DHCP-Delayed-Auth

Protocol Purpose

Delayed entity and message authentication for DHCP

Definition Reference

RFC 3118, http://www.faqs.org/rfcs/rfc3118.html

Model Authors

- Graham Steel, University of Edinburgh, July 2004
- Luca Compagna, AI-Lab DIST University of Genova, November 2004

Alice&Bob style

```
1. C -> S : C, delayedAuthReq, Time1
2. S -> C : S, delayedAuthReq, succ(Time1), KeyID(K),
            H(S, delayedAuthReq, succ(Time1), K)
```

Model Limitations

The RFC describes different options and checks in terms of key words MAY, MUST etc. This model is of the minimum protocol, i.e. only the MUST checks. In real life, message looks like

- 90 (auth requested),
- length,
- 1 (for delayed auth),
- 1 (to indicate standard HMAC algorithm),
- 0 (standard Replay Detection Mechanism, monotonically increasing counter),
- counter value.

We ignore length field (as it cannot be, yet, expressed in HLPSL), use fresh nonce to model RDM, and assume 'DelayedAuthReq' token is enough to specify algorithm, type of auth, and type of RDM.

The server returns the nonce + 1 (or `succ(nonce)` to be exact) instead of a timestamp with a higher value.

Problems considered: 2

Attacks Found

None

Further Notes

Client is the initiator. Sends a DHCP discover and requests authentication

---

HLPSL Specification

```
role dhcp_Delayed_Client (
        C, S     : agent,    % C client, S server
        H        : function, % HMAC hash func.
        KeyID    : function, % get a key id from a key
        K        : text,     % K is the pre-existing shared secret
        Snd, Rcv : channel(dy))
played_by C
def=

  local State : nat,
        Time1 : text,
        Sig   : message

  const delayedAuthReq : protocol_id,
        succ           : function,   % Successor function
        sec_k          : protocol_id
```

```
  init  State := 0

  transition

   1. State = 0
      /\ Rcv(start)
      =|>
      State' := 1
      /\ Time1' := new()
      /\ Snd(C.delayedAuthReq.Time1')

   2. State = 1
      /\ Rcv(S.delayedAuthReq.succ(Time1).KeyID(K).
             H(S,delayedAuthReq,succ(Time1),K))
      =|>
      State' := 2
      /\ Sig' := H(S,delayedAuthReq,succ(Time1),K)
      /\ request(C,S,sig,Sig')
      /\ secret(K,sec_k,{S})

end role
```

```
role dhcp_Delayed_Server (
        S,C      : agent,
        H        : function,  % HMAC hash func.
        KeyID    : function,  % get a key id from a key
        K        : text,
        Snd, Rcv : channel (dy))
played_by S
def=

  local State : nat,
        Time1 : text,
        Sig   : message

  const delayedAuthReq : protocol_id,
        succ           : function   % Successor function
```

```
  init State := 0

  transition

   1. State = 0
      /\ Rcv(C.delayedAuthReq.Time1')
      =|>
      State' := 1
      /\ Sig' := H(S,delayedAuthReq,succ(Time1'),K)
      /\ Snd(S.delayedAuthReq.succ(Time1').KeyID(K).Sig')
      /\ witness(S,C,sig,Sig')

end role
```

```
role session(C, S           : agent,
             H, KeyID       : function,
             K              : text)
def=

  local SA, RA, SB, RB : channel (dy)

  composition
        dhcp_Delayed_Server(S,C,H,KeyID,K,SA,RA) /\
        dhcp_Delayed_Client(C,S,H,KeyID,K,SB,RB)

end role
```

```
role environment()
def=

 const a, b       : agent,
       k1, k2, k3 : text,
       h, keyid   : function,
       sig        : protocol_id

 intruder_knowledge = {a,b,k2,i,delayedAuthReq,
                       keyid,h,succ,
```

```
                        k3}

 composition
        session(a,b,h,keyid,k1)
    /\  session(a,i,h,keyid,k2)
    /\  session(i,b,h,keyid,k3)

end role
```

```
goal
  secrecy_of sec_k

  %DHCP_Delayed_Client authenticates DHCP_Delayed_Server on sig
  authentication_on sig
end goal
```

```
environment()
```

References