Authentication

CS 491/591

Fall 2015

Authentication

- Running example: Bob and the Unix machine

```
Login: bob
Password: hunter2
```

```
Last Login 1/12/13 3:05pm from console
[bob@desktop ~]$ ls /home/bob
Desktop  Documents  Downloads  Music  Pictures
[bob@desktop ~]$ ls /home/joe
ls: cannot open directory /home/joe: Permission denied
```

Authentication

How does the system know that Bob is really who he says?

`Login: bob`

Authentication: Who are you?

- Verify identity based on
  - Something you know
  - Something you have
  - Something you are

Something you know: Passwords

```
Login: bob
Password: hunter2
```

How does the system verify whether Bob's password is correct?

What not to do: The Sony Method

- In 2011, intruders stole information on up to 70 million users of Sony's Playstation Network, **including their PSN login passwords**
- http://blog.us.playstation.com/2011/04/26/update-on-playstation-network-and-qriocity/
- How did this occur? Apparently Sony stored the passwords as "plain text"

```
Login: bob
Password: hunter2
```

What not to do: The Sony Method

- In 2011, intruders stole information on up to 70 million users of Sony's Playstation Network, **including their PSN login passwords**
- http://blog.us.playstation.com/2011/04/26/update-on-playstation-network-and-qriocity/
- How did this occur? Apparently Sony stored the passwords as "plain text"

```
Login: bob
Password: hunter2
```

Username	Password
alice	kittens3
bob	hunter2
charlie	password1
doug	qwerty!
…	…

What not to do: The Sony Method

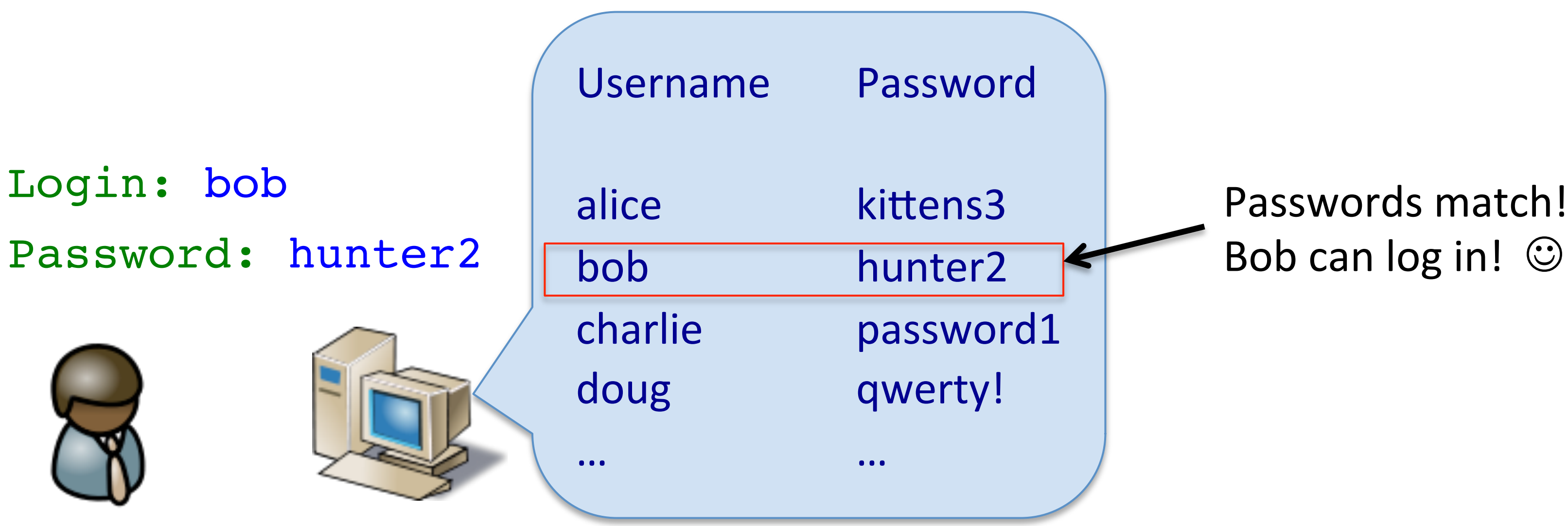
- In 2011, intruders stole information on up to 70 million users of Sony's Playstation Network, **including their PSN login passwords**
- http://blog.us.playstation.com/2011/04/26/update-on-playstation-network-and-qriocity/
- How did this occur? Apparently Sony stored the passwords as "plain text"

```
Login: bob
Password: hunter2
```

Username	Password
alice	kittens3
bob	hunter2
charlie	password1
doug	qwerty!
…	…

What not to do: The Sony Method

- In 2011, intruders stole information on up to 70 million users of Sony's Playstation Network, **including their PSN login passwords**
- http://blog.us.playstation.com/2011/04/26/update-on-playstation-network-and-qriocity/
- How did this occur? Apparently Sony stored the passwords as "plain text"



What not to do: The Sony Method

- In 2011, intruders stole information on up to 70 million users of Sony's Playstation Network, **including their PSN login passwords**
- http://blog.us.playstation.com/2011/04/26/update-on-playstation-network-and-qriocity/
- How did this occur? Apparently Sony stored the passwords as "plain text"

Username	Password
alice	kittens3
bob	hunter2
charlie	password1
doug	qwerty!
…	…

BUT anyone who gets this database can see everyone's passwords! ☹

Better password storage

- Need to verify that the password entered is correct
- Need to protect against attacker who can steal the password database
  - Can't store it in plain text!

Better password storage

- Idea: Store some value computed from the password, instead of the password itself
  - Use some function F() to compute F(password)
  - What function F do we need?
  - Should be hard to find another password that generates the same value
  - Should be hard to recover password from F(password)

What if we encrypt the passwords?

Enc Pwd = E(password, key)

```
Login: bob
Password: hunter2
```

Username	Enc. Pwd
alice	29d930e
bob	3802bc7
charlie	e5281a3
doug	892eb4d
…	…

Key = 0x23401947ba

What if we encrypt the passwords?

Enc Pwd = E(password, key)

`Login: bob`
`Password: hunter2`

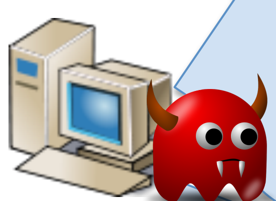
Username	Enc. Pwd
alice	29d930e
bob	3802bc7
charlie	e5281a3
doug	892eb4d
…	…

Key = 0x23401947ba

Bob can log in if
D(3802bc7, key) == “hunter2”
or
E(“hunter2”, key) == 3802bc7

What if we encrypt the passwords?

Problem:
If the adversary can get the encrypted passwords, what prevents him from getting the key?



Username	Enc. Pwd
alice	29d930e
bob	3802bc7
charlie	e5281a3
doug	892eb4d
…	…

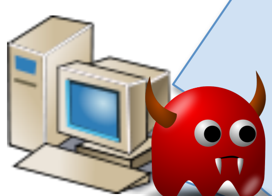
Key = 0x23401947ba

What if we encrypt the passwords?

Given encrypted passwords and the key,
adversary can decrypt EVERYONE's passwords!

Password = D(Enc Pwd, key)

Username	Enc. Pwd	Password
alice	29d930e	kittens3
bob	3802bc7	hunter2
charlie	e5281a3	password1
doug	892eb4d	qwerty!
…	…	…



Key = 0x23401947ba

What if we hash the passwords?

Login: bob

Password: hunter2

Username	PW Hash
alice	a92e08da
bob	bcab21e0
charlie	8703ba0c
doug	53ffd87d
…	…

Bob can log in if
H(password) == hash

That is, if
H(“hunter2”) == bcab21e0

Hashed Passwords

Username	PW Hash
alice	a92e08da
bob	bcab21e0
charlie	8703ba0c
doug	53ffd87d
…	…

Now what happens if an attacker gets the password database?

Exercise: Naïve Password Hashing

Hash Function Pseudocode

```
sum = 0
for c in password:
  sum += ord(c)
hash = sum % 397
```

Password Database

```
root    123
bob     315
joe     202
jane    391
alice   165
```

Attacking Hashed Passwords

- Attacker Strategy 1
  - Break the hash function
  - Work backwards from hash values to derive passwords
- Attacker Strategy 2
  - Brute force search (“password cracking”)
  - Hash lots of possible passwords,
    see which hash to values in the list

Cryptographic Hash Functions

- 3 key properties for a hash function H
  - Preimage resistance
    - Given a hash value h, it should be hard to find x s.t. H(x) == h
  - Second preimage resistance
    - Given H and x, it should be hard to find x2 such that H(x2) == H(x)
  - Collision resistance
    - Given H, it should be hard to find x and x2 such that H(x) == H(x2)

ATTACKS ON PASSWORDS

Attacking Hashed Passwords

- Attacker Strategy 1
  - Break
  - Work ... es to derive passwo

Use of cryptographic hash function makes Strategy 1 ineffective

- Attacker Strategy 2
  - Brute force search ("password cracking")
  - Hash lots of possible passwords, see which hash to values in the list

Cracking Hashed Passwords

Attacker can pre-compute hash values for likely passwords

Username	PW Hash
alice	a92e08da
bob	bcab21e0
charlie	8703ba0c
doug	53ffd87d
…	…

Password	Hash
AAAA	208da48
aaaa	21e0bb3
BBBB	ba0c96e
bbbb	587d368
…	…
hunter2	bcab21e0
…	…

Cracking Hashed Passwords

Attacker can tell when multiple users share the same password

Username	PW Hash
alice	a92e08da
bob	bcab21e0
charlie	8703ba0c
doug	53ffd87d
…	…
herb	bcab21e0

Salted, Hashed Passwords

Login: bob

Password: hunter2

Username	Hash	Salt
alice	a92e08da	7823db
bob	bcab21e0	21023e
charlie	8703ba0c	978b2a
doug	53ffd87d	10cc94
…	…	…
herb	8780bc26	638921

Bob can log in if
H(password|salt) == hash

That is, if
H(hunter2|21023e) == bcab21e0

Salted, Hashed Passwords

Username	Hash	Salt
alice	a92e08da	7823db
bob	bcab21e0	21023e
charlie	8703ba0c	978b2a
doug	53ffd87d	10cc94
…	…	…
herb	8780bc26	638921

Use of the salt forces the attacker to brute-force search each user's password individually

Salted, Hashed Passwords in Python

```
> import hashlib
> m = hashlib.md5()  # construct new MD5 object
> m.update(“hunter2”)
> m.update(“21023e”)
> m.hexdigest()
f369801178c9dbee237d0912655ce2f7
>
> s = hashlib.sha1()  # construct new SHA-1 object
> s.update(“hunter2”)
> s.update(“21023e”)
> s.hexdigest()
0bcf0402c0eaf047cd7a8c5bf262c92d3512aa5c
```

Unix /etc/password

```
root:x:0:0:System Administrator:/root:/bin/bash
bob:x:1001:1001:Bob Jones:/home/bob:/bin/bash
joe:x:1002:1002:Joe Smith:/home/joe:/bin/bash
```

User ID

Primary Group ID

Password
(“x” for shadowed)

User’s home
directory

User’s shell
program

User login name

Shadow Passwords: /etc/shadow

bob:1lls9dl824js0fjsps:1432:0:9999

User login name

Encrypted password

Date of last
password change
(days since 1/1/1970)

Min number of days
between password changes

Max number of days
between password changes

/etc/password must be available to many programs on the system.

/etc/shadow is readable only by programs running as root,
to protect the hashed passwords

Manber's Password Hardening Scheme

- **A Simple Scheme to Make Passwords Based on One-Way Functions Much Harder to Crack**
- By Udi Manber, University of Arizona, 1994.

- Uses two salts
  - One public
  - One private

Manber Password Hardening

- Storing hashed passwords
  - hash = H(password|public|private)
- Checking passwords:
  - Exhaustively check for all possible values p
    - Does hash == H(password|public|p) ?
  - If so, login is successful
  - If no value p matches, login fails

Manber Performance Penalty

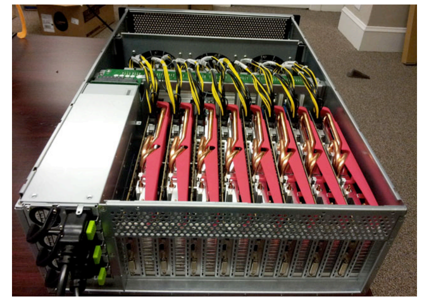
- Password creation:
  - Negligible penalty

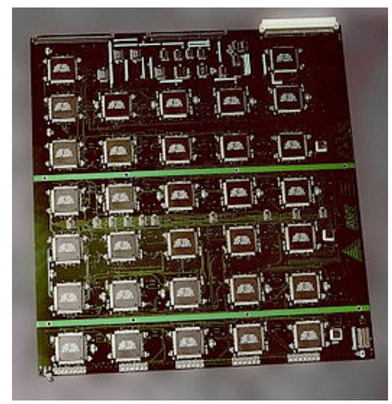
- Password checking:
  - Slowdown of N, where p can take one of N values
  - A few ms (not really noticeable)

- Password cracking:
  - Slowdown of N, where p can take one of N values
  - Hundreds of hours (== pain for the adversary)

Advanced Password Cracking

- GPU’s
  - http://arstechnica.com/security/2012/12/25-gpu-cluster-cracks-every-standard-windows-password-in-6-hours/
  - http://passwords12.at.ifi.uio.no/Jeremi_Gosney_Password_Cracking_HPC_Passwords12.pdf





- Custom hardware
  - EFF DES Cracker
  - https://w2.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/HTML/19980716_eff_des_faq.html
- Cloud computing
  - https://www.cloudcracker.com/
  - http://www.forbes.com/sites/andygreenberg/2012/02/14/moxie-marlinspikes-cloudcracker-aims-for-speedier-cheaper-password-cracking/

CloudCracker

Better Password Hashing

- PBKDF2
  - http://en.wikipedia.org/wiki/PBKDF2
  - RSA Labs' PKCS #5 version 2.0
  - RFC 2898
- bcrypt
  - N. Provos and D. Mazieres. *A Future-Adaptable Password Scheme.* In Proc. USENIX Annual Technical Conference, 1999.
- scrypt
  - C. Percival, *Stronger Key Derivation via Sequential Memory-Hard Functions*. Presented at BSDCan'09, May 2009.

PBKDF2

- http://en.wikipedia.org/wiki/PBKDF2
- Idea:
  - Use a keyed hash, or MAC
  - Hash the password and salt C times
  - Output the XOR of all Ci’s
  - Use large salts (e.g. 64 bits)
  - Use many repetitions (e.g. 4096 times)

bcrypt

- N. Provos and D. Mazieres. *A Future-Adaptable Password Scheme.* In Proc. USENIX Annual Technical Conference, 1999.
  - https://www.usenix.org/legacy/publications/library/proceedings/usenix99/provos.html
  - http://en.wikipedia.org/wiki/Bcrypt

bcrypt

- Minimizes advantage of hardware crackers
  - Uses a modified version of the Blowfish cipher’s slow key setup function
  - Includes a tunable “work factor”
  - Uses instructions that are really fast on general-purpose CPU’s
  - Make life difficult for hardware ASICS
    - Use a big table in memory
    - Modify the table a lot

scrypt

- Design goals similar to bcrypt
  - Be fast in software
  - Be not much faster in hardware
  - Aims at making life difficult for modern FPGA’s

- C. Percival, *Stronger Key Derivation via Sequential Memory-Hard Functions*, presented at BSDCan'09, May 2009. http://www.tarsnap.com/scrypt.html