EAP-Archie 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 EAP Archie Protocol

- ✧ 機能

  PPP 接続において事前共有鍵を用いた相互認証・鍵交換プロトコル。

- ✧ 関連する標準

  Internet Draft (Intended status: Informational)

  http://tools.ietf.org/html/draft-jwalker-eap-archie-01

2. ProVerif の文法による記述

CPVP 技術文書「**EAP-Archie の概要**」で示したロール S (EAP Server) をロール A (Authenticator) で表す。

2.1. プロトコル仕様

```
free c: channel.


type key.
type nonce.
type host.


const Request: bitstring [data].
const Response: bitstring [data].
const Confirm: bitstring [data].
const Finish: bitstring [data].


free hostP, hostA: host.


table keys(host, host, bitstring).
```

```
(* MAC *)
fun mac(bitstring, key): bitstring.
reduc forall k: key, x: bitstring;
      checkmac(mac(x, k), k) = x.
reduc forall k: key, x: bitstring;
      getmsg(mac(x, k)) = x.

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

(**)
fun n2b(nonce): bitstring [data, typeConverter].
fun n2k(nonce): key [data, typeConverter].

(* events *)
event beginPeer(nonce, host, host, bitstring, bitstring).
event endPeer(nonce, host, host, bitstring, bitstring).
event beginAuthenticator(nonce, host, host, bitstring, bitstring).
event endAuthenticator(nonce, host, host, bitstring, bitstring).
event end().

(**)
free secretPeerNa, secretPeerNp,
     secretAuthenticatorNa, secretAuthenticatorNp,
     secretTest: nonce [private].

not attacker(new KEKpa).
not attacker(new KCKpa).
```

```
(* queries *)
query attacker(secretPeerNa).
query attacker(secretPeerNp).
query attacker(secretAuthenticatorNa).
query attacker(secretAuthenticatorNp).

query sid: nonce, p: host, a: host, ena: bitstring, enp: bitstring;
      inj-event(endAuthenticator(sid, p, a, ena, enp)) ==>
      inj-event(beginPeer(sid, p, a, ena, enp)).

query sid: nonce, p: host, a: host, ena: bitstring, enp: bitstring;
      inj-event(endPeer(sid, p, a, ena, enp)) ==>
      inj-event(beginAuthenticator(sid, p, a, ena, enp)).

(*
query attacker(secretTest).
*)

let procPeer =
    in(c, (PeerID: host, Binding: bitstring));
    if PeerID = hostP then
    (* *)
    (* Archie-Request *)
    in(c, (=Request, AuthID: host, SessionID: nonce));
    get keys(=PeerID, =AuthID, (KEK: key, KCK: key)) in
    (* Archie-Response *)
    new NonceP: nonce;
    let encNonceP = encrypt(NonceP, KEK) in
    let MAC1 = mac((Request, AuthID,
                    Response, SessionID, PeerID, encNonceP, Binding),
                     KCK) in
    out(c, (Response, SessionID, PeerID, encNonceP, Binding, MAC1));
```

```
    (**)
    in(c, (=Confirm, =SessionID, encNonceA: bitstring, =Binding,
           MAC2: bitstring));
    if (Request, AuthID,
        encNonceP,
         Confirm, SessionID, encNonceA, Binding) = checkmac(MAC2, KCK)
then
    (* Archie-Finish Message *)
    let MAC3 = mac((Finish, SessionID), KCK) in
    event beginPeer(SessionID, AuthID, PeerID, encNonceA, encNonceP);
    out(c, (Finish, SessionID, MAC3));
    if AuthID = hostA then
    event endPeer(SessionID, AuthID, PeerID, encNonceA, encNonceP);
    let NonceA = decrypt(encNonceA, KEK) in
    out(c, encrypt(secretPeerNa, n2k(NonceA)));
    out(c, encrypt(secretPeerNp, n2k(NonceP)));
    event end().
let procAuthenticator(AuthID: host) =
    (**)
    new SessionID: nonce;
    (*
    new NonceA: nonce;
    *)
    (* Archie-Request *)
    out(c, (Request, AuthID, SessionID));
    (* Archie-Response *)
    in(c, (=Response, =SessionID, PeerID: host, encNonceP: bitstring,
           Binding: bitstring, MAC1: bitstring));
    get keys(=PeerID, =AuthID, (KEK: key, KCK: key)) in
    if (Request, AuthID,
        Response, SessionID, PeerID, encNonceP, Binding)
       = checkmac(MAC1, KCK) then
```

```
    (* Archie-Confirm Message  *)
    new NonceA: nonce;
    let encNonceA = encrypt(NonceA, KEK) in
    let MAC2 =
         mac((Request, AuthID,
              encNonceP,
              Confirm, SessionID, encNonceA, Binding), KCK) in
    event beginAuthenticator(SessionID, AuthID, PeerID, encNonceA,
encNonceP);
    out(c, (Confirm, SessionID, encNonceA, Binding, MAC2));
    (* Archie-Finish Message *)
    in(c, (=Finish, =SessionID, MAC3: bitstring));
    if (Finish, SessionID) = checkmac(MAC3, KCK) then
    if PeerID = hostP then
    event endAuthenticator(SessionID, AuthID, PeerID, encNonceA,
encNonceP);
    out(c, encrypt(secretAuthenticatorNa, n2k(NonceA)));
    let NonceP = decrypt(encNonceP, KEK) in
    out(c, encrypt(secretAuthenticatorNp, n2k(NonceP)));
    event end().

let keyRegistration =
    in(c, (h1: host, h2: host, ks: bitstring));
    if (h1, h2) <> (hostP, hostA) then
    insert keys(h1, h2, ks).

process
         new KEKpa: key;
         new KCKpa: key;
         insert keys(hostP, hostA, (KEKpa, KCKpa));
         (
          (!procPeer)|
```

```
        (!procAuthenticator(hostA))|
        (!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 sid: nonce, p: host, a: host, ena: bitstring, enp: bitstring;
      inj-event(endAuthenticator(sid, p, a, ena, enp)) ==>
      inj-event(beginPeer(sid, p, a, ena, enp)).
(* (2) *)
query sid: nonce, p: host, a: host, ena: bitstring, enp: bitstring;
      inj-event(endPeer(sid, p, a, ena, enp)) ==>
      inj-event(beginAuthenticator(sid, p, a, ena, enp)).
```

(1) A(authenticator)が P(peer)を正しく認証すること。すなわち、A(authenticator)が実行を完了したとき、同じパラメータを用いて実行を完了した P(peer)が存在する。
(2) P(peer)が A(authenticator)を正しく認証すること。すなわち、P(peer)が実行を完了したとき、同じパラメータを用いて実行を完了した A(Authenticator)が存在する。

3. ProVerif による評価結果

3.1. 出力

ProVerif の出力は以下の通り。

```
RESULT              inj-event(endPeer(sid,p,a,ena,enp))              ==>
inj-event(beginAuthenticator(sid,p,a,ena,enp)) is true.
RESULT
inj-event(endAuthenticator(sid_2379,p_2380,a_2381,ena_2382,enp_2383)
) ==> inj-event(beginPeer(sid_2379,p_2380,a_2381,ena_2382,enp_2383))
cannot be proved.
RESULT                                                          (even
event(endAuthenticator(sid_4632,p_4633,a_4634,ena_4635,enp_4636)) ==>
```

event(beginPeer(sid_4632,p_4633,a_4634,ena_4635,enp_4636)) cannot be proved.)

(1)の安全性の評価に失敗した。ProVerif は効率的に評価を行うため、まず近似モデルで評価を行い、攻撃が発見されたらそれが現実のモデルで実行可能かどうかチェックを行う。この安全性については、近似モデルでは攻撃が発見されたが、本来の現実には実行不可能であることが確認された。このため、攻撃の具体例を提示できず、逆に安全であるとの結果も得られなかった。

(2)の安全性は成り立つ。

3.2. 攻撃の解説

前述のとおり、現実に実行可能な攻撃は発見されなかった。

4. 形式化

4.1. 方針

MAC と秘密鍵暗号のみを用いるため、そのまま Dolev-Yao モデル上で記述する。

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 0.1 秒未満であった。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。