マルチメディア通信と分散処理　97－24
コンピュータセキュリティ　8－24
（2000．3．22）

SPROMELAによる認証プロトコルの記述と検証

田中慎也　　佐藤文明
静岡大学大学院理工学研究科　静岡大学情報学部

分散システムでは認証などのセキュリティが犯されることは大変な脅威であり，認証プロトコルは，何らかの形式的な検証方式を用いて正しく設計する必要がある．そこで，本稿では状態遷移に基づいた検証ツールであるPROMELA/SPINによる認証プロトコルの検証方式を提案する．SPINを用いて認証プロトコルを検証するために，セキュリティに関する情報を扱えるようにPROMELAを拡張し(SPROMELA)，SPROMELAをPROMELAに変換するためのプリプロセッサを実装した．評価として実際にSPEOMELAを用いてOtway-Reesプロトコルを検証した結果，既知のセキュリティホールを発見することが可能であった．

Description and Verification of Authentication Protocols in SPROMELA

Shinya Tanaka　　Fumiaki Sato
Graduate School of Science and Technology, Shizuoka University　*Faculty of Information, Shizuoka University*

In a distributed system, a crack of security is a big threat. So, we have to design authentication protocols correctly. We present a new approach, verification of authentication protocols using PROMELA/SPIN that is a general verification tool based on the state transition model. To use SPIN for the verification of authentication protocols, we extend PROMELA to handle the security information. We implement the security enhanced PROMELA(SPROMELA) preproseccor which translates SPROMELA to original PROMELA. SPROMELA is used to verify the Otway-Rees protocol and evaluated to detect the known security hole.

1　はじめに

分散システムでは認証などのセキュリティが犯されることは大変な脅威である．そこで，認証プロトコルを正しく設計することは重要であり，形式的な検証方式を用いてプロトコルを厳密に検証することが必要である．

本稿では状態遷移に基づいた認証プロトコルの検証方式を提案する．本提案方式では，元来通信プロトコルの検証のために設計された言語PROMELAのインタープリタであるSPIN(Simple PROMELA INterpreter)を使用する．SPINを用いて認証プロトコルを検証するために，セキュリティに関する情報を扱えるようにPROMELAを拡張し(SPROMELA)，SPROMELAをPROMELAに変換するためのプリプロセッサを実装した．実際にSPROMELAを用いてOtway-Reesプロトコルを検証した結果，既知のセキュリティホールを発見することが可能であった．

本検証方式では，論理に基づいた検証では抽象化されてしまうようなプロトコルの詳細な動作を検証することが可能であるが，状態爆発等の問題も存在する．

2 認証プロトコルの検証技術

現在，認証プロトコルの検証にはBAN論理[1][2]などの認証の論理を用いる方法が提案され．計算機を用いた認証の論理の自動検証器[5]も存在する．

近年，AudunはSPINを用いた認証プロトコルの検証[10]について提案した．彼は，特にSPINを用いた検証の特徴や攻撃のパターン，さらにはモデルの状態数について述べていて，具体的な検証方式の提案や実現などは行っていない．さらに，本研究で用いているSPINとPROMELAについて説明する．

PROMELA[3][4]は，元来通信プロトコルを形式的に検証するために設計された言語であり，プロトコルをプロセスや通信路や変数などで表現することができる．PROMELAで記述したプロトコルは、PROMELAインタプリタであるSPIN[3][4](Simple PROMELA INterpreter)を用いて，状態遷移に基づいた様々なシミュレーションや検証を行なうことが可能である．SPINを用いた検証では，到達不可能な文の検出したり，表明文や時間的要求を用いることでデッドロックやライブロックを検出したりすることができる．

3 SPROMELAによる認証プロトコルの記述と検証

認証プロトコルは普通の通信プロトコルと比べてプロトコルの手順自体は単純であり，デッドロックなどの検出はそれほど困難ではない．しかし，認証プロトコルが本当にセキュリティを保っているかどうか検証することは困難である．本稿では後者のセキュリティの特性についての検証に注目して論じる．

3.1 検証の概要

認証プロトコルの検証の概要を図1に示す．まず，セキュリティ拡張したPROMELA(SPROMELA)で認証プロトコルを記述し，プリプロセッサを用いてSPROMELAをPROMELAに変換する．次に，SPINでプリプロセッサから出力されたPROMELAを読み込み，従来通りのSPINによる検証を行う．検証で発見されたバグを修正したり検証に与えるパラメータを変更する場合には，最上位のSPROMELAを更新し，プリプロセッサを用いて再び変換し検証を行う．このようなシーケンスを繰り返すことで認証プロトコルの設計を行う．

3.2 SPROMELAとプリプロセッサ

PROMELAは元来通信プロトコルを検証するために設計された言語であるため，セキュリティに関する情報を簡潔に表現するための構文は存在しない．そこで、認証プロトコルを簡潔に表すために必要ないくつかの構文を追加する．追加した構文を表1に示す．これらの構文は，実際はPROMELAの低レベルなデータ表現でなされている暗号化などのデータ構造をユーザに隠蔽し，抽象的なシンボルを用いた検証をユーザに許す．

SPROMELAを普通のPROMELAに変換するプリプロセッサの役割は大きく3つある．本研究では，オブジェクト指向スクリプト言語Rubyを用いてこのプリプロセッサを実装した．

- SPROMELAの構文をPROMELAに変換する．その際に，SPROMELAで記述されている通信路を攻撃者を介した通信路に置き換える．
- 検証に必要な3つのプロセスを自動生成する．その3つのプロセスとは，最初に全てのプロセスを立ち上げ認証シーケンスをスタートさせるためのinitプロセスと，セキュリティが犯されていないか認証プロトコルのシステム全体を監視するモニタプロセス，さらに，認証プロトコルに攻撃を仕掛け認証プロトコルを破ろうとする攻撃者プロセスである．
- 攻撃者の挙動や送信するメッセージ，または通信路のキューのサイズなどの様々なパラメータを検証に与える．

3.3 通信路とセッション

ネットワークを介した通信は攻撃者が盗聴することが可能であると仮定する．本提案方式では，例えばユーザAからユーザBへの通で，攻撃者はユー

表 1: SPROMELA の構文

追加した構文	構文の意味
$message_define(M, flag, ...)	プロトコルで使用するメッセージとそのメッセージを攻撃者が知っているかどうかの定義.
$session_define(...)	セッション名，メッセージ番号，送信者，受信者，攻撃者の挙動などセッションを定義する.
$security_check(...)	セキュリティが犯される条件を定義する.
$set(m, M, ...)	変数 m の値をメッセージ M にする.
$encrypt(m, K, ...)	変数 m を鍵 K で暗号化する.
$decrypt(m, K, ...)	変数 m を鍵 K で復号化する.
$nonce(m1, m2, ...)	m1 と m2 の値が等しいかどうかチェックする.
$send(A, B, N)	ユーザ A から B に N 番目のメッセージを送信する.
$receive(B, A, N)	ユーザ B から A に送信された N 番目のメッセージを受信する.

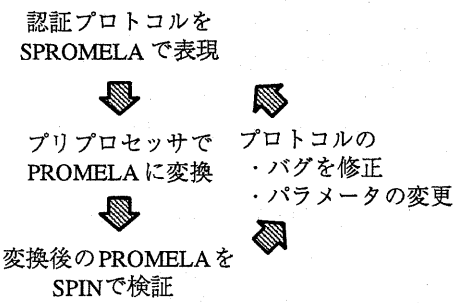


図 1: 認証プロトコルの検証の概要

```
proctype Attacker()
{
    /* ユーザ A, B 間のセッションを示す */
    byte state_AB;

    do
    :: /* ユーザ A → ユーザ B メッセージ 0 */
       (state_AB <= 0) -> state_AB = 1;

       /* メッセージの
        *   受信，復号化，記憶
        *   組み立て，送信
        */

    :: /* ユーザ A → ユーザ B メッセージ 1 */
       (state_AB <= 1) -> state_AB = 2;
            ⋮

    :: /* Give Up !! */
       (state_AB == 4) -> break
    od
}
```

図 2: 攻撃者プロセスの構成

ザ A からのメッセージを一旦受け取り，メッセージを変更する等した上でユーザ B に転送するようにする．SPROMELA では普通にユーザ A からユーザ B への仮想的な通信路で記述し，プリプロセッサによって変換することで実際はユーザ A から攻撃者，攻撃者からユーザ B への通信路で検証する．

複数の認証が並列に動作する状況を考慮して，認証の単位としてセッションを規定する．これは，例えばユーザ A からユーザ B への認証のセッション，攻撃者からユーザ A へのセッションというように用いる．攻撃者がどのセッションの何番目のメッセージを盗聴，変更するという具合に$session_define で定義することができる．

3.4 攻撃者プロセス

攻撃者プロセスの構成を図 2 に示す．攻撃者プロセスは大きな 1 つの do ループで成り立っていて，その do ループの中はセッションを示す変数で制御されたブロックによって構成されている．シミュレーションや検証では，ブロックの先頭の条件が成り立っているブロックはそれぞれランダムに選択される．また，セッションを示す変数がセッションの終わりを示していた場合，攻撃者はそのセッションに干渉することができないとみなし，do ループから抜け出るため検証はストップする．

ブロックの内部には，主にメッセージを盗聴したりメッセージを新たに生成し他プロセスに送信するなどの具体的な動作が記述される．この動作は $session_define で定義することができる．

メッセージの盗聴において，盗んだメッセージを攻撃者自身の知り得る限りの鍵で復号化を試み，復号化可能である場合には復号化した後のメッセージ，復号化不可能である場合には暗号化されたメッセージをメッセージを攻撃者プロセスのローカル変数に記憶する．これらのメッセージは今後の攻撃に使用

することができる．

メッセージの送信において，攻撃者自身が知っているメッセージと過去に盗聴し復号化不可能なメッセージの両方を組み合わせ，さらに攻撃者自身の知っている鍵で暗号化し他のプロセスに送信することができる．メッセージは，可能な限りのあらゆる組み合わせを網羅したり，固定のメッセージを送信するように指定することができる．

3.5 セキュリティが犯される条件

モニタプロセス内に以下のような条件を記述した表明文を記述し，常にシステムを監視する．これらの条件が両立した時，セキュリティが犯された状態と判断する．これらの条件は$security_checkで指定する．

- 本来のプロトコルの動作では獲得し得ないような鍵を攻撃者プロセスが獲得する．
- 正しく鍵が配送できたとみなしてユーザプロセスが認証シーケンスを終了する．

4 評価と今後の課題

本検証方式をOtway-Reesプロトコル[1][6][9]に適用し検証を行うことで評価する．Otway-Reesプロトコルには既知のセキュリティホールが存在し，攻撃をかけることが可能である．

4.1 Otway-Reesプロトコル

Otway-Reesプロトコルは，図3に示すように2人のユーザAとBと1つの認証サーバSによって行われるユーザ間の共有鍵を配送することを目的とした認証プロトコルである．

認証のシーケンスを簡単に説明する．まずAはBにメッセージを送信する．このメッセージにはAのノンスが含まれている．BはAから受け取ったメッセージにB自身のノンスなどを付け加えて認証サーバに転送する．サーバでは新しいセッション鍵Kabを生成し，A，B別々に分けてそれぞれKa，Kbで暗号化しBに送信する．次にBはサーバから返ってきたメッセージのB宛ての部分を復号化し，ノンスが以前送信したものと同じであることを確認する．そして残りのA宛ての部分をAに転送する．AはBから送られてきたメッセージを復号化しノンスを確認する．ここでノンスが正しいものだと確認できた場合には認証が成功したとみなし，KabをAB間のセッションで使用することができる．

4.2 SPROMELAによる記述

Otway-Reesプロトコルに登場するユーザが記述するプロセスとプリプロセッサが自動的に生成するプロセスを図5に示す．次にプロセス内部の記述について説明する．例としてOtway-Reesプロトコルのメッセージ0を取り挙げる。例えば図6のように，変数msg1からmsg4にそれぞれメッセージをセットし鍵Kaでメッセージを暗号化した後でBに送信するというように記述することができる．

4.3 検証 (1) 攻撃者が存在しない場合

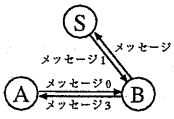
まず，攻撃者プロセスを生成しないようにプリプロセッサに指示して，攻撃者プロセスが存在しないモデルを検証する．ランダム・シミュレーションを数回繰り返すとプロトコルが問題なく予想通りに動作していることが確認できた．

変換後のPROMELAをSPINに読み込み，ランダム・シミュレーションを走らせた結果を図6に示す．縦軸は時間を表わしていて上から下に向かって時間が経過する．また四角の中の数字は探索の深さのステップ数である．次に，検証器を生成し検証を実行した．表明違反や到達不可能な文の検出など様々なフラグを付けて検証を実行したところ，特に問題となることは発見できなかった．

4.4 検証 (2) 攻撃者が存在する場合

攻撃者プロセスを生成するようにプリプロセッサに指示してPROMELAへの変換し，攻撃者が入った場合のプロトコルがうまく動作しているかどうか徹底探索を用いて検証する．本提案方式では，メッセージの組み合わせ方によって状態数が爆発的に増加し検証が困難になってしまう．そこで，試行錯誤しながら検証を行った結果，表明違反を発見することができた．発見した表明違反はファイルに保存され，後でガイド付きシミュレーションでそのエラーを再現することができる．発見した攻撃パターンを

(0) A → B : Na AB {Na}Ka {AB}Ka
(1) B → S : Na AB {Na}Ka {AB}Ka
Nb {Na}Kb {AB}Kb
(2) S → B : Na {Kab}Ka {Na}Ka {Kab}Kb {Nb}Kb
(3) B → A : Na {Kab}Ka {Na}Ka



A,B : ユーザ
S : サーバ
Na,Nb : ノンス
Ka,Kb : ユーザ・サーバ間の共有鍵
Kab : ユーザ間の共有鍵
AB : ユーザ間のセッションを示す情報
{...}Kx : ... を鍵Kxで暗号化

図 3: Otway-Rees プロトコル

(0) A → CB : Na AB {Na}Ka {AB}Ka
(0)' C → A : Nc CA {Nc}Kc {CA}Kc
(1)' A → CS : Nc CA {Nc}Kc {CA}Kc
Na {Nc}Ka {CA}Ka
(1)'' CA → S : Nc CA {Nc}Kc {CA}Kc
Na {Nc}Ka {CA}Ka
(2)' S → CA : Nc {Kca}Kc {Nc}Kc {Kca}Ka {Na}Ka
(3)' CB → A : Na {Kca}Ka {Na}Ka

A : ユーザ
S : サーバ
Na,Nc,Na1 : ノンス
Cx : Xのふりをした攻撃者
Ka,Kc : ユーザ, 攻撃者とサーバ間の共有鍵
Kca : ユーザ間の共有鍵
CA,AB : C と A, A と B のセッションを示す情報
{...}Kx : ... を鍵Kxで暗号化

図 4: Otway-Rees プロトコルの攻撃例

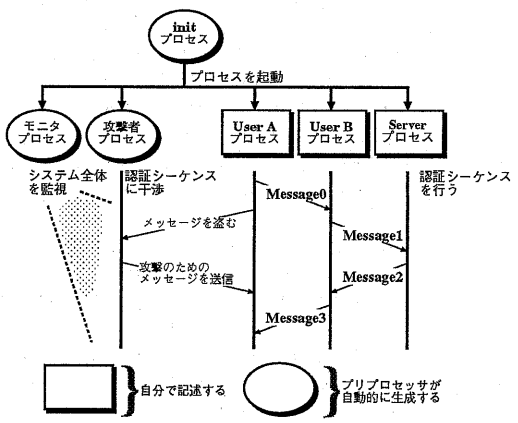


図 5: Otway-Rees プロトコルのプロセス

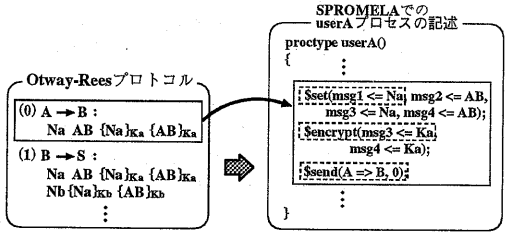


図 6: SPROMELA によるメッセージの記述

表 2: 実行環境

CPU	Pentium Pro 200MHz
メモリ	64Mbyte
OS	Linux
SPIN	Version 3.2.3

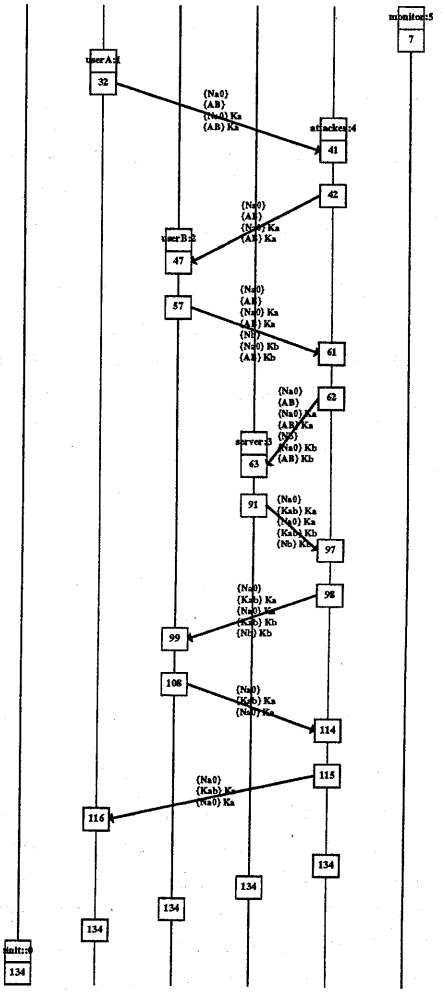


図 6: ランダム・シミュレーションの結果

図4に示す.

4.5 性能評価

今回の検証は表2のような実行環境で行った. プリプロセッサによる変換は2, 3秒で終了した. ランダム・シミュレーションは1, 2分で実行できた. 徹底探索による検証は, 検証に与えるパラメータによって大きく異なるため, 一概に言うことができないが, 概ね数～数十時間, 状態数は 10^6～10^7 程度である.

4.6 問題と今後の課題

本論文では, 指定したセッションの範囲において完全な検証が可能である. しかし, 任意のセッションについては, また完全な検証を行うことはできない.

また, 状態数はおおよそ以下の式で表されるため, メッセージの組み合わせ次第で状態数が爆発に増加し検証が困難になるという問題がある. 現在解決策として, 完全な探索ではなくなるがスーパートレースを使用したり, [11] のような状態数を減少させる方式を検討している.

$$S = C_{AB\_0} \cdot C_{AB\_1} \cdots C_{CA\_0} \cdots$$

S : 状態数
C_{S_N} : セッション S のメッセージ N で送信するメッセージの組み合わせの総数

5 おわりに

本稿では, 認証プロトコルの SPIN による検証方式を提案し, 実際に Otway-Rees プロトコルを例に取り挙げ検証を行い評価した. SPIN を用いた認証プロトコルの検証では, 認証の論理に基づいた検証では抽象化されてしまうようなプロトコルの詳細な動作を検証することができる. しかし, この検証方式は検証の程度が分りにくく, 状態爆発で検証が困難になるといった問題も存在する. これらの問題の解決が今後の課題である.

参考文献

[1] Michael Burrows, Martin Abadi, Roger Needham: logic of authentication, ACM Transactions on Computer System, Vol. 8, No. 1, pp. 18-36, February 1990.

[2] George Coulouris, Jean Dollimore, Tim Kindberg, 水野忠則, 福岡久雄, 齋藤正史, 山口義一 共訳: 分散システム コンセプトとデザイン, 1991.

[3] Gerard J. Holzmann, 水野忠則, 東野輝夫, 佐藤文明, 太田剛 共訳: コンピュータプロトコルの設計法, 1994.

[4] Spin - Formal Verification http://netlib.bell-labs.com/netlib/spin/whatispin.html.

[5] Darrell Kindred, Jeannette M.Wing: Fast, Automatic Checking of Security Protocols the proceeding of the Second USENIX Workshop on Electronic Commerce, November 1996.

[6] Lawrence C. Paulson: The Indective Approach to Verifying Cryptographic Protocols, Computer Laboratory, University of Cambridge, February 1998.

[7] Volker Kessler, Gabriele Wedel: AUTLOG – an advanced logic og authentication, the Computer Security Foundations Workshop VII, pp. 90-99, IEEE Comput. Soc., June 1994.

[8] Rajashekar Kailar: Accountability in electronic commerce protocols, IEEE transactions on Software Engineering, Vol. 22, No. 5, pp. 313-328, May 1996.

[9] Otway D.,Rees O.: Efficient and Timely Mutual Authentication, Operating Systems Review Vol. 21, No.1, pp 8-10, 1987.

[10] Audun Jøsang: Security Protocol Verification using SPIN, SPIN95, the First SPIN Workshop 95, October 1995.

[11] 若原恭, 新田文雄, 宇都宮栄二, 小田稔周, 齋藤博徳: プロセス検証とグローバル検証による高速プロトコル検証, 電子情報通信学会論文誌, B-I, Vol. J81-B-I, No. 6, pp. 378-390, 1998