CMPSC 497

Password Authentication

Trent Jaeger
Systems and Internet Infrastructure Security (SIIS) Lab
Computer Science and Engineering Department
Pennsylvania State University
Security Model - E-voting

• **Who are the principals?**
  ‣ Voters, Admins, Counters, Others

• **Who are adversaries?**

• **Which commands may be threatened (attack surface)?**
  ‣ Start the application
  ‣ Process the vote
  ‣ Count the votes

• **Who must the application trust? To do what?**
  ‣ Of principals above
Principals

- **Principals** are expected system subjects
  - Computers, agents, people, enterprises, …
  - Depending on context referred to as: servers, clients, users, entities, hosts, routers, … - and some may be adversarial
  - Security is defined with respect to these subjects
    - Implication: every principal may have unique view

- A **trusted third party**
  - Trusted by all principals for some set of actions
  - Often used as introducer or arbiter
Challenges

• Distinguish adversaries from trusted principals
  ‣ Suppose the e-voting application receives a command
  ‣ How do we know whether the command may be from an adversary or a trusted principal?

• The security mechanism for identifying principals is
  ‣ **Authentication**
  ‣ The act of confirming the truth of an attribute of a single piece of data claimed true by an entity
  ‣ For an identity, authentication is the process of actually confirming a claimed identity
What is Authentication?

• **Short answer**: establishes identity
  ‣ Answers the question: To whom am I speaking?
• **Long answer**: evaluates the authenticity of identity by proving credentials
  ‣ **Credential** – is proof of identity
  ‣ **Evaluation** – process that assesses the correctness of the association between credential and claimed identity
    • for some purpose
    • under some policy (what constitutes a good cred.?)
Why authentication?

• Well, we live in a world of rights, permissions, and duties
  ‣ Authentication establishes our identity so that we can obtain the set of rights
  ‣ E.g., we establish our identity with Tiffany’s by providing a valid credit card which gives us rights to purchase goods ~ physical authentication system

• Q: How does this relate to security?
Why authentication (cont.)?

- Same in online world, just different constraints
  - Vendor/customer are not physically co-located, so we must find other ways of providing identity
    - e.g., by providing credit card *number* ~ electronic authentication system
  - Risks (for customer and vendor) are different
    - Q: How so?

- Computer security is crucially dependent on the proper design, management, and application of authentication systems.
What is Identity?

• That which gives you access … which is largely determined by context
  ‣ We all have lots of identities
  ‣ Pseudo-identities

• Really, determined by who is evaluating credential
  ‣ Driver’s License, Passport, SSN prove …
  ‣ Credit cards prove …
  ‣ Signature proves …
  ‣ Password proves …
  ‣ Voice proves …

• Exercise: Give an example of bad mapping between identity and the purpose for which it was used.
Exercise

- Classify each of the following as a violation of confidentiality, of integrity, of availability, or of some combination.

- Carol changes the amount of Angelo's check from $100 to $1000
- John copies Mary's homework
- Eve registers the domain name "psu.edu" and refuses to let Penn State buy or use that domain name.

Credentials

- ... are evidence used to prove identity
- Credentials can be
  - Something I am
  - Something I have
  - Something I know

INTERNATIONAL THEOLOGICAL UNIVERSITY

ADMIRISTRATIVE CREDENTIAL

To

YOUR NAME GOES HERE

Given the third day of January, one thousand nine hundred and ninety-six, to

Title

Administrative Credential

Signature

Dr. Mary Brown Eagle, Ph.D.

President of the University

Access Codes
1. Event Level: Decorum
2. Event Level: Media
3. Name Tag
4. Conference
Passwords

• An example of “something you know”
  ‣ Client users must remember passwords to access their data on servers

• Passwords have a checkered history
  ‣ People have often chosen poor passwords
  ‣ Why is that an issue?

• We (security community) assumed (in the 1990s) that passwords would be replaced with another technology to enable users to authenticate
Password Use

- **Naively**: Retrieve password for ID from database and check against that supplied password
  - Baravelli: ...you can't come in unless you give the password.
  - Professor Wagstaff: Well, what is the password?
  - Baravelli: Aw, no. You gotta tell me. Hey, I tell what I do. I give you three guesses. It's the name of a fish.
  - .......

  [Slams door. Professor Wagstaff knocks again. Baravelli opens peephole again.] Hey, what's-a matter, you no understand English? You can't come in here unless you say, "Swordfish." Now I'll give you one more guess.

  - Professor Wagstaff: ...swordfish, swordfish... I think I got it. Is it "swordfish"?
  - Professor Wagstaff: Pretty good, eh?

  [Marx Brothers, *Horse Feathers]*

- How should you store passwords to protect them?
  - Just storing them in a file gives anyone with access to the file your password
Password Storage

• Instead of storing passwords, we store
  ‣ A value that can be computed from the password
    • $F(password) = value$
  ‣ That is highly unlikely to be the same as the value computed from another password (collision-free)
  ‣ From which it is difficult to extract (reverse) the password (one-way)

• What kind of function provides such properties?
Hash Algorithms

• Hash algorithm
  ‣ Compression of data into a hash value
  ‣ E.g., $h(d) = \text{parity}(d)$
  ‣ Such algorithms are generally useful in algorithms (speed/space optimization)

• … as used in cryptosystems
  ‣ One-way - (computationally) hard to invert $h()$, i.e., compute $h^{-1}(y)$, where $y = h(d)$
  ‣ Collision resistant hard to find two data $x_1$ and $x_2$ such that $h(x_1) == h(x_2)$

• Q: What can you do with these constructs?
Password Storage

- Hosts store password hashes in a file
  - Originally, `/etc/passwd` for UNIX systems
  - Now `/etc/shadow`
- Server programs can also store their own users’ password hashes in a file
  - For Apache can store in `/usr/local/apache/passwd`
- What if an adversary can gain access to a password storage file?
Password Cracking

• Attacker can access the hashed password
  ‣ Can guess and test passwords offline
• Called “password cracking”
• Lots of help
  ‣ John the Ripper
• How well do these work?
Cracking Passwords

• How hard are passwords to crack?
• How many 8-character passwords are there given that 128 characters are available?
Cracking Passwords

• How hard are passwords to crack?
• How many 8-character passwords given that 128 characters are available?
  • $128^8 = 2^{56}$
• How many guesses to find one specific user’s password?
  • $2^{56}/2 = 2^{55}$
Cracking w/ Dictionaries

• How hard are passwords to crack?
• How many 8-character passwords are there given that 128 characters are available?
  • $128^8 = 2^{56}$
• Suppose we use a dictionary where there is a 25% chance that that user’s password appears in that password dictionary. How many guesses then? (Assume 1 million dictionary entries)
  • $1/4(2^{19}) + 3/4 (2^{55}) \sim 2^{54.6}$
• However, you probably simply apply the dictionary and accept a 25% chance of recovery
Cracking w/ Dictionaries

• How hard are passwords to crack?
• How many 8-character passwords are there given that 128 characters are available?
  • $128^8 = 2^{56}$
• But, in practice the attacker just needs one password from a set of users - rather than a specific user
• If there are 1024 users, the basic work effort is now
  • $2^{55}/2^{10} = 2^{45}$
• However, given a dictionary, we can simply see if one of the 1024 passwords are in the dictionary
  • About equal to size of dictionary/prob. in dictionary
Password Guessing

- Research in password cracking can improve the cracker’s ability

- **Markov Chains**
  - Guess the next character in order of the probability that it next character to follow
  - Guess the highest probability first character – ‘s’
  - Guess the highest probability character to follow ‘s’

- **Grammars**
  - Guess by most popular password structure
  - Then, fill in characters as above
Password Guessing

- How can you make your password hard to guess?
Password Guessing

• How can you make your password hard to guess?
  ‣ Limited by memorability though

• Suppose computers get faster and faster
  ‣ Ever make password storage obsolete?
Password Guessing

- How can you make your password hard to guess?
  - Limited by memorability though

- Suppose computers get faster and faster
  - Ever make password storage obsolete?
    - Can slow down by hashing many times – $h^{100}(\text{password})$
    - Discussed in CMPSC 443…
Common Passwords

- What if two users have the same password?
  - How will that appear in the password database?
  - How is the problem addressed?
“Salt”ing passwords

• Suppose you want to avoid a offline dictionary attack
  ‣ bad guy precomputing popular passwords and looking at the password file

• A salt is a random number added to the password differentiate passwords when stored in /etc/shadow

\[
salt_1, h(salt_1, pw_1) \\
salt_i, h(salt_2, pw_2) \\
salt_i, h(salt_3, pw_3) \\
\ldots \\
salt_n, h(salt_n, pw_n)
\]

• consequence: guesses each password independently
Project #1

- Store passwords for your server’s users
  - On “set” enter username and password
    - Store pair on first entry of username (unknown_user returns “1”)
  - Compute a salt from OpenSSL random number generator functions
  - Concatenate salt (16 bytes) + password (16 bytes, padded) into input to hash function (32 bytes)
  - Compute a hash using digest_message (cmpsc497-ssl.c)
  - Store in key-value store provided indexed by username
    - Key: username; Value: hash; Tag: salt
Project #1

- Verify a username-password for a later command
  - How do you do that?
Project #1

- Verify a username-password for a later command
  - How do you do that?

- Lookup salt and password hash for user

- Create hash input as before using input password

- Check that this results in expected password hash
Use OpenSSL Library

- The file `cmpsc497-ssl.c` has a set of functions to access the OpenSSL library API
- OpenSSL is a crypto library for implementing the SSL protocol
- Includes code for encryption, hashing, random number generation
- Only need to use `digest_message` from that file, but others may be useful at some point
- Will use OpenSSL to collect salt values also from its random number generator (engine)
Use OpenSSL Library

- Random number generation in OpenSSL
- Uses `engine_init` provided to initialize RNG – in `main` already
- Then, need to extract a random number
- For guidance
- How random does the salt need to be?
  - Just different for everyone ideally, but the salt is not a secret
Take Away

• Security depends on differentiating friend from foe
  ‣ These are called principals
  ‣ Mechanism to identify principals is called authentication

• Authentication mechanisms depend on validating the possession of a secret (credential)
  ‣ Something you have, know, are

• Passwords are an authentication mechanism that validates something you know
  ‣ Passwords are stored as a hash of a combination of the password and a salt value to make cracking harder