開発技術と開発プロセス

DBセキュリティ確保のための監査証跡記録機能

生産品質強化本部
設計・インフラ監理センター
田中　薫

1．はじめに

本プロジェクトで構築したの は金融系の DB であり、また、年金加入者などの個人データを扱うため、そのセキュリティ確保は重要なテーマである。

セキュリティ確保の観点では、次の2つを考慮しておく必要がある。

ひとつは、いかにして侵入や不正アクセスを防ぐかであり、もうひとつは、万一不正なアクセスがあった場合、それを記録に残す手段を講じておくことである。これらを考える上で重要なのは、いずれの場合も、絶対ということはないことである。絶対に侵入されないシステムはありえないし、侵入されたことを必ず記録することも保証できない。また、「絶対」に近付けようとすれば、そのぶんコストがかさみ、システムの使い勝手やパフォーマンスも低下する。したがって、予算、システムの使いやすさ、パフォーマンスなど、システム全体のバランスをとることが重要である。

このような検討のベースになるものが、セキュリティポリシーである。今回は、顧客が設定していたセキュリティポリシーに従い、設定を行った。

DB におけるセキュリティ関連の設定については、Microsoftが提供するSQL Server 自習書シリーズ（http://www.microsoft.com/japan/SQL/techinfo/selfstudy/Self_doc.asp） の9章「セキュリティ設定」を参照するのが良いだろう。

一方、監査証跡の記録については、DB サーバーに対するすべてのアクセスパケットをキャプチャし、記録すれば目的は達するように思われるだろうが、このままではデータ量が膨大となり、後の解析がやりにくい。

そこで、DB に対するベーシックなアクセス、DB の更新処理の単位での記録、およびアプリケーションレベルでのシステムへのアクセス記録など、いくつかのレベルを設けて、それぞれに情報を収集して記録する手段を考えた。それぞれの目的と方法の詳細について、次項 で述べる。

記録に残すとともに、収集した情報を解析して、不正アクセスとみなされるものがあった場合には警告を発することになるが、そのような解析を自動的に行うことは非常に難しく、最終的には人間が判断する必要がある。本システムでも、一次的なフィルタリングを行った結果を不正アクセスリスト（不正アクセスと思われるアクセスの記録）として作成し、運用担当者に提示する方式とした。

2．監査証跡取得の目的と方法

DB へのアクセスについて記録に残すことにより、不正なアクセスを調べることができる。また、そのようなルートを閉じるための情報を集めることも可能だ。アクセスした者に対して何らかの対応（警告する、告発するなど）を行う場合には、その証拠として使うことができる。

ただし、そのような情報を収集するためにはオーバーヘッドが付き物であるから、どのレベルで、どこまでのデータを収集するかをよく検討する必要がある。

年金管理システムでは、これらの情報を収集するために、次の3つのレベルでの情報収集を行っている。

1）DB への個々のアクセスに対応する情報の収集

SQL Server は、このような情報を収集するために C2 監査機能[*1]を持っている。これは、DB に対するすべてのアクセスを SQL Server のプロファイル機能を利用して取得するものである。

このレベルで取得するデータ量は膨大なものになるため、取得後、すぐにフィルタリングして不要なデータを削

除する必要がある。

このレベルでは、DB に対するすべてのアクセスを捕捉することができる。また、C2 監査機能が、DB のパフォーマンスに与える影響は10%程度であり、常時設定するのに特に問題はない。しかし、C2 監査機能で出力されるログファイルは、1日に 10GB～20GB という量になり、バッチの大量更新があると 50GB を超えることもあるので、このログの退避場所には十分な容量を確保しておく必要がある。もしも、このログの退避が間に合わず、SQL Server のインストールされた場所にログが作成され続けると、いずれはディスクがいっぱいになって SQL Server が異常終了することになる。

2）DB への更新時に更新前後の情報を取得

すべての DB に、「DB更新ログ」というテーブルを設け、業務処理用に設定したテーブルに更新 （insert、update、delete） があった場合に、その更新前後の情報をこのテーブルに書き込むようにトリガ*2を設定した。

トリガの中身は、inserted またはdeleted のテーブルの内容を編集して、DB更新ログテーブルに書き込む（insert） ように構成してある。

この方法では、DB に対する検索・参照は監視の対象から外れるが、重要な （着目すべき） データの更新がいつ、どのように行われたのかを追跡することができる。また、1）と異なるのは、SQL 文レベルでの監視ではなく、更新されたデータの内容を捕捉できる点である。その一方、DB サーバーのパフォーマンスに与える影響は無視できない。対象のテーブルの構成にもよるが、影響が10%～50%になる可能性がある。また、バッチでの大量更新時に更新量を倍増させるため、さらにパフォーマンスへの影響が大きい。

DB更新ログには、更新時のデータを取得するテーブルと取得しないテーブルを区別して設定し、 業務上重要な位置づけにあるテーブルは取得し、2次的な位置づけのテーブルは取得しないこととした。

データの更新量にもよるが、DB更新ログのデータ量は1日に 1GB～2GB になるので、毎日のクリーンアップが必要になる。

3）アプリケーション機能による監査

本誌別稿「年金管理システムのITアーキテクチャ」で述べたように、オンライン入力、バッチ入力はすべてエントリー管理のクラスを通してデータが管理されている。このエントリー管理 （テーブル） には、本システムに入力されたすべての情報が蓄積されている。

これを監査証跡として使うために、業務 DB の更新が終了したエントリー管理情報を監査証跡テーブルに移動して蓄積している。また、この目的のため、データの検索など、DB 更新を伴わないオンライン処理においても、処理した内容についてエントリー管理にデータを残すようにアプリケーション機能を構成してある。

このようにすることで、アプリケーションの観点からのシステムへのアクセス情報を蓄積することができる。

3．SQL ServerのC2監査機能

SQL Server の C2 監査機能では、次のような情報を収拾することができる。

＊1）C2 監査とは、1983年にアメリカの国防総省 （DoD） が制定したTrusted Computer System Evaluation Criteria（通称「オレンジブック」に記載されたもので、http://www.radium.ncsc.mil/tpep/library/rainbow/5200.28-STD.html で参照できる）で定義されている。

定義されたクラスは、最下級のD から C1、C2、B1、B2、B3、A1（最上級）まであり、C2 は下から3番目であるが、これでもかなり厳しい基準で概略、次のようなことが要求される。

「個々のユーザーとオブジェクト（ファイルやプログラムなど）間のアクセスを定義･制御する機構を持ち、複数のユーザー間で、特定の個人･グループ （または両方） とオブジェクトを共有することを可能とする。この機構は、オブジェクトをアクセス権限のないユーザーから保護し、アクセスの許可･排除は、個々のユーザー単位に可能である。オブジェクトへのアクセス権の設定は、しかるべき権限を持ったユーザーによってのみ行われる。」

Windows NTなどのACL（アクセスコントロールリスト）を使用したファイル･フォルダへのアクセス制御や、SQL Server の DB 関連オブジェクトへのアクセス制御は、ほぼこの C2 クラスに準じていると言ってよい。

SQL Server の C2 監査機能は、上記の C2 レベルのアクセス制御にかかわるアクセス状況を記録することができるもので、C2 audit trace フラグをONにすることでSQL Server へのすべてのアクセスをログに取得することができる。

＊2）トリガ（Trigger）は、SQL文の文法に定められているRDBMSでのデータ加工の一機能である。

特定のテーブルにトリガを設定しておくと、そのテーブルに更新があった場合、そのトリガが起動されトリガの中で記述した処理内容が実行される。

SQL Server 2000では、トリガを、T-SQL という（SQL文を基礎とする）独自言語で作成するが、次期製品では、.NET系言語で作成することも可能になるといわれている。

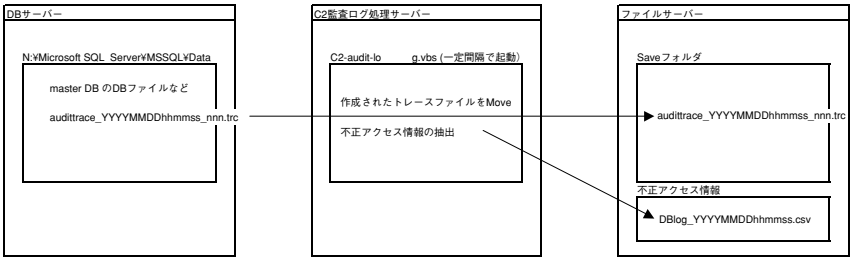

図1　SQL ServerのC2監査機能

・アクセスがあった日付・時刻
・アクセスした SQL 文
・アクセス対象となったオブジェクト名 （DB 名）
・DB のログイン名（アクセスしたアカウントが NT ドメインのアカウントの場合、ドメイン名とアカウント名）
・アクセスを行ったホスト（クライアント）名
・アクセスしたアプリケーション名
・アクセスの内容 （ログイン、ログアウト、SQL 文によるアクセスのタイプ区分）

C2監査によるログ情報は、audittrace_YYYYMMDDhhmmss_nnn.trc という名前で200MB 単位で作成される（YYYYMMDDhhmmssは、SQL Server を起動した日付･時刻、その後のnnn は、作成されたログファイルの連番）。通常の処理状況なら、5分から30分ごとに、この200MBのファイルが1つ作成される。

そこで、20分間隔で起動されるバッチプログラム（WSH で作成したスクリプト）を用意し、作成されているすべてのログファイルについて、DB サーバーから別のサーバー上のログ退避領域に移動し、その内容を解析 （1次フィルタリング）し、不正アクセスリストを作成するようにしている（図1）。

データ内容の解析は、図2に示すように SQL Server の SELECT 文で行っている。

FROM は、C2 監査ログファイルを指定する関数を利用し、WHERE の部分は、正当とみなしたアクセスの内容以外を抽出するSELECT 文になっている。

4．DB更新ログによる不正アクセス監視

DB更新ログに記録した情報は下記のとおりである。
・更新されたテーブルの全項目を文字列に変換し、‘,’で区切ってすべてを連結した文字ストリング
・更新のあった日付・時刻
・テーブル名
・“削除”、“追加”の区別
・更新処理を行った DB ユーザー名

このようなトリガを作成するために次のような機能を持ったプログラムを作成した。

```
SELECT TextData, StartTime, ObjectName, LoginName,
NTUserName, NTDomainName, HostName, ApplicationName,
EventClass, ObjectType, DatabaseName FROM ::fn_trace_get-
table('audittrace_20040428091032_123.trc', 1)
WHERE (not (NTUserName = 'S_XXXUSER' or NTUserName =
'S_XXXADMIN' or NTUserName = 'S_XXXSQLADMIN' or
LoginName = 'S_XXXExternal' or (LoginName = 'sa' and
ObjectName = 'logmarkhistory')))
```

図2　データ内容を解析するSELECT 文

・SQL Server から、DB 内のテーブル定義情報を取得する
・別途用意したテーブル一覧表に、テーブルごとにトリガを作成するかどうかの情報を持たせる
・DB から取得したテーブル名と上記の一覧表に基づき、トリガを作成する場合insert、update、deleteの各トリガ内容を組み立て、トリガ定義のSQL文を出力する
・このSQL文を使用して、DB にトリガを設定する

5．監査証跡記録機能の評価と設計上の留意点

C2 監査機能の実装については、当初懸念していたパフォーマンスに対する影響は10%ほどとそれほど大きくならなかった。また、監査機能自体も汎用的に実装できたので、他の場面で使用できる点でよかったと思う。

しかし、不正アクセスと判定する基準が難しく、この点にさらに工夫を要する点が課題である。

この実装における留意点は、随時作成される C2 監査ログをSQL Server のシステムフォルダ領域から速やかに退避フォルダに移動してやる必要があり、この処理を確実に実行させるようジョブを組む点で困難があることだ。実際に、移動させるジョブが健全に稼動しているかどうかを判定するジョブを後で追加するという修正を行って、サーバーのチェックを強化している。

DB 更新ログによる実装については、当初の目的以外にアプリケーションのデバック時にデータの更新状況を調べることができたという予想外のメリットがあった反面、DB 処理のパフォーマンスに対する影響が無視できず、本番環境では一部のDB更新ログ取得用のトリガをはずす対

応を行った。

もし、今後同様の機能を実装するなら、取得したログ情報を後で加工しやすいように、取得するログ内容・形式を考えるのが有効であろう。

アプリケーション機能による監査については、実装方法をもっと工夫する必要があると考えている。

現在の実装では、アプリケーションの作成時にコーディング上作りこまなければならない処理が多く、作成の負担が大きい。アプリケーションログの取得クラスと一緒に監査機能を検討し、共通機能としてもう一工夫必要であり、今後の課題である。

この実装で一番困った点は、監査記録を格納するときにXMLの形式にシリアライズしてDBに格納しているが、エントリー管理関連のクラスに変更があるとシリアライズされたデータは新しいクラスでは読み込めなくなってしまうことだ。そこで、クラスのバージョンが変わっても後続のバージョンで古いクラスが作成した情報を読めるようにするために、クラスの中にバージョン対応の機能を盛り込む必要があった。これは、シリアライズするデータを極小化することと、データにバージョン情報を埋め込み、デシリアライズする際に処理クラスのバージョンとデータのバージョンを判別してデシリアライズの方法を切り替えることで実現した。