Authentication and Passwords

CMSC 426/626 - Computer Security
Fall 2014

Outline

- Types of authentication
- Vulnerabilities of password authentication
- Linux password authentication
- Windows Password authentication
- Cracking techniques

Goals of Authentication

- *Identification* - provide a claimed identity to the system.
- *Verification* - establish validity of the provided identity.

We're *not* talking about message authentication, e.g. the use of digital signatures.

Means of Authentication

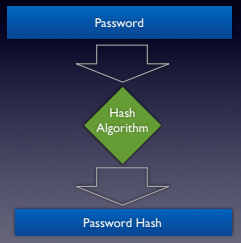
- Something you know, e.g. password
- Something you have, e.g. USB dongle or Common Access Card (CAC)
- Something you are, e.g. fingerprint
- Something you do, e.g. hand writing

Password-Based Authentication

- User provides identity and password; system verifies that the password is correct for the given identity.
- Identity determines access and privileges.
- Identity can be used for Discretionary Access Control, e.g. to give another user access to a file.

Password Hashing

- System stores hash of the user password, not the plain text password.
- Commonly used technique, e.g. UNIX password hashing.



Password Vulnerabilities

Assume the authentication system stores hashed passwords. There are eight attack strategies.

- *Off-line Dictionary Attack* - get hold of the password file, test a collection (dictionary) of possible passwords.
  - Most systems protect the password file, but attackers sometimes get hold of one.

- *Specific Account Attack* - given a specific user account, try popular passwords.
  - Most systems use lockout mechanisms to make these attacks difficult.
- *Popular Password Attack* - given a popular password, try it on multiple accounts.
  - Harder to defend against - have to look for patterns in failed access attempts.

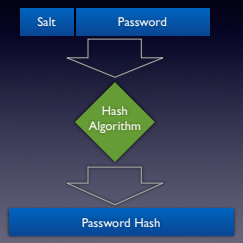
- *Targeted Password Guessing* - use what you know about a user to intelligently guess their password.
  - Counter with password policies and enforcement.
- *Workstation Hijacking* - walk up to an unattended, unlocked workstation and do your business!
  - Umm...don't leave your workstation unattended and unlocked.

- *Exploiting User Mistakes* - Does the user write their password (or clues) on a sticky note? Do they share passwords with other users? Do they forget to change default passwords?
- *Exploiting Multiple Password Use* - crack the password from one account, and you're in to multiple accounts.
  - Technology can encourage stronger password usage by easing the management of multiple, strong passwords.

- *Electronic Monitoring* - watch for passwords or password hashes being sent over the network for remote authentication.
  - Modern protocols avoid this; use vetted protocols.

Salted Hashing

- *Salt* - random quantity prepended to password before hashing.
- Commonly used technique, e.g. UNIX password hashing.



Salt Advantages

- Prevents duplicate passwords from being visible in password file. Different salts ensure different hashes.
- Increases difficulty of off-line dictionary attacks. For *b*-bit salt, dictionary is 2^b times larger.
- More difficult to identify multiple password use. Same password on two systems will produce different hashes.

UNIX Password Authentication

UNIX *crypt*

- 8-character password represented in 7-bit ASCII to produce 56-bit key.
- 12-bit salt introduced via modified DES.
- Modified DES used with key to encrypt 64-bit block of zeroes; iterated 25 times.
- Final output block is the hash.
- Modification to DES makes it one-way.

Problems and Solutions

- Even iterating 25 times, UNIX hash function is no longer sufficient. Dictionary attacks are very effective.
- For password hashing **slow is good**. Add more iterations.
- Use stronger hashes.
- Several variants for UNIX-like OSes.

Most systems are configurable. Choose your hash.

- *Linux, Solaris, FreeBSD* - change hash function to MD5 and use 48-bit salt. Far slower than UNIX *crypt*.
- *OpenBSD* - hash based on Blowfish symmetric algorithm and use128-bit salt. Can vary the cost to compute a hash.
- *Red Hat, Ubuntu, Mac OS* - SHA-2 based scheme.

Protecting Hashes

- On original UNIX systems, the password hashes were stored in `/etc/password`
- `/etc/password` had to be world-readable because it included information such as the user's login shell
- Later systems (incl. Linux) move the hashes to `/etc/shadow`, only readable by root.

/etc/shadow

User name: `sally`

Algorithm ID: `6` (SHA-512)

Salt: `54y91l6O`

Password hash: `9DRcg4cWQy2nTCfnRwNO`...

```
root:$6$012BPz.K$fbPkT6H6Db4/B8cLWbQIlcFjn0R25yqtqrSrFeWfCgybQWWnwR4ks/.rjqyM7Xwh/
seed:$6$OqXAiWQA$AIjctTUkHMECipE8EiAAJh76YZgrvadHKmWs3hQ3BU8vCClbSVv4NhGWw2FsZ0lLiZw0SL6Gc/p8Plw7ShkZR0:15933:0:99999:7:::
chris:$6$TEofkz5q$akVHlZ8A64q2gtnh8RLYm79JPLQl2kvQTKTaegr3lHBmJUipHG6FC3RFOUultItkA5i9zRTfxziyR0cz3AbZy1:16346:0:99999:7:::
sally:$6$54y91l6O$9DRcg4cWQy2nTCfnRwNOfJ8tgSkjhqHBVHYBbOOFE9wPxqUCNclRzjlCZn7Q7uE7Enl.yB/jsEaZCOYj1tbmb0:16346:0:99999:7:::
```

Salt and hash are encoded using the 64-character alphabet `[a-zA-Z0-9./]`.

Windows Password Authentication

Windows LM Hash

- LAN Manager Hash (LM Hash)
  - Pad password to 14 character
  - Convert to upper case
  - Construct two 7-byte DES keys
  - Encrypt fixed string with both keys
- *No Salt - Seriously Weak!*
- Prior to Windows NT

Windows NT Hash

- MD4 hash of encoded password
  - Encoding is UTF-16 little-endian
- Allows for longer passwords
- Doesn't convert to upper case
- *Still no salting!*

NTLMv1 Protocol

- NT Lan Manager Authentication Protocol
  - Server sends eight byte challenge (*C*). Server may just be the OS.
  - Client computes 48 byte response *R* from *C*. Client may just be a login service.
  - Response *R* involves LM and NT hashes of the user password and DES encryptions of the challenge.
  - Server computes expected response; compares to R.
- Still uses LM Hash; DES not secure enough.

NTLMv2 Protocol

- Improvement over NTLMv1
- Eight byte challenge; 32 byte response
- LM Hash *finally* goes away
- No more DES - uses HMAC-MD5
  - *Hash-based Message Authentication Code* built on MD5, keyed with NT Hash

NTLM Weak Nonce

- What happens if the eight-byte challenge (*nonce*) is not “random enough?”
- Repeated nonces allow for replay attacks
  - Attacker records challenges and responses
  - Attempts to authenticate hoping for a repeated challenge; replays response
- Hernan Ochoa & Agustin Azubel, February 2010
  - Most (all?) Win versions affected over 17 years!

Kerberos

- Replaces NTLM…but NTLM still used in some cases, e.g. stand-alone systems
- Kerberos is based on an Authentication Server and separate Ticket Granting Server — we’ll learn more about it later in the semester.

Cracking Techniques

Cracking Approaches

- *Dictionary Attacks* - given a list of possible passwords, hash each one and compare it to the target hash.
- *Rainbow Tables* are a clever way to speed-up certain password recovery attacks.

Time-Memory Trade-Off

- A class of attacks invented by Martin Hellman in 1980.
- Large up-front work requiring substantial storage in exchange for faster password recovery.
- Need a *Return Function* R(*h*), a mapping from the space of hashes back to the space of possible passwords.

TMTO One-Time Work

- Do this **many** times with different starting passwords.
- Keep list of (*start*, *finish*) pairs - end up with a big table.

TMTO Attack

- Given a target hash h_0, compute h_1 = H(R(h_0)) and iterate, i.e. compute h_2, h_3, etc.
    - h_{i+1} = H(R(h_i))
- If h_n matches one of the finish points in our table, compute H(start), H(R(H(start))), etc.
- Probably you will find a password such that H(password) = h_0.

Rainbow Tables

- Rainbow tables are a type of TMTO.
- They solve a particular technical difficulty by using many different return functions.
- The result is a more efficient TMTO.

Note: TMTOs are not effective against salted hashes. Why??

Finished. See the website for exercises.