cefriel
Consorzio per la Formazione e la Ricerca in Ingegneria dell'Informazione
Politecnico di Milano

POLITECNICO DI MILANO

Password

Davide Cerri
CEFRIEL - Politecnico di Milano
cerri@cefriel.it
http://www.cefriel.it/~cerri/

Autenticazione utente

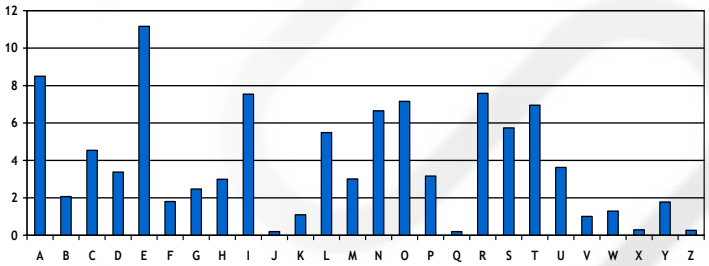
- Per **autenticare un utente** si può utilizzare:
  - ciò che l'utente **conosce** ("what you know");
    - PIN, password
  - ciò che l'utente **possiede** ("what you have");
    - tessera magnetica, smart-card, token…
  - ciò che l'utente **è** ("what you are").
    - biometria
- Per avere un sistema di autenticazione più forte si possono anche **combinare più fattori**.
  - Ad esempio tessera magnetica (oggetto posseduto) più PIN (informazione conosciuta).

Username e password

- Il metodo di autenticazione più semplice è quello basato su **username e password**: l'utente inserisce un nome che lo identifica (lo username), solitamente non segreto, e una parola segreta (la password).
- Vantaggi:
  - semplice per l'utente,
  - economico,
  - non richiede di immagazzinare un segreto lato client.
- Svantaggi:
  - spesso gli utenti scelgono cattive password,
  - spesso i metodi di autenticazione basati su password sono deboli.

Quanto è sicura una password? (1)

- Se una password fosse una **combinazione qualsiasi** delle 26 lettere dell'alfabeto, avremmo un'entropia di $\log_2 26 \approx 4{,}7$ bit per lettera.
- Ma solitamente una password è una **parola**, non una combinazione casuale di lettere, e le lettere **non sono equiprobabili**.



Frequenze tratte dalle voci principali del *Concise Oxford Dictionary* nona edizione (1995)

Quanto è sicura una password? (2)

- Se si considera che le lettere non sono equiprobabili l'entropia della lingua inglese si riduce da 4,7 bit per lettera a circa:
  - 4,2 bit se si considerano le **probabilità delle lettere**;
  - 3,9 bit se si considerano le **probabilità dei digrammi**;
  - ...
  - 1,5 bit se si considera un **intero testo**.
- Ma una password non è un testo, per cui l'ultima stima è eccessivamente pessimista.
- Inoltre i **caratteri possibili** in una password sono ben più di 26: ci sono lettere maiuscole, minuscole, numeri e altri simboli.

Attacchi alle password

- Attacchi possibili:
  - **intercettazione** (se la password passa in chiaro)
  - **guessing/cracking**
    - si può fare un attacco a pura forza bruta...
    - ...o più spesso un "**attacco a dizionario**", provando parole di senso compiuto (o loro minime variazioni) in modo da sfruttare la bassa entropia.
- Una password dovrebbe essere abbastanza lunga, non essere una parola di senso compiuto, e dovrebbe essere cambiata di frequente.
  - Ma questo si scontra con la "comodità" e la "pigrizia" degli utenti...
- Altri attacchi: social engineering, trojan horse...

On-line e off-line password guessing

- Il password guessing può essere condotto "on-line" oppure "off-line":
  - Attacco **on-line**: l'attaccante fa dei tentativi di autenticazione **interagendo direttamente con il sistema**. Il sistema può quindi accorgersi dell'attacco e prendere contromisure (rallentare, bloccare l'accesso, lanciare un allarme) ma bisogna stare attenti a non facilitare in questo modo attacchi di tipo "denial of service".
  - Attacco **off-line**: l'attaccante ha ottenuto una o più password in una forma "offuscata" (ad esempio ha ottenuto degli hash), e procede per tentativi **senza dover interagire con il sistema** di autenticazione.

Scelta della password (1)

- Come fare per evitare che gli utenti scelgano cattive password? Alcune possibilità:
  - **educare gli utenti**: è fondamentale, ma di solito non è sufficiente;
  - **assegnare password generate casualmente**: sequenze casuali di lettere sono però difficili da ricordare, per cui gli utenti le annotano. Si può restringere la generazione a password "pronunciabili", ma questo riduce l'entropia;
  - lasciare scegliere le password agli utenti stessi ma **effettuare dei controlli** (a posteriori oppure al momento della scelta).

Scelta della password (2)

- Controllare le password scelte dagli utenti:
  - **a posteriori**: il sistema periodicamente tenta il cracking delle password, per trovare password deboli (costoso, il cracking è pesante);
  - **all'atto della scelta** da parte dell'utente: il sistema controlla la password quando l'utente la sceglie, rifiutandola se è debole. Tecniche utilizzabili:
    - semplici regole (ad esempio: almeno 8 caratteri, almeno un carattere non alfabetico…),
    - dizionari (se la password è nel dizionario viene rifiutata, per essere efficiente però serve un dizionario molto grande, e quindi costoso in termini di spazio e tempo),
    - tecniche probabilistiche (modelli Markoviani, filtri di Bloom).

Password UNIX (1)

- Esempio: le **password UNIX**.
- Ogni utente ha una password: come si possono immagazzinare le coppie username-password?
  - non in un file in chiaro, perché potrebbe essere letto da altri;
  - in un file cifrato?
    - con quale chiave?
    - dove e come si immagazzina la chiave di cifratura?
- In UNIX non si memorizzano le password, ma i loro **hash (password hashing)**:
  - il sistema **non ha bisogno di conoscere la password**: quando l'utente la fornisce ne calcola l'hash e lo confronta con quello memorizzato.

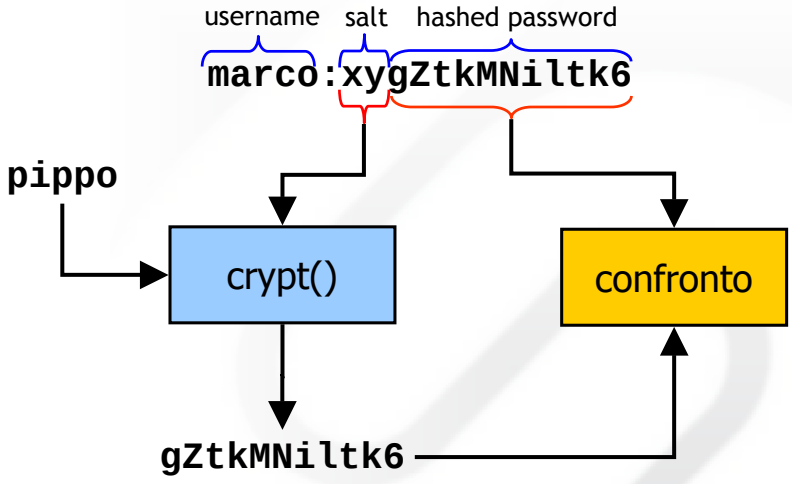
Password UNIX (2)

- L'hash di UNIX (noto come "**UNIX crypt()**") è basato su una **versione modificata del DES**:
  - si utilizza la password (8 caratteri: 7 bit per carattere danno 56 bit) **come chiave** per cifrare 64 bit di zeri, ripetendo la procedura **25 volte**: il risultato costituisce l'hash e viene memorizzato nel file delle password (**`/etc/passwd`**).
  - nell'algoritmo entra anche un numero "casuale" (detto **salt**) di 12 bit, che viene salvato poi in chiaro insieme con l'hash; il salt serve a:
    - rendere più difficili gli attacchi a dizionario;
    - generare hash diversi da password uguali;
    - modificare l'algoritmo DES rispetto a quello standard (il salt modifica la funzione di espansione del DES).

Password UNIX (3)

- L'utilizzo di una versione modificata del DES (per cui non si possono usare le normali implementazioni hardware del DES) e la ripetizione della cifratura 25 volte hanno lo scopo di **peggiorare le prestazioni**:
  - si vuole **rallentare il calcolo dell'hash**, per rendere più difficili attacchi che impiegano molti tentativi.
  - **aveva senso negli anni '70**, oggi molto meno.
- Oggi inoltre si usano le "**shadow password**": dato che il file **`/etc/passwd`** deve essere leggibile da tutti, gli hash delle password vengono memorizzati in un altro file (**`/etc/shadow`**) leggibile solo dall'utente root.
  - per un attacco off-line è necessario avere a disposizione gli hash.

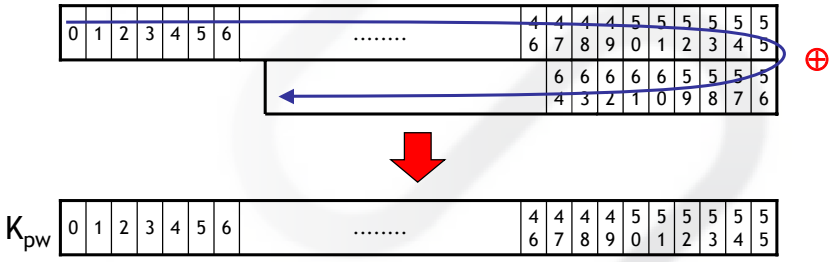
Password UNIX (4)



Password - 13 - Davide Cerri

Password Kerberos (1)

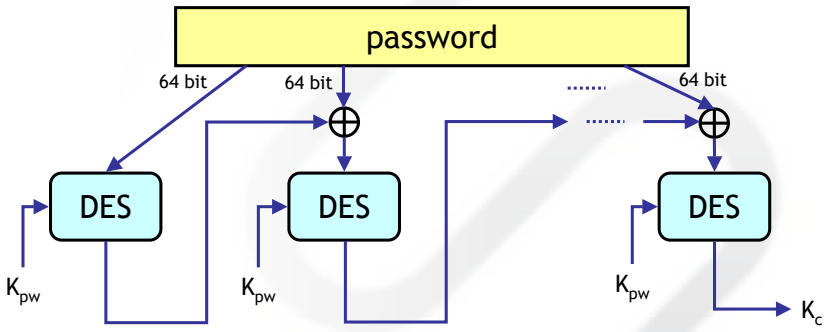
- In **Kerberos** si usa una trasformazione (di tipo hash) per ricavare una chiave dalla password dell'utente.
- Per prima cosa la stringa della password (in codifica ASCII a 7 bit) viene "ripiegata a ventaglio" su una lunghezza di 56 bit, facendo poi lo XOR tra le diverse "pieghe".



Password - 14 - Davide Cerri

Password Kerberos (2)

- I 56 bit risultanti dallo XOR (K_{pw}) vengono utilizzati come chiave DES per cifrare (con DES CBC, 64 bit per volta) la stringa della password. Il risultato è la chiave associata alla password (K_c).



Autenticazione Windows

- Sistemi di autenticazione nelle reti **Windows** (fino a Windows NT4 compreso):
  - LAN Manager (**LANMAN** o **LM**);
  - NT LAN Manager (**NTLM**, **NTLMv2**).
- A partire da Windows 2000 Microsoft ha abbandonato la famiglia LM/NTLM e ha adottato una versione modificata di **Kerberos v5** (protocollo standard già diffuso da anni in ambiente Unix).
- I vecchi protocolli sono comunque **tuttora in uso**, per ragioni di compatibilità con sistemi Windows 9x/NT.
- LM presenta **numerose vulnerabilità**.

Autenticazione Windows: hash LM

- LM utilizza tecniche di **password hashing** e autenticazione a **sfida e risposta**.
- L'hash della password è calcolato in questo modo:
  - la password viene troncata a 14 caratteri, oppure, se è più corta, sono inseriti in coda dei caratteri nulli;
  - i caratteri vengono **convertiti in maiuscolo**;
  - la stringa di 14 caratteri viene **divisa in due stringhe** da 7 caratteri (prima e seconda metà);
  - ognuna delle due stringhe viene utilizzata come **chiave DES** (7 caratteri x 8 bit/carattere = 56 bit) **per cifrare una costante** di 64 bit (**KGS!@#$%**);
  - i due risultati vengono **concatenati** in una stringa di 128 bit che costituisce l'hash.

Autenticazione Windows: debolezze hash LM (1)

- L'hash LM, lungo 128 bit, potrebbe a prima vista sembrare più forte dell'hash Unix (lungo 64 bit).
- In realtà ha **enormi difetti**:
  - la **conversione dei caratteri in maiuscolo** riduce di molto l'entropia della password, che sarà ben lontana dagli 8 bit per carattere considerati;
  - al contrario dell'hash Unix, LM **non prevede un salt**, per cui attacchi a dizionario sono molto più semplici;
  - la **suddivisione della password nelle due metà** semplifica moltissimo il cracking.

Autenticazione Windows: debolezze hash LM (2)

- Dividendo la password nelle due metà accade che:
  - la **dimensione dello spazio d'attacco diminuisce moltissimo**: con una password alfabetica di 14 caratteri lo spazio è grande $26^{14} \approx 6{,}5 \times 10^{19} \approx 65{,}8$ bit, ma con due password alfabetiche di 7 caratteri diventa $2 \times 26^{7} \approx 1{,}6 \times 10^{10} \approx 33{,}9$ bit. Il **cracking delle due metà può infatti essere fatto separatamente**!
  - fare il **cracking della seconda metà della password è spesso facile**, perché spesso questa metà sarà più corta di 7 caratteri (in quanto la password sarà spesso più corta di 14 caratteri). La seconda metà così scoperta potrebbe poi essere d'aiuto per scoprire la prima metà...

Autenticazione Windows: sfida e risposta in LM

- Per l'autenticazione diretta LM prevede un meccanismo a **sfida e risposta**:
  - il server invia una sequenza casuale di 64 bit (sfida);
  - il client calcola l'hash LM della password dell'utente (128 bit), aggiunge 40 bit di zeri, suddivide il risultato in tre blocchi da 56 bit. Ogni blocco viene utilizzato come chiave DES per cifrare la sfida, la risposta è data dalla concatenazione dei tre risultati (192 bit).
- Problemi:
  - con una coppia sfida-risposta si può fare un **attacco a dizionario** per ottenere la password (e notate i padding...);
  - ciò che viene utilizzato per autenticarsi è **l'hash, non la password**: l'hash è un "password equivalent".

Autenticazione Windows: NTLM

- L'hash **NTLM** è calcolato nel modo seguente:
  - la password viene **codificata in Unicode** (16 bit/carattere), preservando maiuscole e minuscole;
  - viene calcolato un **hash MD4** della stringa Unicode (128 bit).
- Si eliminano la conversione in maiuscolo e la divisione a metà, non si introduce però un salt.
- L'autenticazione sfida-risposta è **identica a quella di LM**, utilizzando l'hash NTLM anziché l'hash LM.
  - Il problema è che per compatibilità molti client inviano **sia la risposta NTLM che la risposta LM**.
    - Si fa il cracking dell'hash LM, e una volta ottenuta la password in maiuscole si fa il cracking dell'hash NTLM per scoprire quali lettere sono invece minuscole.

One-time password

- Con il termine "**one-time password**" ci si riferisce a sistemi in cui viene generata una **nuova password ad ogni accesso da parte dell'utente**, per risolvere il problema dell'intercettazione.
- Queste password "monouso" vengono generate sulla base di un **contatore** (esiste quindi una sequenza di password successive) oppure sulla base dell'**istante temporale**.
- Spesso i sistemi one-time password si appoggiano su "**token**", dispositivi hardware che forniscono all'utente la password da inserire (eventualmente sostituiti da implementazioni software dell'algoritmo di generazione delle password).

One-time password più PIN

- Chi entra in possesso del token può autenticarsi… per aumentare la sicurezza si può allora utilizzare un **PIN** combinato con la one-time password.
- Tre possibili strategie:
  - PIN accodato alla one-time password
  - PIN come password per il token
  - PIN come parte del segreto base
- Il PIN può anche essere breve (ad esempio solo 4 o 5 cifre), dato che è pensato come strumento complementare (la sicurezza non è affidata totalmente al PIN).

Hash di Lamport e OTP

- Con lo schema dell'**hash di Lamport** (proposto da Leslie Lamport) si crea una **sequenza di password** che sia **percorribile solo in un senso**, utilizzando una funzione di hash.
- Questa catena di password può essere utilizzata come sistema di **one-time password**.
- La particolarità di questo schema sta nel fatto che **nessun dato segreto** (password o chiave) **deve essere memorizzato sul server**.
- Un sistema di questo tipo è **OTP** (descritto in RFC 2289), che deriva dal sistema **S/KEY** (sviluppato da Bellcore).

OTP: inizializzazione

- L'utente sceglie una **passphrase** (10-63 caratteri) e la concatena con un **seme** fornito dal server, ottenendo il dato iniziale (*S*).
  - La presenza del seme permette di **riutilizzare la stessa passphrase** più volte e/o con più macchine.
- L'utente può calcolare l'*i*-esima one-time password della sequenza di *N* elementi ripetendo *N-i* operazioni di hash su *S*:

$$p_i = \mathrm{H}^{N-i}(S) \qquad 0 \le i \le N-1$$

dove H è una funzione di hash.

- L'utente comunica p_0 (prima password della sequenza) al server attraverso un canale sicuro.

OTP: utilizzo

- Il client invia lo username al server.
- Il server invia il numero di sequenza *i* e il seme.
- Il client concatena la passphrase fornita dall'utente con il seme ottenendo *S*, calcola la one-time password richiesta p_i e la invia al server.
- Il server ha immagazzinato p_{i-1} e calcola $\mathrm{H}(p_i)$: se queste coincidono l'autenticazione va a buon fine.

$$\mathrm{H}(p_i) = \mathrm{H}\left(\mathrm{H}^{N-i}(S)\right) = \mathrm{H}^{N-i+1}(S) = p_{i-1}$$

OTP: problemi

- Permette di autenticarsi un **numero prefissato di volte**, dopo di che richiede una reinizializzazione.
- L'autenticazione **non è mutua** (il server non si autentica) e non genera un segreto (la connessione può poi essere dirottata), per cui è vulnerabile ad attacchi man-in-the-middle.
- È il server a inviare il numero di sequenza. Questo rende possibili **attacchi al numero di sequenza,** in cui l'attaccante invia un numero più alto di quello reale, e una volta ottenuta la password relativa può calcolare le password successive e utilizzarle per autenticarsi un certo numero di volte.

SRP

- **SRP** (Secure Remote Password) è un protocollo di autenticazione basato su password proposto da Thomas Wu nel 1998 e standardizzato in RFC 2945 (la versione descritta è SRP-3).
- L'utente deve ricordare e inserire una **password**, ma lo schema, utilizzando una tecnica simile allo scambio **Diffie-Hellman**, è robusto anche se la password è relativamente debole, in quanto l'intercettazione dei messaggi del protocollo **non è sufficiente** per effettuare attacchi off-line a dizionario.
- Il server deve immagazzinare un "**verificatore**", non la password in chiaro dell'utente.

SRP: caratteristiche

- Il protocollo non è vulnerabile ad **attacchi a dizionario** sulla password da parte di un attaccante attivo o passivo.
- Il server non memorizza un "**password equivalent**", se l'attaccante si impadronisce del database del server deve fare un attacco a dizionario.
- La chiave di sessione non è derivata dalla password (bassa entropia) ma proviene da uno **scambio simile al Diffie-Hellman**.
- La compromissione in un dato momento della password non compromette le sessioni passate (**perfect forward secrecy**).

SRP: funzionamento (1)

- Siano U lo username, P la password, s un salt, H una funzione di hash, m un numero primo molto grande, g una sua radice primitiva (m e g scelti come in Diffie-Hellman). Sia $x = H(s \| P)$, allora il verificatore è $v = g^x \bmod m$. Il server immagazzina U, s e v (non immagazzina né P né x).
- Il client sceglie un numero a, calcola $A = g^a \bmod m$ e invia al server A e U.
- Il server cerca nel suo database s e v corrispondenti a U ricevuto, sceglie due numeri b e u, calcola $B = v + g^b \bmod m$ e manda al client B, u e s.

SRP: funzionamento (2)

- Sia $K_s = g^{b(a+ux)} \bmod m$, la chiave è $K = H(K_s)$.
  - il client la può calcolare così:
    $K_s = (B - g^x)^{a+ux} \bmod m = (g^x + g^b - g^x)^{a+ux} \bmod m =$
    $= g^{b(a+ux)} \bmod m$.
  - il server la può calcolare così:
    $K_s = (Av^u)^b \bmod m = (g^a g^{ux})^b \bmod m = g^{b(a+ux)} \bmod m$.
- Con gli ultimi due messaggi client e server si dimostrano reciprocamente di conoscere la chiave segreta K:
  - il client invia $M = H(H(m) \oplus H(g) \| H(U) \| s \| A \| B \| K)$;
  - il server invia $H(A \| M \| K)$.