Lecture 2:

Basic
PPP authentication mechanisms

PAP, CHAP, +++

Recommended reading:
RFC 1334, October 1992;
RFC 1994, August 1996
Wiley AAA book, chapter 2 (parts)

Authentication in PPP

⇒ Optional phase
  ⇒ After link establishment
  ⇒ after or during link quality determination (if present)

⇒ Authentication mechanism
  negotiated during link establishment
  ⇒ Option sent in configuration request PPP msg
Basic PPP authentication

⇒ LCP option #3 = authentication protocol

<table>
<thead>
<tr>
<th>Type</th>
<th>Length</th>
<th>Auth-protocol</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>03</td>
<td>&gt;=4</td>
<td>c023=PAP, c223=CHAP</td>
<td>...</td>
</tr>
</tbody>
</table>

Specific extra info needed to the considered auth protocol

⇒ Two basic authentication mechanisms initially considered

⇒ PAP: Password Authentication Protocol

<table>
<thead>
<tr>
<th>Type</th>
<th>Length</th>
<th>Auth-protocol</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>03</td>
<td>04</td>
<td>c023=PAP</td>
<td></td>
</tr>
</tbody>
</table>

No extra info

⇒ CHAP: Challenge Handshake Authentication Protocol

<table>
<thead>
<tr>
<th>Type</th>
<th>Length</th>
<th>Auth-protocol</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>03</td>
<td>05</td>
<td>c223=CHAP</td>
<td>05=MD5</td>
</tr>
</tbody>
</table>

Extra info: Hash Function
Basic = MD5

Auth direction

⇒ Independently done on both directions!

⇒ Authentication may differ
⇒ or may apply to a single direction only
⇒ Typically NAS requires user to authenticate
⇒ User does not require NAS to authenticate

⇒ Authenticator = the end of the link that requires the other peer to perform authentication

⇒ Authenticator: sends the Configure-Request, specifying the authentication protocol to be used
⇒ Both sides act in turn as authenticators in the case of mutual authentication
PAP
Password Authentication Protocol

Simplest possible mechanism
Based on the a-priori knowledge, by the authenticator, of the (user_id, password) pair specified during contractual agreement

Two-way handshake

Remote User

Authenticate-Request (User_ID,Passwd)

Authenticate-Ack
Authenticate-Nak

User Database

UID passwd

Giuseppe Bianchi

Giuseppe Bianchi
PAP procedure

- **Starts when link is established**
- **Authenticating peer repeatedly sends Id/Password pair, until:**
  - An ACK is received
  - A NACK is received and/or the connection is terminated

**PAP is a weak authentication method**
- Passwords sent “in clear”
- No protection from playback
- No protection from repeated trial and error attacks
  - Peer is in control of the frequency and timing of the attempts.

---

Giuseppe Bianchi

---

PAP Authentication Protocol option

- **Configure-Request (auth protocol option 03=PAP)**
- **Configure-Ack (auth protocol option 03=PAP)**

There is no data field in the option

---

Giuseppe Bianchi
**PAP packet format: auth-request**

User NAS

- **PAP Authenticate-Request (code 1)**
- **PAP Authenticate-Ack (code 2)**
- **PAP Authenticate-Nak (code 3)**

To match request/reply

<table>
<thead>
<tr>
<th>Flag</th>
<th>Address</th>
<th>Control</th>
<th>Protocol</th>
<th>Information</th>
<th>FCS</th>
<th>Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>11111111</td>
<td>00000011</td>
<td>0xC023 (PAP)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Flag**

- 01

**Address**

- 11111111

**Control**

- 00000011

**Protocol**

- 0xC023 (PAP)

**Information**

- FCS

**User NAS**

- Flag
- Address: 11111111
- Control: 00000011
- Protocol: 0xC023 (PAP)
- Information: FCS

**Auth-request example**

(Real capture)

- Code 01
- ID=09
- User id = 18 bytes: e u 2 5 6 3 6 7 8 @ t e l e 2 . i t
- Password = 8 bytes: 9 d w c - u j n

**ALL UID+PW IN CLEAR!!!!**

---

Giuseppe Bianchi
PAP packet format: auth-Ack/Nak

User

PAP Authenticate-Request (code 1)

PAP Authenticate-Ack (code 2) or
PAP Authenticate-Nak (code 3)

NAS

To match request/reply

Code 1 byte =2 or 03
Identifier 1 byte
Length 2 bytes
Msg len 1 byte
msg ***

Message field: 0+ bytes
its contents are
Implementation-dependent
Intended to be
human readable (ASCII)

Flag
Address 11111111
Control 00000011
Protocol 0xC023 (PAP)
Information
FCS
Flag

Giuseppe Bianchi

CHAP
Challenge Handshake Authentication Protocol

Giuseppe Bianchi
Approach

- **Three-way handshake**
  - Challenge – Response – Success or Failure
- **Uses a Random Challenge, with a cryptographically hashed Response which depends upon the Challenge and a secret key**
- **Secret key never transmitted in clear**
  - Much more safer than PAP
- **Conceptually identical to the approach currently adopted into actual cellular networks**
  - GSM, UMTS

---

**Three-way handshake**

1) Challenge

2) Username, Hash(Challenge+Pw+...)

3) Success, Failure

Three-way handshake performed initially, after link establishment
But it MAY be repeated ANYTIME at RANDOM TIMES after the link is established
With new (different) challenges!!
CHAP pros & cons

**Pros:**
- Protection against playback attack
- Incrementally changing identifier
- Variable challenge value
- Repeated challenges
  - Authentication may be repeated over connection time (unlike PAP where it is performed only once at start)
  - Intended to limit the time of exposure to any single attack
- Authenticator controls frequency and timing of the challenges
  - CHAP does not allow a peer to attempt authentication without a challenge

**Cons:**
- Secret must be available in plaintext form
  - Cannot use irreversibly encrypted password databases
- Hard scalability in large installations
  - Every possible secret must be maintained at both ends of the link

---

CHAP selection

Similar to PAP: during configure-Request

The only required hashing algorithm, in a conforming implementation, is MD5 (but CHAP protocol open to other possible hashing algorithms as well)
CHAP Challenge

- **Identifier:** MUST change at each new challenge
- **Value:** randomly generated - must be designed to be
  - Unique & different per each challenge
  - To avoid replay attacks
  - Not predictable
  - Value size: 1+ bytes
  - In principle arbitrary and independent of the hashing algorithm used

- **Name field:** Identification of the system transmitting the packet

<table>
<thead>
<tr>
<th>Code</th>
<th>Identifier</th>
<th>Length</th>
<th>Challenge</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 byte</td>
<td>1 byte</td>
<td>2 bytes</td>
<td>1 byte</td>
<td>***</td>
</tr>
</tbody>
</table>

**PPP Challenge Handshake Authentication Protocol (REAL TRACE EXAMPLE)**

- **Code:** 0x 01 (Challenge)
- **Identifier:** 0x 01
- **Length:** 0x 00 1f (31 bytes)
- **Value Size:** 0x 10 (16 bytes)
- **Value:** 0x 07 21 e9 b3 30 a6 f8 6f 52 f2 ff 67 7f 07 3d 15 f5
- **Name:** MILZ-LNS-9 (10 bytes)

---

**Response computation**

- **One-Way Hash function**
  - Transform an arbitrary text size into an alphanumeric sequence of given size (digest)
- **MD5 digest = 16 bytes**
- **Response value: one-way hash calculated over:**
  - Identifier, concatenated with the "secret", concatenated with the challenge value

<table>
<thead>
<tr>
<th>identifier</th>
<th>Secret key</th>
<th>Challenge</th>
</tr>
</thead>
</table>

**Original text**

<table>
<thead>
<tr>
<th>Code</th>
<th>Identifier</th>
<th>Length</th>
<th>Value</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 byte</td>
<td>1 byte</td>
<td>2 bytes</td>
<td>1 byte</td>
<td>***</td>
</tr>
</tbody>
</table>

**PPP Challenge Handshake Authentication Protocol (REAL TRACE EXAMPLE)**

- **Code:** 0x 02 (Challenge)
- **Identifier:** 0x 01
- **Length:** 0x 00 27 (39 bytes)
- **Value Size:** 0x 10 (16 bytes)
- **Value:** 0x 4b 70 76 c3 2a b5 21 f0 29 9b 25 72 06 0a e4 ae
- **Name:** eu2563678@tele2.it (18 bytes)

---

Giuseppe Bianchi
Success/Failure

- **Authenticator in turn computes the digest**
  - It has identifier, challenge, and the secret key
  - In fact there is the user id repeated in the "name" field (password from DB lookup)

- **And compares with that received**
  - If OK, send back Success (Code 03)
  - If NO, send back Failure (Code 04) and terminate link

- **Message**: optional field, intended for human

<table>
<thead>
<tr>
<th>Code</th>
<th>Identifier</th>
<th>Length</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>03 or 04</td>
<td>0x 01</td>
<td>0x 00 04 (4 bytes) ***</td>
<td></td>
</tr>
</tbody>
</table>

PPP Challenge Handshake Authentication Protocol (REAL TRACE EXAMPLE)
- Code: 0x 03 (Challenge)
- Identifier: 0x 01
- Length: 0x 00 04 (4 bytes)

\[\text{[no message in the considered implementation!]}\]

---

Giuseppe Bianchi

---

Password based Authentication: Extensions

---

Giuseppe Bianchi
Passwd protection in DB?

Remote User 1) UID, passwd

Authenticator 2) Ack/Nack

User Database

UID  passwd

Passwd DB in clear... Significant vulnerability!

Giuseppe Bianchi

Passwd protection in DB: storing passwd hashes!

Remote User 1) UID, passwd

Authenticator 2) Ack/Nack

User Database

UID  H(passwd)

Authenticator:
1) receives UID & passwd (clear text)
2) computes hash H(passwd) - any locally used Hash OK; Linux = MD5
3) compares with DB entry

Giuseppe Bianchi
Dictionary attack...

- Many users use predictable passwd
- Dictionary attack:
  - Hashing does not help
  - Will see in a dedicated laboratory!

One-time passwd

- Is it possible to extend PAP so that user changes passwd at every (successful) attempt?
  - If it is, would prevent playback attacks

UID="Flavia", passwd="087654" OK
UID="Flavia", passwd="087654" NO!!
One-time passwd: trivial... but...

- N (large) passwd per user
- 10.000.000++ users
- HUGE DB!! Not viable

Idea: hash chains

P[0] = starting point
P[i] = H(P[i-1])
P[N] = last value
One-time passwd: practical

Compute $P[0]...P[N]$ offline

$UID=\text{"Flavia"}, passwd= P[N]$ if $H(P[N])=P[N+1]$ OK; store $P[N]

$UID=\text{"Flavia"}, passwd= P[N-1]$ if $H(P[N-1])=P[i]$ OK; store $P[i-1]$ for $i = 1...N$

One-time passwd benefits

- **Passwd in clear = OK**
- **Authenticator only stores USED passwd**
  - no way to predict next one (1-way hash)
- **Authenticator only stores 1 value**
  - Same complexity as in ordinary PAP

**Issues:**
- Large $N$ to prevent frequent renegotiation
- Client size = vulnerable (must store passwd seed or whole vector)
Issues with CHAP

Basic CHAP vulnerability

- Authenticator MUST store passwd in clear!
  - Otherwise no way to compute H(id, pw, challenge)

- Authenticator storage = straightforward target for attack!
  - Even worse than PAP!!

User Database

<table>
<thead>
<tr>
<th>...</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>flavia</td>
<td>mypass</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
**Idea: “salt”**

- Challenge = 135623; Salt = 9876

**RESPONSE = Flavia, H(id | H(9876,mypass) | 135623)**

**ACK or NACK**

> **ATTACKER MAY ONLY ACCESS TO “SALTED” PASSWD**

- Different salt for different authenticator servers
  - breaking one != breaking all
- Refresh DB periodically
  - Someone must take care of this… more later

**SALT: Same idea can be used also in PAP (of course)**

---

**CHAP and mutual authentication /1**

- ID=2; Challenge = 135623

**RESP: Flavia, H(ID=2, sharedsecret, 135623)**

**ACK**

- ID=3; Challenge = 324567

**RESP: servername, H(ID=3, sharedsecret, 324567)**

**ACK**

Usage of a shared Secret… good idea, Easy to manage!

Good Idea??

---

Giuseppe Bianchi
CHAP and mutual authentication /2
Reflection attack

ID=2; CHALLENGE = 135623

ID=2; CHALLENGE = 135623

RESP: servername, H(ID=2, sharedsecret, 135623)

ACK

RESP: client, H(ID=2, sharedsecret, 135623)

ACK

VERY risky!!
Attacker:
→ replays server challenge
→ accept computed resp
→ uses resp to authenticate!!

Without any info on real client !!

Mutual authentication with Challenge-Response
(just some hints... no full analysis)
Basic idea

Flavia shows knowledge of secret over C1

Boss shows knowledge of secret over C2

Reflection!

→ Does not work

C2!\neq C1

Boss, C1

C1!\neq C2

Boss, H(secret, C2)

Boss, H(secret, C2)

Boss, C1

Flavia, C2, H(secret, C1)

Flavia, C3, H(secret, C2)

Boss, C2

Boss, H(secret, C2)

Giuseppe Bianchi
**What if reflection is prevented by protocol status?**

➔ *Attacker may use “other” party!*

---

**Let's try to fix this**

Chaining challenges! Add dependency between challenges in same handshake

---

**Giuseppe Bianchi**
Does not work

Boss, C1

Flavia, C2, H(secret, C1, C2)

Boss, C2

Flavia, C3, H(secret, C2, C3)

Boss, C3, H(secret, C2, C3)

Too many nonces!!

Minimize nonces

Boss, C1

Flavia, C2, H(secret, C1, C2)

Same challenges, but different values (order!)

Boss, C3, H(secret, C2, C1)

No reflection possible anymore
Challenge-Response in GSM authentication

GSM essential components

- MS: Mobile System
- HLR: Home Location Register
- VLR: Visiting Location Register
- AuC: Authentication Center
- BTS: Base Transceiver Station
- BSC: Base Station Controller
- SIM: Subscriber Identity Module
- MSC: Mobile Switching Center
- EIR: Equipment Identity Register
**Authentication: when**

- **MS** to **BSS/MSC**: Loc. Upd. Request
  - IMSI, LAI
- **BSS/MSC** to **MS**: Update Loc. Area
  - IMSI, LAI
- **MS** to **VLR**: Locat. Upd. Request
  - TMSI Realloc ACK
- **VLR** to **MS**: TMSI Realloc Cmd
  - TMSI Realloc ACK
- **VLR** to **MS**: Update Location
  - IMSI, MSRN
- **VLR** to **HLR**: Insert Subscr. Data
  - IMSI
  - IMSI additional data
  - IMSI, LAI
- **HLR** to **VLR**: Update Location
  - IMSI, MSRN
  - IMSI
  - IMSI, LAI
- **VLR** to **HLR**: Insert Subscr. Data
  - IMSI
  - IMSI additional data
  - IMSI, LAI
- **HLR** to **VLR**: Auth. Info
  - Auth. Parameters
- **VLR** to **MS**: Authentication Request
  - IMSI, AUTHENTICATION
  - Challenge: 128 bit RAND
- **MS** to **VLR**: Authentication Request
  - IMSI, AUTHENTICATION
  - Challenge: 128 bit RAND
- **VLR** to **MS**: Authentication Response
  - SRES
  - Kc
- **MS** to **VLR**: Authentication Response
  - Signed RESult: 32 bit SRES
  - Equal?

**Authentication (managed by VLR)**

- **MS** to **VLR**: Authentication Request
  - Challenge: 128 bit RAND
- **VLR** to **MS**: Authentication Response
  - Signed RESult: 32 bit SRES
  - Equal?

---

**Giuseppe Bianchi**

---

**22**
**Triplets (Authentication Vector)**

- **Idea:** Once in a VLR area, authentication will need to be performed MANY times.

- **Hence deliver N triplets, to be used for N distinct authentications**

- **IMPORTANT:** VLR does NOT need to know authentication algo used (A3, A8)
  - Triplet contains computed result by AuC
  - A3, A8 run inside the SIM (given by operator)

**Authentication: details**

- **Ki** 128 bit secret
- **RAND** 128 bit challenge

- **A38 = A3 and A8 at same time**

- **SRES** 32 bit
- **Kc** 64 bits (well, 54 + ten 0s)

**Challenge response with:**

- Challenge  →  RAND
- Secret  →  Ki
- Hash  →  A3 algorithm
On the A3/A8 algorithms

- **Security by obscurity**
  - A3 algorithm CAN BE operator-specific!
  - But most vendors originally used algo today called COMP128
  - Non disclosed but...
    - Reverse engineered? Leaked out?
- **COMP128 broken in 1998**
  - [http://www.isaac.cs.berkeley.edu/isaac/gsm-faq.html](http://www.isaac.cs.berkeley.edu/isaac/gsm-faq.html)
  - Chosen challenge attack [1998, Briceno, Goldberg, Wagner]
    - Ki disclosed through about 150,000 queries with suitably selected challenges
    - approx 8h in 1998
    - Having Ki means cloning the card!
  - Better attack [2002, Rao, Rohatgi, Scherzer, Tinguely]
    - Less than 1 minute!
- **Lesson learned:**
  - Security by obscurity does NOT work!! Leave hash design to crypto experts

Over the air attach

- **No mutual authentication!**

- **Rogue BTS may easily perform the attach**
  - run it for sufficient time
UMTS authentication: AKA Authentication and Key Agreement

(Simplified for our purposes)

Major differences with GSM

- Mutual authentication!
  - Optimized...
  - Guaranteed freshness for auth parameters
- More comprehensive security
  - More keys and several extra details
  → (we will not focus on this)
- Algorithms FIRST scrutinized by research community, THEN selected
  - The opposite of security by obscurity ☺
**Authentication Vector**

- **MS**
- **RAND, AUTN**
- **RES**
- **N x Auth Vector**

**Authentication Vector:**
- XRES (expected result) as GSM SRES
- RAND as GSM RAND
- AUTN (Netw. Auth. token) as GSM RAND
- CK (cipher key) as GSM Kc
- IK (integrity key) as GSM Kc

**MS authentication**

- **Usual challenge response, but different (public) algorithm**

<table>
<thead>
<tr>
<th>K</th>
<th>RAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>f2</td>
<td>RES 32-128 bit</td>
</tr>
<tr>
<td>f3</td>
<td>CK Cipher key 128 bit</td>
</tr>
<tr>
<td>f4</td>
<td>IK Integrity Key 128 bit</td>
</tr>
</tbody>
</table>
Network Authentication

→ **MS should send a nonce**…
   - 1 extra message

→ **Bright idea: use Sequence number as “implicit” nonce!**
   - Issue: MS and AuC must be (approx) synchronized
   - And robust procedures for resync must be specified

SEQ as nonce: idea

Current SQN-MS stored

Check
SQN = SQN-MS+1 (or in appropriate Tolerance range to come with lost msg)

Use SQN as “implicit” challenge!!

Once auth OK, update local SQN
**Network authentication:**

**AUTN format**

<table>
<thead>
<tr>
<th>SQN</th>
<th>AMF</th>
<th>MAC-A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence number</td>
<td>Auth &amp; key mgmt field</td>
<td>Message Auth code</td>
</tr>
</tbody>
</table>

- **SQN**: 48 bit
- **AMF**: 16 bit: carries info on which algo or key to use if choice available (signalling info)
- **MAC-A**: 64 bit: allows MS to verify authenticity of Network!

MS: has all info needed to check that MAC-A transmitted by network is the same of MAC-A locally computed! SQN guarantees defense against replay.

---

**Minor detail: protecting SQN!**

- **A privacy problem:**
  - By looking at SQN (stepwise increasing), eavesdropper may discriminate and track user!

- **Solution: mask SQN with Anonymity Key**

<table>
<thead>
<tr>
<th>SQN xor AK</th>
<th>AMF</th>
<th>MAC-A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence number</td>
<td>Auth &amp; key mgmt field</td>
<td>Message Auth code</td>
</tr>
</tbody>
</table>

- **SQN xor AK**: 48 bit
- **AMF**: 48 bit
- **MAC-A**: 48 bit

---

Giuseppe Bianchi