モデル検査器を用いたプロトコルの DoS 攻撃耐性解析

Resistance analysis of DoS attacks against protocols, using SPIN model checker

池田 立野 †, 西崎 真也 †
Ritsuya IKEDA, Shin-ya NISHIZAKI

† 東京工業大学大学院情報理工学研究科
Graduate School of Information Science and Engineering, Tokyo Institute of Technology
ritsuya@lambda.cs.titech.ac.jp, nisizaki@cs.titech.ac.jp

インターネットが広く使用されるにともなって，Denial-of-Service 攻撃（DoS 攻撃）が問題となってきている．DoS 攻撃は正当な利用者の活動を妨害する攻撃であり，ソフトウェアの欠陥やプロトコルの脆弱性の悪用する方法が主流である．Spice 計算は DoS 攻撃を形式的に検証する形式体系である．Spi 計算にコスト計測機能を追加した体系である．暗号化処理やデータの送受信処理に対して必要となるコストを定めている．プロセスのリダクションをおこなうと処理に必要とされるコストが発生する．その結果を検証することによって DoS 攻撃耐性を解析する．これまでの Spice 計算は理論的な単なる枠組みであり，解析手法と DoS 耐性に関する性質を記述する方法がなかった．そこで，本研究では Spice 計算を使用した解析手法を提案する．この解析手法は，モデル検査器 Spin を使用する．Spice 計算でプロセスを記述し Spin のモデルに変換し，LTL 式で表現した DoS 攻撃耐性に関する性質を Spin で検証する．最後に，変換系のプロトタイプを作成し，簡易な例による実験を紹介する．

1 はじめに

1.1 DoS 攻撃

インターネットが広く使用されるにともなって，Denial-of-Service 攻撃（DoS 攻撃）が問題となってきている．DoS 攻撃は正当な利用者の活動を妨害する攻撃である．主要な攻撃方法として次のようなものがある．

- ソフトウェアの欠陥を悪用する．バッファーオーバーランなどが要因となる．
- プロトコルの脆弱性の悪用する．

また，1 台の計算機を使用してこれらの攻撃を行うだけでなく，複数の計算機を使用する分散 DoS 攻撃（Distributed DoS attack, DDoS 攻撃）もある．

本研究では，プロトコルの脆弱性を悪用する DoS 攻撃に焦点をあてる．攻撃者はプロトコルの次の特徴を悪用する．

- クライアントは接続要求パケットを少ない資源（CPU，メモリ）で容易に送信できる
- サーバはクライアントからのパケットの処理に資源（CPU，メモリ）を消費する

このようなプロトコルでは，攻撃者はサーバの資源を浪費させることが可能となる．攻撃の例としては，偽造された要求を使用する SYN あふれ攻撃 [2]，正当な要求を使用する Naptha 攻撃 [3]，SSL/TLS への攻撃がある．

1.2 Spice 計算

Spice 計算は DoS 攻撃を形式的に検証する体系 [8, 9] であり，Spi 計算にコスト計測機能を追加した体系である．暗号化処理やデータの送受信処理に対して，それに必要となるコストを定めている．プロセスのリダクションをおこなうと，プロセスの書き換えが起こるとともに，処理に必要とされるコストが発生する．プロトコルの形式化・検証は次の手順でおこなわれる．

- プロトコルに参加するプリンシパルごとにプロセスを記述する（サーバプロセス，クライアントプロセス，攻撃者プロセス等）．
- システム全体を各プロセスを並列合成したものとして表す．そして，プロセス全体を記述したプロセスのリダクションをおこなう．
- リダクションの結果発生したコストを検証する．コストはプロトコルに参加する計算機ごとに計測される．

このようにして攻撃者・被害者のコストの計測をおこない，プロトコルの DoS 耐性を検証する．

Spi 計算との差異として，使用した変数は明示的に解放しなければならない点がある．こうすることによりメモリ使用のコストが明示的に扱える．

1.2.1 Spice 計算の定義

ここでは Spice 計算の定義の抜粋を述べる．定義の詳細は論文 [8, 9] を参照のこと．

n を名前，x を変数とする．値 V は n,x,$hash_V$ などで評価後の結果を表す．項 M は V, $(M+N)$, $\mathbf{hash}(M)$ などである．

プロセスは次のように定義される．

P,Q,R	::=	プロセス
	$\mathsf{out}\ M_{port}\ \langle N_{msg}\rangle; P$	出力
	$\mathsf{inp}\ M_{port}\ (x); P$	入力
	$(P \mid Q)$	並列合成
	$\mathsf{new}\,(n); P$	名前制限
	$\mathsf{repeat}\,P$	複製
	stop	空
	$\mathsf{store}\ x = M; P$	メモリ割当
	$\mathsf{free}\ x; P$	メモリ開放
	$\mathsf{match}\ M\ \mathsf{is}\ N\ \mathsf{err}\{P\}; Q$	マッチング
	$\mathsf{split}\ [x_1,\ldots,x_n]\ \mathsf{is}\ M\ \mathsf{err}\{R\}; P$	

出力と入力は名前 M_{port} を使用しての通信を表している．合成は合成はプロセスの合成を表している．名前制限は名前 n の範囲を P に制限している．複製はプロセスの複製を表している．空はプロセスの停止を表している．メモリ割当と解放は変数 x によるメモリの割当と解放を表している．マッチングは M と N の比較を表している．成功すれば次に Q を実行し失敗した場合は P を実行する．ペア分解は M のペア分解を表している．成功すれば次に Q を実行し失敗した場合は P を実行する．

項の評価は，項 M，値 V，コスト値 c とした $M \downarrow V : c$ で定義されている．規則の例を次に示す．

$$\frac{M \downarrow V : c}{\mathsf{hash}(M) \downarrow hash_V : c + hash}$$

この規則はハッシュ関数を実行するとコスト $hash$ が発生していることを表している．

プロセスの簡約は，プロセス P,Q，コスト値 c とした $P > Q : c$ で定義されている．規則の例を次に示す．

$$\frac{M \downarrow V : c, \quad fv(V) = \emptyset}{\mathsf{store}\ x = M; P > P[V/x] : c + store_x}$$

この規則はメモリ割当を実行するとコスト $store_x$ が発生していることを表している．

プロセス間の複簡約は型 $\mathcal{A}$，プロセス P,Q，コスト割り当て σ とした $\mathcal{A} \vdash P \gg Q : \sigma$ で定義されている．規則の例を次に示す．

$$\frac{\mathcal{A} \vdash P \gg P' : \sigma_1 \quad \mathcal{B} \vdash Q \gg Q' : \sigma_2}{(\mathcal{A} \mid \mathcal{B}) \vdash (P \mid Q) \gg (P' \mid Q') : \sigma_1 + \sigma_2}$$

この規則は部分プロセスで発生したコストの和がプロセス全体のコストとなることを表している．

プロセスのコミットメントは $\mathcal{A} \vdash P \xrightarrow{\alpha} A : \sigma$ で定義されている．規則の例を次に示す．

$$\frac{}{\mathbf{a}{::}\mathcal{V} \vdash \mathsf{inp}\ n\ (x); P \xrightarrow{n} (x)P : \{\mathbf{a} \cdot store_x\}}$$

この規則は名前 n でデータを受信して変数 x に格納し計算機 $\mathbf{a}$ にコスト $store_x$ が発生していることを表している．

複コミットメントは型 $\mathcal{A}$，プロセス P,P'，コスト割り当て σ として $\mathcal{A} \vdash P \longrightarrow P' : \sigma$ で定義されている．

1.2.2 Spice 計算の例

ここでは Spice 計算を使用した検証の例として，TCP に対する SYN あふれ攻撃を紹介する．

最初に攻撃者プロセスを記述する．

$$\begin{aligned} P_A \stackrel{def}{=}\ & \mathsf{new}\,(S_A); \\ & \mathsf{store}\ x_{sa} = S_A; \\ & \mathsf{out}\ c\ \langle (A, B, x_{sa})\rangle; \\ & \mathsf{free}\ x_{sa} \\ & P'_A, \end{aligned}$$

このプロセスはスリーウェイハンドシェイクの 1 番目のステップである SYN の送信だけをおこなう．

次に被害者プロセスを記述する．このプロセスは通常の TCP サーバである．

$$\begin{aligned} P_B \stackrel{def}{=}\ & \mathsf{new}\,(S_B); \\ & \mathsf{inp}\ c\ (q_1); \\ & \mathsf{split}\ [y_a, y_b, y_{sa}]\ \mathsf{is}\ q_1\ \mathsf{err}\{\mathsf{free}\ q_1\}; \\ & \mathsf{free}\ q_1; \end{aligned}$$

match y_b is B err{free y_a, y_b, y_{sa}};
free y_b;
store $y_{sb} = S_B$;
out $c\ \langle(B, y_a, y_{sb}, \mathsf{succ}(y_{sa}))\rangle$;
inp $c\ (q_2)$;
split $[y'_a, y'_b, y'_{sa1}, y'_{sb1}]$ is q_2
  err{free y_a, y_{sa}, y_{sb}, q_2};
free q_2;
match y'_b is B and
y'_a is y_a and
y'_{sa1} is $\mathsf{succ}(y_{sa})$ and
y'_{sb1} is $\mathsf{succ}(y_{sb})$
err{free $y_a, y_{sa}, y_{sb}, y'_b, y'_a, y'_{sa1}, y'_{sb1}$};
free $y'_b, y'_a, y'_{sa1}, y'_{sb1}$;
P'_B

2 行目で SYN を受信，8 行目で SYN-ACK を送信，9 行目で ACK を受信する．スリーウェイハンドシェイクにより TCP のコネクションを生成している．

この 2 つのプロセスを合成して型をつけたシステムは次のものになる．

$$(\mathbf{a}\,|\,\mathbf{b}) \vdash P_A|P_B$$

リダクションをおこなうと次の結果を得る．

$$P_A|P_B \gg P'_A|\,\mathsf{out}\ c\ \langle\ldots\rangle;\ldots : \{\mathbf{b}\cdot 2store\}$$

攻撃者はコストが不要なのに対して，被害者はメモリを消費している．このことから，攻撃者は容易に被害者のメモリを浪費できることがわかる．

1.3 Spin モデル検査器

Spin は Holzmann らによって開発されたモデル検査器 [5] である．ωオートマトン上のモデル検査技法を使用している．モデルは Promela と呼ばれる仕様記述言語で記述する．Promela は，一種の並行プログラミング言語であり，チャンネルを用いたプロセス間の送受信が提供されている．また，変数の代入文など，文一つ一つはアトミックとなっている．

モデルの性質を記述する方法がいくつかある．

アサーション ある状態で満たすべき条件である．

サイクル ある状態が無限回実行されることを示す．

Never claim なってはいけない状態を受理状態とするオートマトンである．Linear Temporal Logic (LTL) の論理式から生成することもできる．

1.4 関連研究

SYN あふれ攻撃の防御策として SYN cookie[4] が普及している．これはサーバ側の初期シーケンス番号をランダムに生成するのではなくクライアントの情報とサーバの秘密情報から生成したハッシュ値を使用する．こうすることによってサーバはクライアントの情報を保持する必要がなくなりメモリ浪費を避けることが可能になる．

一般に，DoS 攻撃耐性を向上する手法として，プロトコルを拡張して DoS 耐性を付加する Client puzzle[1] が提案されている．サーバに負荷がかかっているときにはクライアントに暗号学的なパズルを解くことを要求する．クライアントの計算量を増やすことにより DoS 攻撃を困難にする．

また，DoS 攻撃を形式的に扱う研究として，Meadows による研究 [6, 7] がある．これは，アリスボブ記法にコストを付記した形式体型を提案している．それにより DoS 攻撃を形式的に記述することを可能としている．

1.5 目的

これまでの Spice 計算は理論的な単なる枠組みであり，そのため，解析手法と DoS 耐性に関する性質を記述する方法がなかった．そこで，本研究では Spice 計算を使用した解析手法を提案する．

モデル検査器を使用して，システムが至りうるあらゆる状態を対象とする網羅的な検査を行う．具体的には，モデル検査器として Spin を用い，(1)Spice 計算で記述されたプロセスを Spin のモデルに変換し，(2)DoS 攻撃耐性に関する性質を LTL 式（線形時間論理式）で表現し，(3)Spin で検証する．

Promela ではなく Spice 計算でプロトコルを記述すると，プロトコルの記述に，コストに関する記述を明示的に書かなくてよい．もし，Promela で直接，コストを計算する記述をおこなったとすると，間違いが入り込む可能性がある．Spice 計算から Promela への変換という方法は，言い替えると，コスト記述の自動化ともいえる．

2 解析手法

本節では SPIN を用いた Spice 計算の解析手法を説明する．解析は以下の手順でおこなう（図 1）．

1. Spice 計算のプロセスでプロトコルを記述する
2. Spice 計算のプロセスを PROMELA で記述されたモデルに変換する．この結果を M とする．
3. 検証したい DoS 攻撃耐性の性質を LTL 式で記述する．
4. LTL 式を Never claim に変換する．この結果を N とする．
5. M と N を SPIN へ入力し検証を実行する．

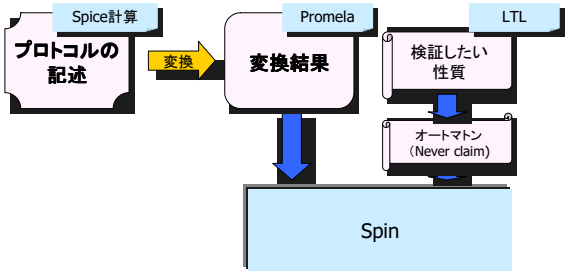

図 1: 解析手法の概要

2.1 Spice 計算から Promela への変換

ここでは Spice 計算から PROMELA で記述されたモデルへの変換について説明する．プロトコルの DoS 攻撃耐性を解析するときに重要なのは，どのような実行パスでも発生するコストを抑えられるかを調べることである．

そのために，システムの全状態を網羅するように全パス探査をおこうなう．変換のポイントは次のようになる．

- 送受信と分岐のみが重要である．
- 項の評価は抽象化して無視する．
- repeat は無限ループで近似する．
- コスト処理を埋め込む．

2.1.1 inp（受信），out(送信)

Spice 計算の送受信 `out %n M`, `inp #n (x)` は PROMELA の送受信`_n!T`, `_n?x` に変換される．通信に使用している Spice 計算の名前は PROMELA のチャネルと変数に変換される．図 2 から図 3 に変換した例を示す．

2.1.2 comp（並列合成）

Spice 計算の並列合成 `(P|Q)` は PROMELA のプロセス宣言 `proctype p1() { P }`と `proctype p2() { Q }`に変換される．さらにこれらのプロセスを生成する `run p1()` と `run p2()` が `init` のなかに生成される．図 2 から図 3 に変換した例を示す．

```
( out %n <0>;
 stop
| inp #n ($x);
 free $x;
 stop  )
```

図 2: 変換前の Spice 計算 (1)

```
byte x;
chan _n = [0] of { int };
byte n;
init {
 COST cost;
 run p1(); run p2()   }
proctype p1(){
 COST cost;
 _n!0;
 STOP();
}
proctype p2(){
 COST cost;
 _n?x;
 cost.store = cost.store + 1;
 FREE();
 cost.store = cost.store + -1;
 STOP();
}
```

図 3: 変換した結果の Promela コード (1)

2.1.3 repeat（繰り返し）

Spice 計算の繰り返し `repeat P` は PROMELA の `do :: P od` による無限ループに変換される．図 4 から図 5 に変換した例を示す．

2.1.4 store（メモリ割当）, free（メモリ解放）

Spice 計算のメモリ割当とメモリ解放は PROMELA では無視される．コスト処理のみが埋め込まれる．例ではわかりやすくするために FREE() を使用しているがこれは何もしない文である．図 4 から図 5 に変換した例を示す.

2.1.5 split（ペア分解）

Spice 計算のペア分解 split [x1,x2,..] は次のような PROMELA の条件文に変換される．

```
if
 ::true -> 成功 ;
 ::true -> 失敗
fi
```

ペア分解に成功した場合と失敗した場合で分岐をするが，どちらもガードが true となっている．こうすることによって全パス探査を実現している．図 4 から図 5 に変換した例を示す.

```
repeat store $x=0;
split [$y,$z] is $x
    err {free $x;stop};
free $x,$y,$z;
stop
```

図 4: 変換前の Spice 計算 (2)

2.2 LTL による性質の表現

ここでは DoS 攻撃耐性に関する性質を LT 式 L でどのように表現するかを具体例を使って説明する．

1 つめに「確保したメモリは必ず解放する」という性質をみる．この性質はサーバプロセスがメモリを使いすぎないことを表現している．LTL 式で記述すると次の式になる．

$$\Box(Alloc \rightarrow \Diamond Dealloc)$$

Alloc は，なんらかのメモリが確保されたことを表す命題である．*Dealloc* は，すべてのメモリが解放されたことを表す命題である．したがって，上式は,「『もし，メモリが確保されたならば，いずれは解放される』ということが常に成り立つ」ということを意味する．

```
byte z,y,x;
active proctype my_init() {
  int cost_pair, cost_store,
      cost_match, cost_hash;
  do
  ::x = 0;
    cost_store = cost_store + 1;
    if
    ::true -> cost_store = cost_store + 1;
              cost_store = cost_store + 1;
              FREE();
              cost_store = cost_store + -1;
              STOP();
    ::true -> FREE();
      cost_store = cost_store + -1;
      FREE();
      cost_store = cost_store + -1;
      FREE();
      cost_store = cost_store + -1;
      STOP();
    fi
  od;
}
```

図 5: 変換した結果の Promela コード (2)

2 つめは「いつでもサーバのコストはクライアントのコストより小さい」という性質を考える．この性質はプロトコルに参加するプロセス間のコスト量を制限している．攻撃者が攻撃を成功するにはサーバ以上の資源が必要となる．

$$\Box(C_{server} < C_{client})$$

C_{server} は，サーバーのコストを表す変数であり，C_{client} は，クライアントのコストを表す変数である．すなわち，サーバーのコストは常に，クライアントのコストにより押さえられているということを意味する．

2.3 解析例

ここでは SPIN を使用した解析の例を示す.

図 6 が解析対象のプロトコル記述した Spice 計算のプロセスである．クライアントがデータをサーバに送信し，サーバは受け取ったデータのハッシュ値を計算してクライアントに送信するプロトコルである．これを PROMELA に変換した結果が図 7 である.

解析したい性質は「確保したメモリは必ず解放する」である．SPIN への入力は否定をとり，命題を命題変数とした LTL となる（図 8）．実際には命題変数は#define で定義する.

```
(Spin Version 4.2.7 -- 23 June 2006)
        + Partial Order Reduction

Full statespace search for:
        never claim             +
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness disabled)
        invalid end states      - (disabled by never claim)

State-vector 84 byte, depth reached 31, errors: 0
      20 states, stored (25 visited)
       8 states, matched
      33 transitions (= visited+matched)
       0 atomic steps
hash conflicts: 0 (resolved)

2.622   memory usage (Mbyte)
```

図 9: Spin による解析結果

Spin による解析結果を図 9 に示す．この例ではエラーが発生していないので「確保したメモリは必ず解放する」という性質が満たされていることを確認できた．

```
( {@a,[]} | {@b,[]} ) |>
( out %n <0>;
    inp #n ($y);
    free $y;
    stop
| inp #n ($x);
    out #n <hash($x)>;
    free $x;
    stop)
```

図 6: Spice 計算によるプロトコル記述

3 まとめと今後の課題

3.1 まとめ

本研究では Spice 計算を用いた解析手法を提案した．この手法では Spin を用いて解析をおこうなう．そのために Spice 計算から Promela で記述されたモデルへの変換と DoS 攻撃耐性の性質を LTL 式で表現する手法を示した．また，変換系のプロトタイプを作成し，簡易な例による実験をおこなった．

```
byte y,x;
chan _n = [0] of { int };
byte n;
active proctype my_init() {
    int cost_pair, cost_store,
       cost_match, cost_hash;
    run p1(); run p2()
}
proctype p1(){
    int cost_pair, cost_store,
       cost_match, cost_hash;
    _n!0;
    _n?y;
    cost_store = cost_store + 1; 1;
    FREE();
    cost_store = cost_store + -1; 1;
    STOP();
}
proctype p2(){
    int cost_pair, cost_store,
       cost_match, cost_hash;
    _n?x;
    cost_store = cost_store + 1; 1;
    _n!hash(x);
    cost_hash = cost_hash + 1; 1;
    FREE();
    cost_store = cost_store + -1; 1;
    STOP();
}
```

図 7: 変換した結果の Promela コード

```
!([](p -> <>q))
p = (p2:cost_store > 0)
q = (p2:cost_store == 0)
```

図 8: 解析する性質を表現した LTL 式

3.2 今後の課題

本研究では Spice 計算から PROMELA で記述されたモデルへの変換を示した．この変換の理論的な性質を示す必要がある．

- 変換の正当性．変換結果がコスト等価である．
- Spice 計算の意味論と PROMELA の意味論との関係．DoS 攻撃耐性に関する性質を，変換された PROMELA で記述されたモデルに関する LTL 式で表現した．この LTL 式を書く際には，変換されたモデルを読み，理解して記述することが必要となる．本来は，Spice 計算のプロセスに関する論理式を記述するのが望ましい．Spice 計算で記述されたプロセスに関する性質を記述するための論理体系（図 10）を検討することが課題となっている．

また変換した結果の実用性の検証する．そのために実際のプロトコルの検証をおこない，検証の実行時間を計測して性能評価をする．

さらに時間モデルの導入も考えられる．

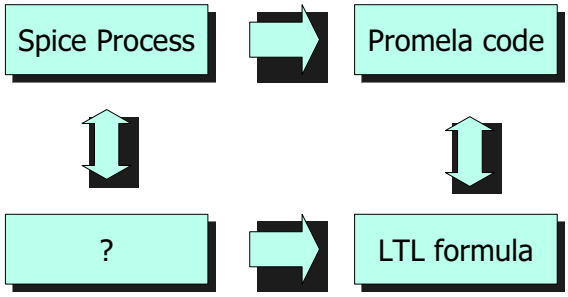

図 10: Spice 計算と Promela の対応

参考文献

[1] Tuomas Aura, Pekka Nikander, and Jussipekka Leiwo. DOS-resistant authentication with client puzzles. In *Security Protocols, 8th International Workshop*, Vol. 2133 of *Lecture Notes in Computer Science*, pp. 170–177. Springer-Verlag, 2001.

[2] CERT. CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks, 1996.

[3] CERT. CA-2000-21 Denial-of-Service Vulnerabilities in TCP/IP Stacks, 2000.

[4] D. J. Bernstein. SYN cookies. `http://cr.yp.to/syncookies.html`.

[5] Gerard J. Holzmann. *The Spin model checker: primer and reference manual*. Addison-Wesley, 2003.

[6] Catherine Meadows. A formal framework and evaluation method for network denial of service. In *Proceeding of the 12th IEEE Computer Security Foundations Workshop*, pp. 4–13, 1999.

[7] Catherine Meadows. A cost-based framework for analysis of denial of service networks. *Journal of Computer Security*, Vol. 9, No. 1/2, pp. 143–164, 2001.

[8] Daigo Tomioka, Shin-ya Nishizaki, and Ritsuya Ikeda. A cost estimation calculus for analyzing the resistance to denial-of-service attack. In *Proceedings of International Symposium on Software Security 2003, Springer-Verlag, LNCS 3233, 2004*, 2002.

[9] 冨岡大悟, 池田立野, 西崎真也. 通信プロトコルにおけるサービス不能攻撃耐性解析のための型付き π 計算. コンピュータソフトウェア, Vol. 23, No. 3, pp. 66–84, 2006.