Javaによるイベント認証を用いた機器制御プロトコル *

5N－1

嶺 行伸† 山崎 航† 溝口 文雄†

東京理科大学 理工学部‡

1 はじめに

インターネット，イントラネットの発達によって，セキュリティーの重要性が認識されつつある．例えば，ネットワーク内部のサービスに、ユーザーの権限に応じたアクセス制御を導入しようとすると、ユーザーが誰であるかを知り、権限を調べる仕組みが必要となる。このような問題に対して，Javaのセキュリティフレームワークである Gateway Security Model[4] が知られている．これは、カセットと呼ばれる署名されたアプリケーションだけに様々な権限を与えて、E-commerceのクライアントを拡張するフレームワークである。しかしながら，Gateway Security Modelでは、認証されているのはアプリケーションであってユーザーではない。

本研究では、ネットワークに様々なサービスがあり、ユーザー(グループ)毎に利用権限を設定する問題を扱う．そのために，分散イベント処理をベースとしたシステムにイベント発信者と受信者の認証を導入した．本稿では，権限に応じてイベントを接続する仕組みを述べる．

2 分散イベント処理API

本研究で用いている分散イベント処理は、独自に開発したAPIによって実現されている。このAPIは、JavaのDelegation Event Modelに基づいたイベントをそのまま分散化する事に主眼を置いており、イベント発信者(Event Source)と受信者(Event Listener)との間に、Event Pitcher(送信)とEvent Catcher(受信)のペアを割り込ませて、ネットワーク経由でイベントを送る。Event PitcherとEvent Catcherの機能は、次に示す通りである。

- Event Pitcherは、Event Sourceが発信したイベントをEvent Catcherに送信する
- Event Catcherは、受信したイベントをEvent Listenerに渡してイベント処理を行わせる
- Event PitcherとEvent Catcherのペアは、イベント接続管理サーバーによって互いに接続される．

本研究が対象とするのは、このAPIを拡張してEvent PitcherからEvent Catcherへのイベント送信にチケットを導入し、Event Source(ユーザー)とEvent Listener(機器)の認証を行う(アイデンティティを確認する)方法である。

3 セキュリティの方針

3.1 セキュリティモデル

本研究のセキュリティのベースとなっているのは、機器のデータの参照、コマンド送信等の権限を設定し、権限の無いユーザーからのリクエストは受け付けないというパーミッション設定型のセキュリティである。パーミッションの設定には、次の様なものがある。(1)ユーザー毎に設定する.(2)ユーザーをグループ化し、グループ毎に設定する．(3)上記2つのの混合型．我々のシステムでは、2番目のグループ毎の設定を採用した。更に、グループ間に階層構造を持たせる事により、権限のレベルが設定できる様にした。例えば、システムAのユーザーX，Aのサブシステム Bのユーザー Y、という階層的なグループ分けを行い、例えばAへのアクセスはYに許可される．これによって権限の範囲が把握し易くなる．また、ユーザーの所属するグループを変更すれば、関連する権限が一度に変更される．

3.2 セキュリティシステムの構成

チケットベースのユーザー認証システムとして、Kerberos認証システム[3]がある。本システムは、これをベースにするが，イベントの認証に適用するための変更を行った。システムの構成要素は、次の通りである。

ユーザー認証サーバー ユーザーが登録されている事を確認し、チケット交付用チケットを発行する。

チケット交付サーバー チケット交付を依頼したユーザーが属するグループを調べ、そのグループの権限で、指定されたEvent SourceからEvent Listenerへのイベント送信が可能ならば、チケットを交付して両者を接続する。

---

*

†Yukinobu MINE, Wataru YAMAZAKI, Fumio MIZOGUCHI

‡Dept.of Industrial Admin.Faculty of Sci.and Tech. Science University of Tokyo

Event Source ユーザーからのリクエストとして、チケットと共にイベントを発信する。

Event Listener チケットを調べてイベントを受信し、正しければイベント処理を実行する。

4 プロトコルの実装

プロトコルは、Kerberosのプロトコルをベースとして、公開鍵暗号方式とJavaに対応させて設計した.

本プロトコルは、認証によってイベントを接続するまでは公開鍵暗号方式を使い、接続後は、共有鍵方式(DES)のセッションキーを使う。これは、1回のイベント処理に必要な時間を減らすためには、計算時間の短い共有鍵暗号方式が有利だからである[2]. イベントの接続処理は何度も実行する訳ではないので、認証に有利な公開鍵暗号方式を採用した。

プロトコルの内容を以下に記す. なお, ASはユーザー認証サーバー, TGSはチケット交付サーバーSはEvent Source(ユーザー), Lは Event Listenerを表す.

1. SはASに、ASの公開鍵で暗号化したユーザー名(S)及び署名、オーセンティケーターを送信する.ASは、次の操作によって、Sとオーセンティケーターを確認する.
2. ASはSに、Sの公開鍵とTGSの公開鍵で暗号化した、チケット交付用チケット及びSとTGSの共有鍵を送信する.Sは、返答があった事によってASを確認し、チケット交付用チケット及びSとTGSの共有鍵を取り出す。
3. SはTGSに、取り出したチケット交付用チケット、SとTGSの共有鍵で暗号化したオーセンティケーター、Event Listener名(L)を送信する.TGSは、チケット交付用チケットを解読してSを知り、チケット内からSとTGSの共有鍵を取り出し、それを使ってLとオーセンティケーターを取り出す。
4. TGSはSに、Sの公開鍵で暗号化した、TGSとLの共有鍵で暗号化したチケット及びその共有鍵をLの公開鍵で暗号化した物、さらにSとLの共有鍵(セッションキー)、乱数を送信する. Sは、返答があった事によってTGSを確認し、チケット、セッションキー、乱数を取り出す。
5. SはTGSに、TGSの公開鍵で暗号化した乱数を送信する. TGSは乱数を解読し、Sが正しくチケットを取り出せた(本物だった)事を確認する。そして、SとLを接続する。
6. SはLに、チケット解読キーをLの公開鍵で暗号化した物を送信する. Lは、チケット解読キーを得る.
7. SはLに、暗号化されたチケット、セッションキーで暗号化したオーセンティケーターを送信する. Lは、取得したチケット解読キーでチケットを解読し、チケットからセッションキーを取り出してオーセンティケーターを解読し、確認する.

5 Kerberosとの相違

Kerberosは、発行したチケットが本物であることを2者間による鍵の共有という前提によって確認する. そのため、ユーザーやサービスが増加すると、共有するべき鍵を増やさなければならず、それら全てをサーバーが適切に管理することになる.

それに対して本システムは、認証までは全ての通信を相手の公開鍵で暗号化するため, 正当な相手でなければ通信が成立することは無く、長期に渡って共有しなければならない鍵は存在しない(共有するのはセッションキーのみ)。改竄から守られた公開鍵のデータベースが1つあれば、それで十分である。

さらに、チケット交付用チケットを取得する時、署名によってユーザーを確認するので、攻撃者によって情報が収集されるのを防止する事ができる。これは、初期のKerberosの大きな問題点であった[1]. また、オーセンティケーターを付加するので、再生攻撃にも対処できる。

6 まとめ

本論文は、分散イベント処理にチケットによるユーザー認証を導入する方法について述べた。これによって、イベント駆動型のソフトウェアコンポーネントで構成されるアプリケーションの安全性を高める事ができる。また、ユーザーのグループに階層構造を持たせることで、権限の範囲の把握を容易にするとともに、権限の設定を柔軟にする事ができる。

参考文献

[1] Steven *et.al.*, Limitations of the Kerberos authentication system: *Proc.of USENIX Conference*,1991.

[2] William R and athers. Firewalls and Internet Security. AT&T Bell Laboratories, Inc. 1994.

[3] R. M. Needham *et.al.*, Using encryption for authentication in large networks of computers. *Communications of the ACM*,1978.

[4] Ted Goldstein, The Gateway Security Model in the Java Commerce Client, Sun Microsystems, Inc. 1998.