Autenticazione

- Fondamenti
- Password
- Challenge-Response
- Biometria
- Localizzazione
- Metodi Multipli

Fondamenti

- Autenticazione: associare un'identità ad un soggetto
  - L'identità è quella di un'entità esterna (la mia identità, RDP, etc.)
  - Il soggetto è un'entità su un computer (un processo, etc.)

Stabilire l'identità

- Si può sfruttare uno o più dei seguenti attributi:
  - **Cosa conosce** l'entità (es. password)
  - **Cosa ha** l'entità (es. badge, smart card)
  - **Cosa è** l'identità (es. impronta digitale, caratteristiche della retina)
  - **Dove** si trova l'entità (es. davanti ad uno specifico terminale)

Sistema di autenticazione

- Quintupla (A, C, F, L, S)
  - A: informazione che attesta l'identità
  - C: informazione conservata su un computer ed usata per validare l'informazione di autenticazione
  - F: funzione di complementazione; $f : A \rightarrow C$
  - L: funzione che prova l'identità
  - S: funzione che permette ad un'entità di creare o alterare le informazioni in A o in C

Esempio

- Password system, con password di tipo testo memorizzate in chiaro
  - *A* insieme delle stringhe che compongono le password
  - *C* = *A*
  - *F* insieme singleton (valori singoli) della funzione identità {*I*}
  - *L* funzione del test di uguaglianza {*eq*}
  - *S funzione per impostare/cambiare password*

Password

- Sequenza di caratteri
  - Esempi: 10 cifre, una stringa di lettere, etc.
  - Possono essere generate in maniera casuale da un utente, da un computer con l'input di un utente
- Sequenza di parole
  - Esempio: pass-phrase
- Algoritmi
  - Esempi: challenge-response, one-time password (OTP)

Memorizzazione

- Memorizzare in chiaro
  - Se il file delle password è compromesso, **tutte** le password sono rivelate
- File cifrato
  - Richiede la memorizzazione delle chiavi di cifratura e di decifratura in memoria
  - Si riduce al problema precedente
- Memorizzare l'*hash* della password
  - Se il file è letto, l'attaccante può soltanto indovinare la password, oppure invertire la funzione hash

Esempio

- Sistema standard di UNIX per l'*hash*
  - A partire dalla password, crea un'hash di 11 caratteri, utilizzando una delle 4096 funzioni hash
- Il sistema di autenticazione:
  - *A* = { stringa di 8 o meno caratteri }
  - *C* = { 2 caratteri per l'id della funzione hash || 11 caratteri per l'hash della password }
  - *F* = { 4096 versioni modificate del DES }
  - *L* = { *login*, *su*, … }
  - *S* = { *passwd*, *nispasswd*, *passwd+*, … }

Anatomia di un attacco

- Scopo: trovare $a \in A$ tale che:
  - Per qualche $f \in F$, $f(a) = c \in C$
  - c è associata ad un'entità
- Due modi per determinare se a soddisfa questi requisiti:
  - Approccio diretto: come sopra
  - Approccio indiretto: dato che $I(a)$ è verificato se e soltanto se $f(a) = c \in C$ per qualche c associato ad un'entità, calcolare $I(a)$

Prevenire gli attacchi

- Come prevenire:
  - Nascondere uno fra *a*, *f* o *c*
    - Previene gli attacchi di cui sopra
    - Esempio: UNIX/Linux oscura il file delle password
      - Nasconde *c*
  - Bloccare gli accessi a tutte le $l \in L$ oppure ai valori *l(a)*
    - Previene che l'attaccante scopre la password se indovina il valore di *l(a)*
    - Esempio: prevenire *tutti* i login provenienti dalla rete ad un account
      - Previene la conoscenza del valore di *l* (oppure l'accesso a *l*)

Attacco con dizionari

- Trial-and-error (Tentativi ed errori) da una lista di possibili password
  - *Off-line:* conoscendo f e c, provare ad indovinare ripetutamente diversi valori $g \in A$ fino a terminare la lista o ad indovinare la password
    - Esempi: *crack*, *john-the-ripper*
  - *On-line*: avere l'accesso alle funzioni in *L* e provare ad indovinare *g*, finché *l(g)* ha successi
    - Esempi: provare ad eseguire un login, indovinando una password

Usare il tempo

Formula di Anderson:

- P probabilità di indovinare una password in un determinato periodo di tempo
- G numero di tentativi esperibili in una unità di tempo
- T numero di unità di tempo
- N numero possibile di password ($|A|$)
- Allora $P \approx TG/N$

Esempio

- Goal
  - Password prese da un alfabeto di 96 caratteri
  - 10^4 tentativi possibili in un secondo
  - La probabilità di un successo è 0.5 in un periodo di 365 giorni
  - Qual è la minima lunghezza della password?
- Soluzione
  - $N \geq TG/P = (365 \times 24 \times 60 \times 60) \times 10^4 / 0.5 = 6.31 \times 10^{11}$
  - Scegliere s tale che $\Sigma^s_{j=1} 96^j \geq N$
  - Dunque, $s \geq 6$, ovvero le password devono essere lunghe almeno 6 caratteri

Scegliere una password: Approcci

- Scelta casuale
  - Ogni possibile password di *A* ha la stessa probabilità di essere scelta
- Password pronunciabili
- Selezione delle password da parte dell'utente

Password pronunciabili

- Generare fonemi in maniera casuale
  - Un fonema è un'unità di suono, es. cv, vc, cvc…
  - Esempi: *helgoret, juttelon*; non lo sono *przbqxdfl, zxrptglfn*
- Problema: sono troppo pochi
- Soluzione: key crunching
  - Dare in pasto delle chiavi abbastanza lunghe a funzioni hash e convertirle in sequenze stampabili
  - Usare come password queste sequenze

Scelta dell'utente

- Problema: di solito si scelgono password facilmente indovinabili
  - Basate sul nome dell'**account**, sul nome utente, sul nome del computer, sul nome di luoghi
  - Parole da **dizionario** (anche rovesciate, con le dispari maiuscole, con caratteri di controllo, "elite-speak", coniugazioni o declinazioni, parolacce, parole della Bibbia, del Corano, della Torah…)
  - Troppo **corte**, fatte di **soli numeri** o di **sole lettere**
  - Basate su targhe automobilistiche, acronimi, numeri della sicurezza sociale
  - Basate su caratteristiche personali o manie (nome dell'animale, soprannomi, lavori)

Scegliere buone password

- "lIMm*2^Ap"
  - Nomi di membri di due famiglie
- "CoddPA/O"
  - Le lettere delle parole iniziali dell'Odissea, seguite da "/", seguite dalle iniziali dell'autore
- Quello che va bene da una parte potrebbe non andare bene da un'altra
  - "DMC/MHmh" non va bene per Dartmouth ("Dartmouth Medical Center/Mary Hitchcock memorial hospital"), va bene altrimenti
- Perché queste password non vanno bene?

Controllo proattivo delle password

- Analizzare la password proposta in base alla sua bontà
  - Deve essere sempre fatto
  - Può scoprire e segnalare password cattive, per appropriate definizioni di "cattivo"
  - Può essere specializzato in base all'utente o a un luogo
  - Necessita un sistema di pattern matching
  - Ha bisogno di sottoprogrammi e ne usa l'output
    - Ad esempio analizzatori lessicali
  - E' facile da preparare e da integrare nel sistema di selezione della password

Esempio: OPUS

- Goal: controllare la password contro l'uso di ampi dizionari
  - Si da in pasto ogni parola del dizionario a k diverse funzioni hash $h_1, \ldots, h_k$, producendo valori più piccoli di n
  - Si settano i bit $h_1, \ldots, h_k$ nel dizionario OPUS
  - Per controllare la nuova parola proposta, si genera il vettore dei bit e si controlla l'insieme di tutti i bit corrispondenti
    - Se è così, la parola è nel dizionario con un certo grado di probabilità
    - Se non è così, la parola non è nel dizionario

(Bloom filter)

Esempio: *passwd+*

- Fornisce un piccolo linguaggio per descrivere la verifica proattiva
  - test length("$p") < 6
    - Se la password è più corta di 6 caratteri, la rifiuta
  - test infile("/usr/dict/words", "$p")
    - Se la password è nel file /usr/dict/words, la rifiuta
  - test !inprog("spell", "$p", "$p")
    - Se la password viene accettata come input di un programma di divisione in sillabe, la rifiuta (poiché è una parola sillabata correttamente)

Salting

- Goal: rallentare gli attacchi con dizionario
  - Metodo: perturbare le funzioni hash
  - Esempio: usare il salt come la prima parte dell'input di una funzione hash

Indovinare tramite L

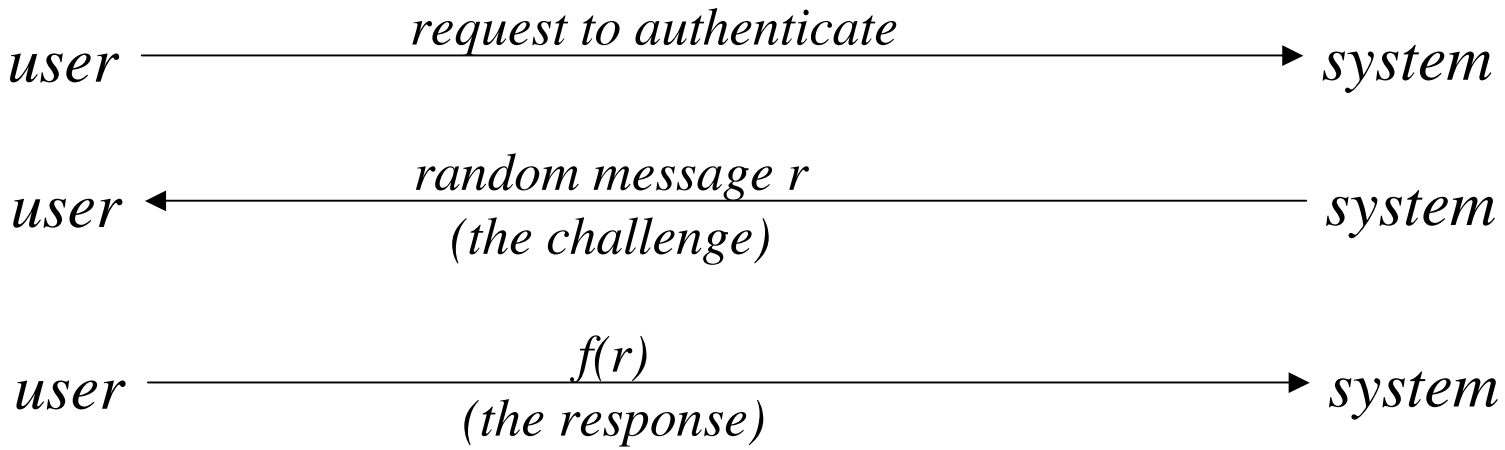
- Non può essere evitato
  - Altrimenti, gli utenti legittimi non potrebbero fare il log in
- Può essere rallentato
  - Backoff
  - Disconnessione
  - Disabilitazione
    - Bisogna prestare attenzione con gli account amministrativi!
  - Jailing
    - Abilitare solo per attività limitate

Invecchiamento delle password

- Obbligare l'utente a cambiare la password dopo che sia trascorso un certo periodo
  - Come forzare l'utente a non riutilizzare le password?
    - Memorizzare le password obsolete
  - Dare all'utente il tempo necessario a scegliere buone password
    - Non forzarlo a cambiare prima che abbia eseguito il login
    - Avvertirlo prima del giorno di scadenza

Challenge-Response

- L'utente e il sistema condividono una funzione segreta *f* (in pratica, *f* è una funzione nota, con parametri non noti, come ad esempio la chiave di cifratura)



Pass Algorithm

- Challenge-response dove anche la funzione *f* è un segreto
  - Esempio:
    - Il Challenge è una stringa casuale di caratteri, come "abcdefg", "ageksido"
    - Il Response è una qualche funzione su questa stringa, come "bdf", "gkip"
  - L'algoritmo può cambiare in base ad informazioni ausiliarie
    - Le connessioni da rete sono come sopra, mentre i dial-up potrebbero richiedere "aceg", "aesd"

One-Time Password

- Password che possono essere usate soltanto una volta
  - Dopo l'uso, sono rese immediatamente non valide
- Meccanismo Challenge-response
  - Challenge è il numero di autenticazioni; response è la password per quel particolare numero
- Problemi
  - Sincronizzazione dell'utente con il sistema
  - Generazione di password "davvero" casuali
  - Problema della distribuzione delle password

S/Key

- Schema di One-time password, basato su un idea di Lamport
- *h* funzione hash non invertibile (MD5 o SHA-1, ad esempio)
- L'utente sceglie il seme iniziale *k*
- Il sistema calcola:

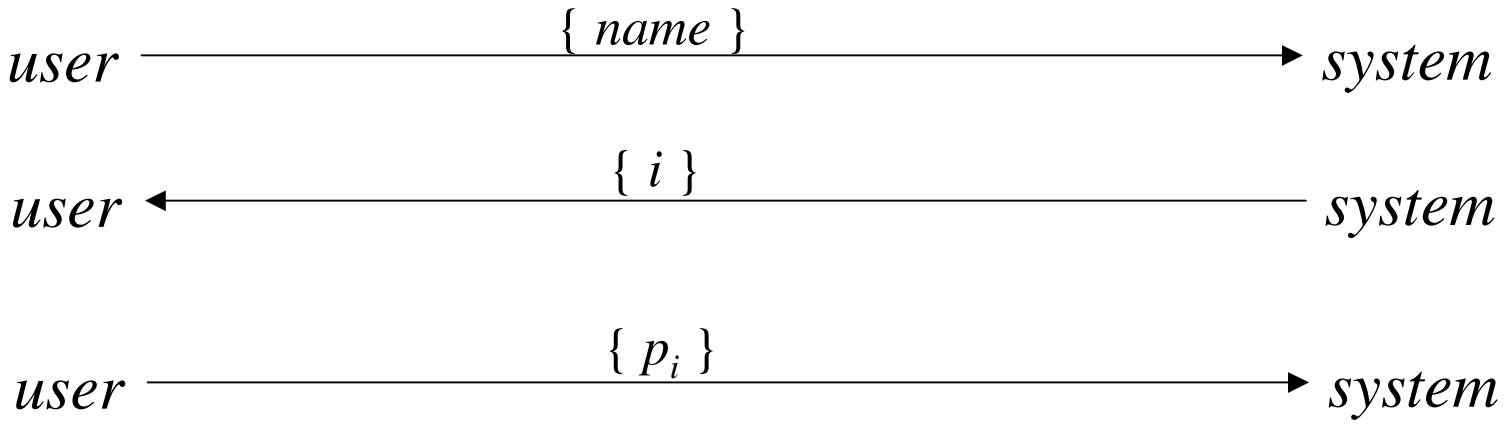
$$h(k) = k_1,\ h(k_1) = k_2,\ \ldots,\ h(k_{n-1}) = k_n$$

- Le password sono nell'ordine inverso:

$$p_1 = k_n,\ p_2 = k_{n-1},\ \ldots,\ p_{n-1} = k_2,\ p_n = k_1$$

Protocollo S/Key

Il sistema memorizza il massimo numero di autenticazioni n, il numero della prossima autenticazione i e l'ultima password correttamente inserita p_{i-1}.



Il sistema calcola $h(p_i) = h(k_{n-i+1}) = k_{n-i} = p_{i-1}$. Se coincide con le informazioni memorizzate, il sistema rimpiazza p_{i-1} con p_i ed incrementa i.

Supporto Hardware

- Token-based
  - Usato per calcolare il response di un challenge
    - Può cifrare o fare l'hash di un challenge
    - Può richiedere all'utente l'inserimento di un PIN
- Temporally-based
  - Ogni minuto (o altro) mostra un diverso numero
    - Il computer sa quale numero aspettarsi e quando
  - L'utente inserisce il numero e la password fissata

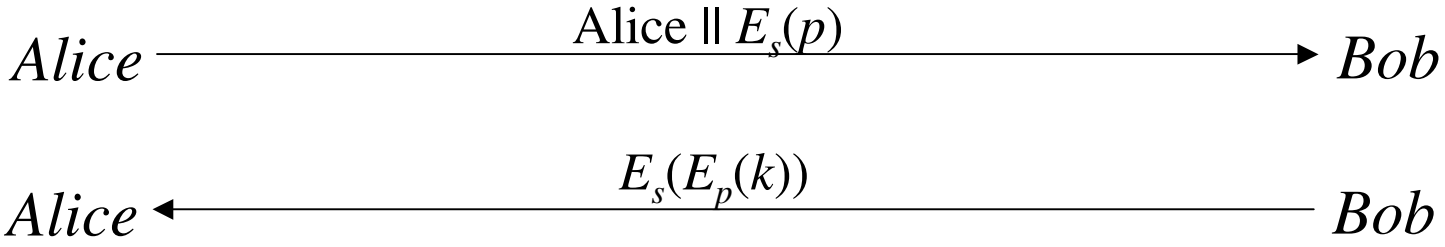
C-R e Attacchi con Dizionario

- Come per le password fissate
  - L'attaccante conosce il challenge *r* e il response *f(r)*; se conosce anche la funzione di cifratura *f*, può provare diverse chiavi
    - Può solo aver bisogno di sapere il formato della risposta; l'attaccante può verificare se ha indovinato controllando la presenza di oggetti della giusta forma
    - Esempio: Kerberos versione 4 utilizzava il DES, ma le chiavi avevano 20 bit casuali; gli attaccanti indovinavano velocemente le chiavi, poiché il testo decifrato (ticket) aveva un insieme fissato di bit in alcune posizioni

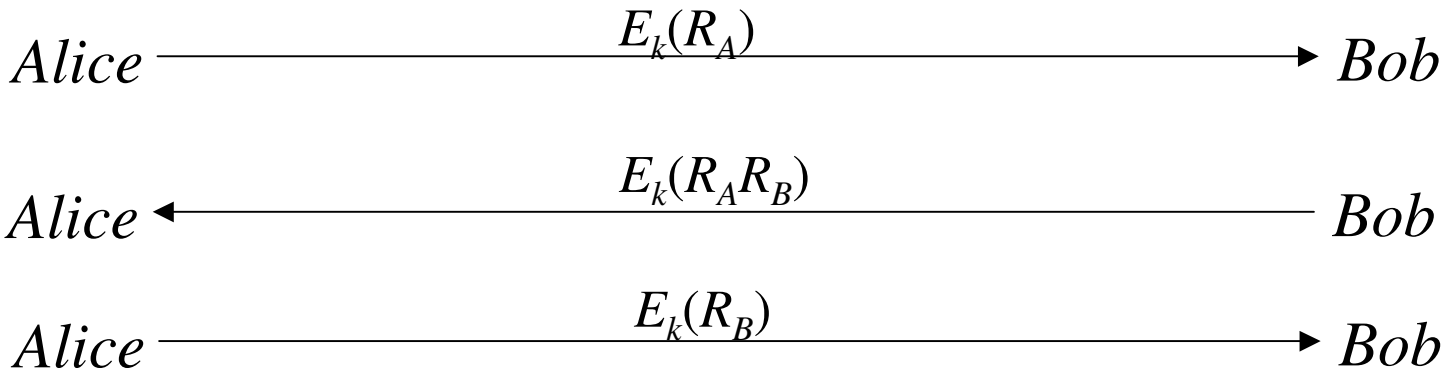
Scambio cifrato di Chiavi

- Vanifica gli attacchi *off-line* con dizionari
- Idea: challenge casuale e cifrato, in modo che l'attaccante non può verificare la corretta decifratura del challenge
- Ad esempio, Alice e Bob condividono la password segreta *s*
- Nel seguito, Alice ha bisogno di generare una chiave pubblica casuale *p* e una corrispondente chiave privata *q*
- Inoltre, *k* è una chiave di sessione generata casualmente e R_A e R_B sono challenge casuali

EKE Protocol



Adesso Alice e Bob condividono una chiave segreta di sessione k, generata in maniera casuale



Biometria

- Misure automatizzate di caratteristiche biologiche che permettono di identificare una persona
  - Impronte digitali: tecniche ottiche o elettriche
    - Associare le impronte a grafi e compararli con un database
    - Poiché le misure sono imprecise, si usano algoritmi per associazioni approssimate
  - Voce:
    - **Verifica**: si usano tecniche statistiche per controllare che chi parla è esattamente chi dice di essere (speaker dependent)
    - **Riconoscimento**: si controlla il contenuto delle risposte (speaker independent)

Altre caratteristiche

- Si possono usare diverse altre caratteristiche
  - Occhi: lo schema delle iridi è unico
    - "Measure pattern", determinano se le differenze sono casuali; oppure associare immagini usando test statistici
  - Volto: immagine o caratteristiche specifiche, come la distanza fra il naso e il mento
    - La luce, la vista di faccia o altre interferenze possono impedire il riconoscimento
  - "Keystroke dynamics": si pensa siano uniche
    - Intervalli di keystroke, pressione, durata del movimento, dove la chiave è colpita
    - Sono usati test statistici

Precauzioni

- Questi test possono essere ingannati!!
  - Si assume che il dispositivo per la misura biometrica sia preciso *nell'ambiente dove viene usato*
  - La trasmissione dei dati al sistema di verifica deve essere corretto, a prova di manomissione

Localizzazione (Location)

- Se si sa dove è l'utente, si assicura l'identità controllando se l'entità da autenticare è dove è l'utente
  - Richiede dispositivi dedicati per localizzare l'utente
    - GPS (global positioning system), dispositivo che fornisce la locazione firmata dell'identità
    - Host usa LSS (*location signature sensor*) per ritirare la firma dall'entità

Metodi Multipli

- Esempio: "dove sei" richiede che l'entità abbia LSS and GPS, così come "cosa hai"
- E' possibile assegnare diversi metodi per diversi scopi
  - Poiché gli utenti svolgono molti e diversi compiti delicati, devono autenticarsi in diversi modi (presumibilmente, più severi
- Pluggable Authentication Modules (PAM)

PAM

- Idea: quando un programma ha bisogno di autenticare un'entità, accede ad un deposito centrale per caricare il metodo da usare
- Library call: *pam_authenticate*
  - Accede al file con il nome del programma in */etc/pam_d*
- I moduli realizzano il controllo dell'autenticazione
  - *sufficient*: riesce se il modulo riesce
  - *required*: fallisce se il modulo fallisce, ma tutti i moduli devono essere eseguiti prima di riportare il fallimento
  - *requisite*: come *required*, ma non controlla tutti i moduli
  - *optional*: viene invocato soltanto se tutti i moduli precedenti falliscono

Sommario

- L'autenticazione non è crittografia
  - Si devono considerare le componenti del sistema
- Devono esserci le password
  - Forniscono una base per quasi tutte le forme di autenticazione
- I protocolli sono importanti
  - Possono rendere il mascheramento molto complicato
- I metodi di autenticazione possono essere combinati
  - Esempio: PAM