単一の文字列として解釈され、パケットの検査に利用する。
- depth
  content オプションで指定された文字列を検索する範囲をバイト数で指定する。
- nocase
  content オプションに指定された文字列の大文字／小文字を無視するよう指定するものである。
- pcre
  Perl[6] 互換の正規表現を使用して記述することができるパターン部分である。
- reference
  外部に存在する情報への参照を記載する。
- classtype
  不正アクセスのタイプによってルールを分類し、管理者の素早い理解とイベントの適切な優先順位づけを促す。classtype オプションが提供するクラスは、その重要性に応じて優先度を示す値を持っている。
- sid
  各シグネチャを一意に識別するための ID。
- rev
  シグネチャのリビジョン番号を表す。シグネチャを改訂した場合などに、その記録を残すために与える。

オプションはこれ以外にもあるがパターンを用いるのは pcre のみであるため、今回の調査はオプション pcre のパターンに対して行うこととする。

3.2.2.pcre

pcre は Perl 互換の正規表現を使用して記述することができるパターン部分である。以下に pcre のフォーマットを示す。

```
pcre:[!]"(/<regex>/|m<delim><regex><delim>)
[ismxAEGRUBPHMCOIDKYS]";
```

!は否定の意味を表しており、パターンマッチが成功しなかった場合に alert などの処理を行う。pcre でパターンは以下に示すどちらかのような形で表される。説明のため今回の例でパターンは abcdefg とする。

- `/abcdefg/`
  /で囲まれた部分をパターンとする。
- `m<delim>abcdefg<delim>`
  `<delim>`とはデリミタのことで先頭と末尾の区切り文字を表す。 例えば `m{abcdefg}`といった形である。

`[,]` で囲まれた部分はパターンマッチングの際のマッチングオプションを示している。例えば大文字と小文字を区別しない、改行文字を含める、などがある。

3.3. 調査方法

本調査は 2 段階に分かれている。まず最初の段階として対象ルールセットよりパターンを切り出し、次の段階として取り出したパターンが拡張文字列で定義されているかを調査する。コンパイラは yacc/lex[7] を用いて作成した。

パターンの切り出しのためのプログラムの概要としては、pcre:”で始まり”; で終わるパターンを探すというプログラムである。通常 yacc/lex でのパターンマッチングでは最長一致であり pcre より後のオプションの”; をマッチしてしまうが、最初に出てくる”; までを出力することによってその問題を解決している。

次に拡張文字列で定義できるパターンを受理し、受理できないパターンの原因となる文字をカウントするプログラムを用いた。

プログラムでは (,), |, ^,$,\b,\B を受理しない文字として定めこれらの文字がパターンに現れた場合にそのパターンを不受理とし、現われた文字をカウントする。上記の文字が現れた場合でも文字の直前に\がついている場合や文字クラスの中身だった場合は受理される。

3.4. 調査結果

snort のシグネチャから重複を省いたパターンを抽出し、それを拡張文字列を受理するコンパイラに入力として与えた結果を示す。重複を省いた対象パターン数が 6297、コンパイラで受理したパターン数が 1452 となった。結果をまとめたものを以下の表 1 に示す。

4. 考察

4.1. 特殊文字の扱い

表 1 の結果をうけて不受理とした文字について対策を考える。

4.1.1.（、）、| の扱い

正規表現のパターンが意味的に拡張文字列で表せる場合がある。例えば、

$$(abc) \Longleftrightarrow abc, a(b|c)d \Longleftrightarrow a[bc]d$$

などのパターンである。今回の調査ではこのようなパターンは受理していないので、これを考慮することで何割かのパターンを受理することが可能になるだろう。

しかし、(、)、| を受理できないパターンも多く残ると考えられる。例えば、

(abc)+ , ab|cd

のようなものである。これらの正規表現と等価な拡張文字列は存在しない。

そこで正規表現を含有する形で、拡張文字列で表現することを考える。例えば

$$\begin{array}{ll} a(bc)+d & [abcd, abcbcd, abcbcbcd....] \\ \Downarrow & \\ abc[bc]*d & [abcd, abcbd, abccd, abcbcd....] \end{array}$$

のような形である。上記のように誤りを許すことによって正規表現を拡張文字列で表現することが可能になる。multiple-ESA 法のアプローチでも誤りを許しており、同じ方向性の false positive の処理であることからこのような処理も可能ではないかと考えられる。

4.1.2.^、 $の扱い

^は文字列の先頭にマッチ、$は文字列の末尾にマッチする特殊文字である。例えば^abc とあれば abc で始まる入力テキストにマッチする。また^はパターンの先頭に、$はパターンの末尾にのみ現われる。これらの対策としては multiple-ESA 法のアルゴリズムを改善することが挙げられる。

- ^をパターンに見つけた場合
  パターン長 L_i しか入力テキストを読み込まない。
- $をパターンに見つけた場合
  入力テキスト長 n からパターン長 L_i を引いた pos から読み込みを始める。

いずれの場合も入力テキスト長と読み込みを始める pos を調整することによって^と$は処理できるようになることになると考えられる。

4.1.3.\b、\B の扱い

\b は単語の区切りに、\B は単語の区切り以外にマッチする特殊文字である。ここで単語の区切りとは [0-9a-z_A-Z] とそれ以外の文字の境界のことを指す。文字とのマッチングではなくあくまで文字の境界でのマッチングである。そのため、拡張文字列で単語の区切りを表現することは出来ない。

表 1 を参照すればわかるように\b、\B の出現確率は 0.04%と非常に小さい確率であった。よって、\b、\B が出てくるパターンは無視することで誤りは発生するが IDS で利用可能となる。

4.2. 判定方法

本研究では拡張文字列の形をしているパターンのみを受理できるコンパイラを作成し調査を行った。しかし、この手法では拡張文字列で表すことが出来る正規表現、つまり 4.1.1 などで示されている例を見逃してしまう。

これを受理するために正規表現を受理するコンパイラを考える。コンパイラには意味的に拡張文字列で表

表 1: 調査結果

	受理	不受理							総計
		(	^	$	|	\b	\B	小計	
個数	1452	3409	1321	109	4	2	0	4845	6297
割合	23.1%	54.1%	21.0%	1.73%	0.0635%	0.0318%	0%	76.9%	100%

現できる正規表現を判別できる能力を持たせる。今回作成したコンパイラはほぼ構文規則だけで作られているが意味規則を利用することによって意味的な判別が可能になる。

例えば () で囲まれた部分や | の両側の単語の文字数をカウントしたり () のあとに特殊文字が続いているかどうかを確認したりすることによって判別可能になるものもあると考える。

また、正規表現で表されるパターンを含有するように、拡張文字列で表現することを考えても正規表現を受理するコンパイラを用いた方が調査を行いやすい。どの程度の変換を行うかを意味規則に与えればよい。例えば「$(ab)+$ などの場合に () 間の文字数が 2 文字以内であれば許容する」などとできる。実際の設定に関しては実装した場合に起こる誤り率などを考慮しながら行う必要がある。

5. 結論

5.1. まとめ

本研究では、multiple-ESA 法の IDS への適用可能性についてパターンの表現力という観点から調査した。調査結果としてパターンのうち 23%のみが拡張文字列であることがわかった。よってそのまま変換するのでは multiple-ESA 法の IDS への適用は不可能であると判断できる。

しかし、今回の調査によって適用できないと判断された 77%のうち 23%は multiple-ESA 法の改良で利用可能になると推測され、残りの 54%についても利用可能になる見込みはある。

IDS を用いた侵入検知はセキュリティの観点から非常に有用なものであり、今後も利用され続けていくと考える。それに伴いパターン数の増大や IDS の処理負荷の増大は更に進むと予測されるが、multiple-ESA 法を実装できればこれらの問題は解決できる。

5.2. 今後の課題

適用可能性を向上するために様々な課題がある。

- 意味的に拡張文字列で表現できる正規表現を判別できる能力の開発。
- 正規表現のパターンを含有する形への拡張文字列の変換、これに伴う適度な誤り率の設定なども考えなくてはならない。

また、速度や誤り率の評価のために、multiple-ESA 法の snort などの IDS へ向けた実装も行いたい。

参考文献

[1] 柳瀬葵, 今泉貴史. IDS での利用に適したパターンマッチアルゴリズム. 学術情報処理研究, Vol. 17, pp. 85–92, 2013.

[2] Karl Abrahamson. Generalized string matching. *SIAM J. Comput.*, Vol. 16, No. 6, pp. 1039–1051, 12 1987.

[3] Gonzalo Navarro and Mathieu Raffinot. *Flexible Pattern Matching in Strings: Practical On-Line Search Algorithms for Texts and Biological Sequences*. Cambridge University Press, New York, NY, USA, 1st edition, 2007.

[4] 今泉貴史, 水野恵祐. IDS に特化した文字列探索アルゴリズム. 情報処理学会研究報告. IOT, [インターネットと運用技術], Vol. 2009, No. 21, pp. 107–112, 2 2009.

[5] Snort. http://www.snort.org/.

[6] Larry Wall and Randal L. Schwartz. Perl プログラミング. ソフトバンク株式会社, 1993.

[7] 原田賢一. コンパイラ構成法. 共立出版株式会社, 1999.