行動表現文法 ELL を用いた攻撃シナリオ再構築技術

千田忠賢[1,a)]　鐘本楊[1]　青木一史[1]　三好潤[1]

概要：インシデント調査では，分析者が被害の発生した環境からネットワーク・端末レベルのログを収集・分析し，攻撃ベクトルや情報漏えいの有無など，攻撃者の一連の行動 (攻撃シナリオ) を再構築することが必要となる．しかし，既存手法では自動で攻撃シナリオを再構築できておらず，分析者は依然としてインシデント調査に多くの時間を費やしている．本論文では，インシデントが発生した際に収集したログから，攻撃シナリオを自動で再構築する手法を提案する．インシデントに関係するログを抽出する手法は既に多く提案されているが，攻撃シナリオを再構築するものはいまだに存在していなかった．提案手法では，攻撃者の行動を形式文法で定義する．形式文法で定義された行動を，収集したログから検出し行動間の関連付けを行った後，攻撃シナリオとして再構築する．評価では，実際に起きた攻撃を再現し収集したログを用い，既存手法と比較することにより提案手法の有効性を示す．

キーワード：デジタルフォレンジック，形式文法，ログデータ分析

A Formal Grammar-based Approach Toward Attack Scenario Reconstruction

Nariyoshi Chida[1,a)]　Yo Kanemoto[1]　Kazufumi Aoki[1]　Jun Miyoshi[1]

Abstract: In an incident investigation, the investigator collects network and endpoint logs and analyzes the logs to reconstruct the attack scenarios. Despite the advances made by techniques that make the investigation more efficient, the reconstruction is still largely a manual process. As a result, the investigation takes an enormous amount of time. In this paper, we present *ELL*, a formal grammar to define attack scenarios over logs, and show an algorithm to reconstruct the attack scenarios from the logs. Our evaluation with an actual ATP attack demonstrates that *ELL* has an expressiveness to define the attack scenarios over logs and the algorithm can reconstruct the attack scenarios from the logs.

Keywords: digital forensics, formal grammars, log data analysis

1. はじめに

標的型攻撃に見られるように，近年の攻撃は，標的について綿密に調査した上でソーシャルエンジニアリング（例えばスピアフィッシング）などの技術を利用して実施される[*1]．それらの技術では調査で得た情報をもとに標的を騙すことで攻撃を成功させるため，システム的なセキュリティを高めていても，インシデントが発生してしまう．そのため，インシデントを未然に防ぐことを目的とした従来の対処だけでは，インシデントによる被害を抑えることが困難になりつつある．この様な状況から，事後対処に重きを置いたインシデント調査への需要が高まっている．

インシデント調査は，ネットワーク・端末レベルのログを分析者が監視・調査することでインシデントの検知や攻撃者の一連の行動 (**攻撃シナリオ**) の明確化を行う．これにより端末内での攻撃活動を察知し，インシデントによる被害の最小化に向けた対応を実現する．

現在，インシデント調査におけるログの調査の大部分は人手で行われているが，調査対象のログ量は人手で網羅的

[1] NTT セキュアプラットフォーム研究所
NTT Secure Platform Laboratories
[a)] nariyoshi.chida.nx@hco.ntt.co.jp
[*1] https://github.com/kbandla/APTnotes

© 2018 Information Processing Society of Japan

に調査できる量を超えている[*2]．そこで，分析者は SIEM 製品や既存技術 [4][5][7][9][10] を用いてログの関連付けを行い，インシデントに関係のあるログのみを抽出し調査対象としている．

しかし，そのような対処をとっているにも関わらずインシデント調査には多くの時間を要する [9]．この理由として，抽出されたログには攻撃シナリオを明確にする上で必要なログがどこにあるのかを示す情報が含まれていないことが挙げられる．抽出されたログには調査項目への答えを明らかにする上では不要なログも含まれているため，分析者は調査の際に，自身の勘と経験を頼り必要なログを探さなければならず，調査に時間を要する．

本論文では，ログから攻撃シナリオを見つけ出す手法を提案する．より具体的には，攻撃シナリオのモデル化 (**攻撃モデル**) についての研究が既に多く存在することから [1][6]，それらのモデルに合致するログを抽出し，モデルと当てはめた上で攻撃シナリオとして再構築し分析者へ提示する手法を提案する．ログを攻撃モデルと当てはめた上で攻撃シナリオとして再構築し分析者へ提示することで，分析者は自身の勘と経験に頼らずとも攻撃シナリオを明確にする上で必要なログを見つけ出すことができる．提案手法における攻撃シナリオ再構築の具体的な例は第 2 章で示す．

本論文の貢献は次の通りである．

設計． 既存の形式文法を拡張し，ログから攻撃シナリオを定義するための文法 *ELL* を設計した．

実装． *ELL* で記述された攻撃シナリオをログから検出し，攻撃モデルとログを紐付けた上で攻撃シナリオとして再構築する手法を示した．また，実証用プログラムを開発した．

評価． 実証用プログラムを用いて評価実験を行ない，提案手法を用いて攻撃シナリオを再構築できていることを確認した．また，本論文で提案した手法が現実的な時間内で動作することを確かめた．

2. Motivating Example

本研究における問題意識とアイデアを明確にするため，例を示し説明する．

図 1 はインシデントの発生した環境から収集したログ (Windows Event Log[*3]) を簡略化し一部抜粋したものである．抜粋前のログ量 (エントリ数) は 150,038 行であった．インシデント調査では，分析者はこのようなログの中から攻撃シナリオを明らかにしていかなければならない．

攻撃シナリオを明らかにするため，既存研究ではインシデントと関係するログのみを見つけ出し分析者に提示する．しかし第 1 章で説明した通り，依然としてインシデント調査には多くの時間を要している．


```
<Data="InitialTime">45167873332</Data><Data="Status">3221225524</Data><Data="KeyName">LocalizedName</Data>
<Data="InitialTime">45167873339</Data><Data="Status">3221225524</Data><Data="KeyName">Icon</Data>
<Data="FileName">b.zip:Zone.Identifier</Data><Opcode>FileCreate</Opcode>
<Data="InitialTime">45167873347</Data><Data="Status">3221225524</Data><Data="KeyName">Security</Data>
<Data="InitialTime">45167873816</Data><Data="Status">3221225524</Data><Data="KeyName">StreamResource</Data>
<Data="FileName">a.doc.crdownload</Data><Opcode>FileCreate</Opcode>
<Data="PID">0x8F0</Data><Data="PPID">0x1F4</Data><Data="CMD">svchost.exe -k WerSvcGroup</Data>
<Data="PID">0xB3C</Data><Data="daddr">192.168.12.113</Data><Data="saddr">192.168.12.115</Data>
<Data="PID">0xA78</Data><Data="PPID">0x198</Data><Data="CMD">conhost.exe</Data>
<Data="KeyName">chrome.exe</Data>
<Data="InitialTime">48281713626</Data><Data="KeyName">B9E9E544C3EFFF5F12F246ADBCE1</Data><Opcode>Open</Opcode>
<Data="FileName">a.doc</Data><Opcode>FileCreate</Opcode>
<Data="InitialTime">45167873832</Data><Data="Status">3221225524</Data><Data="KeyName">StreamResourceType</Data>
<Data="FileName">a.doc:Zone.Identifier</Data><Opcode>FileCreate</Opcode>
<Data="InitialTime">45167873837</Data><Data="Status">3221225524</Data><Data="KeyName">LocalRedirectOnly</Data>
<Data="PID">0xC58</Data><Data="PPID">0x928</Data><Data="CMD">SearchFilterHost.exe 0</Data>
<Data="CommandLine">WINWORD.EXE /n a.doc</Data>
<Data="InitialTime">45167873843</Data><Data="Status">3221225524</Data><Data="KeyName">Roamable</Data>
<Data="PID">0x670</Data><Data="ImageFileName">EQNEDT32.EXE</Data><Opcode>Start</Opcode>
<Data="PID">0x520</Data><Data="daddr">192.168.12.111</Data><Data="saddr">192.168.12.112</Data>
<Data="CommandLine">chrome.exe --type= --field-trial-handle= --disable-gpu-compo --service-pipe-token=</Data>
<Data="InitialTime">45167873848</Data><Data="Status">3221225524</Data><Data="KeyName">PreCreate</Data>
<Data="PID">0xB3C</Data><Data="daddr">192.168.12.111</Data><Data="saddr">192.168.12.112</Data>
<Data="FileName">AppCrash_EQNEDT32.EXE</Data><Opcode>FileCreate</Opcode>
<Data="FileName">d.mp3:Zone.Identifier</Data><Opcode>FileCreate</Opcode>
<Data="PID">0x3B0</Data><Data="ProcessName">svchost.exe</Data>
<Data="InitialTime">45167873854</Data><Data="Status">3221225524</Data><Data="KeyName">Stream</Data>
<Data="PID">0xC9C</Data><Data="PPID">0x670</Data><Data="CMD">regsvr32 http://192.168.5.1/n.sct s.dll</Data>
<Data="KeyName">SYSTEM\CurrentControlSet\Services\Tcpip6\Parameters\Interfaces\{846EEEE-EEE-11EE-9D2E}</Data>
<Data="PID">0x3B0</Data><Data="ProcessName">svchost.exe</Data>
<Data="PID">0x544</Data><Data="PPID">0xC9C</Data><Data="CMD">powershell.exe Download(http://192.168.5.1))</Data>
<Data="FileName">c.EXE:Zone.Identifier</Data><Opcode>FileCreate</Opcode>
<Data="PID">0xD1C</Data><Data="PPID">0x544</Data><Data="CMD">powershell.exe -noni -nop</Data>
<Data="PID">0x3B0</Data><Data="ProcessName">svchost.exe</Data>
<Data="PID">0xD1C</Data><Data="daddr">192.168.5.1</Data><Data="saddr">10.0.2.15</Data>
<Data="PID">0x3B0</Data><Data="ProcessName">svchost.exe</Data>
<Data="PID">0x5E8</Data><Data="PPID">0x928</Data><Data="CMD">SearchProtocolHost.exe Global\FltPipeMssGthrPipe10</Data>
<Data="PID">0xD1C</Data><Data="daddr">192.168.5.1</Data><Data="saddr">10.0.2.15</Data>
<Data="InitialTime">48281713473</Data><Data="Status">0</Data><Data="KeyName"></Data>
<Data="InitialTime">0</Data><Data="KeyName">2A59603F598FFF2B6FB6716A3AF</Data><Opcode>KCBCreate</Opcode>
<Data="InitialTime">48281713626</Data><Data="KeyName">B9E9E544C3EFFF5F2F246ADBCE1</Data><Opcode>Open</Opcode>
```


図 1 収集したログ (一部抜粋)

そこで本研究では，攻撃モデルを用いてログから攻撃シナリオを見つけ出す．例えば，攻撃モデルの 1 つであるサイバーキルチェーンと当てはめて上記のログから攻撃シナリオを再構築すると図 2 に示す通りになる．

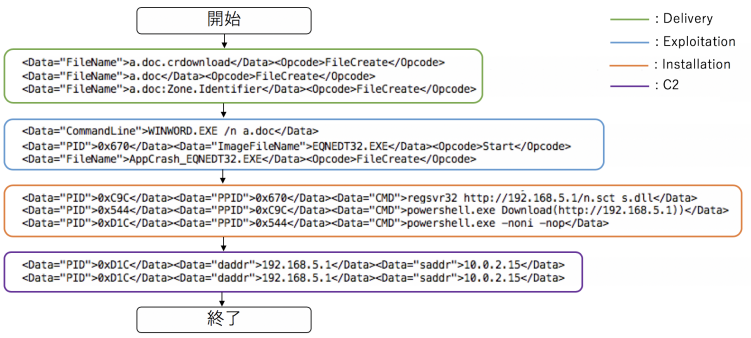


図 2 サイバーキルチェーンに当てはめて再構築された攻撃シナリオ

この手法を実現するには，次の 2 つの課題を解決する必要がある．

(1) ログで攻撃モデルを定義できなければならない

(2) 定義に従ってログから攻撃シナリオを再構築できなければならない

攻撃モデルを定義するものは，次の 3 つの要件を満たす必要があると考えられる．

1. 順序性の表現

攻撃モデルは順序性を持つ複数のフェーズから構成されることが多いため，そのような順序性を表現できること．

2. 曖昧性の表現

ログ中に攻撃モデルと合致する箇所が複数存在することが考えられるため，そのような曖昧性を表現できること．

3. 状態の和/積の表現

[*2] https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-dec-2016.pdf

[*3] https://msdn.microsoft.com/ja-jp/library/windows/desktop/aa385780(v=vs.85).aspx

© 2018 Information Processing Society of Japan

IOC などで用いられるように，特定の状態を表現するために複数の状態の和や積をとることがあるため，状態の和や積を表現できること．

これらを全て満たすものとして，形式文法 Parsing Expression Grammars with Unordered Choices(PEGwUC)[3] がある．ただ，PEGwUC に限らず形式文法はログを入力として扱うものではなく，また攻撃シナリオを定義する上で役立つ演算子を持たない．そこで，本論文では PEGwUC を拡張し，新たに行動表現文法 *ELL* を設計した．*ELL* の詳細は第 3 章で説明する．また，*ELL* で記述した攻撃シナリオをログから再構築する方法は第 4 章で説明する．

3. 行動表現文法 *ELL*

本章では行動表現文法 *ELL* の形式的な定義を述べる．*ELL* は PEGwUC を拡張した形式文法であり，入力がログである点，文法定義に変数が利用できる点，そして攻撃シナリオ再構築用の演算子を含む点が PEGwUC と異なる．また，それらの拡張に合わせて意味論や表現力も変わっている．まず，本論文中で用いる記号の記法や意味を第 3.1 節で述べる．それ以降で *ELL* の定義とその意味論を順に説明する．

3.1 準備

ログ．ログの集合は Σ で表す．a, b, c はそれぞれ Σ の 1 要素であり，1 つのログを表す．例えば，Windows の Event Log であれば，**<Event>**タグで囲まれる箇所が 1 つのログに該当し，Linux の Auditd であれば，auditd.log に記録される 1 行が 1 つのログに該当する．ログは表 1 に示されるような要素を持つ．要素は全て文字列として扱われ，ログ a が要素 v を持つとき，その要素の値は $a.v$ を用いて表される．例えば，a が要素 pid を持つとき，要素の値は $a.pid$ として表される．x，$y \in \Sigma^*$ はログの列を表す．

ELL ではログを図 3 に示す構文に従い記述する．ログ log は，$v =$ “re” または $v =$ “re” $\rightarrow t$ の繰り返しで表現される．v はログの要素であり，re は要素の値の定義である．re は，変数 t を含むことを除いては正規表現と同様に記述され，$v =$ “re” は v が re にマッチするものであることを意味する．また，$v =$ “re” $\rightarrow t$ は v が re にマッチし，なおかつ re がキャプチャした文字列を変数名 t で表される変数に代入することを意味する．v に対応する re を表す際に，$v.re$ を用いることがある．例えば，$v =$ “$(.^*)$” なら，$v.re = (.^*)$ である．

環境 E．*ELL* では，解析中にログの特定の要素の値を変数に代入し，その値をその後の解析で利用することが可能である．この機能を実現するため，*ELL* は環境 E を持つ．E は変数名とその値の組からなるリストであり，解析中に利用できる変数の状態を表す．例えば，$E = [(t,$10.0.2.15$)]$ とすると，環境には変数 t が値 10.0.2.15 を代入された状態で存在し，それが解析中に利用できることを意味する．

log	::=	ε	空文字
	|	$v =$ “re”	要素
	|	$v =$ “re” $\rightarrow t$	変数代入付き要素
	|	log , log	連接
re	::=	ε	空文字
	|	**a**	文字
	|	.	任意の 1 文字
	|	t	変数名
	|	$re\ re$	連接
	|	re^*	非貪欲な繰り返し
	|	$re\|re$	優先度無し選択
	|	(re)	キャプチャ

図 3 *ELL* におけるログの構文

表 1 ログに含まれる要素の一例

要素名	説明
pid	プロセス識別子
ppid	親プロセス識別子
cmd	コマンドライン
saddr	送信元アドレス
sport	送信元ポート番号
fname	ファイル名
tstamp	タイムスタンプ
regkeyname	レジストリーのキー名
imagefilename	実行したファイルの名前

E は **(1) 変数の追加**と **(2) 変数に代入された値の参照**に利用される．**(1) 変数の追加**では，*ELL* で定義された文法の解析中に変数へ値が代入されると，E の末尾に変数名とその値の組が追加される．例えば，$E = [(t_1,$10.0.2.15$)]$ とする．変数 t_2 に値\$HOME\Download\malware.doc が代入された場合，$E = [(t_1,$10.0.2.15$)]$ は $E' = [(t_1,$10.0.2.15$), (t_2,$\$HOME\Download\malware.doc$)]$ となる．環境への変数の追加を行う関数として ψ を定義する．ψ は *ELL* におけるログの記述 $[log]$ が含む変数代入付き要素の解析結果に合わせて環境を更新する関数である．ψ は環境 E，*ELL* におけるログの記述 $[log]$ と入力のログ a を受け取り，環境 E' を返す．$[log]$ が変数代入付き要素を含む場合，その変数が環境に追加される．例えば，$\psi([(t_1, 123)], [pid =$ “$(.*)$” $\rightarrow t_2], [pid = 234]) = [(t_1, 123), (t_2, 234)]$ となる．この例では，*ELL* におけるログの記述が $[pid =$ “$(.*)$” $\rightarrow t_2]$ であり変数 t_2 に 234 が代入されるため，それらの組 $(t_2, 234)$ が環境に追加される．

(2) 変数に代入された値の参照では，E から参照したい変数名を持つ組を見つけ，その値が返される．参照したい変数名が複数存在する場合は，末尾に近いものが選ばれる．E が変数 t を要素に持つとき，その値への参照は $E[t]$ として表される．例えば，$E =$

© 2018 Information Processing Society of Japan

$[(t_1$,10.0.2.15), (t_2,\$HOME\Download\malware.doc), (t_1, 192.168.0.2)] とする．このとき，$E[t_1]$ =192.168.0.2 となる．

マッチング関数 (matching function). *ELL* は，一般的な形式文法とは異なり，ログを入力とする．ログを扱うため，*ELL* で記述されたログと入力のログとのマッチング関数 ϕ を定義する．ϕ は環境 E，*ELL* におけるログの記述 $[log]$ と入力のログ a を受け取り，マッチングの結果を表すブール値 $true$ か $false$ を返す関数である．ϕ の定義を以下に示す．

$$\phi(E,[log],a) = \begin{cases} true \\ (\forall v \in V.\text{正規表現 } cv(v.re) \text{ が} \\ \text{文字列 } a.v \text{ にマッチする}) \\ \\ false(\text{その他}) \end{cases}$$

ここで，V は $[log]$ が含む要素の集合を表す．例えば，$[log]$ が $[pid =$ “1.∗”, $ppid =$ “2.∗”] であるならば，$V = \{pid, ppid\}$ である．関数 cv は，$v.re$ に含まれる変数をその値で置換し，正規表現に変換する関数である．cv の定義を図 4 に示す． ϕ 関数の使用例を示す．$\phi([(t, 123)], [pid =$ “t”$], [pid = 123]) = true$．この例では，t の値が 123 であるため，*ELL* におけるログの記述 $[pid =$ “t”] は $[pid =$ “123”] となり，入力のログ $[pid = 123]$ にマッチする．

$$\begin{aligned} cv(\varepsilon) &= \varepsilon \\ cv(\mathbf{a}) &= \mathbf{a} \\ cv(.) &= . \\ cv(t) &= E[t] \\ cv(re_1 re_2) &= cv(re_1)cv(re_2) \\ cv(re^*) &= cv(re)^* \\ cv(re_1 \mid re_2) &= cv(re_1) \mid cv(re_2) \\ cv((re)) &= (cv(re)) \end{aligned}$$

図 4　正規表現への変換関数 cv

3.2　*ELL* の定義

ELL の定義を定義 1 に示す．

定義 1　行動表現文法 *ELL* は 5 つ組 $G = (N, \Sigma, R, e_S, E)$ で定義される．ここで，N は非終端記号の有限集合，Σ はログの集合，R は構文規則の有限集合，e_S は開始表現，E は環境を表す．構文規則 $r \in R$ は非終端記号 A から表現 e への写像であり，$A = e$ として表される．非終端記号 A に対応する表現 e は $R(A)$ と記述する．

表現 e の構文を図 5 に示す．

ELL の持つ演算子のほとんどは，適用する対象が文字ではなくログであることを除けば，PEGwUC の演算子と

e	::=	ε	空文字
	|	$[log]$	終端記号（ログ）
	|	.	任意のログ
	|	A	非終端記号
	|	ee	連接
	|	e/e	優先度付き選択
	|	$e \mid e$	優先度無し選択
	|	$e*$	貪欲な繰り返し
	|	$e+$	貪欲な 1 回以上の繰り返し
	|	$e?$	オプション
	|	$\&e$	肯定先読み
	|	$!e$	否定先読み
	|	$\{\#Tag\ e\}$	タグ付きキャプチャ

図 5　表現の構文

同様である．つまり，ε は空文字にマッチする．A は表現 $R(A)$ を試みる．e_1e_2 は 2 つの表現の連接を表し，どちらかがマッチに失敗した場合は e_1 を呼び出した時の位置までバックトラックする．e_1/e_2 はまず e_1 を試し，失敗した場合に限り e_2 を試す．$e_1 \mid e_2$ は e_1 と e_2 をそれぞれ試す．$e*$ は e にマッチする限り繰り返し e を試す．$\&e$ は現在のログの位置から先が e にマッチするかどうかを試す．その際にログは消費しない．$!e$ は現在のログの位置から先が e にマッチしないかどうかを試す．その際にログは消費しない．

PEGwUC には存在しない演算子として，ログ $[log]$ とタグ付きキャプチャ $\{\#Tag\ e\}$ がある．$[log]$ は現在の位置のログが $[log]$ で表されるログにマッチするかどうかを試す．$\{\#Tag\ e\}$ は，解析時には e と同様に振る舞い，攻撃シナリオ復元時には e にマッチするログに Tag を付加する．ここで，Tag はタグ名を表す空でない文字列を表す．

3.3　糖衣構文

任意のログ . は Σ の全ての要素を優先度付き選択でつないだ表現 $(a/b/.../c)$ として扱うことができる．本節では，. と同様に，糖衣構文として扱うことのできる演算子を紹介する．

$$\begin{aligned} e* &= A, \text{ただし } A = eA/\varepsilon \\ e+ &= ee* \\ e? &= e/\varepsilon \end{aligned}$$

3.4　*ELL* の意味論

本節では，PEGwUC の意味論を拡張し，*ELL* の意味論を定義する．$\overset{ELL}{\rightsquigarrow}$ は *ELL* におけるマッチングの関係を示す．$G[e]$ は *ELL* G の開始表現を e で置き換えたものとする．$G[e, E]$ は *ELL* G の開始表現を e で，環境を E で置き換えたものとする．このとき，マッチングの関係 $G[e, E]\ x \overset{ELL}{\rightsquigarrow} \langle y, E' \rangle$ は，$G[e]$ の解析器が入力のログ x を環境 E において解析した結果，ログ y が残り，環境が

© 2018 Information Processing Society of Japan

E' に変化した，と読むことができる．解析に失敗した状態を表す際に fail を用いる．つまり，$G[e, E]\ x \overset{ELL}{\rightsquigarrow}$ fail は $G[e]$ の解析器が入力のログ x を環境 E において解析できないことを意味する．

ELL は優先度無し選択を持つため，解析結果が複数通りになることがある．そのような状態を表すため，我々は $\langle v_1, E_1\rangle \vee ... \vee \langle v_2, E_n\rangle$ という記法を用いる．例えば，$G[[pid = \text{“}(.^*)\text{”} \rightarrow v_1] \mid [pid = \text{“}(.^*)\text{”} \rightarrow v_1][pid = \text{“}(.^*)\text{”} \rightarrow v_2], []][pid = 1][pid = 2][pid = 3] \overset{ELL}{\rightsquigarrow} \langle [pid = 2][pid = 3], [(v_1, 1)]\rangle \vee \langle [pid = 3], [(v_1, 1), (v_2, 2)]\rangle$ となる．

解析結果には以下の関係が成り立つ．

$$\langle x, E_1\rangle \vee \langle y, E_2\rangle \equiv \langle y, E_2\rangle \vee \langle x, E_1\rangle \quad (1)$$

$$\text{fail} \vee \langle x, E_1\rangle \equiv \langle x, E_1\rangle \vee \text{fail} \quad (2)$$

$$\langle x, E_1\rangle \vee \langle y, E_2\rangle \equiv \langle x, E_1\rangle (x = y \text{ かつ } E_1 = E_2) \quad (3)$$

$$\text{fail} \vee \text{fail} \equiv \text{fail} \quad (4)$$

(1) と (2) は $\vee$ が可換であるであることを意味する．(3) と (4) は解析結果に含まれる重複を取り除くことを意味する．

簡単のため，任意の解析結果を X と記述する．つまり，X は次の 3 つの解析結果のうちいずれか 1 つを表すものとする．

- $\langle x_1, E_1\rangle \vee ... \vee \langle x_n, E_n\rangle$
- fail
- fail $\vee \langle x_1, E_1\rangle \vee ... \vee \langle x_n, E_n\rangle$

ELL の意味論を図 6 に示す．SPLIT は曖昧な解析結果を分割し，1 つずつ処理するための規則である．

4. 攻撃シナリオ再構築

本章では，入力として与えられたログから攻撃シナリオを再構築する手法を説明する．

手法の概要． 攻撃シナリオ再構築の処理手順を図 7 に示す．処理手順は 3 つの処理からなる．各処理の内容は次の通りである．

処理 *1*．ログパーサを用いてログから *ELL* で行動を記述する際に利用する要素 (例えば，表 1 に示される要素など) を抽出する．ログパーサは用いるログに合わせて既存のものを利用する．

処理 *2*．*ELL* パーサを用いて *ELL* で記述された行動定義ファイルから抽象構文木 (abstract syntax tree, AST) を生成する．AST の各頂点は図 5 に示される各表現と対応している．また AST の各頂点には，それぞれの頂点を一意に識別できるように異なる自然数の番号が割り当てられている．行動検出モジュールでは，AST と要素のみから成るログを用いて，行動定義ファイルに記述された攻撃シナリオがログに含まれるかどうか確認する．また，解析後に攻撃シナリオを再構築できるように，AST 上での移動経路を

$$\frac{}{G[\varepsilon, E]\ x \overset{ELL}{\rightsquigarrow} \langle x, E\rangle} \text{(EMPTY)}$$

$$\frac{\phi(E, [log], a) = true \qquad \psi(E, [log], a) = E'}{G[[log], E]\ ax \overset{ELL}{\rightsquigarrow} \langle x, E'\rangle} \text{(LOG.1)}$$

$$\frac{\phi(E, [log], a) = false}{G[[log], E]\ ax \overset{ELL}{\rightsquigarrow} \text{fail}} \text{(LOG.2)}$$

$$\frac{}{G[[log], E]\ \varepsilon \overset{ELL}{\rightsquigarrow} \text{fail}} \text{(LOG.3)}$$

$$\frac{R(A) = e \qquad G[e, E]\ x \overset{ELL}{\rightsquigarrow} X}{G[A, E]\ x \overset{ELL}{\rightsquigarrow} X} \text{(NT)}$$

$$\frac{G[e_1, E]\ x \overset{ELL}{\rightsquigarrow} \text{fail}}{G[e_1 e_2, E]\ x \overset{ELL}{\rightsquigarrow} \text{fail}} \text{(SEQ.1)}$$

$$\frac{G[e_1, E]\ x \overset{ELL}{\rightsquigarrow} \langle x_1, E_1\rangle \vee ... \vee \langle x_n, E_n\rangle \qquad G[e_2]\ \langle x_1, E_1\rangle \vee ... \vee \langle x_n, E_n\rangle \overset{ELL}{\rightsquigarrow} X}{G[e_1 e_2, E]\ x \overset{ELL}{\rightsquigarrow} X} \text{(SEQ.2)}$$

$$\frac{G[e_1, E]\ x \overset{ELL}{\rightsquigarrow} \text{fail} \vee \langle x_1, E_1\rangle \vee ... \vee \langle x_n, E_n\rangle \qquad G[e_2]\ \langle x_1, E_1\rangle \vee ... \vee \langle x_n, E_n\rangle \overset{ELL}{\rightsquigarrow} X}{G[e_1 e_2, E]\ x \overset{ELL}{\rightsquigarrow} \text{fail} \vee X} \text{(SEQ.3)}$$

$$\frac{G[e_1, E]\ x \overset{ELL}{\rightsquigarrow} \langle x_1, E_1\rangle \vee ... \vee \langle x_n, E_n\rangle}{G[e_1 / e_2, E]\ x \overset{ELL}{\rightsquigarrow} \langle x_1, E_1\rangle \vee ... \vee \langle x_n, E_n\rangle} \text{(ORDER.1)}$$

$$\frac{G[e_1, E]\ x \overset{ELL}{\rightsquigarrow} \text{fail} \qquad G[e_2, E]\ x \overset{ELL}{\rightsquigarrow} X}{G[e_1 / e_2, E]\ x \overset{ELL}{\rightsquigarrow} X} \text{(ORDER.2)}$$

$$\frac{G[e_1, E]\ x \overset{ELL}{\rightsquigarrow} \text{fail} \vee \langle x_1, E_1\rangle \vee ... \vee \langle x_n, E_n\rangle \qquad G[e_2, E]\ x \overset{ELL}{\rightsquigarrow} X}{G[e_1 / e_2, E]\ x \overset{ELL}{\rightsquigarrow} \langle x_1, E_1\rangle \vee ... \vee \langle x_n, E_n\rangle \vee X} \text{(ORDER.3)}$$

$$\frac{G[e_1, E]\ x \overset{ELL}{\rightsquigarrow} X \qquad G[e_2, E]\ x \overset{ELL}{\rightsquigarrow} Y}{G[e_1 \mid e_2, E]\ x \overset{ELL}{\rightsquigarrow} X \vee Y} \text{(UNORDER)}$$

$$\frac{G[e, E]\ x \overset{ELL}{\rightsquigarrow} \langle x_1, E_1\rangle \vee ... \vee \langle x_n, E_n\rangle}{G[\&e, E]\ x \overset{ELL}{\rightsquigarrow} \langle x, E\rangle} \text{(AND.1)}$$

$$\frac{G[e, E]\ x \overset{ELL}{\rightsquigarrow} \text{fail} \vee \langle x_1, E_1\rangle \vee ... \vee \langle x_n, E_n\rangle}{G[\&e, E]\ x \overset{ELL}{\rightsquigarrow} \text{fail} \vee \langle x, E\rangle} \text{(AND.2)}$$

$$\frac{G[e, E]\ x \overset{ELL}{\rightsquigarrow} \text{fail}}{G[\&e, E]\ x \overset{ELL}{\rightsquigarrow} \text{fail}} \text{(AND.3)}$$

$$\frac{G[e, E]\ x \overset{ELL}{\rightsquigarrow} \langle x_1, E_1\rangle \vee ... \vee \langle x_n, E_n\rangle}{G[!e, E]\ x \overset{ELL}{\rightsquigarrow} \text{fail}} \text{(NOT.1)}$$

$$\frac{G[e, E]\ x \overset{ELL}{\rightsquigarrow} \text{fail} \vee \langle x_1, E_1\rangle \vee ... \vee \langle x_n, E_n\rangle}{G[!e, E]\ x \overset{ELL}{\rightsquigarrow} \text{fail} \vee \langle x, E\rangle} \text{(NOT.2)}$$

$$\frac{G[e, E]\ x \overset{ELL}{\rightsquigarrow} \text{fail}}{G[!e, E]\ x \overset{ELL}{\rightsquigarrow} \langle x, E\rangle} \text{(NOT.3)}$$

$$\frac{G[e, E]\ x \overset{ELL}{\rightsquigarrow} X}{G[\{\#Tag\ e\}, E]\ x \overset{ELL}{\rightsquigarrow} X} \text{(CAP)}$$

$$\frac{G[e, E] \overset{ELL}{\rightsquigarrow} X}{G[e]\ \langle x, E\rangle \overset{ELL}{\rightsquigarrow} X} \text{(SPLIT.1)}$$

$$\frac{n \geq 2 \quad G[e, E_1]\ x_1 \overset{ELL}{\rightsquigarrow} X \quad G[e]\langle x_2, E_2\rangle \vee ... \vee \langle x_n, E_n\rangle \overset{ELL}{\rightsquigarrow} Y}{G[e]\ \langle x_1, E_1\rangle \vee ... \vee \langle x_n, E_n\rangle \overset{ELL}{\rightsquigarrow} X \vee Y} \text{(SPLIT.2)}$$

図 6 *ELL* の意味論

経路テーブルに記録しておく．行動検出モジュールでの処理内容については第 4.1 節で説明する．

処理 *3*．処理 *2* でログから攻撃シナリオが検出された場合

© 2018 Information Processing Society of Japan

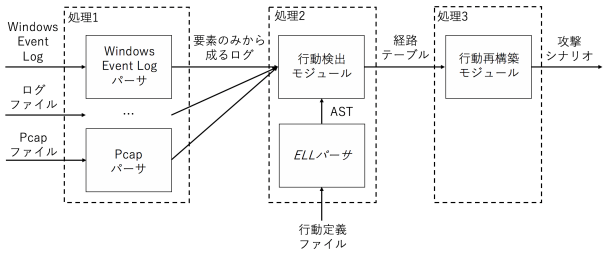


図 7　処理手順

は，経路テーブルから攻撃シナリオの再構築を行う．再構築についての詳細は第 4.2 節で説明する．攻撃シナリオが検出されなかった場合は処理を行わず終了する．

4.1　行動検出

行動検出モジュールでは，図 6 に示した意味論に従い AST 上を移動し，解析を行う．その過程で，攻撃シナリオ再構築のための AST 上での移動経路を経路テーブル T に記録する．本節では，AST の頂点での処理内容について説明するため，頂点が終端記号と対応するものであった場合を例に挙げて説明をする．

終端記号の解析の擬似コードを図 8 に示す．擬似コード

```
1  parse_log(ptr_cur, ID_pre, E)
2      ptr_pre = ptr_cur
3      if φ(E,[log],logs[ptr_cur]) = true
4          T[ID_pre][ptr_pre] ← (ID_cur,ptr_cur+1) #経路を記録
5          return {(ptr_cur+1, ψ(E,[log],logs[ptr_cur]))}
6      else
7          return {fail}
```

図 8　終端記号 $[log]$ の解析

において，処理 *1* から得られたログはリスト logs に格納されているものとし，i 番目のログは logs[i] で表す．また，擬似コード中の#はその行のそれ以降がコメントであることを表す．解析には parse_e の形式で表される関数が用いられる．関数は引数として，現在のリスト内の位置 ptr_{cur}，この関数を呼び出した頂点の番号 ID_{pre} と環境 E を持つ．終端記号が環境 E においてログ $\mathrm{logs}[\mathrm{ptr}_{cur}]$ とマッチするのであれば，経路テーブル T に移動経路を記録する．この場合，$(\mathrm{ID}_{pre}, \mathrm{ptr}_{pre})$ から $(\mathrm{ID}_{cur}, \mathrm{ptr}_{cur}{+}1)$ への遷移が経路として記録される．経路を記録した後は，環境 E を更新した上で次のログへと解析を進める．マッチしないのであれば，解析失敗を表す fail を返す．同様にして，他の頂点でも解析と経路の記録が行われる．

4.2　行動再構築

行動検出を終えると，解析に成功した logs 上の位置が集合として得られている．解析の最後に訪れる AST 上の頂点は開始表現 e_S に対応する頂点であるから，その頂点番号 ID_{last} と logs 上の最後の位置 ptr_{last} を用いて移動経路の復元が可能である．なぜなら，$\mathrm{T}[\mathrm{ID}_{last}][\mathrm{ptr}_{last}]$ には解析時における 1 つ前の状態が記録されているため，それを用いて再帰的に解析時における 1 つ前の状態を知ることができるからである．復元した経路のうち，タグ付きキャプチャに囲まれる区間がそのタグの表す行動により記録されたログである．また，どのタグ付きキャプチャにも囲まれていない区間は攻撃シナリオとは関係ないログである．復元された経路上からタグが付加されていない区間を取り除いたログが，提案手法により再構築された攻撃シナリオである．

5.　実装・評価

5.1　実装

我々は *ELL* の有効性を評価するため，第 4 章で説明した攻撃シナリオ再構築手法の実証用プログラムを作成した．実証用プログラムの全てのコンポーネントは Java で実装されており，LoC は約 6.4K である．

5.2　評価

評価では，実際に発生した攻撃を再現して収集したログに対して，*ELL* を用いて攻撃シナリオを記述し正しく攻撃シナリオを再構築できるか確認する．また，その結果を既存のログ関連付け手法を用いた場合の結果と比較することにより提案手法の有効性を確認する．それに加え，評価では本提案手法の解析時間を計測する．提案手法は解析時にメモ化を用いないため，解析には最悪の場合入力長に対して指数関数時間かかるが，本評価により，実用上は現実的な時間内で処理できることを確かめる．

まず，本評価で用いた攻撃シナリオを説明する．

攻撃シナリオ．本評価で用いる攻撃シナリオは Tropic Trooper と呼ばれている，台湾およびフィリピンの政府機関および企業を標的とした攻撃活動[*4]である．攻撃シナリオの概要を以下に示す．

(1) 標的は武器化された doc ファイル (msf.doc) が添付されたメールを受信
(2) 標的はメールに添付された msf.doc を実行
(3) エクスプロイトが実行され攻撃者が侵入
(4) 攻撃者はバックドアを設置
(5) 攻撃者は標的の端末内を探索し，撤退

msf.doc は CVE-2017-11882 を悪用するものであり，Metasploit[*5]で作成した．ペイロードには reverse_tcp を選択してある．ローカル環境でメールの送受信を行うため，メールサーバには MailHog[*6]を用いた．

[*4] https://blog.trendmicro.co.jp/archives/11465
[*5] https://www.metasploit.com/
[*6] https://github.com/mailhog/MailHog

© 2018 Information Processing Society of Japan

次に，評価を行なった環境と評価のため収集したログについて説明する．

実験環境と収集したログ．本実験の環境は次の通りである．

攻撃者環境：

- OS : Kali Linux
- CPU : Intel Core i7-7567U 3.50GHz
- RAM : 2GB RAM

標的環境：

- OS : Windows 7 Professional x64
- CPU : Intel Core i7-7567U 3.50GHz
- RAM : 4GB RAM

標的環境では，関連研究 [10] を参考に 4 種類のログを選定し，それらを収集した．3 つは端末内の情報を記録した Windows Event Log の Application，Security と NT Kernel Logger であり，1 つはネットワークの情報を記録した pcap ファイルである．収集された各ログの量を表 2 に示す．

表 2 ログ量 (エントリ数)

Applocation	Security	NT Kernel Logger	pcap(パケット数)
6	456	147,145	2,451

評価のため，攻撃シナリオに関係するログがどれなのかを予め人手で明らかにしておき，該当するログには正解を表すラベルを付加する．このラベルを用いて，手法により正しくログが抽出されているかどうかを判定する．この評価方法は，Pei らの評価方法 [10] を踏襲したものである．

最後に評価結果を示す．

評価結果．本評価では，サイバーキルチェーンをモデルに *ELL* で攻撃シナリオを定義し，評価を行なった．*ELL* による攻撃シナリオの定義の一部を図 9 に示す．図 9 はサイバーキルチェーンにおける Delivery のフェーズを *ELL* で定義したものであり，外部からファイルをダウンロードした際に，端末内にダウンロードされたファイルとその代替ストリームが新たに作成されたことを記録するログを検出するように定義している．同様にして他のフェーズについても定義した．

```
Delivery = FileDownload

FileDownload = { #Delivery [fname="(.*[\])+(.*)"->filename, opcode="FileCreate"] }
               (![fname=".*"+filename+"[:]Zone[.]Identifier", opcode="FileCreate"] .)*
               { #Delivery [fname=".*"+filename+"[:]Zone[.]Identifier", opcode="FileCreate"] }
```

図 9 *ELL* による Delivery の記述

この定義を用いて正しく攻撃シナリオを再構築できるか確認した結果を表 3 に示す．サイバーキルチェーンの各フェーズについて，関係するログを 1 つ以上検出できているものに ✓ がつけられている．

次に，再構築された攻撃シナリオと既存手法により再構築された攻撃シナリオを比較した結果を示す．既存手法には，HERCULE[10] を用いた．

表 3 行動毎の精度評価結果

Delivery	Exploitation	Installation	C2
✓	✓	✓	✓

HERCULE は攻撃に関係するログを検出するため，ログに含まれる実行ファイルを Bitblaze[*7] や VirusTotal[*8] でスキャンしマルウェアを含むかどうか判断する．また，ログに含まれる URL が既知の悪性 URL のブラックリストに含まれるかどうかも判断する．もしいずれかに該当するのであれば，そのログに異常を表すタグを付加する．そして，コミュニティ検出後に，異常を表すタグを含むコミュニティに属するログを攻撃者による行動としている．本評価における HERCULE では，マルウェアと攻撃者との通信に用いられた IP は全て検出されているものとし，該当するログには異常を表すタグを付加している．また，HERCULE によって付加される意味は正常か異常の 2 値のみであり各フェーズ毎の分析はできないため，提案手法により得られた結果も 2 値として扱う．

HERCULE と提案手法について，それぞれの手法により攻撃シナリオとして検出されたログ量，適合率と再現率を計測した．その結果を表 4 に示す．

表 4 評価結果

	ログ量 (エントリ数)	精度	
		適合率	再現率
HERCULE	5,508	0.002	0.647
提案手法	14	1	0.705

最後に，提案手法が攻撃シナリオの再構築に要した処理時間を表 5 に示す．構文解析時間は入力のログから *ELL* で記述された攻撃シナリオを検出するまでに要した時間である．これは，図 7 における行動検出モジュールの処理時間である．再構築時間は経路テーブルから攻撃シナリオを再構築するまでに要した時間である．これは，図 7 における行動再構築モジュールの処理時間である．

表 5 処理速度評価結果

構文解析時間 (秒)	再構築時間 (秒)
49.0	0.2

6. 議論

精度について．評価実験において，提案手法を用いた場合の再現率は 0.705 であった．検出できなかったログが存在した原因は行動再構築にある．それらのログは，先読み演算子内では見つけられていたものの，経路復元の際に先読み演算子内は復元されないことから攻撃シナリオとして残らなかった．そのため，先読み演算子内でもログをキャプ

*7 http://bitblaze.cs.berkeley.edu/
*8 https://www.virustotal.com/#/home/

© 2018 Information Processing Society of Japan

チャできるようにすれば全てのログが検出できるようになる．また，検出できなかったログが含むプロセス識別子やファイル名などは検出されたログには含まれていたため，それらをキーワードとして grep 等でログを検索することで，残りのログを全て見つけることも可能である．

ELL の解析に要する計算量． *ELL* の解析にはメモ化なしのナイーブなバックトラッキングを用いたため，最悪の場合入力長に対して指数関数時間かかる．しかし，メモ化なしの解析でも実用上は現実的な時間内で処理が可能であるという主張があるため [11]，我々は *ELL* の解析も実用上は現実的な時間内で処理ができかどうか確認を行なった．今後は解析速度向上のため，メモ化の適用や部分的な DFA 化による高速化 [2] などを用いることを検討している．

7. 関連研究

ログ関連付け技術． ログ関連付け技術の研究は 15 年程前から行われている．2003 年に King らは，ログを関連付けて依存グラフを構築し，インシデントと関係のあるログを明らかにする手法を BackTracker として提案した [7]．BackTracker は，ログが含むプロセスやファイルの情報を用いてログを関連付け依存グラフを構築する．しかし BackTracker は，サーバプロセスのように，複数のユーザからのクエリを 1 つのプロセスに集め処理するようなプロセスが走っていた場合には関係のないログが関連付き依存グラフの状態数が爆発するという問題点があった．2013 年に Lee らは，この問題点を指摘し，その解決策として BEEP を提案した [8]．BEEP は，サーバプロセスにクエリが入る瞬間と出てくる瞬間を監視し紐付けることで，インシデントと関係のないログが関連付かないようにした．2016 年に Pei らは，ログを関連付けるだけでなく，攻撃における行動毎にログをまとめる技術として HERCULE を提案した [10]．HERCULE は，事前に定めたルールに従いログを関連付けてグラフに変換した後，コミュニティ検出を行う．その結果同じコミュニティに属した頂点と対応するログを同一の行動によるものとした．HERCULE は，著者らの知る限り，ログから行動の再構築を試みた最初の研究である．しかし，HERCULE で定義できる行動は正常か異常の 2 値であり，それだけで攻撃シナリオを解明するのは困難であった．これらの研究は，インシデントと関係のあるログを抽出し分析者に提示することで攻撃シナリオの再構築を援助するものであり，攻撃シナリオの再構築は分析者が行う．本研究では，攻撃シナリオをログから再構築する技術を提案した．これにより，分析者は自信でログを精査しなくとも，*ELL* で攻撃シナリオを定義することで攻撃シナリオを解明できると考えられる．

8. おわりに

本論文では，ログで攻撃シナリオを表現できるようにするため，行動表現文法 *ELL* を設計した．また，*ELL* で記述した攻撃シナリオをログから検出・攻撃シナリオとして再構築する手法を示した．評価では，実際に発生した攻撃を再現して収集したログに対して提案手法を用いることで，正しく攻撃シナリオを再構築できることを確かめた．また，提案手法が実用上は現実的な時間内で処理できることを実験により確認した．

今後の課題としては，評価実験の拡充が挙げられる．また，特定の行動を記録したログからそれを受理するような *ELL* を自動生成する手法などがあれば，提案手法の実用性はより高まると考えられる．

参考文献

[1] Aldridge, J.: Remediating Targeted-threat Instrusions, Technical report, MANDIANT (2012).

[2] Chida, N. and Kuramitsu, K.: Linear Parsing Expression Grammars, *Language and Automata Theory and Applications - 11th International Conference, LATA 2017, Umeå, Sweden, March 6-9, 2017, Proceedings*, pp. 275–286 (2017).

[3] Chida, N. and Kuramitsu, K.: Parsing Expression Grammars with Unordered Choices, *Journal of Information Processing*, Vol. 25, pp. 975–982 (2017).

[4] Hassan, W. U., Bates, A. and Moyer, T.: Towards Scalable Cluster Auditing through Grammatical Inference over Provenance Graphs, *Network and Distributed System Security Symposium, NDSS 2018* (2018).

[5] Hossain, M. N., Milajerdi, S. M., Wang, J., Eshete, B., Gjomemo, R., Sekar, R., Stoller, S. and Venkatakrishnan, V.: SLEUTH: Real-time Attack Scenario Reconstruction from COTS Audit Data, *26th USENIX Security Symposium (USENIX Security 17)*, Vancouver, BC, pp. 487–504 (2017).

[6] Hutchins, E. M., Cloppert, M. J. and Amin, R. M.: Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains, *Leading Issues in Information Warfare & Security Research*, Vol. 1, p. 80 (2011).

[7] King, S. T. and Chen, P. M.: Backtracking Intrusions, *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, New York, NY, USA, ACM, pp. 223–236 (2003).

[8] Lee, K. H., Zhang, X. and Xu, D.: High Accuracy Attack Provenance via Binary-based Execution Partition., *NDSS*, The Internet Society (2013).

[9] Liu, Y., Zhang, M., Li, D., Jee, K., Li, Z., Wu, Z., Rhee, J. and Mittal, P.: Towards a Timely Causality Analysis for Enterprise Security, *Network and Distributed System Security Symposium, NDSS 2018* (2018).

[10] Pei, K., Gu, Z., Saltaformaggio, B., Ma, S., Wang, F., Zhang, Z., Si, L., Zhang, X. and Xu, D.: HERCULE: Attack Story Reconstruction via Community Discovery on Correlated Log Graph, *Proceedings of the 32Nd Annual Conference on Computer Security Applications*, ACSAC '16, New York, NY, USA, ACM, pp. 583–595 (2016).

[11] Redziejowski, R. R.: Parsing Expression Grammar As a Primitive Recursive-Descent Parser with Backtracking, *Fundam. Inf.*, Vol. 79, No. 3-4, pp. 513–524 (2007).

© 2018 Information Processing Society of Japan