TEU00311

What is the Internet doing to me? (witidtm)

Stephen Farrell
stephen.farrell@cs.tcd.ie

https://github.com/sftcd/witidtm
https://down.dsg.cs.tcd.ie/witidtm

(Mostly) About Passwords

- Background on passwords and their uses
- Password handling and a bit on password cracking
- Exercise for you today: think about how you handle your passwords

More detail exists...

- In 2018/2019 I taught a 4th year/MSc module on scaleable computing where the students spent a lot of time cracking password hashes
- These slides evolved from a lecture from that module, which is similar but more technical:
  - https://github.com/sftcd/cs7ns1/blob/master/lectures/about-passwords.pdf
- Hopefully I've adjusted the content correctly for you, but do let me know
  - I've left in some bits that we'll just skim over to give a flavour of what's possible (for an attacker) and in case your future self has to care about that kind of thing
  - I've added some content on password managers

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 or manager

Passwords are horrible (2)

- Password entry user interfaces lend themselves to phishing
  - Distinguishing browser “chrome” from web content is very hard for a user
- Mandatory “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! (SSH == secure shell, a command line thing)
- No need for h/w tokens or special s/w tools
- People can change passwords (even if they don’t)
- Passwords can link authentication 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://tools.ietf.org/html/rfc7486

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)
  - Careful of man-on-the-side attacks!
- Attack password entry (phishing, keylogging)
  - Can be done at scale, if you can get malware to host
- Masquerade as server, e.g. via borked DNS in coffee shop
  - Recall the HTTP->HTTPS demo in the lab
- Grab a copy of the password verifier database
  - And then dictionary attack that – 1% success rate can be enough!

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
  - If joe.bloggs@accounts.example.com has password “123456” then it’s worth trying variants of that password with a possible account like joe.bloggs@example.net
- 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?

Firefox (nightly) sez...

- It has 273 “logins” stored for me (was 270 a yeay ago, 265, 259 the years before that)
- Many are duplicates, e.g. same login for different cs.tcd.ie web pages, where URLs change from time to time
- Many are outdated, e.g. my home-router before last
- Many are for conference technical programme committees that change from year to year
- Almost all have auto-generated crap for passwords (so not human memorable), some also for usernames (if the password isn’t memorable, maybe the username also doesn’t need be)
- None are (IMO) highly sensitive: Not banking, not github, etc
- Of my other browsers:
  - Opera has home network logins and one online service that uses certificate-based TLS client authentication
  - Firefox, Chromium, Brave, Vivaldi have zero stored logins (I hope!)

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?
- Two-factor authentiction (2FA) if you can and are willing/able

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 being important for you
- Firefox (nightly) and maybe other browsers now warn when you access a site that (they reckon) has had a password DB leak or other similar incident
  - Collaboration with https://haveibeenpwned.com/
- Do not kick the bucket!
  - Post-mortem credential handling is kinda hard (but a real issue)

Server side considerations

- It’s maybe useful to think a bit about the sysadmin view, or server side, here because:
  - You’ll hear some of these terms, e.g. if there’s a data-leak from some service you use
  - You may end up being a sysadmin or the boss of some sysadmins some day
    - For your home network, you almost certainly kind-of will be a sysadmin
  - It might help you choose between services when you have a choice
  - It might help you whine at a sysadmin if they’re being dumb:-)
- Don’t worry if the detail here is too much...

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.
  - You won’t ever know all possible attacks, so just make sure it’s someone’s job to be on top of that
- 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 “IT” or a mega-scaler or service provider
  - But – centralisation of the web is a real problem, as is control
- 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 hard problems with passwords are **not** really cryptographic ones
  - But... some service providers are getting keen on PAKEs

What’s a sysadmin to do? (3)

- Use 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
  - 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 recently
  - https://www.passwordping.com/surprising-new-password-guidelines-nist/

Biometrics eh?

- Beware of biometrics!
  - You cannot easily change your fingerprint, retina, toe-smell!
  - And biometric verifiers can be fooled as much as anything (and haven't been tested near as much as some things)
- CCC 2017: 55 minute video of breaking biometrics
  - Fingerprint stuff about 24 mins in
  - https://www.youtube.com/watch?annotation_id=annotation_2684251971&feature=iv&src_vid=pIY6k4gvQsY&v=VVxL9ymiyAU

- Fiebig, Tobias, Jan Krissler, and Ronny Hänsch. "Security Impact of High Resolution Smartphone Cameras." WOOT. 2014.
  - https://www.usenix.org/system/files/conference/woot14/woot14-fiebig.pdf

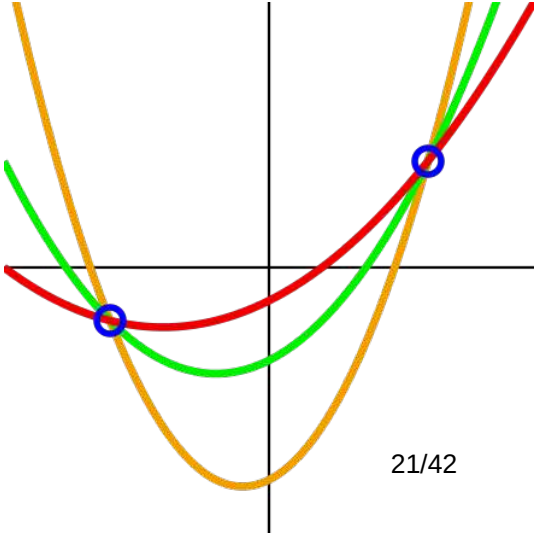
Aside: secret sharing

Aside: shamir secret sharing

- Nice function to give intuiton into how cryptographic functions tend to work
- HOWTO create “n shares” so that combining any k-of-n allows you to re-calculate a secret
- Shamir, Adi. "How to share a secret." Communications of the ACM 22.11 (1979): 612-613.
  - According to Google scholar: “Cited by 17484”
  - https://web.mit.edu/6.857/OldStuff/Fall03/ref/Shamir-HowToShareASecret.pdf

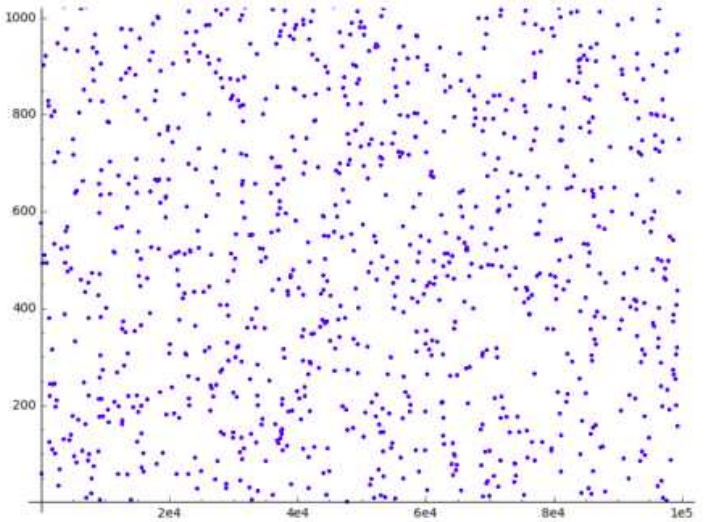
Shamir Secret Sharing

- Given a secret, how can I give n people a value such that once at least k (<=n) people get together they can reconstruct the secret?
- Shamir's scheme: If I have a polynomial of degree k-1, and I give each person one point (e.g. "i,f(i)" for i=1..n) then if any k of them get together they can find the formula for the polynomial and hence the secret
  - Usually the secret is "f(0)"
- Intuition: I need two points to know the formula for a line. I need three points to find the formula for a quadratic, etc.
  - https://en.wikipedia.org/wiki/Shamir%27s_Secret_Sharing
  - Figures on this and next slide from that Wikipedia page



More on Shamir

- For security reasons we choose polynomials over a finite field e.g. F(p) with p a prime that's big enough for the given k-of-n scenario
  - With a polynomial over the reals, k-1 points can allow guessing of the k-th point a bit too easily, e.g. if secret value were "small" guessing may be cryptographically efficient
- Intuition: polynomial curves over finite fields are more "random" looking
- We end up dealing with polynomical coefficients that are maybe 64 or 128 bit values
  - Hey, it's crypto:-)



Ok, here comes the password hashing nitty-gritty....

Again, don’t worry:-)

Our goal is that you properly understand stories such as:

https://arstechnica.com/information-technology/2019/09/doordash-hack-spills-loads-of-data-for-4-9-million-people/

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 those be?
- System uses u to retrieve 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 as part of f, 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

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

Cryptographic Hashes (2)

- Finding collisions is **much** easier than pre-images due to birthday paradox
  - https://en.wikipedia.org/wiki/Birthday_problem
- 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
  - Note: 2^128 is a **lot** less than 2^256!
- Hash function names:
  - 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 recent competition related to that kind of function

MD5

- MD5 is an old hash function, probably older than most in this room:-)
  - See RFC 1321 (https://www.rfc-editor.org/rfc/rfc1321)
  - 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, 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

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://www.rfc-editor.org/rfc/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
  - RFC9106 - https://datatracker.ietf.org/doc/html/rfc9106
- All very nice, but after PHC win some issues discovered, "fixed" by Argon2d (Argon2i won the PHC), and deployment is (so far) v. limited

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

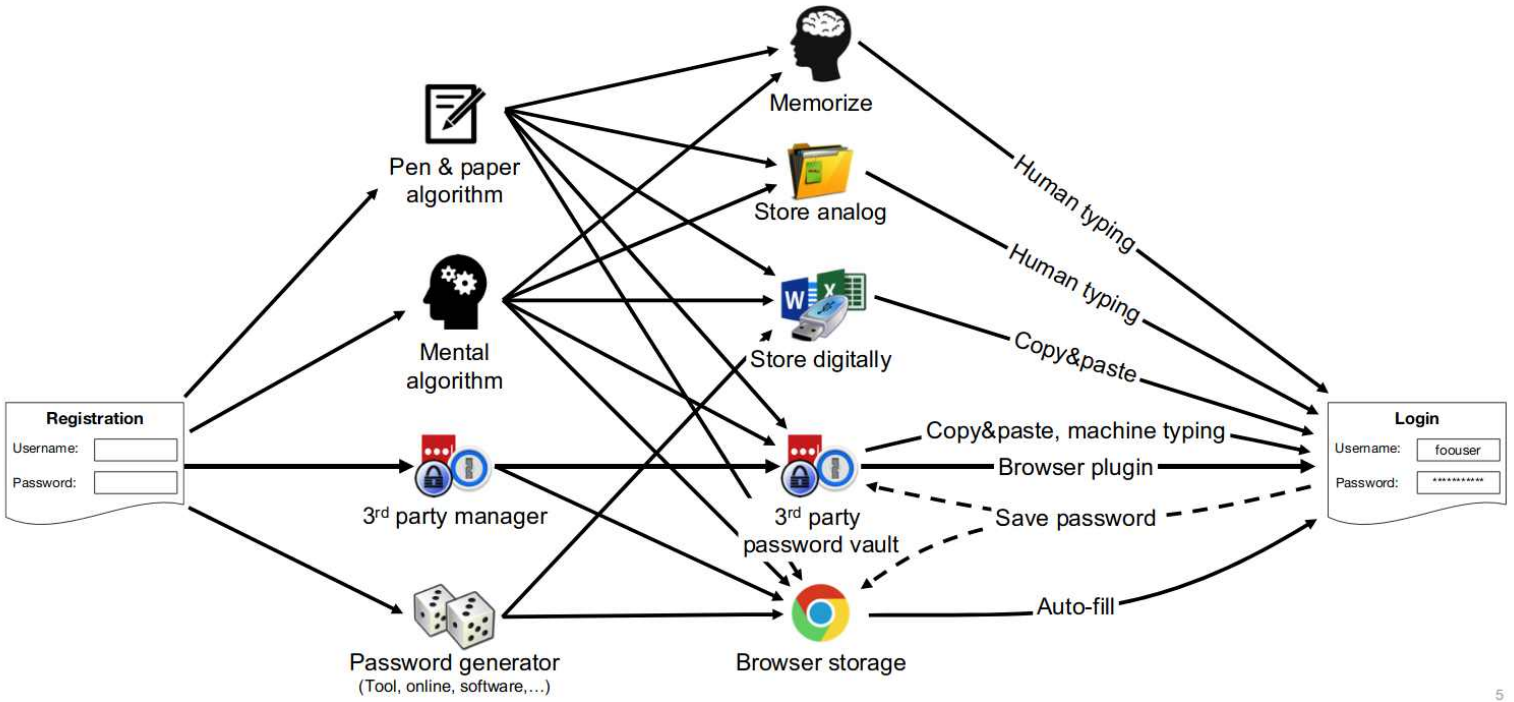
- Describes challenges/opportunities in distributed computation for hash cracking with a nice description of implementation and measurement issues

Password Managers

Password Managers

- Basic idea: humans are crap at managing usefully complex passwords and services are crap at not letting PVDB's leak, so why not have a bit of client-side software handle all your passwords?
- This can make sense, but there are trade-offs
- For me, so far, I've not gone down this route, but I might, depending...
  - My current setup works, but would make for a lot of work if my laptop were stolen

Password Handling Strategies



Lyastani, Sanam Ghorbani, et al. "Better managed than memorized? Studying the Impact of Managers on Password Strength and Reuse." 27th {USENIX} Security Symposium ({USENIX} Security 18). 2018. https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-lyastani.pdf

Some Resources (accessed 2022-10-18)

- Wikipedia page with a list and nice characterisation of products and features:
  - https://en.wikipedia.org/wiki/List_of_password_managers
- More commercial lists/reviews:
  - Caveat: I am not recommending specific products!
  - Some of the article authors here make money from people buying these products/services
  - Tom’s guide:
    - https://www.tomsguide.com/us/best-password-managers,review-3785.html
  - CNET “best of”:
    - https://www.cnet.com/tech/services-and-software/best-password-manager/
  - PC mag “best of”:
    - https://www.pcmag.com/picks/the-best-password-managers

Factors to consider

- Are you ok with just letting your browser(s) do it? Some can sync over >1 device within the browser (e.g. if you create a Firefox account or are a dedicated Apple-fan)
- How much time do you want to invest in start-up/testing costs?
  - **BUT** how much might you lose if some of your passwords leak?
- Support for your preferred current/future platforms (OSes, browser prefs, ...)
- Cost, limitations (e.g. syncing)
- Form filling for the forms you care about (usually via browser extensions)
- Open/closed source (important for me)
- Developer has history of fixing problems quickly?
- Server sync needed or just client-side?
- How many eggs do you like in how many baskets?
  - https://www.zdnet.com/article/lastpass-bug-leaks-credentials-from-previous-site/

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
- You should think about whether you ought use a password manager
- Re-use passwords less!

Questions? Things to add?

- <your text here>