Kerberos with forwardable ticket の ProVerif による評価結果

国立研究開発法人 情報通信研究機構

1. 基本情報

- ✧ 名前

  The Kerberos Network Authentication Service (V5), Kerberos with forwardable ticket

- ✧ 機能

  信頼できる第三者機関（TTP, Trusted Third Party）の存在を前提とする、オープンなネットワークにおけるサーバ・クライアント間でのネットワーク認証・鍵交換プロトコル。 特徴は認証後にクライアントの IP アドレスを変更可能であること。暗号として共通鍵暗号を利用。

- ✧ 関連する標準

  RFC4120 (https://www.ietf.org/rfc/rfc4120.txt)

2. ProVerif の文法による記述

2.1. プロトコル仕様

```
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 secretGf_K_CG: bitstring [private].
free secretTest: bitstring [private].

(* constants *)
const FORWARDABLE: bitstring [data].

(* 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(secretGf_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) =
    in(c, (IP_ADDR: bitstring, S: host));
    get keys(=C, =A, K_CA) in
    (* 1 *)
    new N_1: nonce;
    out(c, (C, G, N_1));
    (* 2 *)
    in(c, (Ticket_1: bitstring, ct2: bitstring));
    let (=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 N_2: nonce;
    new T: bitstring;
    out(c, (IP_ADDR, S, N_2, Ticket_1, encrypt((C, T), K_CG),
FORWARDABLE));
    (* 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 *)
    out(c, (S, N_2, Ticket_2, encrypt((C, T), K_CG)));
    (* 6 *)
    in(c, (=C, Ticket_3: bitstring, ct6: bitstring));
    let (=S, =K_CS, =Tstart2, =Texpire2, =N_2)
        = decrypt(ct6, K_CG) in
    (* 7 *)
    new T2: bitstring;
    event beginC(C, S, K_CS);
    out(c, (Ticket_3, encrypt((C, T2), K_CS)));
    (* 8 *)
    in(c, ct8: bitstring);
    if T2 = decrypt(ct8, K_CS) then
    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, 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((G, K_CG, Tstart, Texpire, N_1), K_CA)));
    event end().
```

```
let procG(G: host, A: host, K_AG: symkey) =
    (* 3 *)
    in(c, (IP_ADDR: bitstring, S: host, N_2: nonce,
           Ticket_1: bitstring, ct3: bitstring, =FORWARDABLE));
    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((IP_ADDR, C, S, K_CS, Tstart2, Texpire2,
                            FORWARDABLE), 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 procGf(G: host, A: host, K_AG: symkey) =
    in(c, Ticket_1: bitstring);
    let (C: host, =G, K_CG: symkey,
         Tstart1: bitstring, Texpire1: bitstring) = decrypt(Ticket_1,
K_AG) in
    (* 5 *)
    in(c, (NEW_IP_ADDR: bitstring, S: host, N_2: nonce,
           Ticket_2: bitstring, ct5: bitstring));
    get keys(=G, =S, K_GS) in
    let (IP_ADDR: bitstring, =C, =S, K_CS: symkey,
```

```
         Tstart2: bitstring, Texpire2: bitstring, =FORWARDABLE)
        = decrypt(Ticket_2, K_GS) in
    let (=C, T: bitstring) = decrypt(ct5, K_CG) in
    (* 6 *)
    let Ticket_3 = encrypt((NEW_IP_ADDR, C, S, K_CS,
                            Tstart2, Texpire2, FORWARDABLE), K_GS) in
    out(c, (C, Ticket_3, encrypt((S, K_CS, Tstart2, Texpire2, N_2),
K_CG)));
    if C = hostC then
    out(c, encrypt(secretGf_K_CG, K_CG)); (* !!! for describing security
*)
    event end().

let procS(S: host, K_GS: symkey) =
    (* 7 *)
    in(c, (Ticket_3: bitstring, ct7: bitstring));
    let (IP_ADDR: bitstring, C: host, =S, K_CS: symkey,
         Tstart2: bitstring, Texpire2: bitstring, flag: bitstring)
        = decrypt(Ticket_3, K_GS) in
    let (=C, T2: bitstring) = decrypt(ct7, K_CS) in
    (* 8 *)
    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);
          (
                    (!procC(hostC, hostA, hostG))|
                    (!procA(hostA))|
                    (!procG(hostG, hostA, K_AG))|
                    (!procGf(hostG, hostA, K_AG))|
                    (!procS(hostS, K_GS))|
                    (!keyRegistration)
          )
```

2.2. 攻撃者モデル

上述の記述に含まれる。

2.3. セキュリティ要件

上述の記述の (* queries *) に該当する。

```
(* (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).
query attacker(secretGf_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。

3. ProVerif による評価結果

3.1. 出力

いずれも安全であることが確認できた。ただし、(1)のセキュリティ要件の単射性は成立しない可能性がある。すなわち、サーバが暗号プロトコルを複数回実行（認証処理を実施）しているにも関わらずクライアントは暗号プロトコルを 1 回しか実行（認証処理を実施）していない可能性がある。

```
RESULT inj-event(endC(C_52,S_53,K)) ==> inj-event(beginS(C_52,S_53,K))
is true.
RESULT inj-event(endS(C_21818,S_21819,K_21820)) ==>
inj-event(beginC(C_21818,S_21819,K_21820)) cannot be proved.
RESULT (but event(endS(C_39514,S_39515,K_39516)) ==>
event(beginC(C_39514,S_39515,K_39516)) is true.)
RESULT not attacker(secretGf_K_CG[]) 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.
```

3.2. 攻撃の解説

前述のとおり、攻撃は発見されなかった。

4. 形式化

4.1. 方針

CPVP 技術文書「**Kerberos with forwardable ticket の概要**」で引用した AVISPA ライブラリの記述に沿って形式化を行なった。

4.2. 妥当性

特になし。

4.3. 検証ツールとの相性

プロトコル仕様、攻撃者モデル、セキュリティ要件を ProVerif で記述するにあたって、特に制限はなかった。

4.4. 検証ツール適用時の性能

検証時間は 1.6 秒であった。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. 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。