Authentication

3 parts to this talk

- How to work out who a user is
- How to remember who a user is
- Attacks on systems that fail to do this properly

Principle of least privilege

- Doesn’t matter how good security is if users can do anything
- “Do not allow users to do anything they do not explicitly need to do”
- Two methods for this:
  - Whitelist: secure but difficult to maintain
  - Blacklist: insecure but easy to maintain

Who are you?

Plaintext password

- Just save it to a file
- User is authenticated if they send you the same password

USER	PASSWORD
admin	secret
clive	passwordpassword123

Plaintext password **NO BAD**

- bad
- BAD
- NOOOOOOOOO BAD

Plaintext password **NO BAD**

- bad
- BAD
- NOOOOOOOOO BAD

BAD BAD BAD BAD BAD

Plaintext password

- Why bad?
- Timing attacks
- File system read immediately leaks all passwords
- Mess up and leak them yourself
  - Accidentally commit them to git repo
  - Bell labs emailing all passwords on a system to the entire internet
- Encrypted plain passwords are pretty much just as bad, so don’t do that either

Hashed password

- Probably most common
- Receive the password
- Hash it
- Compare against DB

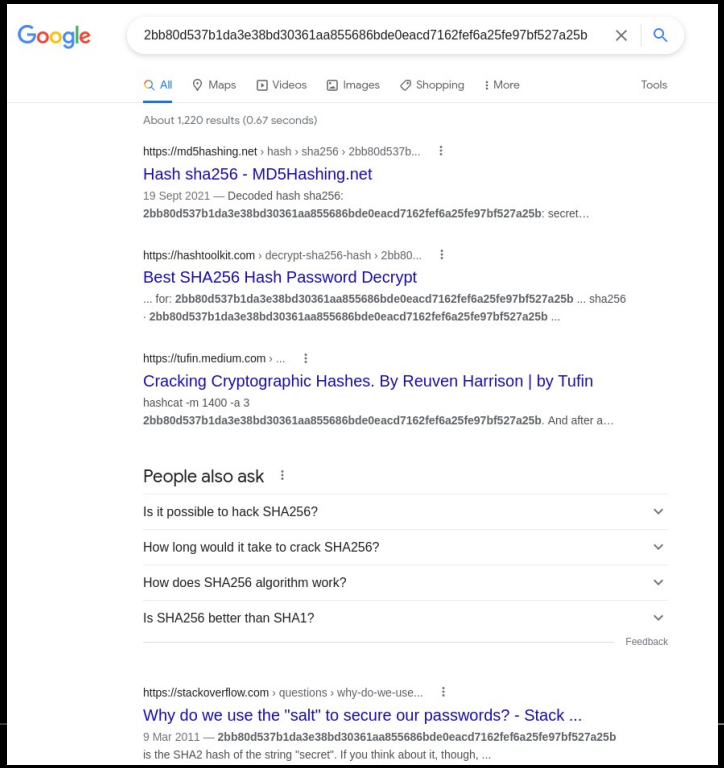
USER	PASSWORD
admin	2bb80d537b1da3e38bd30361aa855686bde0eacd7162fef6a25fe97bf527a25b
clive	17fef9e04fcdd058d06bf29988884f920db31d57364e04fbf159d27c5f924f11

Hashed password

- Most hash functions are designed to be fast
- Therefore passwords are easy to brute force
- Insecure hash functions allow you to cheat
- Rainbow tables are easy to compile
  - Big lists of known hashes for common passwords

Hashed password

- Often weak hashes can literally be googled



Manually Salted + Hashed passwords

- *Append* a random string to a password (to stop length-extension attacks)
- Store the random string and hash in the DB

USER	PASSWORD	SALT
admin	5a66dcc4d277152078adabbd016bade3b2bac6b92f2ce035b4ecf6b04a6c9d62	63862bf5f342f2d6429070c30559b657
clive	a72e9fed631aa759b81654dd189e443b47075db06dd514a3c4335aa0a4218bda	32cd31aa9c3bbad7bb47b716b2836231

Manually Salted + Hashed passwords

- This is the first acceptable method
- Prone to mistakes
- Still pretty easy to brute force, but needs effort

Secure password hash

- Use a prebuilt system such as scrypt, argon2, bcrypt (if you have to), PBKDF2 (as a last resort)
- It is designed to be slow, so harder to brute force
- It has salt processing built in, so easier to use
- Often comes built-in, so easier to use

Salted + peppered password hash

- For when you *really* care (which should be always)
- Mix the passwords with a static key kept outside DB
  - Can be prepended/appended to salt
  - Can be encrypted
  - Doesn’t really matter
- Requires attacker to have more than DB access
- AKA “secret salt”, as pepper can mean other things

Password storage conclusion

- Setup
  - Create static pepper, and store it somewhere else (like a simple text file)
- When creating password:
  - Salt password
  - Pepper password
  - Hash password
  - Store hash + salt, throw away original password
  - ***Don’t store pepper in DB!!!***

Password storage conclusion

- To check password:
  - Read hash and salt from database
  - Load pepper from other location
  - Run the same method used to create the hash on the provided password (salt, pepper, hash)
  - Compare the results
- That’s as good as you can *reasonably* get with just passwords

Extra authentication

- Sometimes you want more security than just a password
  - High traffic
  - Make people feel safe
  - Just really private data
- In that case, you have two main extra options

Digital signatures

- Users are given magic numbers
- They use maths to make these magic numbers to sign things
- More explanation to come in crypto talk
- Theoretically a lot more secure
- *But* users have to save these somewhere, which just moves the problem to somewhere else

2FA

- Opt-in for pretty much everything now
- Requires an app/text message/secure hardware widget thing
- Significantly harder to cheese
- Can be bypassed by:
  - Social engineering to steal phone number (common)
  - Stealing the authentication device
  - Replay attacks

OAUTH

- Get someone else to manage passwords for you
- Can be difficult to properly set up
- Puts complete dependence on another service (usually google)
- Frequently subverted to get personal information from users

Who are you again?

Session cookies

- This is the best way of doing it
- When a user successfully logs in
  - Generate a *secure* random value (at least 16 bytes of cryptographically random bytes), and set it as a cookie
  - Store the token and the username somewhere (a python dict, redis key/value, etc.)
  - MAKE SURE TO SET HTTPONLY!!!
- To check, simply read the cookie, and set them as whatever user the map has in it
- Very hard to attack if done properly
- Often built-in to the language (PHP's $_SESSION)

JWTs (and equivalents)

- When a user logs in, securely sign the username, and set it as a cookie
- To check, first check the signature, then load the value
- Ideally is just as good
  - Even slight mistakes in verification/signing makes massive difference
  - Leak of signing key is *bad*
  - If not encrypted, can leak internal server information to user
  - Personally don’t like it

Rigorous checking

- Checking the user should be able to access what they’re asking for
- Have a system in place, some ideas:
  - Check to see if the URL path starts with /admin/…, and if it does, stop anyone but admins accessing it, so anything there is protected by default
  - Always start with an authentication check
  - Require any endpoint to explicitly state it’s authentication requirements when registered
- Whenever you update something, make sure that no new powers have been given to users who shouldn’t have them
- Repeatedly audit (pentest/redteam/whatever) your endpoints
- EXERCISE THE PRINCIPLE OF LEAST PRIVILEGE

Attacks

Cracking hashes

- Hashcat:
  - Uses GPU and is faster, but might not work if you don’t have a gpu
  - hashcat -m <alg> <hash> <wordlist>
  - Get <alg> by hashcat --help, <hash> can be a file or the hash itself
  - For the wordlist, generally /usr/share/wordlists/rockyou.txt is good
    - (run sudo gunzip /usr/share/wordlists/rockyou.txt.gz on kali to generate it)
  - hashcat 0 5d41402abc4b2a76b9719d911017c592
  - Can specify wordlist with --wordlist

Cracking hashes

- John
  - Supports more formats and is portable, but slower
  - john --show --format=<alg> <file of hashes>
  - Get <alg> by googling, but sometimes john can just guess the format
  - echo 5d41402abc4b2a76b9719d911017c592 > /tmp/hash
  - john --show --format=raw-md5 /tmp/hash
  - Can also add a wordlist with –wordlist=<file> if built-in list doesn’t find it

Replay attacks

- Very, VERY common
- Attacker spoofs trusted site
- User types in password/2fa token/signature
- Attacker uses this to authenticate as user

Replay attacks

- Probably the most common attack in the wild
- Can be fixed at a protocol level
  - SSH, TLS, etc
- Shared private data (such as email confirmation) can be used
- Educating users to *check the actual URL*

Broken access control

- Forgetting to check, or not properly checking, who someone is
- Very, *very* common in the real-world
- Often very simple
- Common on websites with internal REST APIs

Broken access control

- Common format:
  - Page is properly authenticated, but javascript somewhere on page asks API for data
  - API doesn’t check permissions, and trusts whatever is handed to it
  - Attacker just submits the request without accessing the page
- For instance (real-world but fixed):
  - Timetable page is authenticated, but fetches timetable from unauthenticated endpoint, along the lines of “/api/timetable?user=harlan”
  - Attacker can just use “/api/timetable?user=tom” to get different timetable
- Often just leaks information, but can somethimes be used to modify the user’s credentials:
  - Password reset form has a hidden text-box set by the server containing the username of the current user
  - Setting this to a different username allows the attacker to change another user’s password.

Spraying

- Get a list of users (or just guess)
- Try common passwords
- Chances are, one will work
- Now you’ve logged in

Your turn

- Sharksellers: ctf.cybersoc.cf