外的環境変化と複雑な問い合わせの双方に対応できるセキュリティデータ管理方式に関する諸検討

村上　　直†

† 高エネルギー加速器研究機構（KEK） 計算科学センター
〒 305–0801 茨城県つくば市大穂 1–1
E-mail: †tadashi.murakami@kek.jp

あらまし　ネットワークセキュリティを取り巻く環境や技術は日進月歩であり、またネットワークセキュリティを維持する社会的要請が強まっている。これにより、セキュリティ装置のログデータなど、セキュリティデータの管理や有効活用の重要性が増している。一般的に、このログデータはフォーマットが定まっており、また、しばしばサイズが膨大となる。セキュリティ装置は複数の観点からのログを出力しているため、ログ同士の連携も重要である。このようなデータは、スキーマを定めて適切に設計すると、複雑な問い合わせに対応できるなど有効活用できるが、装置のバージョンアップや変更、また時間あたりのデータサイズが増すなど、外的環境が変化した場合の対応が困難となる。スキーマを定めない場合、外的環境変化への対応は容易になるが、複雑な問い合わせに対応するのが難しくなる。そこで本研究では、外的環境変化と複雑な問い合わせの双方に対応できる、セキュリティデータ管理方式に関して諸検討を行う。

キーワード　セキュリティ、時系列ログデータ、スキーマ管理

1. はじめに

ネットワークセキュリティの管理は、計算機をネットワークに接続しているあらゆる組織において、重要である。コンピュータネットワーク上の機器の脆弱性や人の心理につけ込んだサイバー攻撃から組織を守るべく、セキュリティ担当者は、情報収集や最新鋭のセキュリティ装置導入に常に追われている。

セキュリティ装置の発する観測データの多くは時系列データであり、これと組織内における機器管理台帳が連携付けられることで、有用なデータとなることが通常である。典型的な例を挙げる（図 1）。セキュリティインシデントが疑われる通信が発生した場合は、通信機器の所有者を割り出す必要がある。ファイアウォールなどの通信記録は、観測された通信の通信元及び通信先 IP アドレスに対して、送受信バイト数、通信プロトコル、脅威情報などの組をもつが、この記録には所有者に関する情報は含まれないのが通常である。さらに、この IP アドレスは

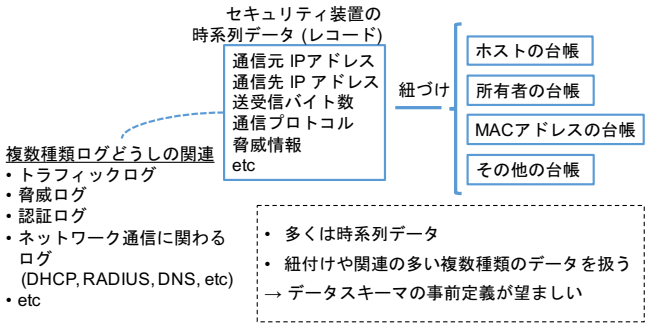

図 1　ログ管理の典型的な状況

DHCP で動的に管理される場合が多く、この場合、ひとつの IP アドレスの利用者は時間の経過により変わる。そこで、通信記録の IP アドレスと該当時刻における DHCP サーバのログから該当する通信の MAC アドレスを割り出し、この MAC アドレスで機器管理台帳に照会することで、はじめて所有者が判明する。このような作業を、調査対象の全ての通信に対して行う必要が出てくる。また、問題ありと疑われる複数の IP アドレスに対して通信した機器の所有者と、各々の通信回数、通信総バイト数を調べたいといった要求も典型的である。このように、セキュリティに関するデータを扱う際には、複数種類の台帳の照合を行い、かつ複数のキーを使って集計するような要求が頻繁に発生する。このような複雑な要求に応えるためには、データスキーマを事前に定義して、データスキーマの扱いに長けたデータベースを利用するのが望ましい。

また、ネットワークトラフィックが増大することを考えると、一定時間あたりに受け付けるログデータは運用するごとに増大する一方である。一定時間あたりのログデータが増大しても、いわゆるビッグデータ [4], [12] に属する技術などを活用して、応答の性能を維持する必要がある。このような状況を踏まえ、OS のログ、セキュリティデバイスのログ、ウェブサーバやメールサーバなど各種サーバのログなど、様々なログデータを一ヶ所に集めて管理することを骨子とした、SIEM [5] (Security Information and Event Management) とよばれるセキュリティ管理方式の重要性が指摘されている。

一方で、セキュリティ技術は日進月歩であり、現在のセキュリティ装置や手法が数年で役に立たなくなることが多々ある。最新鋭のセキュリティ装置を導入しても、頻繁なバージョン

アップが発生し、さらに、数年毎に新しい方式の装置への入替が必要となる。このとき、装置が発する観測データの形式もしばしば変更される。もとより、ネットワークセキュリティを管理するためのシステムは、収集、解釈、検索のための各種ツールの適材適所な寄せ集めであることが多く、各々のツールにデータに関する定義が分散している。システムにデータスキーマをもたせるとデータ構造の変更に対応させるのが難しくなることを考えると、セキュリティ装置の発するデータにデータスキーマをもたせると、装置のバージョンアップや入替がしづらくなり、また、日々変わるセキュリティ情勢を解析基盤に反映させるのも容易ではなくなる。

これらを踏まえると、セキュリティ装置の発する観測データを、他の装置の観測データや、組織内の具体的なネットワークリソース、日々変わる情勢に対応したセキュリティ運用方針、過去のインシデント情報などの組織内環境と関連づけて運用を続けるのは困難であり、各々のデータはばらばらに扱わざるを得ない。

このように、ネットワークセキュリティ管理の重要性はここ数年で飛躍的に増しているにもかかわらず、セキュリティに関わるデータ構造がしばしば変更されることを前提としたデータ管理の枠組みの研究は、盛んには為されていない。たとえば文献 [2] では、セキュリティインシデントの解析や対応のための協調作業ツールが報告されているが、データスキーマの頻繁な変更に対応することの困難さへの解決策は示されていない。また産業界においても、セキュリティ装置の各ベンダが総合的なセキュリティ管理手法を盛んに提案しているが、提案間の互換性は無い。したがって、セキュリティに関する機能の有効活用のためには、ネットワーク機器の大部分を提案するベンダ製品でそろえる必要がある。また、ベンダの想定しないデータ活用には、困難を伴う。

まとめると、ネットワークセキュリティにおけるデータの管理においては、下記の 3 点が課題となる。

1. 一定時間あたりのログデータは運用するごとに増大する一方である。増大しても、応答の性能を維持する必要がある。
2. 複数種類の台帳の照合を行い、かつ複数のキーを使って集計するような要求が頻繁に発生するため、データスキーマ、集計演算、結合演算の扱いに長けたデータベースを利用するのが望ましい。
3. セキュリティ技術は日進月歩であり、セキュリティの運用では、バージョンアップやセキュリティ機器の入れ替えがしばしば発生する。この際、機器の発するデータ構造が変わることもある。データスキーマを事前に定めると、この対応が難しくなる。

この課題のうち、1. を満たすには、いわゆる NoSQL [8] とよばれるデータベースもしくは関係データベースが適している。この二つのデータベースについてみると、NoSQL のデータベースでは 2. を満たすことができない。いっぽうで関係データベースは 2. の扱いは得意だが 3. を不得意としている。すなわち、1, 2, 3 はいずれも、ネットワークセキュリティにおけるデータの管理において重要であるが、これを同時に満たすのは難しい。

これらの考察を踏まえ、本研究では、1, 2, 3 を満たすことができる、すなわち外的環境変化と複雑な問い合わせの双方に対応できる、セキュリティデータ管理方式に関して諸検討を行う。

2. 関 連 研 究

本節では、SIEM に代表されるセキュリティログの収集や格納を扱った関連研究について述べる。

Madani らは文献 [7] で、セキュリティログ収集に必要な要素を調査し、これを満たすための、ログ収集サーバとログ格納サーバによるシステム構成と、各コンポーネントに求められる要件をまとめている。また、法令遵守のために最短一年最長七年のログ保存が必要であることが示されている。

Fedaghi らは文献 [1] で、各々のログ管理システムがそれぞれ、特定のハードウェアやシステムに依存していることを指摘している。その上で、これらに依存しない理論的な枠組みを提案している。

Rieke らは文献 [10] で、侵入検知システム (IDS) について、用意された 4 種類のシナリオ (文献 [9]) を実行した。その経験に基づき、SIEM に備わると望ましい要件を抽出している。

これらの文献 [1], [7], [10] について、ログ生成元のデータフォーマットが変化しうることに対する考察は為されていない。

Kotenko らは文献 [6] で、大量のデータを関係データベースで扱い、複雑なデータをオントロジーで扱う、ハイブリッドアプローチを提案している。この研究では、関係データベースに格納されるデータについて、1 フィールド 1 行で扱っており、リソースのデータ形式が変更されても、フィールド名とデータを正しく読み取れれば、データの格納を続行できる。しかし、リソースのデータ形式が変更されたことを把握せずに格納されたデータを利用し続けることは不可能であるため、ログ生成元のデータフォーマットが変化しうることに対応しきれていない。

3. セキュリティデータの管理方式に関する問題点の分析

本節では、ネットワークセキュリティに関するデータの管理方式に関する問題点の分析を行う。

セキュリティ装置が発する通信ログは、通信の発生時刻における通信元と通信先の IP アドレスについて記録されることが多い。セキュリティ事象を扱う際、通信機器の使用者の特定は重要である。IP アドレスの運用方法によって、IP アドレスとその機器の使用者の関係には、複数パターンがある。また、ひとつの組織において複数の方式が採用されることも多い。さらに、IP アドレスと使用者の関係が時刻により変化する場合も多い。このように、通信ログは複数の台帳との対応関係をもつ。

このような複数の対応関係をもつデータは、たとえば下記のようなシナリオで利用される。

- ある DHCP 配下の IP アドレスについて、異常な通信が検知された。DHCP ログから通信機器の MAC アドレス

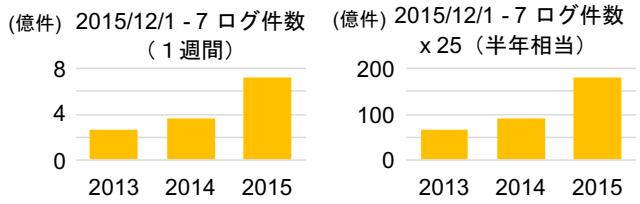

図 2　ログ件数 2015 年 12 月 1 日～12 月 7 日

を割り出し、利用者を特定。次に、この利用者が行った直近 3 ヶ月の通信を洗い出し、詳細な検討を行う。IP アドレスは DHCP で動的に割り当てられるため、利用者が使っている IP アドレスは時刻により変化する。

- 上記の調査の結果、この利用者の通信機器はマルウェアに感染していた。また、不審な外部 IP アドレスが複数判明した。同様の感染が組織内に蔓延していないかどうか、該当の複数 IP アドレスと通信している組織内の通信機器を割り出す必要がある。組織内では、IP アドレスを固定で持っている利用者、DHCP により MAC アドレスは固定だが IP アドレスは動的に割り当てられる利用者、ログイン認証により MAC アドレスも IP アドレスも動的に割り当てられる利用者がいる。
- 上記の通信について、通信回数と送受信バイト数を通信機器ごとに割り出すことで、端末毎の状況の深刻度に軽重を付けて調査に当たりたい。

セキュリティに関するデータのうち、通信ログなど件数が多くなるデータは、時系列である場合が多い。高速に検索できるような通信ログの格納テクニックは、クラスタ化、スタースキーマ化、列指向化など、複数存在する。また、キー属性の条件により格納先を変えることで 1 テーブルあたりのデータ量を減らすクラスタ化においては、1 テーブルあたりに格納させるデータ量の検討なども必要である。ログ件数は年々増える場合が多く、定期的な見直しが必要である（図 2(注1)）。

組織におけるセキュリティの運用は長期間にわたるため、バージョンアップや機器の入替などによりデータ形式が変更される。このため、データ形式が変わってもデータを引き続き受け入れられる必要がある。また、データ形式の変更前に作成したクエリ文を、変更後も引き続き扱えることが望ましい。

このようなシナリオについて、関係データベースを用いると複雑なクエリを扱うことができるが、性能がスケールアウトしない。いっぽう、いわゆる NoSQL データベースを用いると、性能はスケールアウトするが複雑なクエリを扱えない。本研究では、関係データベースを念頭におき、セキュリティデータの管理方式の検討を行う。

ログの調査対象は一定期間に限られることが多いと考えられるため、通信時刻をキーにしてクラスタ化することは有効である [3]。RDBMS においては、1 テーブルを指定したキーによってクラスタ化するパーティション化機能を備えているものが多いが、高価なオプション機能としての提供であったり、処理の並列化が十分でなかったり、扱いが複雑であったりと、利用には困難を伴う。また、通信機器が変更されたりデータ格納方法が変更されるなど、テーブルの構造が変化した場合に、一つのパーティション化テーブル内で複数のテーブル構造を扱うことはできず、運用の柔軟性は阻害される。

(注1)：当機構で運用している基幹ファイアウォールを通過する通信ログについて、12 月 1 日から 12 月 7 日の通信件数の合計を 2013 年、2014 年、2015 年について調べたところ、それぞれ約 2.6 億件、3.7 億件、7.2 億件であった。

	NoSQL	RDB	検討方式（目標）
複雑な問い合わせ	△	◎	◎
外部環境変化への対応	△	×	○
処理性能	◎	△	○

表 1　従来方式と検討する方式の比較（目標）

このように、ネットワークセキュリティに関するデータを扱うためには、下記のような要件を満たすことが必要である。

1. （複雑な問い合わせ）複雑なクエリや、複数の台帳に渡る集計処理を行える必要がある。
2. （外部環境変化への対応）運用は長期間にわたるため、バージョンアップや機器の入替などによりデータ形式が変更されることを想定し、データ形式が変わっても引き続き受け入れられる必要がある。
3. （処理性能）大量の時系列データを扱う必要がある。セキュリティ環境の悪化により、ログは年々増加するケースが多い。ある時期のデータに基づいてデータ保存方法のチューニングを行っても、数年で見直しが必要になる。見直した場合、たとえば列指向化とスタースキーマのように、データ構造が異なることになっても、透過的に扱えることが望ましい。

2. と 3. は、近年発展が著しいいわゆる NoSQL に属するデータベースシステムが得意とするところだが、1. の要求に応えるのが難しい。また、データスキーマを定めない場合、データ構造が変化した場合に従来のクエリ文を受け付けられるかを検証するのが難しい。そこで、本研究では関係データベースシステムを用いつつ、2. と 3. を克服できる方式に関して検討を行う。検討する方式の目標を表 1 に示す。

4. 外的環境変化と複雑な問い合わせの双方に対応できるセキュリティデータの管理方式

4.1 概　　要

本節では 3. 節を踏まえて、外的環境変化と複雑な問い合わせの双方に対応できるセキュリティデータの管理方式について検討する。

本研究の中核をなすのは、フィルタとレコードの組み合わせにより、ネットワークセキュリティに必要とするデータを管理する解析基盤と、それを記述する言語である。図 3 に、記述言語の記述例を示し、図 4 に、その処理の流れを模式的に示す。

本研究において、データはセキュリティ装置やセキュリティ情報源などのリソースから発せられると考える。ネットワーク

```
                                                    # (1) 変数の定義
$db_conn_str = jdbc:postgresql://localhost/test?user=fred&password=secret

[record_name_1]                                     # (2) レコードの定義（ファイルから読み取りの例）
input = tail: file://...                            # (2-1) 入力リソースを指定
filter = CSV: !date<date>,id,src_ip,dst_ip,appname,... # (2-2) フィルタの動作を順番に指定　! は、index 付与
filter = partition: date(1, "day")                  #   partition: パーティション化
filter = func: ip2bit(src_ip), ip2bit(dst_ip)       #   func: ユーザ定義関数 例)ip2bit IP を 32bit 整数に
filter = toColumn: src_ip,dst_ip                    #   toColumn: カラムストア化
filter = toStar: appname                            #   toStar  : スタースキーマ化
output = rdb: $db_conn_str                          # (2-3) 出力リソースを指定
;;                                                  # (2-4) merge: 同じレコードに格納する別のリソース指定
input = ...
filter = ...

[record_name_2, record_name_3]                      # (2) レコードの定義（dispatch: フィルタの定義に
                                                    #                    より、複数の型のレコードを出力）
...

[record_sql]                                        # (2') レコードの定義（SQL クエリの例）
input = rdb: $db_conn_str
input.cron = 02 9 * * *
filter = SQL: select src_ip,dst_ip from table ...
output = sh: | /home/cronmail/bin/mail_notify.sh

[recgrp:grp_name]                                   # (3) 複数レコード定義のグループ化
target = record_name_1, record_name_2               # (3-1) グループ化対象のレコードを指定
assoc = ...  , ...                                  # (3-2) 名称が異なるフィールド組（結合）を指定
index = .... , ....                                 # (3-3) 名称が異なるフィールド組（インデックス）を指定
ref = .... , ....                                   # (3-4) 名称が異なるフィールド組（クエリ内参照）を指定
```

図 3　記述言語の記述例

構造や機器情報、セキュリティインシデントなど、組織に固有の情報も、これに含まれる。生データのリソースに対してフィルタを通過させることで、スキーマをもつレコードに変換される。このレコードに対しても、フィルタ処理をかけることができる。

レコードは、最終的にはデータベースに格納されるか、若しくはレポーティングやアラートの形で出力される。格納されたデータベース上のデータにクエリをかけてレポーティングやアラートに出力することもできる。

検討する方式は下記に示す特徴を備える。

- 従来方式では困難であった、外的環境変化と複雑な問い合わせの双方に対応できる方式であること
- この方式を、リソース、フィルタ、レコード、データベースを中心とした簡潔な記述言語として設計したこと
- 関係データベースの特徴である複雑なクエリへの対応を享受しつつ、弱点である外的環境変化への対応と処理性能について、対応できる方式を示したこと

4.2　セキュリティログの生成と利用のモデル

本研究では、セキュリティログの生成と利用のモデルを、下記のように、リソース、フィルタ、レコード、データベースを用いて表す。

- セキュリティログの生データを、リソースとよぶ
- リソースに対してフィルタを通過させると、リソースはスキーマをもつレコードに変換される
- dispatch フィルタを定義し、一種類のリソースから複数種類のレコードを出力できるようにする
- merge フィルタを定義し、複数種類のリソースを一種類のレコードにまとめられるようにする
- レコードに対してフィルタを通過させると、別の形式のレコードに変換される

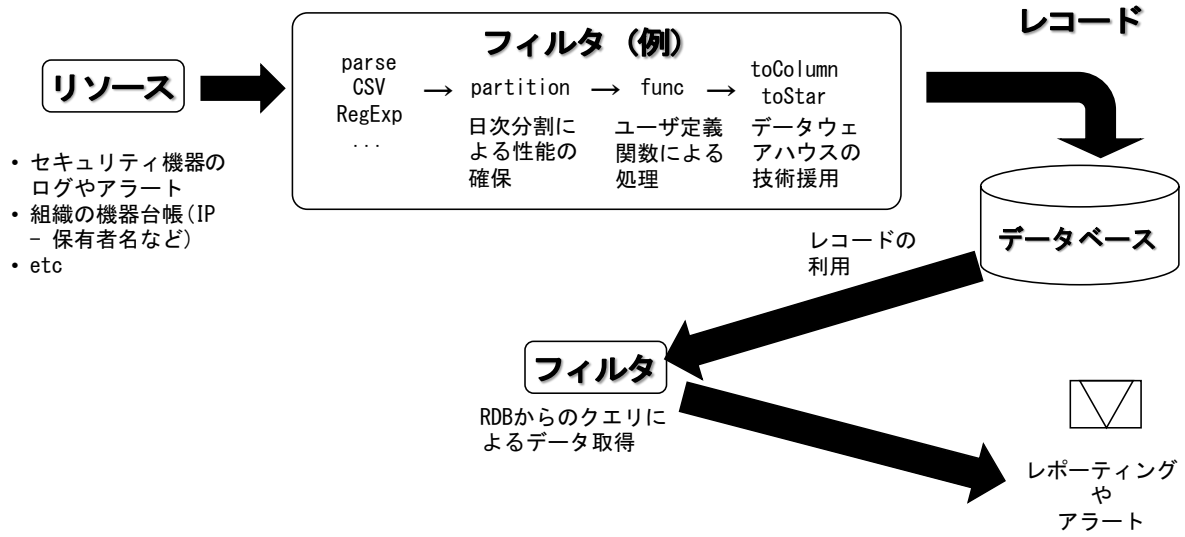

図 4　図 1 の処理 ((3) を除く)

- レコードは、永続化されデータベースに格納されるか、若しくはレポーティングやアラートの形で出力される
- データベースに格納されたデータそのものもレコードとして扱い、データベースに対してクエリをかけた結果をレポーティングやアラートにすることもできる

リソースは、セキュリティ装置やセキュリティ情報源などが記録する生データにより構成される。リソースの対象となるデータは、各々のセキュリティ装置やセキュリティ情報源などが、各々の独自の方法で生成したものである。これらのリソースに対してフィルタを通過させることで、スキーマをもつレコードに変換され、本研究における管理対象となる。

フィルタは、リソースまたはレコードを目的とする形式のレコードに変換する方法を定義する。フィルタによるリソースの処理方法は、一括読み出し若しくはストリーム処理を想定する。フィルタが提供する変換の方法は、下記のような例を考えることができる。

- 正規表現や CSV を用いた、生データから組データへの変換
- カラムストア化 [11]、スタースキーマ化
- 整数型など、型の指定
- インデックスの付与
- クエリ言語を用いた、スキーマをもつデータから所望とするデータの抽出
- ユーザ定義関数やプログラム埋め込みによる任意処理

レコードは、出力方法を指定することができる。現在想定している出力方法は、下記に列挙する通りである。

- データベースに格納することで、レコードの永続化を行う
- レポーティングやアラートによる出力

データベース上で永続化されたレコードには、検索の高速化を目的としたインデックスを付与することができる。また、クエリ言語などにおいて、複数のレコードを結合するための、関連フィールドがある。上記のインデックス、関連フィールドおよび、参照されるフィールドは、それぞれフィルタやレコード内でひも付けられる。

ネットワークセキュリティを取り巻く環境や技術は日進月歩であるので、セキュリティ装置のバージョンアップや変更が発生した際に、リソースの形式がしばしば変更される。セキュリティ装置が発するログデータを活用すればするほど、一般的にはリソースの形式変更への対応は困難となる。本研究で提案する管理方式では、これに対応するために、レコードグループを導入する。レコードグループは複数のレコード定義から構成され、グループ内のレコードは、別のデータフォーマットであっても同種のレコードとして扱う。これは、レコードグループ内において、同種として扱うフィールドをグループ化することにより実現する。グループ化の具体的な指定方法は、4.3.3 節で扱う。

4.3　記述言語の設計

本節では、4.2 節で示した生成と利用のモデルを実現するための記述言語について述べる。1. 節で述べたようにこの記述言語は、データベースの専門知識をもたないセキュリティエンジニアが扱えることを意識している。図 3 に、記述言語の記述例を示し、図 4 に、その処理の流れを模式的に示す。以下、図 3 の例に沿って、記述言語の設計を示す。

4.3.1　変数の定義

この記述言語では、図 3 の (1) に示すように、任意の文字列を変数として定義できる。これにより、データベース接続文字列などの定義の記述が分散することを防ぐ。

4.3.2　レコードの定義

図 3 の (2) のように、[] で囲んだ文字列により、レコードを定義する。レコード定義には、入力リソース (2-1)、フィルタの動作 (2-2)、出力リソース (2-3) を含めることができる。

入力リソース (2-1) や出力リソース (2-3) では、cat（ファイルを指定して読み込む）、tail（ファイルを指定して読み込み、末端まで達すると入力待ち）、rdb（関係データベース）などを指定できる。

フィルタの動作 (2-2) では、フィルタによるリソースの読み込み方法を、処理の順番に指定する。CSV、RegExp、SQL などを指定できる。(2-2) の指定では、フィールド名を併せて指定するものとする。ここで、処理高速化のための変換も実施できる。例えば下記のような機能が考えられる。

- フィールド名の頭に ! を付与すると、そのフィールドにはインデックスが付与される。
- partition: 指定したキーを用いたパーティション化
- func: ユーザ定義関数を定義できる。例は、IP アドレスの文字列を、32bit の符号無し整数に変換
- toColumn: カラムストア化
- toStar: スタースキーマ化
- <date>, <int>, ...: 日付型や整数型など、型を指定する

(2-4) に示すように、;; のみの行を記述することで、一種類のレコードに対して入力リソースやフィルタを複数種類定義できる。

図 3 の (2') に、SQL クエリの例を示す。この例では、毎日午前 9:02 に、指定した SQL が発行され、mail_notify.sh のその結果が渡される。

4.3.3 複数レコード定義のグループ化

図 3 の (3) のように、複数レコード定義をグループ化できる。これを用いることで、ログデータの仕様が変更された場合でも、既に稼働している SQL クエリなどをひきつづき利用できる。

ログデータの仕様が変更された場合、はじめに、新しいレコードを定義する。次に、旧レコード定義と新レコード定義をレコードグループに入れる (3-1)。その次に、レコードグループ内の新旧のフィールドについて、同種として扱うフィールドをグループ化する。

ここで、旧レコード定義で既に用いられている結合（assoc）、インデックス（index）、クエリ内の参照（ref）が、新レコードで定義されているかどうかを調べる。新レコード定義に対応するフィールドが存在しなければ、措置が必要となる。措置を下記に列挙する。

- assoc が無い場合：同じクエリ定義を続けて利用するのは不可能である。新レコード定義にあったクエリを定義する必要がある
- index が無い場合：同じクエリ定義を続けて利用できるが、パフォーマンスが劣化するなどの影響がありうるので、試行や再検討が必要である
- ref が無い場合：同じクエリ定義を続けて利用できるが、レコードのフィールドに欠損が生じるので、再検討が必要である

5. 評価のシナリオ

本節では、本研究の評価のシナリオを述べる。骨子は下記の通りである。

- 提案システムのプロトタイプを構築し、実データを投入し、運用する
- 環境を変更させるシナリオをいくつかつくり、提案システムにおいて変更が極小にとどまるかどうかを検証する

環境を変更させるシナリオは、例えば下記のようなものを検討している。

- セキュリティデバイスのログを取り始める。但し、ログのフォーマット仕様は明らかではない。ここでは、時刻、ログ名、フィールド全体、という非定型なデータを、レコードとしてデータベースに格納する
- ログの仕様が明らかになったが、複数種類のフォーマットが提供されており、全てへの対応を一度に考えるのは難しい。数日はテスト実行する必要がある
- セキュリティ装置が変わり、フォーマットが大幅に変更された。セキュリティ業務をすでに遂行しており、固定的なクエリが多数ある。これらはフォーマット変更の影響を受ける。できるかぎり属性を引き継ぎ、フォーマット変更前の解析ツールとの整合性を保ちたい

6. 予備実験とその結果

本節では、現時点での到達状況、予備実験の内容、およびその結果を示す。

4. 節で示した提案のうち、ログデータをデータベースに格納する基本モジュールを実装した。この基本モジュールでは、設定ファイルに記述したフォーマットのログを読み込み、データベースに格納できる。指定するフォーマットにおいては、ログファイルを読み取る正規表現と、正規表現で指定した要素に対応したフィールド名を指定できる。指定されたフィールドにおいて、整数型や日付型を指定することもできる。読み取ったログデータは、PostgreSQL と MongoDB に格納可能である。

この基本モジュールを用いて、性能測定の予備実験を実施した。実施結果を表 2 に示す。対象データは、2015 年 12 月 1 日に発生した当機構の通信ログであり、合計 114,036,033 件である。この通信ログは 49 列のフィールドを持つ。実験においては、全列の格納 (all) と主要列 12 列の格納 (core) を行い、その各々について性能測定を行った。また、索引有無の両方について実験を行った。性能測定の対象としたデータベースエンジンは、PostgreSQL 9.5 と MongoDB 3.2 である。

実施したシナリオは 3 種類ある。

1) 全件検索カウント：宛先 IP アドレスと通信成否 (allow/deny) でグループ化して全件を集計演算し、結果のうち上位 20 件を取得
2) 検索 1 IP 件数小：件数の小さい宛先 IP (897 件) について、件数を集計

単位：秒			キャッシュ無(1回目)			キャッシュ有(2-4回目)		
	table	索引	Postgres	Mongo	時間比	Postgres	Mongo	時間比
1) 全件検索カウント	core	無	72.63	31.04	2.34	72.94	32.62	2.24
	all	無	102.68	72.03	1.43	126.07	70.56	1.79
2) 検索 1 IP 件数小	core	無	48.27	15.03	3.21	50.01	14.66	3.41
	core	有	0.27	0.01	27.00	0.25	0.01	25.00
	all	無	156.64	42.81	3.66	61.78	41.50	1.49
	all	有	55.50	0.01	5550.00	2.74	0.01	274.00
3) 検索 1 IP 件数大	core	無	148.24	15.05	9.85	48.50	14.74	3.29
	core	有	84.05	0.23	365.44	97.95	0.23	425.87
	all	無	62.07	41.85	1.48	62.03	41.51	1.49
	all	有	232.67	0.23	1011.61	181.75	0.24	757.29

表 2　応答時間の計測結果

3) 検索 1 IP 件数大：件数の大きい宛先 IP (11,457,152 件) について、件数を集計

各々の測定は、4 回実施した。そのうち 1 回目をキャッシュ無しとみなし、2 回目以降をキャッシュ有りとみなして集計した。キャッシュ有りとみなした 3 回については、その平均値を集計結果とした。

表 2 の実施結果から、以下を読み取ることができる。

- 全般的に、MongoDB のほうが性能で優位であった。特に索引付きでは大きな差異を示した。
- 但し MongoDB においても、1 ヶ月以上の複数観点で集計した場合は、性能は十分とは言えない。
- 全件検索した場合の性能差は比較的小さかった。
- 列数を絞った場合の性能向上の効果は高かった。

現在、他のスキーマパターンでの実験を準備中である。特に、PostgreSQL においてスタースキーマを導入した場合の効果は高くなると予想している。

7. まとめと今後の予定

本研究では、外的環境変化と複雑な問い合わせの双方に対応できる、セキュリティデータ管理方式に関して諸検討を行い、これを実現できる方式の設計を示した。示した方式は下記に示す特徴を備える。

- 従来方式では困難であった、外的環境変化と複雑な問い合わせの双方に対応できる方式であること
- この方式を、リソース、フィルタ、レコード、データベースを中心とした簡潔な記述言語として設計したこと
- 関係データベースの特徴である複雑なクエリへの対応を享受しつつ、弱点である外的環境変化への対応と処理性能について、対応できる方式を示したこと

示した方式が実現されれば、セキュリティデータの管理において重要であり、かつ従来方式では両立の難しかった、下記の課題を同時に解決できると期待できる。

- （複雑な問い合わせ）複雑なクエリや、複数の台帳に渡る集計処理
- （外部環境変化への対応）長期間にわたる運用で避けられない、バージョンアップや機器の入替などによりデータ形式が変更されることを想定し、データ形式が変わっても引き続き受け入れ可能とすること
- （処理性能）ネットワークトラフィックの増大により避けられない、セキュリティデータの増大に対応した、大量の時系列データの処理

現在、示した設計の実装を進めており、6. 節にて予備実験の結果を示した。全般的に MongoDB のほうが性能が高い結果となったが、PostgreSQL における性能向上案を残しており、今後進めていく予定である。その上で、プロトタイプシステムの実装と実運用環境への投入、5. 節に示した評価シナリオの実施、性能評価により、本研究の有効性を評価し、その結果を踏まえて提案内容の改善を図る予定である。

文　献

[1] S. Al Fedaghi and B. Mattar. On security log management systems. *Global Journal of Computer Science and Technology*, 10(6), 2010.

[2] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37, 1996.

[3] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2 edition, 2008.

[4] H. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi. Big data and its technical challenges. *Communications of the ACM*, 57(7):86–94, 2014.

[5] K. Kent and M. Souppaya. Guide to computer security log management. *NIST special publication*, 92, 2006.

[6] I. Kotenko, O. Polubelova, A. Chechulin, and I. Saenko. Design and implementation of a hybrid ontological-relational data repository for siem systems. *Future Internet*, 5(3):355–375, 2013.

[7] A. Madani, S. Rezayi, and H. Gharaee. Log management comprehensive architecture in security operation center (soc). In *Computational Aspects of Social Networks (CASoN), 2011 International Conference On*, pages 284–289. IEEE, 2011.

[8] C. Mohan. History repeats itself: sensible and nonsensql aspects of the nosql hoopla. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 11–16. ACM, 2013.

[9] Project MASSIF. MASSIF FP7 Project (2013). `http://www.massif-project.eu/`.

[10] R. Rieke, L. Coppolino, A. Hutchison, E. Prieto, and C. Gaber. Security and reliability requirements for advanced security event management. In *Computer Network Security*, pages 171–180. Springer, 2012.

[11] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, et al. C-store: a column-oriented dbms. In *Proceedings of the 31st international conference on Very large data bases*, pages 553–564. VLDB Endowment, 2005.

[12] A. R. Zope, A. Vidhate, and N. Harale. Data minding approach in security information and event management. *J. Future Comput. Commun*, 2013.