暗号プロトコル評価結果

独立行政法人　情報通信研究機構

1.プロトコル名：Kerberos with ticket caching

2.関連する標準

IETF：RFC4120 (The Kerberos Network Authentication Service (V5)), July 2005.
http://www.rfc-editor.org/rfc/pdfrfc/rfc4120.txt.pdf

3.使用したツール：Proverif

4.評価の概要：Proverif による評価では、攻撃は発見されなかった。

5.Proverif による評価

本暗号プロトコルのアプリケーションサーバは受け取った認証情報を保存し、次のセッションで使用する。このような情報を蓄積する暗号プロトコルは ProVerif ではうまく評価できないため、クライアント及びアプリケーションサーバに新しいタイムスタンプを与え、タイムスタンプが一致する場合にのみ認証情報を受理するようにすることで、擬似的にリプレイの禁止を実現した。

5.1.シーケンス記述

```
free c: channel.

(*types*)
type host.
type symkey.
type nonce.

(*SKE*)
fun encrypt(bitstring, symkey): bitstring.
reduc forall x: bitstring, k: symkey;
      decrypt(encrypt(x, k), k) = x.

(* table *)
table keys(host, host, symkey).
```

```
(* events *)
event end().
event beginC(host, host, symkey).
event endC(host, host, symkey).
event beginS(host, host, symkey).
event endS(host, host, symkey).

(* free names *)
free hostC, hostA, hostG, hostS: host.
free secretC_K_CS: bitstring [private].
free secretS_K_CS: bitstring [private].
free secretC_K_CG: bitstring [private].
free secretG_K_CG: bitstring [private].
free secretTest: bitstring [private].

(* assumptions *)
not attacker(new K_CA).
not attacker(new K_AG).
not attacker(new K_GS).



(* queries *)
query attacker(secretC_K_CS).
query attacker(secretS_K_CS).
query attacker(secretC_K_CG).
query attacker(secretG_K_CG).
(*
query attacker(secretTest).
*)
query C: host, S: host, K: symkey;
      inj-event(endS(C, S, K)) ==> inj-event(beginC(C, S, K)).
```

```
query C: host, S: host, K: symkey;
      inj-event(endC(C, S, K)) ==> inj-event(beginS(C, S, K)).



(* processes *)
let procC(C: host, A: host, G: host, T2: bitstring) =
    in(c, S: host);
    get keys(=C, =A, K_CA) in
    (* 1 *)
    new Lifetime_1: bitstring;
    new N_1: nonce;
    out(c, (C, G, Lifetime_1, N_1));
    (* 2 *)
    in(c, (Ticket_1: bitstring, ct2: bitstring));
    let (=C, =G, K_CG: symkey, Tstart: bitstring, Texpire: bitstring,
=N_1) = decrypt(ct2, K_CA) in
    out(c, encrypt(secretC_K_CG, K_CG)); (* !!! for describing security
*)
    (* 3 *)
    new Lifetime_2: bitstring;
    new N_2: nonce;
    new T: bitstring;
    out(c, (S, Lifetime_2, N_2, Ticket_1, encrypt((C, T), K_CG)));
    (* 4 *)
    in(c, (=C, Ticket_2: bitstring, ct4: bitstring));
    let (=S, K_CS: symkey, Tstart2: bitstring, Texpire2: bitstring,
=N_2)
        = decrypt(ct4, K_CG) in
    (* 5 *)
    event beginC(C, S, K_CS);
    out(c, (Ticket_2, encrypt((C, T2), K_CS)));
    (* 6 *)
```

```
    in(c, ct6: bitstring);
    if T2 = decrypt(ct6, K_CS) then
    (* security check *)
    if S = hostS then
    out(c, encrypt(secretC_K_CS, K_CS));
    event endC(C, S, K_CS);
    event end().


let procA(A: host) =
    in(c, (C: host, G: host, Lifetime_1: bitstring, N_1: nonce));
    get keys(=C, =A, K_CA) in
    new K_CG: symkey;
    new Tstart: bitstring;
    new Texpire: bitstring;
    get keys(=A, =G, K_AG) in
    let Ticket_1 = encrypt((C, G, K_CG, Tstart, Texpire), K_AG) in
    out(c, (Ticket_1, encrypt((C, G, K_CG, Tstart, Texpire, N_1),
K_CA)));
    event end().

let procG(G: host, A: host, K_AG: symkey) =
    (* 3 *)
    in(c, (S: host, Lifetime_2: bitstring, N_2: nonce, Ticket_1:
bitstring, ct3: bitstring));
    let (C: host, =G, K_CG: symkey, Tstart: bitstring, Texpire:
bitstring)
        = decrypt(Ticket_1, K_AG) in
    let (=C, T: bitstring) = decrypt(ct3, K_CG) in
    (* 4 *)
    new Tstart2: bitstring;
    new Texpire2: bitstring;
```

```
    new K_CS: symkey;
    get keys(=G, =S, K_GS) in
    let Ticket_2 = encrypt((C, S, K_CS, Tstart2, Texpire2), K_GS) in
    out(c, (C, Ticket_2, encrypt((S, K_CS, Tstart2, Texpire2, N_2),
K_CG)));
    if C = hostC then
    out(c, encrypt(secretG_K_CG, K_CG)); (* !!! for describing security
*)
    event end().

let procS(S: host, K_GS: symkey, T2: bitstring) =
    (* 5 *)
    in(c, (Ticket_2: bitstring, ct5: bitstring));
    let (C: host, =S, K_CS: symkey, Tstart2: bitstring, Texpire2:
bitstring)
        = decrypt(Ticket_2, K_GS) in
    let (=C, =T2) = decrypt(ct5, K_CS) in
    (* 6 *)
    event beginS(C, S, K_CS);
    out(c, encrypt(T2, K_CS));
    (* security check *)
    if C = hostC then
    out(c, encrypt(secretS_K_CS, K_CS));
    event endS(C, S, K_CS);
    event end().

let keyRegistration =
    in(c, (h1: host, h2: host, k: symkey));
    if (h1, h2) <> (hostC, hostA) &&
       (h1, h2) <> (hostA, hostG) &&
       (h1, h2) <> (hostG, hostS) then
    insert keys(h1, h2, k).
```

```
process
        new K_CA: symkey;
        insert keys(hostC, hostA, K_CA);
        new K_AG: symkey;
        insert keys(hostA, hostG, K_AG);
        new K_GS: symkey;
        insert keys(hostG, hostS, K_GS);
        (
                (!new T2: bitstring;(
                        (procC(hostC, hostA, hostG, T2))|
                        (procS(hostS, K_GS, T2))
                )) |
                (!procA(hostA))|
                (!procG(hostG, hostA, K_AG))|
                (!keyRegistration)
        )
```

5.2.攻撃者モデル

シーケンスの表記に含まれる。

5.3.セキュリティプロパティの記述

```
(* (1) *)
query C: host, S: host, K: symkey;
      inj-event(endS(C, S, K)) ==> inj-event(beginC(C, S, K)).
(* (2) *)
query C: host, S: host, K: symkey;
      inj-event(endC(C, S, K)) ==> inj-event(beginS(C, S, K)).
(* (3) *)
query attacker(secretC_K_CS).
query attacker(secretS_K_CS).
(* (4) *)
```

```
query attacker(secretC_K_CG).
query attacker(secretG_K_CG).
```

(1) サーバによるクライアントの認証。

(2) クライアントによるサーバの認証。

(3) クライアントとサーバが共有した鍵 K_CS の秘匿性。

(4) クライアントとゲートウェイ(KDC)が共有した鍵 K_CG の秘匿性。

5.4.検証結果

○評価ツールの出力

```
RESULT inj-event(endC(C_39,S_40,K)) ==> inj-event(beginS(C_39,S_40,K))
is true.
RESULT inj-event(endS(C_7145,S_7146,K_7147)) ==>
inj-event(beginC(C_7145,S_7146,K_7147)) is true.
RESULT not attacker(secretG_K_CG[]) is true.
RESULT not attacker(secretC_K_CG[]) is true.
RESULT not attacker(secretS_K_CS[]) is true.
RESULT not attacker(secretC_K_CS[]) is true.
```

いずれも安全であることが確認できた。

○攻撃に関する解説

攻撃は発見されなかった。

5.5.モデル化

○モデル化プロセス

本暗号プロトコルのアプリケーションサーバは受け取った認証情報を保存し、次のセッションで使用する。このような情報を蓄積する暗号プロトコルは ProVerif ではうまく評価できないため、クライアント及びアプリケーションサーバに新しいタイムスタンプを与え、タイムスタンプが一致する場合にのみ認証情報を受理するようにすることで、擬似的にリプレイの禁止を実現した。

5.6.モデル化の妥当性

上記のようなモデル化のため、モデル化した暗号プロトコルは現実の暗号プロトコルより強くなっている恐れがある。

5.7.評価ツールとの相性

〇暗号プロトコルの記述可能性

モデル化の項目に記載したように、ProVerif では状態を蓄積する暗号プロトコルはうまく扱えないため、擬似的に類似の動作を行うように記述した。

〇セキュリティプロパティの記述可能性

特に制限はなかった。

5.8.評価ツールの性能

評価は瞬時に完了した。

実行環境は以下のとおり。

・Intel Core i7 L620 2.00HGz

・Windows7 上の VirtualBox 仮想マシン上の Ubuntu Linux 12.04.1 LTS

・メモリ 512MB

・ProVerif 1.86pl3

5.9.備考

本文書は、**総務省「暗号・認証技術等を用いた安全な通信環境推進事業に関する実証実験の請負　成果報告書**」からの引用である。