Computer Security Symposium 2015
21 - 23 October 2015

形式検証を用いた攻撃分析フレームワークの提案

大久保 隆夫† 海谷 治彦‡ 鷲崎 弘宣¶ 吉岡 信和§

†情報セキュリティ大学院大学
221-0835 横浜市神奈川区鶴屋町 2-14-1
okubo@iisec.ac.jp

‡神奈川大学
259-1293 神奈川県平塚市土屋 2946
kaiya@kanagawa-u.ac.jp

¶早稲田大学
169-8050 東京都新宿区戸塚町 1 丁目 104
washizaki@waseda.jp

§国立情報学研究所
101-0003 東京都千代田区一ツ橋 2-1-2
nobukazu@nii.ac.jp

あらまし プロトコルなどの仕様の誤り，ないし仕様の誤った実装によるソフトウェア脆弱性や，脆弱性をついた攻撃が問題となっている．また，ソフトウェアを安全にするためには技術的対策だけではなく、運用時に人的な管理策が必要となる場合もある．本稿では，ソフトウェアの開発の途中段階あるいはプロトコルの実装段階を，脆弱性や外部環境に依存する形式でモデル化し，段階が進んだ時に脆弱性により攻撃が可能になるかどうかを，脆弱性や外部環境によりモデルを変化させつつ形式検証により検査するフレームワークを提案する．更に，提案フレームワークを用いれば検査の結果，開発や運用の際に防がなければならない脆弱性や管理策が明らかになることを示す．

Proposal of Attack Analysis Framework Using Formal Verification

Takao Okubo† Haruhiko Kaiya‡ Hironori Washizaki¶
Nobukazu Yoshioka§

†Institute of Information Security
2-14-1 Tsuruyamachi, Kanagawa-ku, Yokohama 221-0835, JAPAN
okubo@iisec.ac.jp

‡Kanagawa University
2946 Tsuchiya, Hiratsuka-shi, Kanagawa 259-1293, JAPAN
kaiya@kanagawa-u.ac.jp

¶Waseda University
104 Totsukacho, Shinjuku-ku, Tokyo 169-8050, JAPAN
washizaki@wasda.jp

§National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, 101-0003, JAPAN
nobukazu@nii.ac.jp

Abstract There are numbers of security incidents because of attacks exploiting vulnerability of the wrong implementation of a protocol, or because of the error of the protocol itself. Some software requires management methods for security. In this paper we propose a formal verification for checking attack potential of the intermediate state of developing software, using formal model depending on conditions or environments for security.

1　はじめに

ソフトウェアの開発時に，その仕様の安全性(セキュリティ）を評価することは重要であるが，たとえ要求が安全であったとしても，設計や実装で，脆弱性が作りこまれると，完成したソフトウェアは元々の要求仕様を満たせないことになる．しかし，このような後工程における脆弱性の発生を予期することの困難さが，要求策定時や設計時にある仕様の完成時セキュリティリスクを評価する困難さの原因となっている．また，運用時においては，システム外の人的管理策の不備が脆弱性と同様に，セキュリティ事故リスクの原因となるため，システム全体の運用時リスクはシステム安全性と管理策等外的要因の双方を考慮に入れる必要がある．

本稿では，形式検証を用いて，ソフトウェア開発時や運用時のセキュリティを評価するフレームワークを提案する．フレームワークは，次の２点の機能を持つ．

- 要求仕様の形式記述，および実装仕様または管理策の形式記述を読みこみ，攻撃により実装仕様または管理策が要求仕様に違反していないかを検証する
- 実装仕様または管理策に脆弱性状態を与え，攻撃が可能か検証する

本稿の貢献は，上記の機能により，開発者，運用者が要求仕様に基づいて選択した実装仕様やアーキテクチャ，管理策に問題がないかを，アーキテクチャに脆弱性の仮定を置き，攻撃の成功の可否を形式検証により検証することで，攻撃実現可能性によるリスク評価を行うことができる点にある．

筆者らは提案したフレームワークをSPIN/Promelaによるモデル検査を用いて実装し，認証認可の仕様とそれに対する既知攻撃のモデル化および検証ができることを確認した．

2　背景と関連研究

セキュリティにおいて，ソフトウェアの設計，実装や運用を行う場合，それがセキュリティの要求仕様を満たしているかという問題は，2つに分けられる．1つは，ソフトウェアの要求仕様が設計，実装に反映されているかというトレーサビリティの問題，もう1つは，設計や実装が脆弱性を持たない，すなわち，想定される攻撃に対して耐性を持つかという問題である．特に後者の問題を扱うためには，セキュリティリスクの評価が重要になってくる．

ソフトウェアシステムのセキュリティリスクは，そのソフトウェアが包含する脆弱性に依存する．このため，脆弱性データベース CVE などにおいても，そのリスク評価に共通脆弱性評価システム (CVSS) という脆弱性を評価する手法が用いられている．また，リスクは同様に稼動しているシステムの場合，その運用方法にも依存する．このようなシステムの外的条件は，ソフトウェアシステムを評価するコモンクライテリア (CC) においても，運用環境による前提条件を基本設計書に記載する項目として表われている [1].

また，CC 中の保証レベル (EAL)4 以上においては，設計，実装の妥当性を検証する手段として形式検証が要求されており，ソフトウェアのセキュリティ実現の妥当性検証手段として形式検証を用いる技術や研究は数多く行われている [2][3]．しかし一方，脆弱性が存在する時にそれがどのようなリスクの影響を与えるかという観点で形式検証を用いる研究は少ない．阿部らは，シーケンス図と攻撃パターンをもとに脅威を抽出する手法を提案している [4] が，抽出した脅威のリスク評価までは行っていない．

原田らは，ネットワーク構成を用い，ネットワークシステムの脆弱性の影響範囲を測定する手法を提案している [5]．原田らの提案手法では，情報漏洩や改ざんなどの脅威の性質に応じて影響範囲を 1 意に決定している．しかし，情報漏洩や改ざんといった事象は既に攻撃を受けてしまった結果であるから，その攻撃が可能であるかという評価ではない．

ソフトウェアの脅威を詳細化し，リスクを分析する手法としては，故障木分析 (Fault Tree Analysis: FTA) がある [6]．FTA では故障の原因となる各事象の故障率を用いて，全体の故障

率を算出している．しかし，各事象が 1 つの攻撃に相当する場合，その攻撃の成功率が原因の分解だけでは困難な場合もあるため，FTA によるリスク評価には限界があると考えられる．

また，ソフトウェア開発において，設計時やアーキテクチャ構成時におけるそのセキュリティリスクの評価を行う場合，リスク評価の精度が問題となる．設計仕様やアーキテクチャが不明確であるほど，具体的な攻撃手法の特定やその成功可能性の評価が困難になるためである．大久保らは，この問題を解決するため，従来のウォーターフォール型開発のように要求分析完了後に設計，アーキテクチャ分析を行うのではなく，両者を交互に詳細化する手法を提案している [7][8]．しかし，この手法においても，依然として，あるセキュリティ仕様を適切に満たす設計仕様やアーキテクチャを選択するため，これらのリスク評価が必要となる．

筆者らは，まず，リスク値を単純に (脅威の発生確率×被害の大きさ) で算出するのではなく，その状況においてある攻撃が実現可能かという項を追加し，下記の式を用いることとした．

リスク値＝脆弱性の発生確率×攻撃成功可能性×被害の大きさ

ここで，攻撃成功可能性は，攻撃が可能または不可能のバイナリ値をとる．この計算式を用いれば，従来特定が困難であった攻撃の確率の問題を回避し，脆弱性の可能性を扱えばよい．また，ある状況において具体的に攻撃が可能であることを示すことは，システムのステークホルダに，より現実的なリスクとして認識してもらえる効果があると筆者らは考えた．

3　提案フレームワーク

前節で述べたリスク評価式算出のため，筆者らはある脆弱性や運用，管理策の不備が存在する時，それらが原因で想定される攻撃が可能になるかを検証するフレームワークを提案する．フレームワークの構成を図 1 に示す．フレームワークは要求仕様，設計/実装仕様および脆弱性/運用，管理策不備の条件を入力とし，形式検証エンジンを用いて攻撃可能性の検証を行い，結果を出力する．

- 要求仕様
  要求仕様は，要求策定者によって記述された形式記述で与えられる．要求仕様は，正常系のふるまいの他に，攻撃などの結果，セキュリティ上起きてはいけないことを宣言として記述する．

- 実装仕様
  実装仕様は，要求仕様を実現する具体的な設計，実装の仕様である．実装仕様についても，その実装担当者が形式記述を行う．ある要求に対する実装仕様は複数の方式を持つ可能性があるが，フレームワークに入力できるのは 1 回に 1 つの方式のもののみである．
  実装方式は，データフロー図のように，コンポーネント間や外部エンティティ(人など)，データストアのデータの流れとして記述される．

- 脆弱性条件
  ここにおいて，脆弱性条件とは，ソフトウェアの脆弱性，人的脆弱性 (運用時の管理策の不備など) の双方を指す．脆弱性条件は，実装仕様上の各要素 (通信路, データストア, コンポーネント) において，セキュリティ上脅威となる条件の発生する可能性，例えば，情報漏洩や改ざんなどが起きる条件が記述されている．

- 攻撃モデル生成器
  攻撃モデル生成器は，脆弱性条件を入力し，検証するための具体的な攻撃のふるまいをするモデル (形式記述) を生成する．この際，攻撃データベースを参照し，既知の攻撃の種類のいずれかを選択しその攻撃に応じた攻撃モデルを生成する．

- 攻撃モデル
  攻撃生成器によって生成された攻撃を行う形式記述モデル．

- 検証器
  要求モデル，実装モデル，攻撃モデルを入力し，要求モデルの条件に違反する反例が

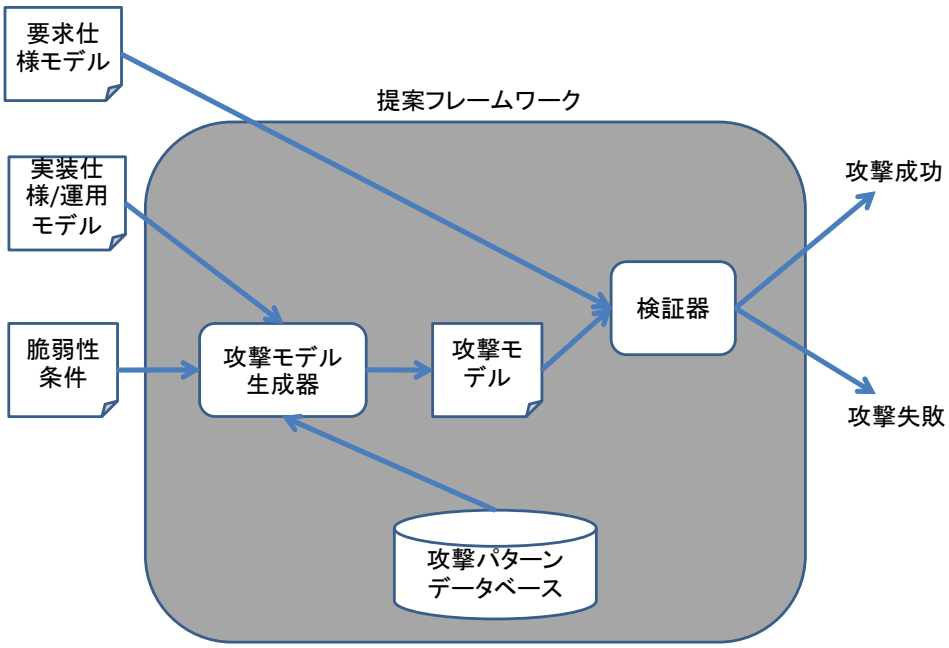


図 1: 提案フレームワークの構成

発生しないかを形式検証ツールによって検証する．反例が発生した場合，それは攻撃が成功したことを意味し，反例が発生しなければ，攻撃は成功しなかったと見なす．

- 攻撃データベース
  既知攻撃の種類ごとにそのふるまいを格納したデータベース．

4 フレームワークの利用と期待される効果

提案フレームワークの利用方法と，利用による効果を次に示す．

4.1 開発時

提案フレームワークを用いたリスク評価の流れを図 2 に示す．

要求策定後，設計に進む際に利用者は，いくつかの設計仕様の選択肢の中からどれかを選ぶが，それぞれの選択肢にどのようなセキュリティリスクがあるか分からないとする．その際，開発者は，要求仕様モデルおよび選択した設計の実装仕様モデルを用意し，想定される攻撃ごとに，攻撃が成功する可能性があるか，フレームワークを用いて検証する．

また，実装仕様を検証する際に，設計仕様中のアークテクチャ要素，例えばデータストアや通信路などに脆弱性が存在することが，フレームワークを用いた別の攻撃の検証によって判明している場合，その脆弱性を条件として入力し，検証を行う (図 2(a) の流れ)．

このように，ある箇所の脆弱性について，再帰的に攻撃可能性を検証することで，精度の高い評価を行うことが可能となる．

一方，要求策定直後では，これから設計実装するコンポーネント上の脆弱性が不明である場合が想定される．この場合は，そのコンポーネント上での脆弱性について仮説を立てた上で検証を行う．例えば，あるデータストア上でデータが漏洩する確率が 1%存在するという仮説を立てる．その仮説を前提として，フレームワークによる検証を行う (図 2(b) の流れ)．この場合，脆弱性条件が成立した時，攻撃が成功するかどうかが分かる．仮に攻撃が成功するとすると，上記の例では 1%の確率で攻撃が成功することになる．仮に，2 つの実装仕様双方で，あ

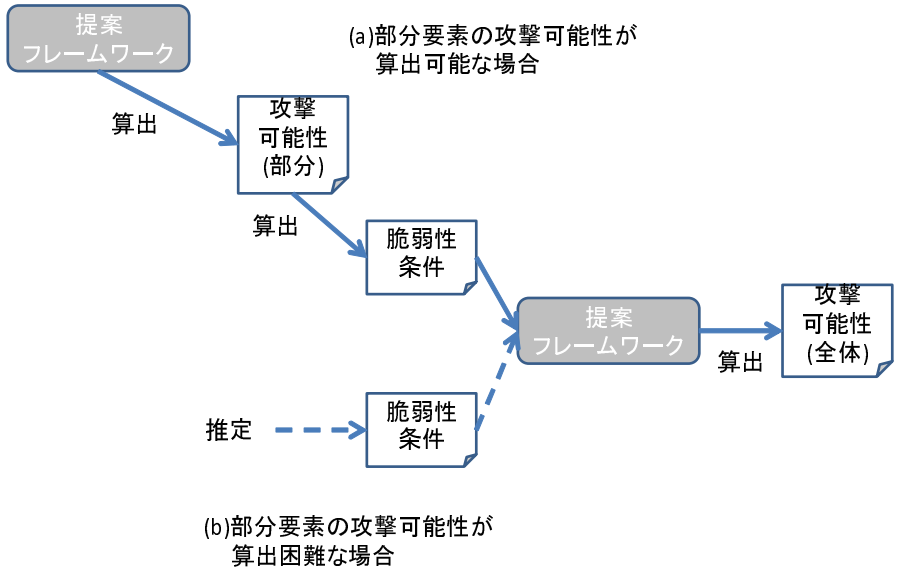


図 2: 提案フレームワークを用いたリスク評価

る脆弱性条件において攻撃が成功し，かつ，実装によってで脆弱性の発生確率が異なるとすれば，開発者はフレームワークによる検証を行った後で，脆弱性の発生確率の低い実装を選択することができる．

また，脆弱性の発生確率が高い脆弱性について攻撃が成功したとすると，そのソフトウェアは完成時にその攻撃によって被害にあう可能性があることになる．このリスクを低減させるには，脆弱性の発生を抑制する実装が必要になる．したがって開発者は，選択した実装を進める際に，脆弱性発生を抑制させる制約をルールとして設定する．これは，通常セキュリティの設計規約ないしコーディング規約として認識されるものである．このように，開発者は事前に開発対象ソフトウェアの弱点を認識した上で，リスクを高めない規約をフレームワークの出力を利用して定めることができる．

4.2 運用時

運用時においても，運用者は開発時と同様にフレームワークによるリスク評価を行うが，システムの脆弱性の代わりに，人的原因による脆弱性 (管理策の不備など) を用いる．また，脆弱性の発生確率が高い脆弱性について攻撃が成功した場合，設計/コーディング規約ではなく運用の際の規約や管理策として，脆弱性発生確率を低減させる施策を実施する．

提案フレームワークを用いた開発，運用のイメージをアクティビティ図で表したものを図 3 に示す．

5 提案フレームワークの実装

筆者らは，提案フレームワークの実効性を確認するため，フレームワークの試作を行った．また，一般的な利用者認証，認可を検証の対象とし，想定される攻撃としては認証，認可に対する既知の攻撃を用意した．

試作フレームワークでは，検証器に定理証

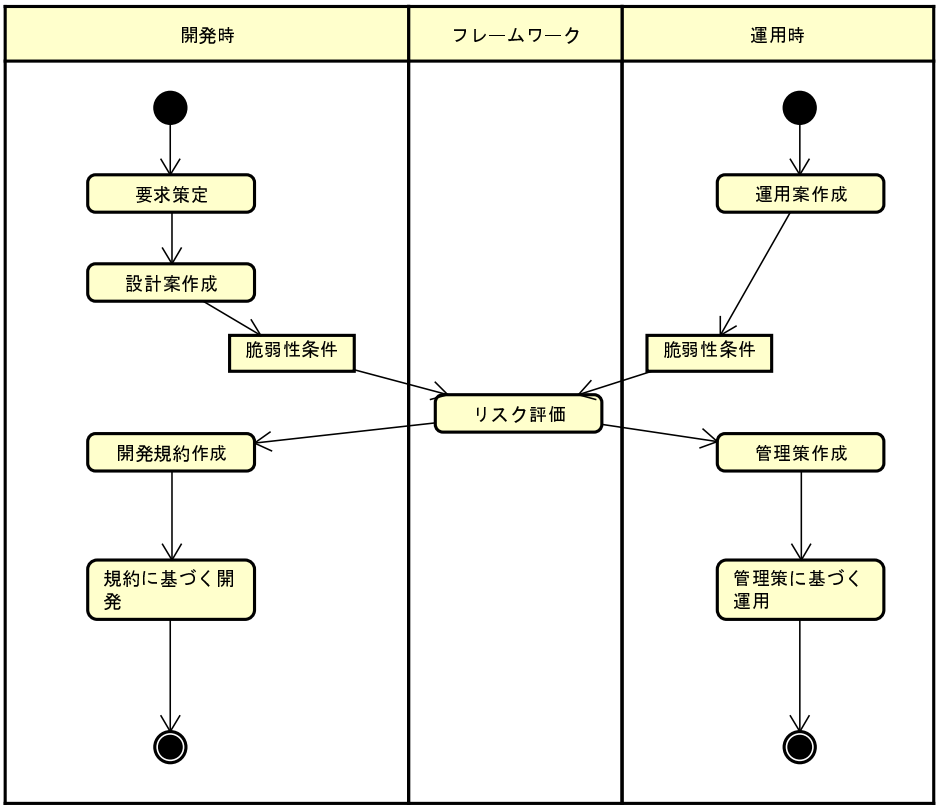


図 3: 提案フレームワークを用いたリスク評価

明系ではなく，モデル検査を用いることとし，SPIN(Windows, ver.6.4.3) による実装を行った．SPIN を採用した理由は，検証対象の認証のプロトコルがアクターやコンポーネントの間のデータの通信によって行われるため，SPIN 記述言語の PROMELA によって容易に記述できること，要求仕様を assert によって宣言記述し，その違反により攻撃成功検証が可能であること，および，状態を網羅できるので，攻撃のバリエーションの検査に適していると判断したことである．

また、攻撃生成器は Java を用いて実装した．

5.1 要求仕様の表現

認証，認可における要求仕様は，credential を所持する者 (US) が認証サーバ (RZ) から認証された後，リソースサーバ (RS) によりリソースへのアクセスを認可されるとすると，それぞれ次の条件を満たすものであると定義できる．

- 正当な credential を所持していない US は認証されてはならない．
- AZ によって認証されていない US，またはリソースにアクセス権限のない US はアクセスを認可されてはならない．

上記の条件は，条件を満たさないユーザを US’ とすると，次のような assert として記述することにより，検査時に SPIN が，違反があった場合に検出することができる．なお，authenticated()，authorized() はそれぞれ，引数のユーザの認証，認可の可否について true または false を返すものとする．

```
assert(!authenticated(US'))
assert(!authorized(US'))
```

5.2 実装仕様

ブラウザ (CL) を用いて，認証を行う場合を PROMELA によって記述できる．

CL(を使うユーザ) は自身の credential をもって，AZ に認証を要求する．AZ はその結果を，セッショントークンとともに返す．CL は受けとったセッショントークンを CL 内に保存しておき，RS へのリソース要求時に添付して送る．RS はセッショントークンにより認証を受けているユーザが存在し，かつ要求されているリソースへのアクセス権が存在する場合のみ，アクセスを認可する．

上記の CL，AZ，RS 間のやりとりは Promela のチャネルを使い実現可能である．また，CL のセッショントークンを保存するストア (ブラウザの cookie に相当) をテーブルで表現する．

5.3 脆弱性条件

次の脆弱性条件を考える．

- 取得したセッショントークンを第三者が参照可能な状態 (セッションハイジャック)

5.4 攻撃モデル

攻撃モデルは，正規 credential を持たないユーザ (CLSHJ とする) をまず実装条件に追加する形で用意する．次に脆弱性条件の表現として，正規 UA が取得したセッショントークンのテーブルをグローバル変数化し，CSHJ が参照し，取得したトークンにより認可要求を送信できるものとする．

5.5 検証

実装仕様の PROMERA 記述と攻撃モデルの Promela 記述を結合し，モデル検査を行う．タイミングによっては，正規ユーザ CL の認証によるセッショントークン取得後に CLSHJ がセッショントークンを取得し，認可要求により認可を受けることに成功する．この場合，要求仕様に挙げた assert に違反するため，SPIN は assert に違反する反例を出力する．反例の出力により，フレームワークは攻撃が成功したと判断する．

実装モデルと攻撃モデルを結合した Promela を図 4 に示す．

6 おわりに

筆者らは，ソフトウェアのある状況 (脆弱性，管理策不備) の想定下において，攻撃の成功可能性を形式検証により検証するフレームワークを提案した．また，フレームワークのプロトタイプの実装を行い，効果について確認した．

提案フレームワークは，リスク評価をある一定条件下での攻撃成功可能性によって方針に基づき，脆弱性条件を入力として，実装仕様の要求仕様に対する違反を攻撃の成否で検出する．

提案フレームワークを用いることにより，開発者，運用者は設計時や運用時に，脆弱性の影響評価をすることができ，その情報をもとに適切な実装や管理策を選択することが可能になる．また，検証結果をもとに，リスクを低減させるための設計規約や新たな管理策を設定することができるようになる．

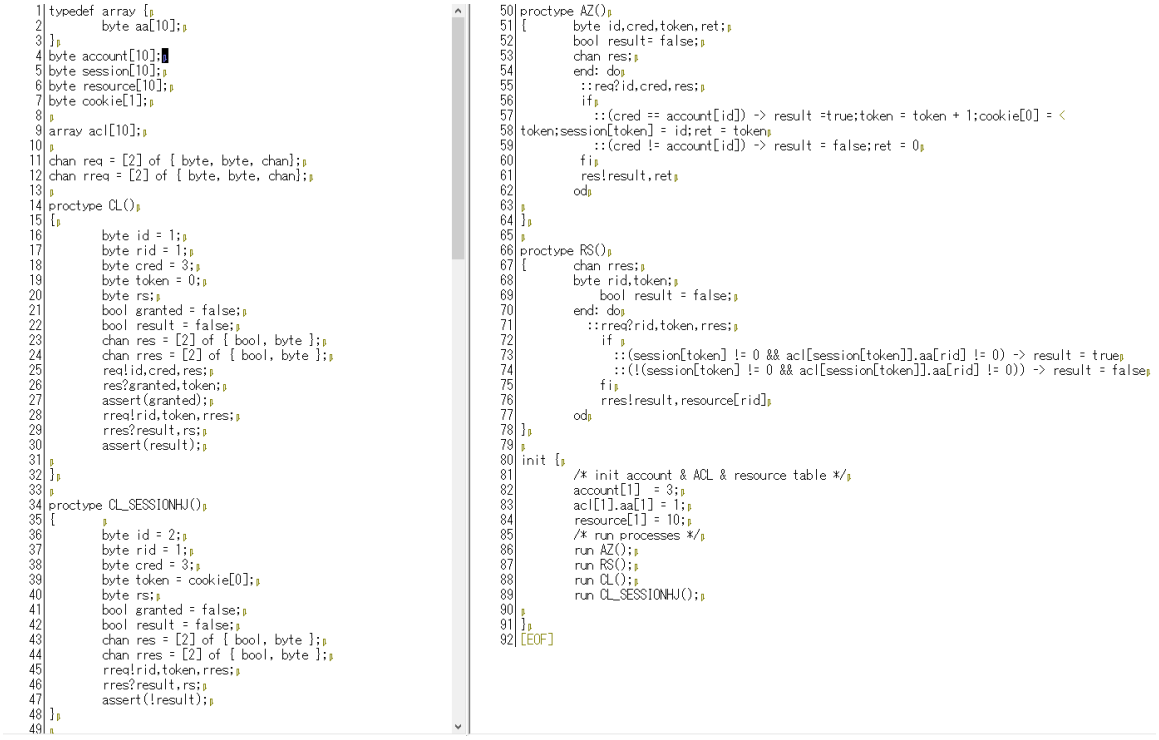
本稿では，主に認証問題を対象にのプロトタイプ実装と評価を行った．しかし認証問題はセキュリティ全体においては一部の問題であり，今回のフレームワーク実装をすべてのセキュリティ問題に適用できるかどうかは，今後解決すべき課題である．また，効果で上げた多段階攻撃の，標的型攻撃などの実例での評価などについても今後の課題となる．

謝辞

本研究を進めるにあたって，株式会社富士通研究所の協力をいただきました．ここに謝意を表します．

参考文献

[1] NIST: Common Criteria, "`http://csrc.nist.gov/cc/` (last accessed August 24, 2015)".



```
typedef array {
        byte aa[10];
};
byte account[10];
byte session[10];
byte resource[10];
byte cookie[1];

array acl[10];

chan req = [2] of { byte, byte, chan};
chan rreq = [2] of { byte, byte, chan};

proctype CL()
{
        byte id = 1;
        byte rid = 1;
        byte cred = 3;
        byte token = 0;
        byte rs;
        bool granted = false;
        bool result = false;
        chan res = [2] of { bool, byte };
        chan rres = [2] of { bool, byte };
        req!id,cred,res;
        res?granted,token;
        assert(granted);
        rreq!rid,token,rres;
        rres?result,rs;
        assert(result);

}

proctype CL_SESSIONHJ()
{       
        byte id = 2;
        byte rid = 1;
        byte cred = 3;
        byte token = cookie[0];
        byte rs;
        bool granted = false;
        bool result = false;
        chan res = [2] of { bool, byte };
        chan rres = [2] of { bool, byte };
        rreq!rid,token,rres;
        rres?result,rs;
        assert(!result);
}

proctype AZ()
{       byte id,cred,token,ret;
        bool result= false;
        chan res;
        end: do
         ::req?id,cred,res;
         if
           ::(cred == account[id]) -> result =true;token = token + 1;cookie[0] = <
token;session[token] = id;ret = token
           ::(cred != account[id]) -> result = false;ret = 0
         fi
         res!result,ret
        od
}

proctype RS()
{       chan rres;
        byte rid,token;
            bool result = false;
        end: do
          ::rreq?rid,token,rres;
            if
              ::(session[token] != 0 && acl[session[token]].aa[rid] != 0) -> result = true
              ::(!(session[token] != 0 && acl[session[token]].aa[rid] != 0)) -> result = false
            fi
            rres!result,resource[rid]
        od
}

init {
        /* init account & ACL & resource table */
        account[1]  = 3;
        acl[1].aa[1] = 1;
        resource[1] = 10;
        /* run processes */
        run AZ();
        run RS();
        run CL();
        run CL_SESSIONHJ();

}
[EOF]
```

図 4: 実装、攻撃モデル例

[2] 岩本智裕，荒井研一，金子敏信:ProVerif による Theft DoS Attack に耐性のあるワンタイムパスワード認証方式の形式的検証，暗号と情報セキュリティシンポジウム (SCIS)2015 (2015).

[3] 吉田真紀，水野修:暗号プロトコル安全性検証の可視化に向けて，暗号と情報セキュリティシンポジウム (SCIS)2015 (2015).

[4] 阿部達也，林　晋平，佐伯元司:シーケンス図のパターンに基づくセキュリティ脅威の検出法 (再利用とプログラミング)，電子情報通信学会技術研究報告. SS, ソフトウェアサイエンス，Vol. 113, No. 24, pp. 1–6 (2013).

[5] 原田敏樹，金岡　晃，加藤雅彦，勝野恭治，岡本栄司:ネットワークシステムにおける脆弱性影響の測定手法とシステム実装，情報処理学会論文誌，Vol. 52, No. 9, pp. 2613–2623 (2011).

[6] Leveson, N. G. and Harvey, P. R.: Software fault tree analysis, *Journal of Systems and Software*, Vol. 3, No. 2, pp. 173–181 (1983).

[7] Okubo, T., Kaiya, H. and Yoshioka, N.: Mutual Refinement of Security Requirements and Architecture Using Twin Peaks Model, *COMPSAC Workshops*, pp. 367–372 (2012).

[8] Okubo, T., Yoshioka, N. and Kaiya, H.: Requirements Refinement and Exploration of Architecture for Security and Other NFRs, *Advanced Information Systems Engineering Workshops - CAiSE 2014 International Workshops, Thessaloniki, Greece, June 16-20, 2014. Proceedings*, pp. 286–298 (2014).