情報処理学会論文誌 Vol.61 No.2 500–509 (Feb. 2020)

推薦論文

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]

受付日 2019年5月15日, 採録日 2019年11月7日

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，攻撃者の一連の行動である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すで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，攻撃者の行動を行動表現文法 *ELL* で定義する．*ELL* 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，実際のインシデント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]

Received: May 15, 2019, Accepted: November 7, 2019

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 incident 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うえでソーシャルエンジニアリング（たと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．このような状況から，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である**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

[1] NTT セキュアプラットフォーム研究所
NTT Secure Platform Laboratories, Musashino, Tokyo 180–8585, Japan
[a)] nariyoshi.chida.nx@hco.ntt.co.jp

[*1] https://github.com/kbandla/APTnotes
本論文の内容は 2018 年 10 月のコンピュータセキュリティシンポジウム 2018（CSS2018）で報告され，同プログラム委員長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．

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分析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], [8], [11], [13] 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かかわらずインシデント調査には多くの時間を要する [11]．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うえ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あげられる．抽出されたログには調査項目（たとえば，攻撃ベクトルの特定や情報漏えいの有無など）への答えを明らかにするうえ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再構築する手法を提案する．より具体的には，攻撃シナリオのモデル化（**攻撃モデル**）についての研究がすでに多く存在することから [1], [7]，それらのモデルに合致するログを抽出し，モデルとあてはめたうえ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あてはめたうえ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なくても攻撃シナリオを明確にするうえ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 2 章で示す．

本論文の貢献は次のとおりである．

設計． 既存の形式文法を拡張し，ログ上での行動定義を記述するための文法 *ELL* を設計した．

実装． *ELL* で記述された行動定義をログから検出し，攻撃モデルとログを紐付けたうえ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．

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 Windows Event ログ[*3]を簡略化し一部抜粋したものである．抜粋前のログ量（ログエントリ数）は 150,038 行であった．

[*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


```
<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">4828171347</Data><Data="Status">0</Data><Data="KeyName"></Data>
<Data="InitialTime">0</Data><Data="KeyName">2A59603F590FFF2B6FB6716A3AF</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）
Fig. 1 An excerpt of the audit logs.

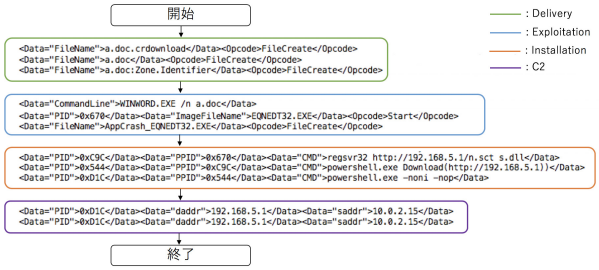


図 2 サイバーキルチェーンにあてはめて再構築された攻撃シナリオ
Fig. 2 An attack scenario reconstructed with the cyber kill chain.

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 1 章で説明したとおり，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．

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．たとえば，攻撃モデルの 1 つであるサイバーキルチェーン [7] とあ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** に示すとおりになる．

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) ログを入力とする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 4 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** Indicator of Compromise（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．
4. **文脈依存な構文の表現** 攻撃モデルに合致するログの中には一部文脈依存な構文が含まれる場合があるため，そのような文脈依存な構文を表現できること．

これらのうち 3 つを満たすものとして，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，攻撃シナリオを定義するうえで必要な演算子（たとえば，文脈依存を表現するための演算子など）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 3 章で説明する．また，*ELL* で記述した行動定義をログから再構築する方法は 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 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． ログエントリの集合は Σ で表す．a，b，c はそれぞれ Σ の 1 要素であり，ログエントリを表す．たとえば，Windows の Event Log であれば，**<Event>**タグで囲まれる箇所がログエントリに該当し，Linux の Auditd であれば，audit.log に記録される 1 行がログエントリに該当する．ログエントリは**表 1** に示されるような要素を持つ．要素はすべて文字列として扱われ，ログエントリ 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．たとえば，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（ログエントリの列）を表す．

表 1 ログエントリに含まれる要素の一例
Table 1 Example elements of log entries.

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

ELL ではログエントリを**図 3** に示す構文に従い記述する．ログエントリ log は，$v = re$ または $v = re \rightarrow t$ の繰返しで表現される．v はログエントリの要素であり，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る．$v = re$ 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 が最後に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．ここで，キャプチャは自動正規表現におけるキャプチャと v に対応する re を表す際に，$v.re$ を用いることがある．たとえば，$v =$ "(.*)" なら，$v.re = (.*)$ である．

環境 E． *ELL* では，解析中にログエントリ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 は変数とその変数に代入された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．たとえば，$E = [(t, 10.0.2.15)]$ 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．

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 の末尾に変数とその変数に代入された値の組が追加される．たとえば，$E = [(t_1, 10.0.2.15)]$ 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* におけるログエントリの記述 $[log]$ 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* におけるログエントリの記述

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	::=	"str"	文字列
	|	t	変数
	|	$re +$ "str"	連接（文字列）
	|	$re + t$	連接（変数）
str	::=	ε	空文字
	|	**a**	文字
	|	.	任意の 1 文字
	|	$re\ re$	連接
	|	re^*	繰り返し
	|	$re \mid re$	選択
	|	(re)	キャプチャ

図 3 *ELL* におけるログエントリの構文
Fig. 3 Syntax of a log entry on *ELL*.

$[log]$ と入力のログエントリ 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．たとえば，$\psi([(t_1, 123)], [pid = \text{"}(.*)\text{"} \rightarrow t_2], [pid = 234]) = [(t_1, 123), (t_2, 234)]$ となる．この例では，ELL におけるログエントリの記述が $[pid = \text{"}(.*)\text{"} \rightarrow t_2]$ 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．たとえば，$E = [(t_1, 10.0.2.15), (t_2, \$HOME\backslash Download\backslash malware.doc), (t_1, 192.168.0.2)]$ とする．このとき，$E[t_1] = 192.168.0.2$ となる．また，E が t を要素に持たないとき，$E[t]$ は E が t を要素に持たないことを表す特殊な文字 $\bullet$ を返す．

マッチング関数． ELL は，一般的な形式文法とは異なり，ログエントリを入力とする．ログエントリを扱うため，ELL で記述されたログエントリと入力のログエントリとのマッチング関数 ϕ を定義する．ϕ は環境 E，ELL におけるログエントリの記述 $[log]$ と入力のログエントリ 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．たとえば，$[log]$ が $[pid = \text{"}1.*\text{"}, ppid = \text{"}2.*\text{"}]$ 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．例 (1) $\phi([(t, 123)], [pid = t], [pid = 123]) = true$．この例では，$cv(t) = 123$ であるため，ELL におけるログエントリの記述 $[pid = t]$ は $[pid = 123]$ となり，入力のログエントリ $[pid = 123]$ にマッチする．例 (2) $\phi([(t_1, 123)], [pid = t_2], [pid = 123]) = false$．一方，この例では t_2 が E に存在しないため $cv(t_2) = \bullet$ となり，ELL におけるログエントリの記述 $[pid = t_2]$ は $[pid = \bullet]$ となる．そのため，入力のログエントリ $[pid = 123]$ にマッチしない．

$$\begin{aligned} cv(\text{"}str\text{"}) &= str \\ cv(t) &= E[t] \\ cv(re + \text{"}str\text{"}) &= cv(re)\ str \\ cv(re + t) &= cv(re)\ E[t] \end{aligned}$$

図 4 正規表現への変換関数 cv
Fig. 4 Definitions of the function 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 4 つ組 $G = (N, \Sigma, R, e_S)$ で定義される．ここで，N は非終端記号の有限集合，Σ はログエントリの集合，R は構文規則の有限集合，e_S は開始表現，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エントリであることを除けば，PEGwUC の演算子と同様である．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 を呼び出したとき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．$\&e$ は現在のログエントリの位置から先が 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．$!e$ は現在のログエントリの位置から先が 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，ログエントリ $[log]$ 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エントリが $[log]$ で表されるログエントリ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

任意のログエントリ $.$ は Σ のすべての要素を優先度付

e	::=	ε	空文字
	|	$[log]$	終端記号（ログエントリ）
	|	$.$	任意のログエントリ
	|	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
Fig. 5 Syntax of an expression.

き選択でつないだ表現 $(a/b/\ldots/c)$ 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．

$e* = A$, ただし $A = e\ A \,/\, \varepsilon$

$e+ = ee*$

$e? = e/\varepsilon$

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 で置き換えたものであり，環境は E であることを表す．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 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，$G[e, E]\ x \overset{ELL}{\rightsquigarrow} \mathsf{fail}$ 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 は優先度なし選択を持つため，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 $\langle x_1, E_1 \rangle \vee \ldots \vee \langle x_n, E_n \rangle$ という記法を用いる．たとえば，$G[[pid = \text{"(.*)"} \rightarrow t_1] \mid [pid = \text{"(.*)"} \rightarrow t_1][pid = \text{"(.*)"} \rightarrow t_2], []][pid = 1][pid = 2][pid = 3] \overset{ELL}{\rightsquigarrow} \langle [pid = 2][pid = 3], [(t_1, 1)] \rangle \vee \langle [pid = 3], [(t_1, 1), (t_2, 2)] \rangle$ となる．

解析結果には以下の関係が成り立つ．

$$\langle x, E_1 \rangle \vee \langle y, E_2 \rangle \equiv \langle y, E_2 \rangle \vee \langle x, E_1 \rangle \tag{1}$$

$$\mathsf{fail} \vee \langle x, E_1 \rangle \equiv \langle x, E_1 \rangle \vee \mathsf{fail} \tag{2}$$

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

$$\mathsf{fail} \vee \mathsf{fail} \equiv \mathsf{fail} \tag{4}$$

式 (1) と (2) は $\vee$ が可換であることを意味する．式 (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．

- $\langle x_1, E_1 \rangle \vee \ldots \vee \langle x_n, E_n \rangle$
- fail
- $\mathsf{fail} \vee \langle x_1, E_1 \rangle \vee \ldots \vee \langle x_n, E_n \rangle$

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．

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

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

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

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

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

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

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

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

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

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

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

$$\frac{G[e_1, E]\ x \overset{ELL}{\rightsquigarrow} X_1 \qquad G[e_2, E]\ x \overset{ELL}{\rightsquigarrow} X_2}{G[e_1 \mid e_2, E]\ x \overset{ELL}{\rightsquigarrow} X_1 \vee X_2}\ (\text{UNORDER})$$

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

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

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

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

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

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

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

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

$$\frac{n \geq 2 \quad G[e, E_1]\ x_1 \overset{ELL}{\rightsquigarrow} X_1 \quad G[e]\langle x_2, E_2 \rangle \vee \ldots \vee \langle x_n, E_n \rangle \overset{ELL}{\rightsquigarrow} X_2}{G[e]\ \langle x_1, E_1 \rangle \vee \ldots \vee \langle x_n, E_n \rangle \overset{ELL}{\rightsquigarrow} X_1 \vee X_2}\ (\text{SPLIT.2})$$

図 **6** *ELL* の意味論

Fig. 6 Semantics of the matching relation.

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

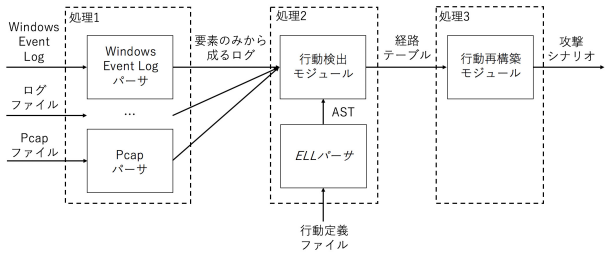


図 7 処理手順
Fig. 7 Overview of the processing.

```
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
Fig. 8 Pseudocode of a terminal [*log*].

とおりである．

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（たとえば，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 と要素のみからなるログを用いて，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 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 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，プログラムは図 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あげて説明をする．

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 番目のログエントリは `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` 関数が用いられる．`parse` 関数は引数として，解析対象の表現 e，現在のリスト内の位置 $\mathtt{ptr}_{cur}$，この関数を呼び出した頂点の番号 $\mathtt{ID}_{pre}$ と環境 E を持つ．終端記号が環境 E においてログエントリ $\mathtt{logs}[\mathtt{ptr}_{cur}]$ 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，$(\mathtt{ID}_{pre}, \mathtt{ptr}_{pre})$ から $(\mathtt{ID}_{cur}, \mathtt{ptr}_{cur}+1)$ 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 を更新したうえ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 $\mathtt{ID}_{last}$ と `logs` 上の最後の位置 $\mathtt{ptr}_{last}$ を用いて移動経路の復元が可能である．なぜなら，$\mathrm{T}[\mathtt{ID}_{last}][\mathtt{ptr}_{last}]$ 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，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すべ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

性能評価．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* を用いて行動定義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．

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 を閲覧．
(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 を選択した．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．

次に，評価を行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とおりである．

攻撃者環境：

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

標的環境：

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

標的環境では，関連研究 [13] 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．これらのログの収集に要した時間は 6 分 35 秒である．収集された各ログの量を**表 2** に示す．

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あらかじ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 [13] を踏襲したものである．

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* で行動を定義し，評価を行った．*ELL* による行動の定義の一部を**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．

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．本評価では，サイバーキルチェーンの各フェーズについて検出されたログエントリの数，適合率と再現率を計測した．ここで，表 3 のログ量（人）は人手でラベル付けした正解のログエントリの数を表し，ログ量（*ELL*）は *ELL* により検出されたログエントリの数を表す．

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 [13] を用いた．

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 アドレスはすべて検出されているもの

```
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
Fig. 9 Delivery phase written in *ELL*.

表 2 ログ量（ログエントリ数）
Table 2 The number of log entries.

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

表 3 行動ごとの精度評価結果
Table 3 Accuracy of each phase.

	ログ量（人）	ログ量（*ELL*）	精度	
			適合率	再現率
Delivery	3	2	1.000	0.667
Exploitation	6	4	1.000	0.667
Installation	8	5	1.000	0.625
C2	3	3	1.000	1.000

[*4] https://blog.trendmicro.co.jp/archives/11465
[*5] https://www.metasploit.com/
[*6] https://github.com/mailhog/MailHog
[*7] http://bitblaze.cs.berkeley.edu/
[*8] https://www.virustotal.com/#/home/

表 4 評価結果
Table 4 Accuracy comparison.

	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	1.000	0.700

表 5 処理速度評価結果
Table 5 The results of the running time.

構文解析時間（秒）	再構築時間（秒）
49.0	0.2

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 ではコミュニティ検出の対象となるグラフへの重み付け方法を 5 つ紹介しているが，本評価ではそのうちの 1 つである Feature Weight Summation を採用した．Feature Weight Summation を採用した理由は，正常な行動と異常な行動から記録されたログから構成されるデータセットが不要な重み付け方法であるためである．他の重み付け方法ではデータセットが必要となるが，データセットは公開されていない．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ごと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．

HERCULE の適合率が低くなっているが，これは HERCULE のログ関連付け方法に原因がある．より具体的には，HERCULE はログを関連付ける際に同じ親プロセス ID を記録しているログどうしを関連付けるため，親プロセスが攻撃シナリオには含まれない行動をした際のログまですべて関連付けられていた．

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* で記述された行動定義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．

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 0.7 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チャできるようにすればすべてのログが検出できるようになる．

実環境から取得したログを用いた場合について．5 章の評価や 7 章で述べられている既存研究では，攻撃者の行動の痕跡がすべてログに記録されていることを前提としている．しかし，現実的にはこの前提を満たすことは容易でない．なぜなら，すべての痕跡を記録するためにはペタバイトにものぼる量のログを保存しておかなければならないからである [6], [15]．そのため実用化に向けては，攻撃者の行動の痕跡がすべてログに記録されていることを前提としない，つまり実環境で収集したログを用いて攻撃シナリオを再構築できるかどうかの評価が必要と考えている．また実用化に向けては，分析者が *ELL* を用いて行動定義を記述する際にかかる負担の軽減のため，より簡単に文法を定義するための糖衣構文の実装や攻撃シナリオ定義時に頻繁に使われるようなパターンのテンプレート化などが必要になると考える．

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 [14]，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るかどうか確認を行った．今後は解析速度向上のため，メモ化の適用，部分的な DFA 化 [2] やカット演算子 [12] による高速化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 年ほど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 BackTracker を提案した [8]．BackTracker は，ログが含むプロセスやファイルの情報を用いて，インシデントが発覚したきっかけとなったログから時系列逆順にログを関連付け依存グラフを構築する．しかし BackTracker には，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 [9]．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．

同年に Lee らは，収集したログからインシデントと関係のないものを取り除く手法として LogGC も提案した [10]．LogGC は，他のプロセス，ファイルやソケットに影響を与えることがなく，またそれらから影響を受けることもな

いログはインシデント調査に影響を与えないという点に着目し，それらを収集したログから取り除くことで調査対象のログの量を削減した．BEEP との主な違いは削減する対象である．つまり，BEEP は依存グラフの状態数を削減するが，LogGC は収集するログの量を削減する．

2016 年に Pei らは，ログを関連付けるだけでなく，攻撃における行動ごとにログをまとめる技術として HERCULE を提案した [13]．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．

2018 年には Liu らにより，ログを関連付けたグラフを，異常と思われる箇所から優先的に探索する手法として PRIOTRACKER が提案されている [11]．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．

BackTracker や BEEP は，インシデントと関連するログどうしを関連付けることに注力しており，そのインシデントがどういった攻撃シナリオにより発生したものなのかを明らかにするのは分析者である．HERCULE では行動を定義できるものの，正常か異常かでしか定義できず，異常な行動が何であるかは分析者がログを精査し解明する必要がある．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，分析者は自身でログを精査しなくても，*ELL* で行動定義を記述することで攻撃シナリオを解明できると考えられる．また，LogGC は調査対象のログ量を削減する手法であり，インシデント調査を開始する前に用いることが可能である．そのため，事前に LogGC により調査対象のログ量を削減しておくことで，より効率的にインシデント調査を進めることができると考えられる．特に，本研究で提案した技術の処理速度は調査対象のログ量に依存するため，LogGC を用いてログ量を削減しておくことで，より高速に処理を実施できると考えられる．

形式文法． *ELL* は，既存の形式文法である PEGwUC に変更を加えて攻撃者の振舞いを表現している．より詳細には，PEGwUC に以下の 2 点の変更を加えている．1 点目として，入力を文字列ではなくログに変更している．また，攻撃者の振舞いを形式化するうえで必要となる文脈に依存した表現ができるような変更を加えている．具体的には，ログに含まれる部分文字列を変数に記録し，後の解析でその変数に束縛された値を参照して解析時の振舞いを変更できるようにしている．これにより，PEGwUC では表現ができない，もしくは表現することが難しい攻撃者の振舞いが表現できるようになっている．図 9 に示されている *ELL* がその例である．異なるログ上の箇所に出現するファイル名が同じことを PEGwUC で表現することは容易ではないが，*ELL* では図 9 に示されているとおり，変数に束縛することでそれを実現している．

2 点目として，*ELL* により定義した振舞いのうち，必要な箇所だけをログの中から抽出できるように変更を加えている．PEGwUC では記述した表現に入力がマッチした場合，入力の先頭からマッチした箇所までのすべてが抽出される．しかし，インシデント調査において必要となる箇所はその中のごく一部である．そこで，*ELL* ではタグ付きキャプチャを導入し，インシデント調査において必要となる箇所のみを抽出できるようにした．

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* で記述した行動定義をログから検出・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．

今後の課題としては，評価実験の拡充があげられる．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, *Proc. Language and Automata Theory and Applications — 11th International Conference, LATA 2017*, 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*), pp.487–504 (2017).

[6] Hossain, M.N., Wang, J., Sekar, R. and Stoller, S.D.: Dependence-Preserving Data Compaction for Scalable Forensic Analysis, *27th USENIX Security Symposium* (*USENIX Security 18*), pp.1723–1740, USENIX Association (2018).

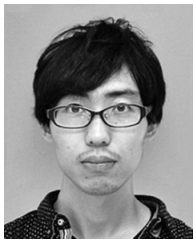
[7] Hutchins, E.M., Cloppert, M.J. and Amin, R.M.: Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains, *Leading Issues in Information Warfare & Se-*

curity Research, Vol.1, p.80 (2011).
[8] King, S.T. and Chen, P.M.: Backtracking Intrusions, *Proc. 19th ACM Symposium on Operating Systems Principles*, *SOSP '03*, pp.223–236, ACM (2003).
[9] Lee, K.H., Zhang, X. and Xu, D.: High Accuracy Attack Provenance via Binary-based Execution Partition, *NDSS*, The Internet Society (2013).
[10] Lee, K.H., Zhang, X. and Xu, D.: LogGC: Garbage collecting audit log, *Proc. 2013 ACM SIGSAC Conference on Computer & Communications Security*, *CCS '13*, pp.1005–1016 (online), DOI: 10.1145/2508859.2516731, ACM (2013).
[11] 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).
[12] Mizushima, K., Maeda, A. and Yamaguchi, Y.: Packrat Parsers Can Handle Practical Grammars in Mostly Constant Space, *Proc. 9th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, *PASTE '10*, pp.29–36 (online), DOI: 10.1145/1806672.1806679, ACM (2010).
[13] 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, *Proc. 32nd Annual Conference on Computer Security Applications*, *ACSAC '16*, pp.583–595, ACM (2016).
[14] 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).
[15] Tang, Y., Li, D., Li, Z., Zhang, M., Jee, K., Xiao, X., Wu, Z., Rhee, J., Xu, F. and Li, Q.: NodeMerge: Template Based Efficient Data Reduction For Big-Data Causality Analysis, *Proc. 2018 ACM SIGSAC Conference on Computer and Communications Security*, *CCS '18*, pp.1324–1337 (online), DOI: 10.1145/3243734.3243763, ACM (2018).

推薦文

多量に生成されるログから攻撃シナリオを再構築する，というタイムリーかつ重要な課題に着目し，その解決策として，攻撃シナリオのモデル化をログをベースに行うための中間表現を提案し，これを用いて事前に表現した攻撃シナリオと，実際のログとの突き合わせから攻撃シナリオを検知・再構築する手法を提案している．従来手法が攻撃を再現するのに対し，提案手法は攻撃シナリオを再現する点で新規性があり，また提案している文法が抽象化を行うために必要十分な表現力があることを実験で示しており，新規性・信頼性ともに十分である．

（コンピュータセキュリティシンポジウム 2018（CSS2018）
プログラム委員長　吉岡 克成）



千田 忠賢　（正会員）

2015 年会津大学コンピュータ理工学部卒業．2017 年横浜国立大学大学院工学府博士課程前期修了．同年日本電信電話株式会社入社．サイバー攻撃対策技術に関する研究に従事．

鐘本 楊

2011 年名古屋大学工学部卒業．2013 年同大学大学院情報科学研究科博士課程前期修了．同年日本電信電話株式会社入社．サイバー攻撃対策技術に関する研究に従事．

青木 一史　（正会員）

2004 年東北大学工学部卒業．2006 年同大学大学院情報科学研究科博士課程前期修了．同年日本電信電話株式会社入社．主任研究員．サイバー攻撃対策技術に関する研究に従事．電子情報通信学会会員．

三好 潤

1993 年京都大学工学部卒業．1995 年同大学大学院工学研究科修士課程修了．同年日本電信電話株式会社入社．主幹研究員．サイバー攻撃対策技術に関する研究に従事．電子情報通信学会会員．