Security & Privacy

About Passwords and Hashes

CS7NS5/CSU440032
Stephen Farrell
stephen.farrell@cs.tcd.ie

~~Scaleable Computing~~

~~About Passwords and Hashes~~

~~CS7NS1/CS4400~~

~~Stephen Farrell~~
~~stephen.farrell@cs.tcd.ie~~

~~https://github.com/sftcd/cs7ns1/~~
~~Note: PRs for repo are welcome!~~

Contents

- Background on passwords and their uses
- Password handling and a bit on password cracking

Passwords are horrible (1)

- Weak (guessable) passwords are inevitable – implicit in allowing “human memorable” values
  - 123456 almost always the most-used
- Re-use of (related) passwords is inevitable
  - Das, Anupam, et al. "The Tangled Web of Password Reuse." NDSS. Vol. 14. 2014. https://www.cs.cmu.edu/~anupamd/paper/NDSS2014.pdf
- Attacker often does not need all passwords, just a few good ones
  - e.g. belonging to an admin

Passwords are horrible (2)

- Password entry user interfaces lend themselves to phishing
  - Distinguishing application “chrome” from content is v. Hard
- Forced “hard” password policies or password rotation force users to game the system
- Accessibility? What if you can’t use a keyboard?

But... passwords are also great!

- Can be human memorable
  - Try that with your SSH private key!
- No need for h/w or special s/w
- People can change passwords (even if they don’t)
- Can link between diverse systems
  - E.g. IPsec VPN access governed by ActiveDirectory login
- Fallback is required in any system – and you can fallback to closing a loop for a password reset
  - Main fail of a scheme I helped with called HOBA – RFC 7468
    - https://datatracker.ietf.org/doc/html/rfc7486
  - Closest successor to that is probably WebAuthn – another lesson is “things browser makers like tend to happen”
    - https://webauthn.guide/

But... passwords leak (1)

- Attackers very rarely, if ever, try to grab passwords as they go by in a network packet (but they might, so don’t allow that and be careful of man-on-the-side attacks)
  - In fact, other meta-data snarfing is much more dangerous and the reason to encrypt as much as possible
- Attack password entry (phishing, keylogging)
  - Can be done at scale, if you can get malware to host or re-direct to phising site
- Masquerade as server, e.g. via borked DNS in coffee shop
- Grab a copy of the password verifier database
  - And then dictionary attack that – 1% success rate can be enough!
  - Or, just sell the DB

But... passwords leak (2)

- Trickle brute-force attacks on online services
  - Esp. sshd but anything really – that happens **all** the time for all services exposed to the Internet
- Access/purchase a DB of leaked passwords and try those elsewhere or on the original leaky site
- Bugginess: user enters password in username field; system logs failed login by user “123456”; “oops!” says user, then successfully logs in as “joeblow”; system logs get centralised; logs contain cleartext passwords easily correlated with usernames
- How else might passwords leak?

2008 count of my paswords

Category	Count	Known	Examples
Logins	10	6	Laptops, host systems (incl. root accounts)
Devices	5	0	DSL router, home print/file servers, sensor nodes
Network access	4	0	Work n/w, WLAN, ISP, etc
Protocol	14	1	Outbound HTTP proxy, IMAP, Jabber, skype, etc.
Service	21	4	Mainly web sites with password stored outside browser
PINs	7	4	Bank cards, door access codes...
Total	61	15	

Your passwords?

- How many? What kinds? How chosen? How protected? How long-lived?
- <your input here>

What’s a person to do? (1)

- Use password managers
  - Pros/cons?
- Avoid new accounts
  - Can you? Do you? Why? Why not?
- Avoid passwords where possible
  - SSH, USB tokens, even HOBA:-)
  - Key management?
- 2FA if you can and are willing/able
  - Getting **much** more common these days, esp for critical services

What’s a person to do? (2)

- Try hard to never use a terribly weak password
  - You never know if the system for which you create/update a password will end up becomming important for you
- Do not kick the bucket!
  - Post-mortem credential handling is kinda hard (but a real issue)

What’s a sysadmin to do? (1)

- Try not use passwords, esp. for admin purposes
  - Does a user **really** need to create an a/c? Is the minimal marketing benefit worth the risk of adding possibly toxic data to your DB?
- Try hard to insulate users from client-side attacks: CORS, XSS, etc.
- Monitor and protect your systems
  - fail2ban, denyhosts, other intrusion detection systems (IDSes)
- Try (but fail) to deploy universal single-sign-on (SSO)

What’s a sysadmin to do? (2)

- Avoid holding the password verifier DB, e.g. outsource to mega-scaler or service provider
  - But – centralisation of the web is a real problem, as is control
  - And even mega-scalers have outages (can you afford to be offline in that case?)
- Ensure best-practices for password verifier DB
  - More on that in a minute
- Maybe: use password authenticated key exchange (PAKE) schemes where that makes sense
  - Caveat: I’m a skeptic for many claimed uses of PAKEs – the problems with passwords are **not** really cryptographic ones

What’s a sysadmin to do? (3)

- Force two-factor authentication (2FA)... which does help
- Need caution in ensuring authentications are really out-of-band of one another
  - SMS messages – SS7 protocol attacks: Holtmanns, Silke, and Ian Oliver. "SMS and one-time-password interception in LTE networks." Communications (ICC), 2017 IEEE International Conference on. IEEE, 2017. https://ieeexplore.ieee.org/document/7997246/
- Google USB token claiming success against phishing
  - https://krebsonsecurity.com/2018/07/google-security-keys-neutralized-employee-phishing/

What’s a sysadmin not to do?

- Do not follow “traditional” password policies
  - Read this instead: Florêncio, Dinei, Cormac Herley, and Paul C. Van Oorschot. "An Administrator's Guide to Internet Password Research." LISA. Vol. 14. 2014.
    https://www.usenix.org/system/files/conference/lisa14/lisa14-paper-florencio.pdf
  - But main result is there’s a huge gap between online guessable and offline dictionary attackable – do require systems/user-pwds to not be online guessable but no more, and do make sure offline dictionary attacks (and hence passwords) are hard-enough but don’t care much more than that
- Do not enforce password “quality” requirements
  - Do encourage (not require) higher-entropy passwords e.g. via meters: Egelman, Serge, et al. "Does my password go up to eleven?: the impact of password meters on password selection." Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, 2013.
    https://www.guanotronic.com/~serge/papers/chi13b.pdf
- NIST recanted on decades-old guidance a while back
  - https://www.passwordping.com/surprising-new-password-guidelines-nist/

Password verifiers

- User enters username (u) and password (p)
- System transmits u & p to verification system
  - Rarely sends a processed password – could do, but system problems often mean p is in clear at the application layer even if encrypted via TLS at the transport layer – what system problems might cause that?
- Verifying system uses u to get password verifier (pv) value from database (call that PVDB)
- System checks if f(p,pv)==’ok’ for some function ‘f’
  - You need to be able to replace ‘f’ without changing all passwords so really PVDB contains some form: of: u,f,pv

Salts

- PVDB has many u,f,pv entries
- What if two users have the same password?
- Don’t want u1,f,pv1 and u2,f,pv1 in the same PVDB as that’d help an attacker quite a bit
  - New attack enabled: if I can read *“/etc/passwd/”* then I could keep changing my password until my pv is the same as someone else’s
- So we use a salt – a randomly chosen string that’s set whenever the password changes
  - Now we’re storing some form of u,f,s,pv
- Longer salts make dictionary attacks harder
- One possibility is to use a cryptographic hash function to generate pv

PVDB Protection

- Is it useful to encrypt the entire PVDB?
- What else might we do to protect the PVDB?
  - Hint: consider the risk as being something like:

    probability(PVDB leaks) x cost of the leak
- There are some PAKEs that allow one to distribute the PVDB in cryptographically clever ways, but AFAIK those aren’t in real-world use
  - There was IPR on that IIRC

Aside: Cryptographic Hashes (1)

- Cryptographic hashes are one-way functions that take arbitrary input and produce a fixed-size output
  - Reversing hash should be cryptographically “hard” - say 2^128 units of work for some sensible unit
- Many uses in cryptography, usually we want hash functions to be highly efficient
  - Signature on LARGE file...
- Hash functions should be:
  - Collision resistant: finding x,y s.t. H(x)==H(y) is hard
  - Pre-image resistant: given x, s.t. x==H(y) finding y is hard
  - 2nd-pre-image resistant: given x, finding y s.t. H(x)==H(y) is hard

Aside: Cryptographic Hashes (2)

- Finding collisions is **much** easier than pre-images due to birthday paradox
- If hash output length is n, then 2^(n/2) work to find a collision, and 2^n to find a pre-image for a perfect hash function
  - It can be important to understand that difference
  - 2^128 is a **lot** less than 2^256!
- Hash functions: MD5, SHA-1, **SHA-256**, SHA-512, SHA-3 family, ...

Properties of ‘f’?

- Do we want ‘f’ to be speedy?
- Do we want ‘f’ to use as little memory as possible?
- Do we want ‘f’ to be easy to parallelise?

- ‘f’ is often called a password hashing algorithm but is really a verifier, the password hashing alg produces pv given p and s (and sometimes u)
  - It’s ok to not be terminologically pure:-)
- See https://password-hashing.net/ for much more on a competition related to that kind of function

MD5

- MD5 is an old OBSOLETE hash function, probably older than most in this room:-)
  - See RFC 1321
  - MD5 has 128 bit output, is very fast and has been **broken** for collisions (with ~2^18 work)
- But let’s say we base f on a simple use of the MD5 hash function...
  - That’s really dumb! But is done all the time.
- In that case ‘f’ is something like:
  - (pv==md5(salt||p)?”ok”:”not-ok”)
    - Where || is catenation
- Unsalted versions are also used (OMG!)
  - (pv=md5(p)?”ok”:”not-ok”)
- Really easy to reverse via dictionary attack

Dictionary Attack (1)

- Say you have md5(x), what's x?
  - While MD5 is broken for collisions, pre-images are still much harder so we won't try reverse the hash directly (yet!)
- But we can guess x, esp if x is human memorable
  - Human memorable => 40 bits or less of entropy
    - => 2^40 search space => easy
- Algorithm:
  - Define the search space; guess=first_guess();
  - while f(guess,s,pv)!=”ok”) guess=next_guess();
- Note: next-guess() might do more than just take the next word from a list, e.g. if current guess==”password” next_guess() might return “password01” or “passw0rd”

Dictionary Attack (2)

- Can be easier still if:
  - We know about the possible/likely alphabet
  - We have a set of substrings that may be (part of) the password
  - We know something about the length of the password
- Dictionary attacks: The set of algorithms that base guesses on a dictionary of words, guessing the next variant and keep going ‘till we’ve run out of search space, computational resources or we’ve won the game.
- If the “words” used are just random strings from an alphabet less than some length then we’d call that a **brute force** attack
- If ‘f’ uses a salt that we don’t know, then we need to explore the space of ‘s’ via brute force, but the space of ‘p’ could still use a dictionary
  - So the salt slows down the dictionary attack, in proportion to the length of the salt

Dictionary Attack (3)

- There are **excellent** tools for running clever and speedy (e.g. using GPUs) dictionary attacks on leaked PVDBs
  - John the ripper
    - https://www.openwall.com/john/
  - Hashcat
    - https://hashcat.net/hashcat/

Rainbow tables (1)

- Another form of ‘f’ is to use ‘p’ as an encryption key so that pv=e(p,”fixed-string”)
  - Or pv=e(p||s,”fixed-string”) or pv=e(p,s||”fixed string”) or similar
- If pv=e(p,”fixed-string”) then we can use a dictionary attack as always, but we could also leverage a time-memory trade-off using rainbow tables.
  - Oechslin, Philippe. "Making a faster cryptanalytic time-memory trade-off." Annual International Cryptology Conference. Springer, Berlin, Heidelberg, 2003. https://infoscience.epfl.ch/record/99512/files/Oech03.pdf

Rainbow Tables (2)

- Size can be dozens to hundreds of GB
- Rainbow tables are sets of chains of passwords where we only store the first and last elements of each chain
  - Next element in chain derived via password hash function and a ‘reduction’ function
- Searching: iterate reduction/hashing of ‘pv’ until you find a value in some chain, at that point ‘p’ is (likely) somewhere in the chain, and you can begin from the chain start until you find pv somewhere in the chain.

So MD5 is crap, how’s SHA-512?

- Standard linux hashes of the “sha-512” (or “$6”) variety:
- See ``man 3 crypt`` on your local linux
- Uses SHA-512 hash, iterated 5000 times
  - ‘s’ and ‘p’ are chars from [a-zA-Z0-9/]
  - ‘s’ is 16 chars, default random per ‘p’
- Output from ``mkpasswd -m sha-512 foo`` has format:

  6<salt>$<hash> and could be :

  6m45hbNT1w/f$gSX6x7nnEwkYTWskeNmz.j7XALhcOVcLL/
  c5oxMfrZy4bbYZsKa2la2yQGPfs0zgSQnPjCtW3mDkPgVqTFqHh.
- Details are gnarly and may be described by:
  - https://www.akkadia.org/drepper/SHA-crypt.txt
  - Seems like a less clean PBKDF2 (RFC8018, https://datatracker.ietf.org/doc/rfc8018/)

So we had crap, then messy, ...

- Unsalted MD5 is terrible because it’s so quick
- sha-512 flavour crypt is slow but attacks can benefit from parallelism, e.g. multiple GPUs
- Argon2 won the password hashing competition and is explicitly designed to be a better ‘f’ that’s memory intensive and time-memory trade-off resistant
  - Argon2: RFC9106 https://datatracker.ietf.org/doc/html/rfc9106
- All very nice, but even after PHC win some issues discovered, “fixed” by Argon2d (Argon2i won the PHC), and deployment is (so far) limited (but I’ve not checked, and stats for PV deployments aren’t easy to gather)

A good password recovery paper...

- Hranický, Radek, Martin Holkovič, and Petr Matoušek. "On Efficiency of Distributed Password Recovery." Journal of Digital Forensics, Security and Law 11.2 (2016): 5.

  https://commons.erau.edu/cgi/viewcontent.cgi?article=1380&context=jdfsl

- Describes challenges/opportunities in distributed computation for hash cracking with a nice description of implementation and measurement issues
  - Should be a fine model for part of an end-of-module report
  - And should provide plenty of hints as to HOWTO approach practical assignments
  - But, don’t take this as me saying “do what they did” your problems and the assets you can bring to bear to solve problems will (likely:-) differ

Conclusion

- Passwords are unavoidable and crap
- They can be handled more or less well at a systems level and cryptographically
- Dictionary attacks (of various forms) will likely have some percentage success
- There are scaling and distributed computing issues in password cracking