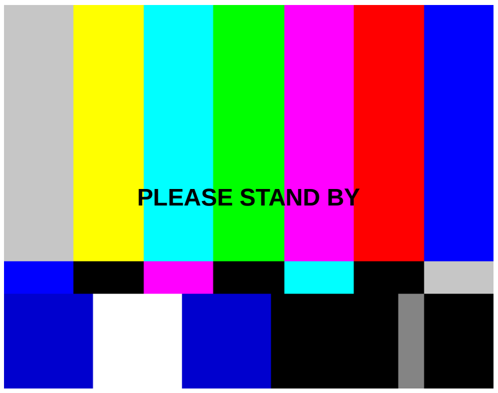
Insidious Implicit Windows Trust Relationships

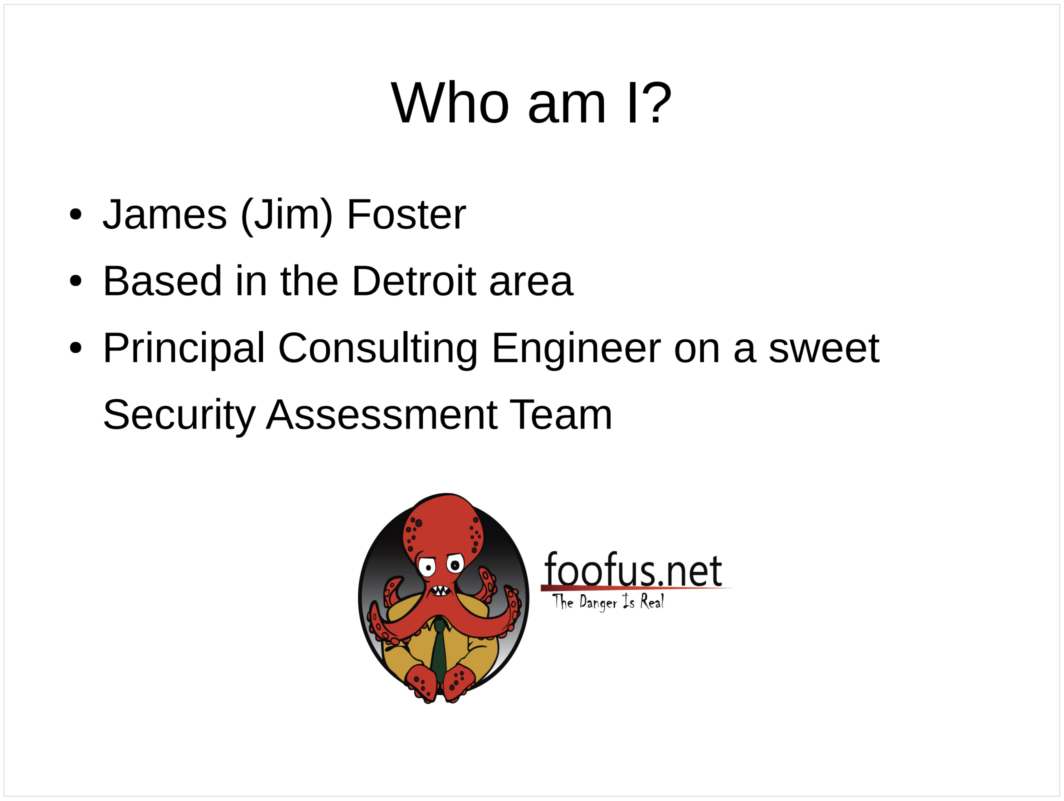


7 June 2013 – BSides Detroit
James Foster

Note about where to get these slides later and whether or not this is being recorded.

Note to folks reading these slides and notes directly: the word TRANSITION in the notes is just a reminder to me that I have animations or transitions on that slide. You can ignore it in the PDF version.

None of the information presented here is original work, it's all stuff other people have figured out. I'm just trying to spread the knowledge.



Who am I?

- James (Jim) Foster
- Based in the Detroit area
- Principal Consulting Engineer on a sweet Security Assessment Team

foofus.net
The Danger Is Real

This Security Assessment Team has been around ~15 years, including the authoring of some well known tools and related awesomeness.

Who am I?

- I do:
  - Security assessments / penetration tests
  - Incident response
  - Whatever other security related stuff comes up
- I have:
  - ~18 years of experience in various IT roles; the last ~9 in IT security, doing lots of different stuff
  - BSCS, CISSP, GCIH

Intended audience

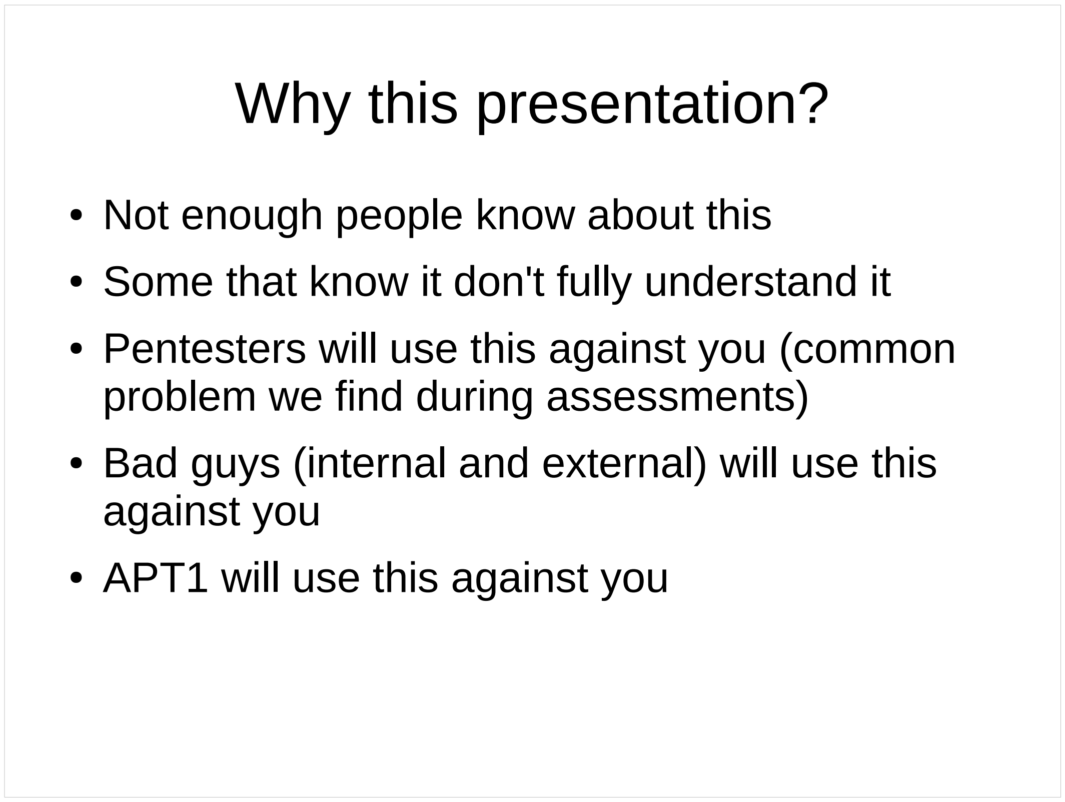
- Not pentesters (they already know this)

- Non-security IT folks
- IT security folks who are busy with other things
- Those tasked with supporting / defending Windows systems, especially

Show of hands

How many folks here are:
- new to all this?
- straight IT (no security)?
- IT with some security responsibility?
- full time IT security?
- attackers?
- anything else?

Primarily tasked with supporting an Active Directory domain, or domain-member systems? Of people with their hands down, how many of their systems authenticate against a domain, or are administered from a domain-member system?



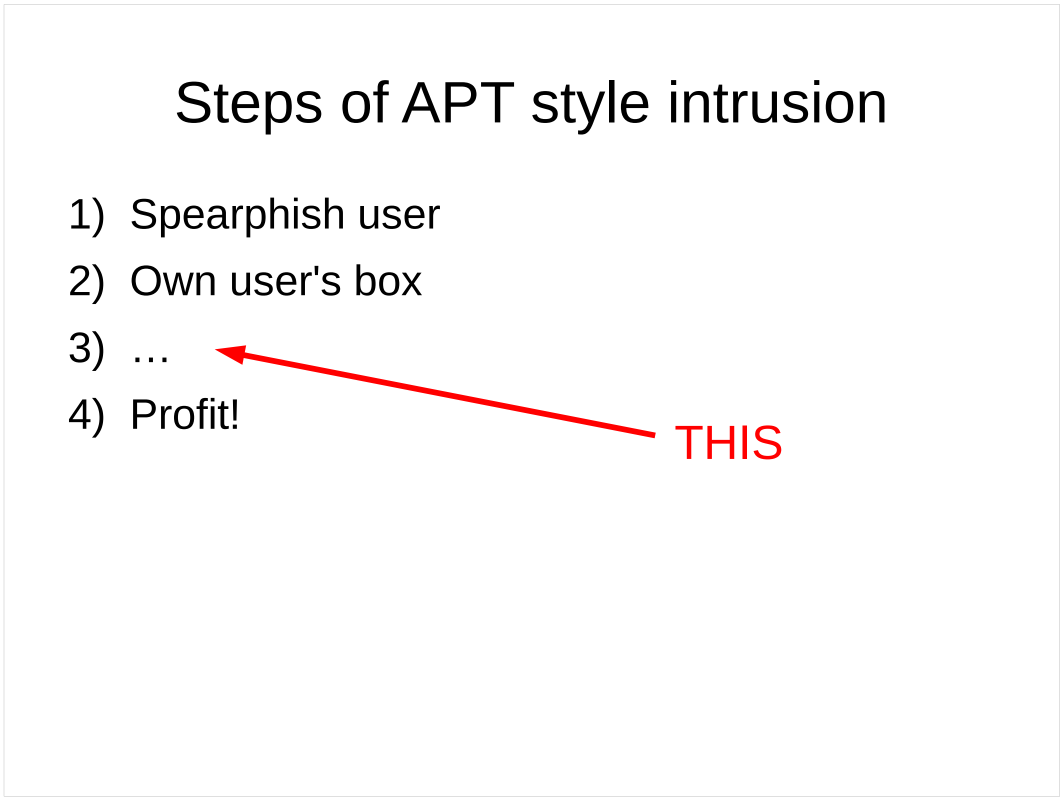
Why this presentation?

- Not enough people know about this
- Some that know it don't fully understand it
- Pentesters will use this against you (common problem we find during assessments)
- Bad guys (internal and external) will use this against you
- APT1 will use this against you

TRANSITION

Before I was an attacker, I fell into the second category.

Remember: I didn't discover any of the information presented here. I'm just good at summarizing and explaining.



Steps of APT style intrusion

1) Spearphish user
2) Own user's box
3) …
4) Profit!

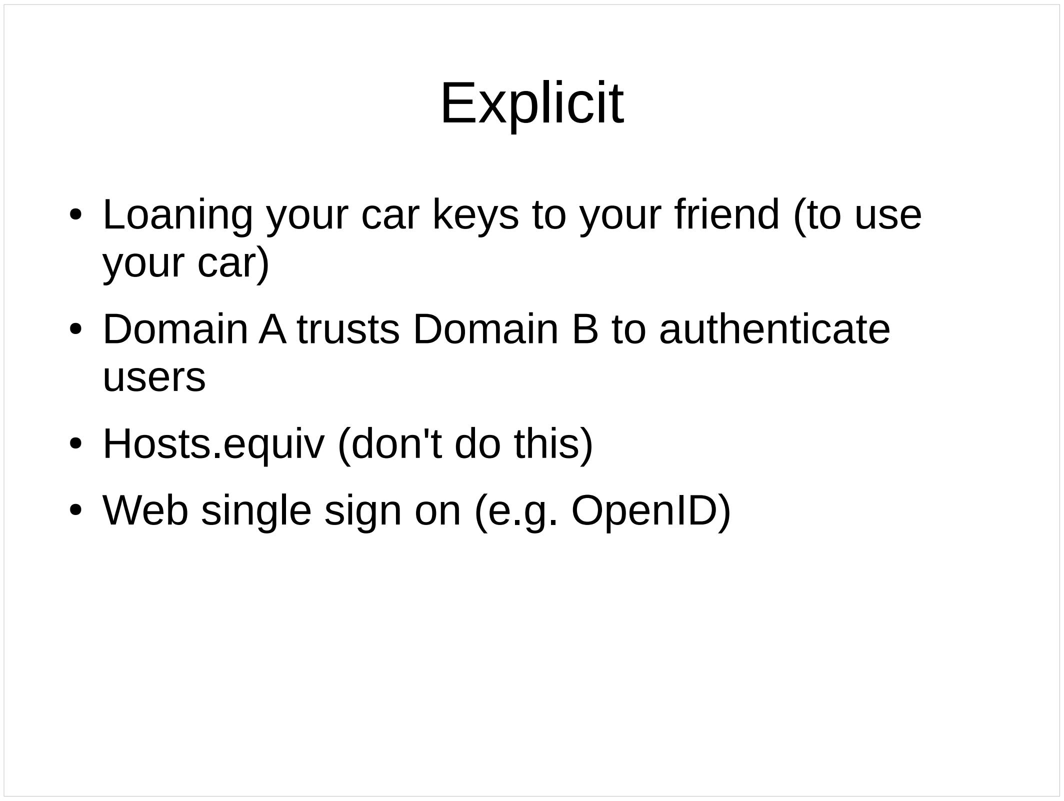
TRANSITION

Part of step 3 is exploiting implied trust relationships. Often this is a big part of “move laterally” and “escalate privileges”.

Now that you're fully convinced how important this topic is, let's go.

Two kinds of trusts

- Explicit – these you intend to exist
- Implicit (implied) – these you don't

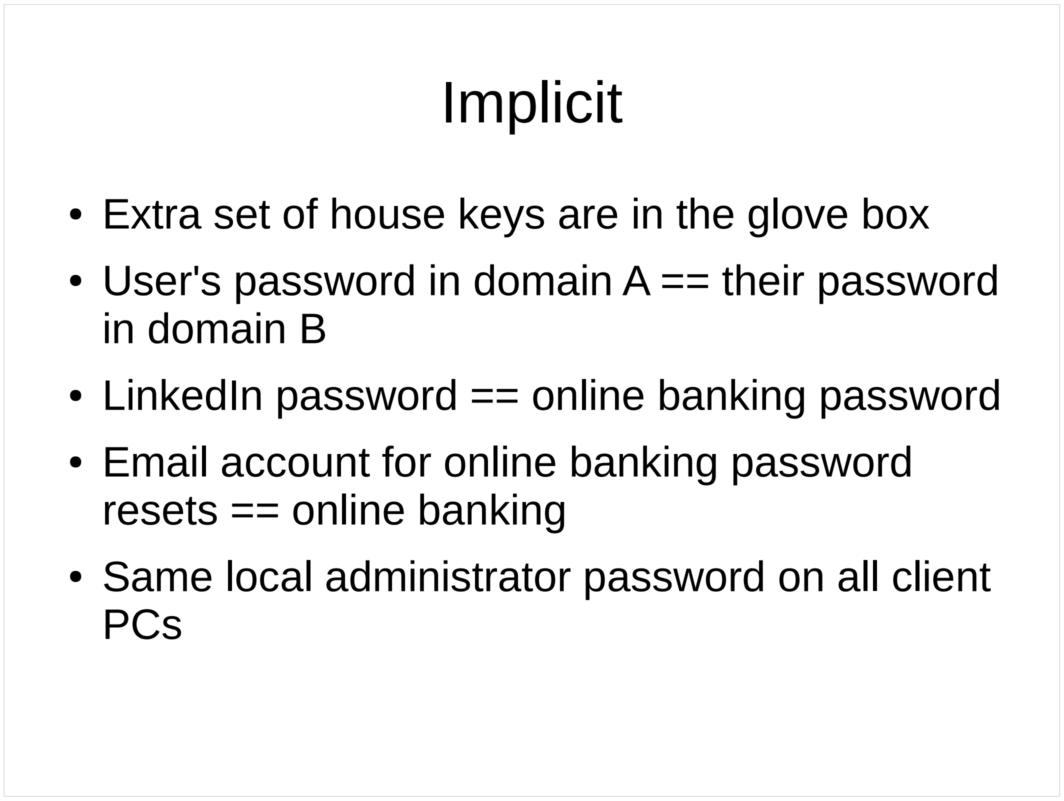


Explicit

- Loaning your car keys to your friend (to use your car)
- Domain A trusts Domain B to authenticate users
- Hosts.equiv (don't do this)
- Web single sign on (e.g. OpenID)

TRANSITION

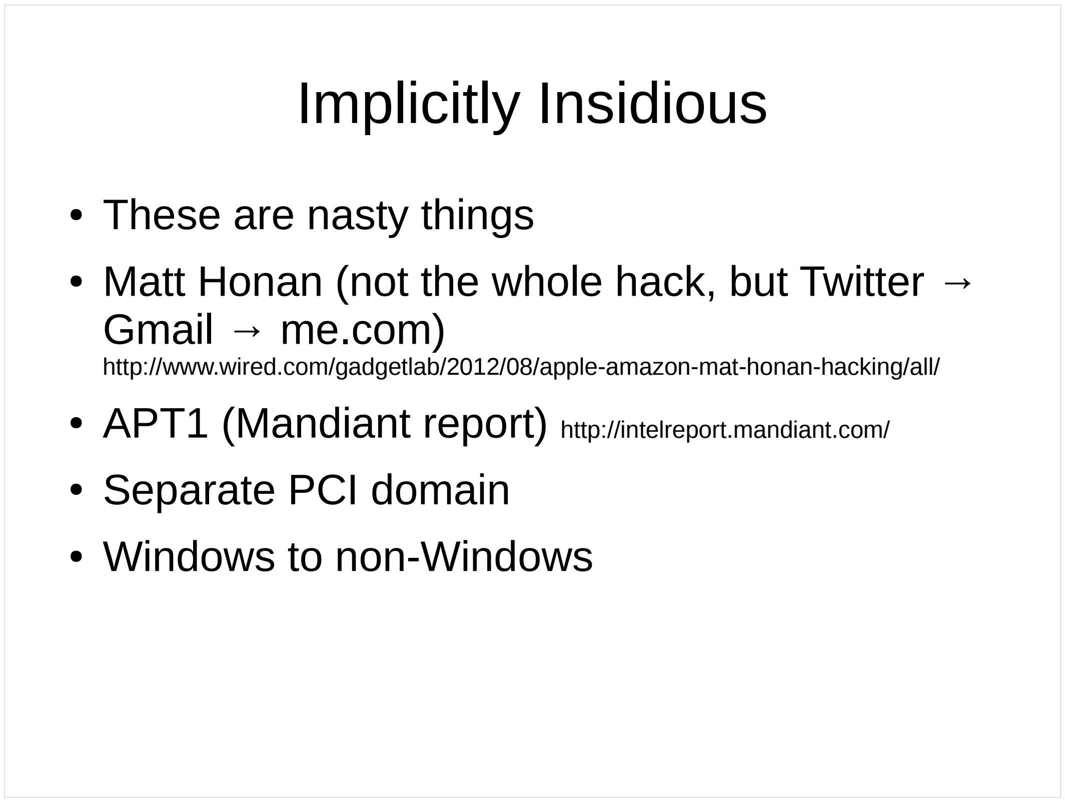
In the last case, the Relying Parties (websites you're trying to login to) explicitly trust third-party Identity Providers (eg, Google) to authenticate you.



Implicit

- Extra set of house keys are in the glove box
- User's password in domain A == their password in domain B
- LinkedIn password == online banking password
- Email account for online banking password resets == online banking
- Same local administrator password on all client PCs

TRANSITION



Implicitly Insidious

- These are nasty things
- Matt Honan (not the whole hack, but Twitter → Gmail → me.com) http://www.wired.com/gadgetlab/2012/08/apple-amazon-mat-honan-hacking/all/
- APT1 (Mandiant report) http://intelreport.mandiant.com/
- Separate PCI domain
- Windows to non-Windows

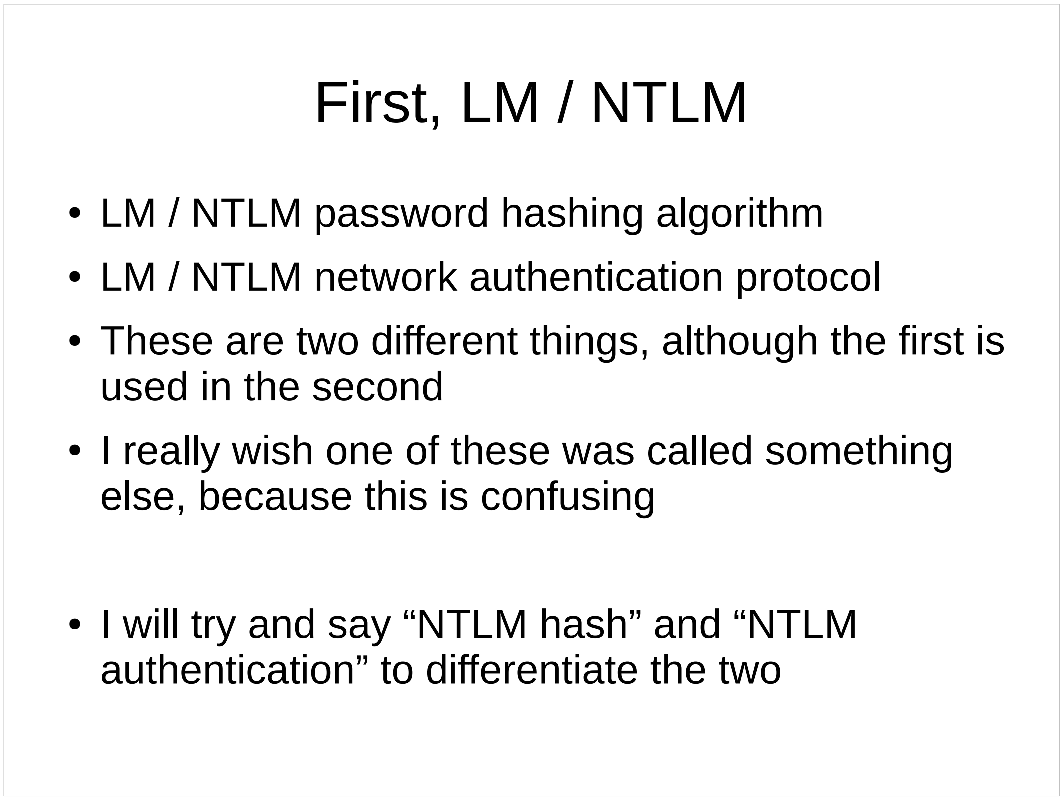
TRANSITION

They wanted Matt's Twitter account. It used his Gmail account for password reset/recovery, so they needed that first. His Gmail account used his me.com (Apple) account for password reset/recovery. They knew how to get Apple accounts (a different attack), so once they got that, they got his Gmail and Twitter for free.

The last two are stories from assessments.

Focus on Windows

- You've got the general idea
- Let's see why implicit trusts matter so much in Windows



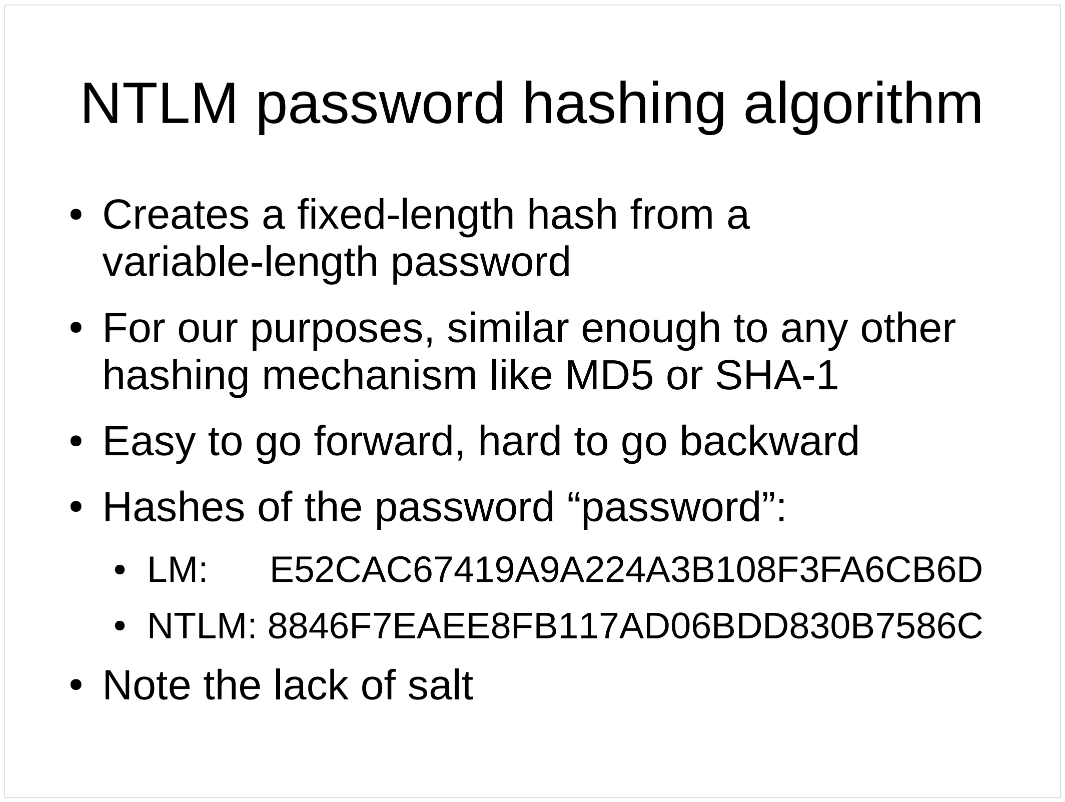
First, LM / NTLM

- LM / NTLM password hashing algorithm
- LM / NTLM network authentication protocol
- These are two different things, although the first is used in the second
- I really wish one of these was called something else, because this is confusing

- I will try and say “NTLM hash” and “NTLM authentication” to differentiate the two

TRANSITION

I'm just going to say NTLM from now on since it's easier, and this all applies to environments that “aren't using LM anymore” anyway.



NTLM password hashing algorithm

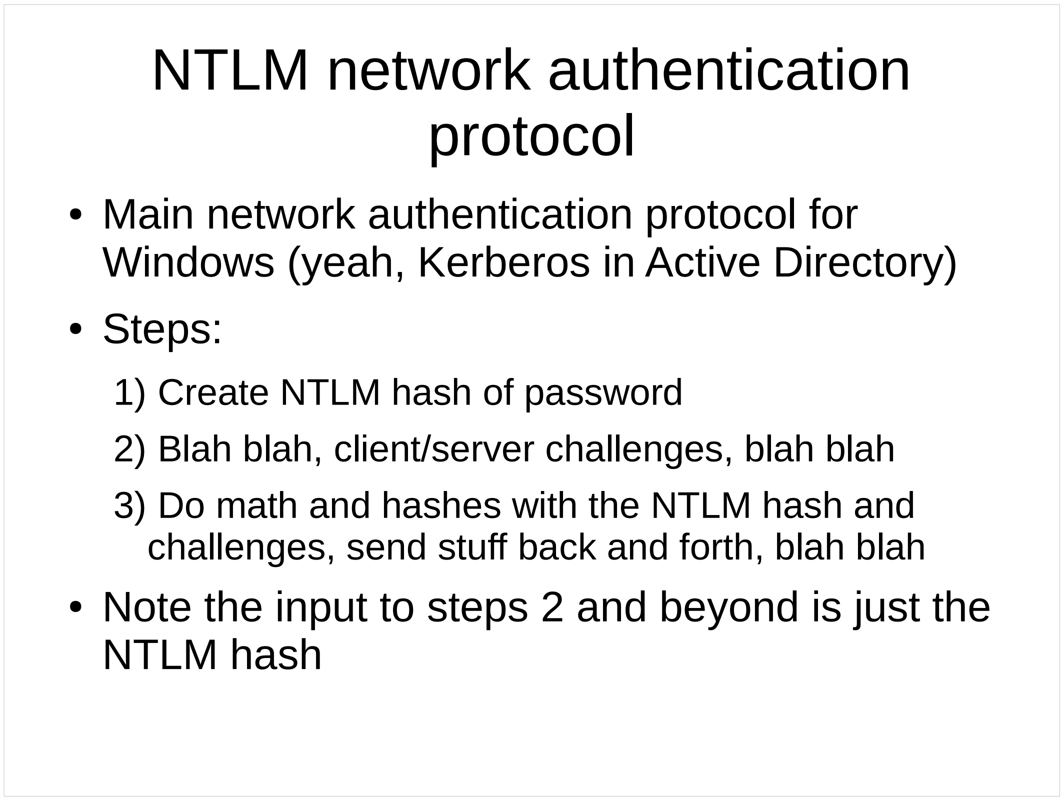
- Creates a fixed-length hash from a variable-length password
- For our purposes, similar enough to any other hashing mechanism like MD5 or SHA-1
- Easy to go forward, hard to go backward
- Hashes of the password “password”:
  - LM: E52CAC67419A9A224A3B108F3FA6CB6D
  - NTLM: 8846F7EAEE8FB117AD06BDD830B7586C
- Note the lack of salt

TRANSITION

What does the lack of salt mean?

Can use rainbow tables.

Users with the same password will have the same hash, regardless of username, system, domain, version of Windows, language, etc.



NTLM network authentication protocol

- Main network authentication protocol for Windows (yeah, Kerberos in Active Directory)
- Steps:
  1) Create NTLM hash of password
  2) Blah blah, client/server challenges, blah blah
  3) Do math and hashes with the NTLM hash and challenges, send stuff back and forth, blah blah
- Note the input to steps 2 and beyond is just the NTLM hash

TRANSITION

The details of steps 2 and beyond don't matter for our purposes. The output of step 1 is just the NTLM hash of the password, with nothing else added.

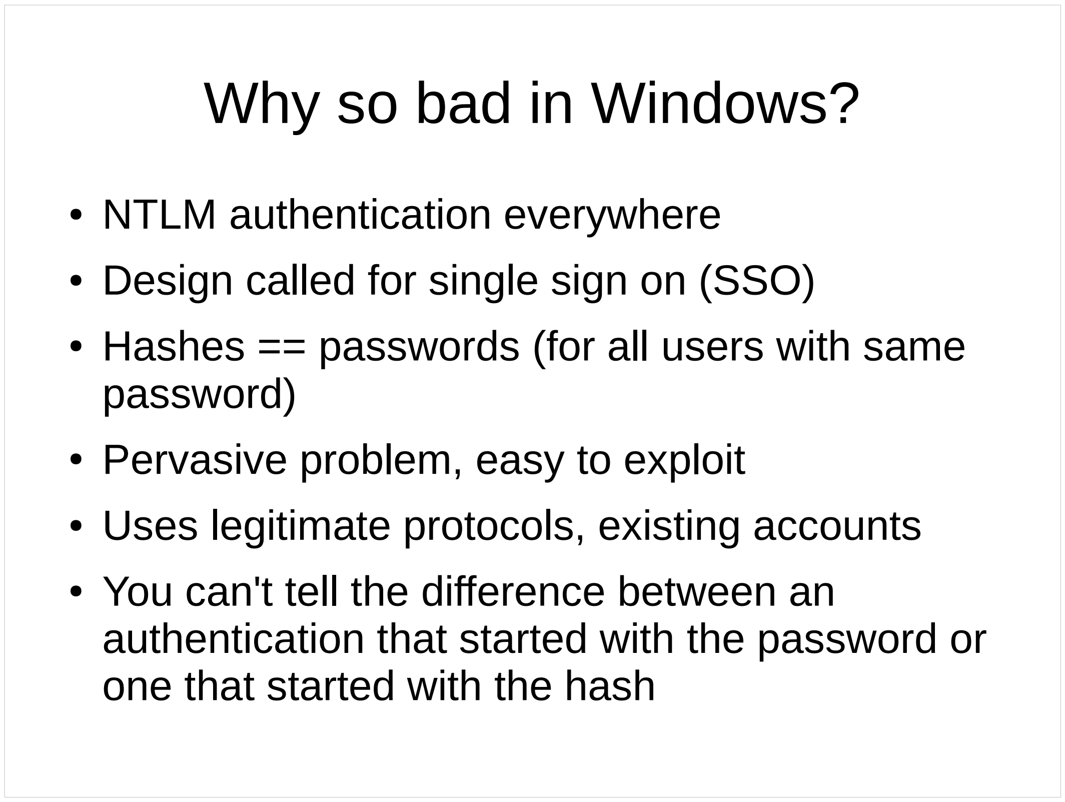
So what does this mean for NTLM authentication?

What if we have a user's hash, but don't know their password (couldn't crack it, whatever).

Doesn't matter, because the hash works just as well.

Hashes == passwords

- (for NTLM authentication)
- Often called “pass-the-hash” (PTH)
- And not just for the one user – for all users who have the same password



Why so bad in Windows?

- NTLM authentication everywhere
- Design called for single sign on (SSO)
- Hashes == passwords (for all users with same password)
- Pervasive problem, easy to exploit
- Uses legitimate protocols, existing accounts
- You can't tell the difference between an authentication that started with the password or one that started with the hash

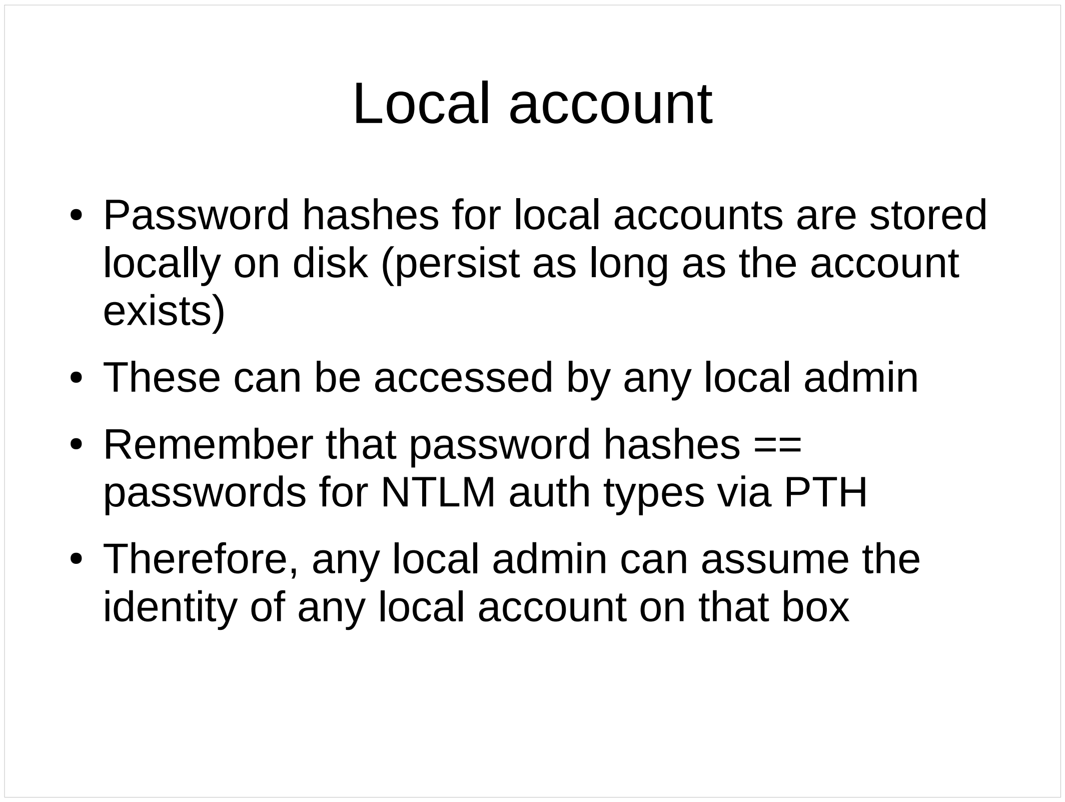
TRANSITION

After all, do you want to have to re-type your password for every new Windows resource your system connects to?

For SSO to work, the system has to either know your password (or its hash in NTLM authentication) or have some token (like in Kerberos).

Windows implicit trust relationship types

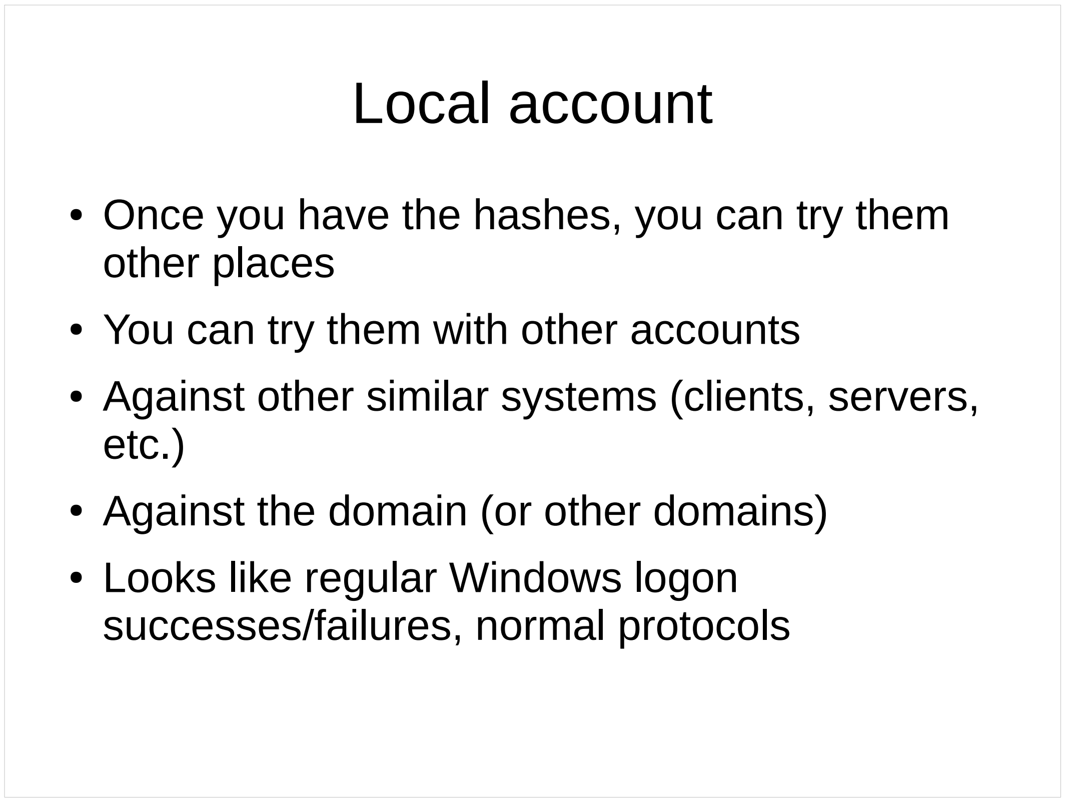
- Local account
- Cached credential
- Access token



Local account

- Password hashes for local accounts are stored locally on disk (persist as long as the account exists)
- These can be accessed by any local admin
- Remember that password hashes == passwords for NTLM auth types via PTH
- Therefore, any local admin can assume the identity of any local account on that box

TRANSITION



Local account

- Once you have the hashes, you can try them other places
- You can try them with other accounts
- Against other similar systems (clients, servers, etc.)
- Against the domain (or other domains)
- Looks like regular Windows logon successes/failures, normal protocols

You might guess that this password (hash) is the same for this same username on other systems. Often you'd be right.

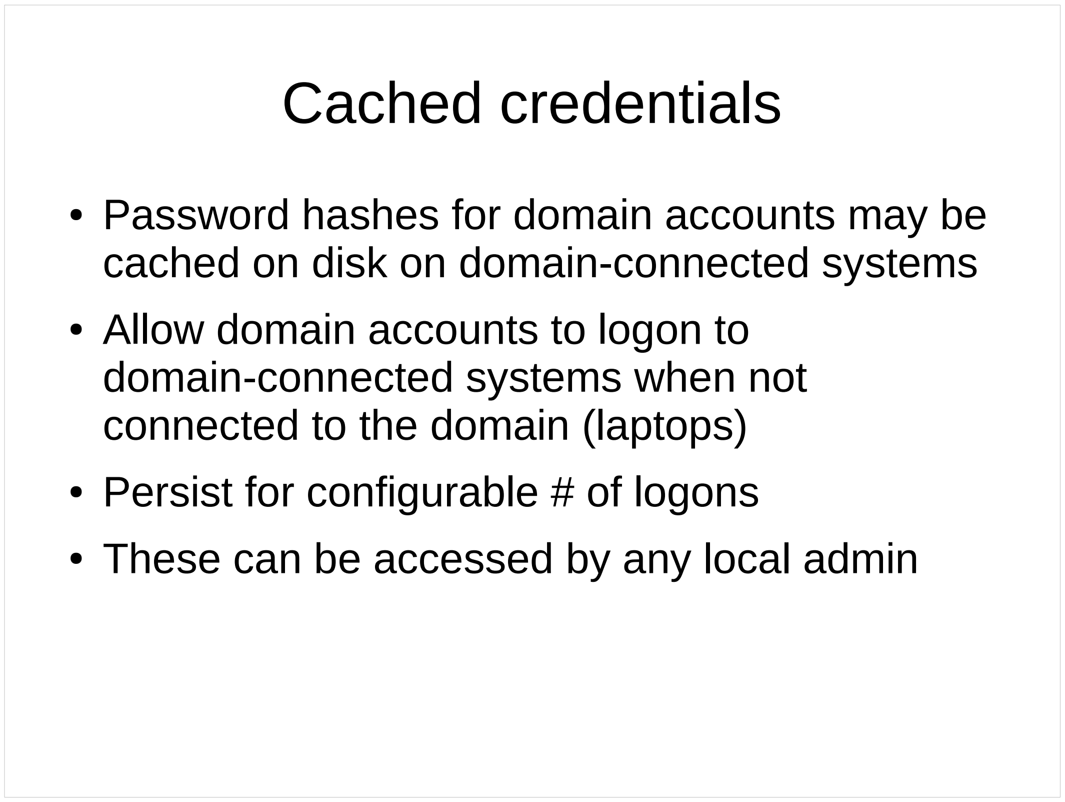
You might guess that this password (hash) is the same for other usernames on other systems. Sometimes you'd be right.

Sometimes it's the same in the domain, too.

Local account

- On a domain controller, all domain accounts are just “local accounts” in this sense
- Get “local admin” on a domain controller, get the hashes of all domain accounts
- Yay!

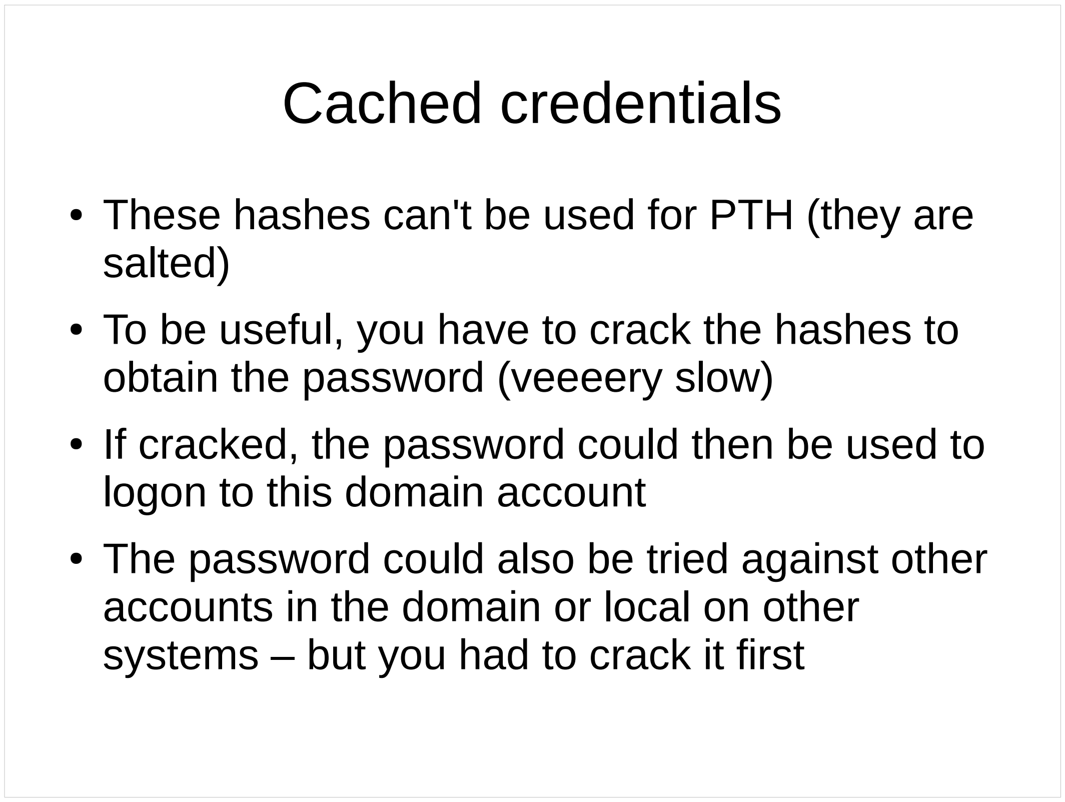
This may seem obvious. In order to compromise an entire domain and steal all of its users' hashes, you just need to compromise a domain controller and you're done.



Cached credentials

- Password hashes for domain accounts may be cached on disk on domain-connected systems
- Allow domain accounts to logon to domain-connected systems when not connected to the domain (laptops)
- Persist for configurable # of logons
- These can be accessed by any local admin

TRANSITION



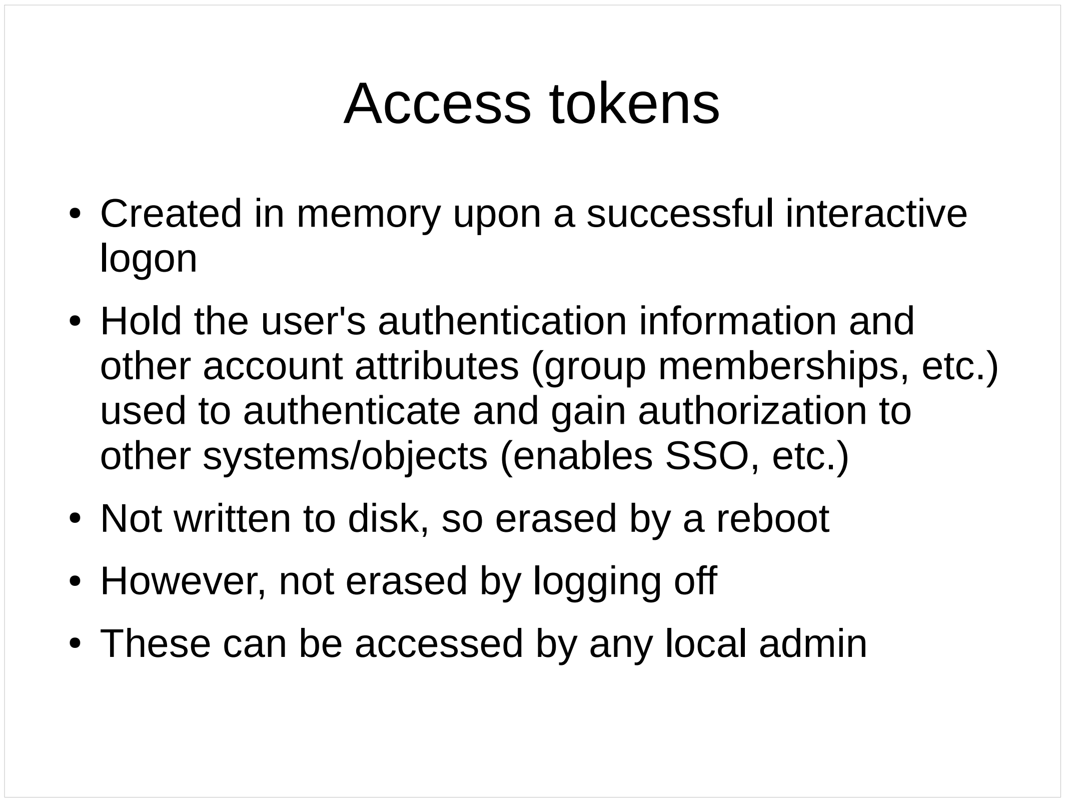
Cached credentials

- These hashes can't be used for PTH (they are salted)
- To be useful, you have to crack the hashes to obtain the password (veeeery slow)
- If cracked, the password could then be used to logon to this domain account
- The password could also be tried against other accounts in the domain or local on other systems – but you had to crack it first

TRANSITION

Of course, if the password is trivial, it doesn't matter if the cracking is “slow”, you'll get it in a few seconds anyway.

This has allowed me to compromise a domain, but it's really the least useful of the three kinds of implied trusts. So we'll move on.



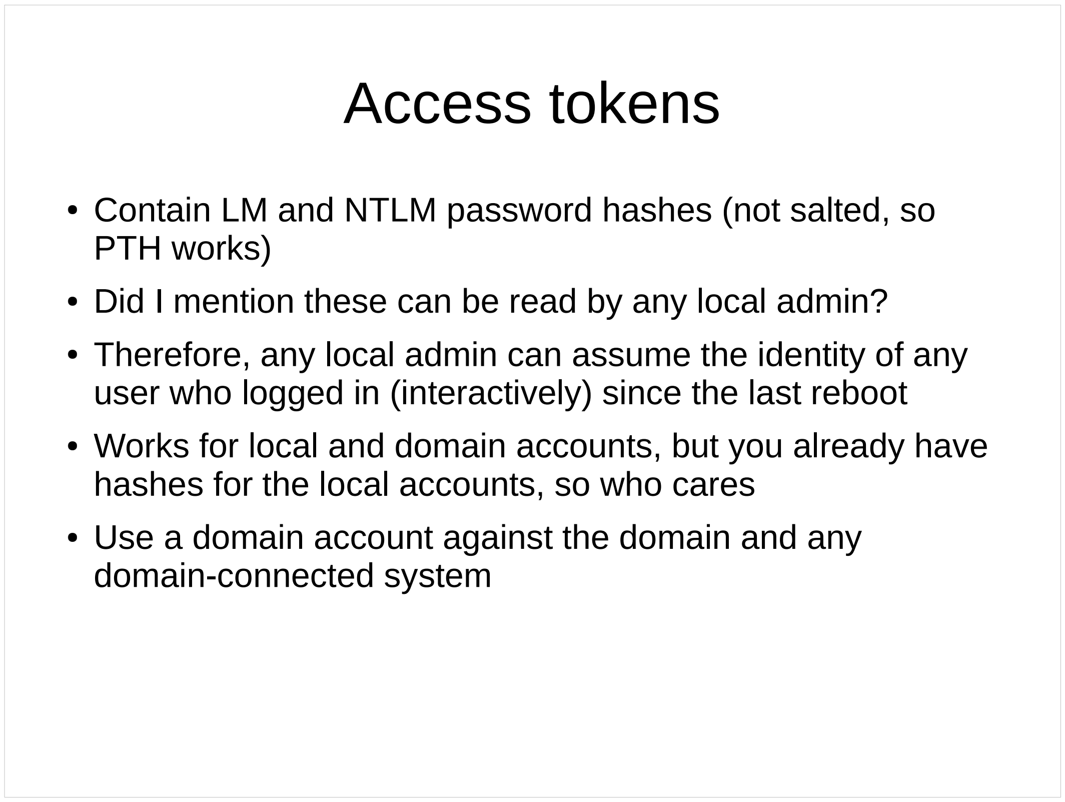
Access tokens

- Created in memory upon a successful interactive logon
- Hold the user's authentication information and other account attributes (group memberships, etc.) used to authenticate and gain authorization to other systems/objects (enables SSO, etc.)
- Not written to disk, so erased by a reboot
- However, not erased by logging off
- These can be accessed by any local admin

TRANSITION

Sometimes people call these “account tokens”, “logon tokens”, or just “tokens”.

Note that just mapping a drive does not create an interactive logon to the target, so this is not enough to create an access token.



Access tokens

- Contain LM and NTLM password hashes (not salted, so PTH works)
- Did I mention these can be read by any local admin?
- Therefore, any local admin can assume the identity of any user who logged in (interactively) since the last reboot
- Works for local and domain accounts, but you already have hashes for the local accounts, so who cares
- Use a domain account against the domain and any domain-connected system

TRANSITION

We just love access tokens belonging to domain admins. Tasty.

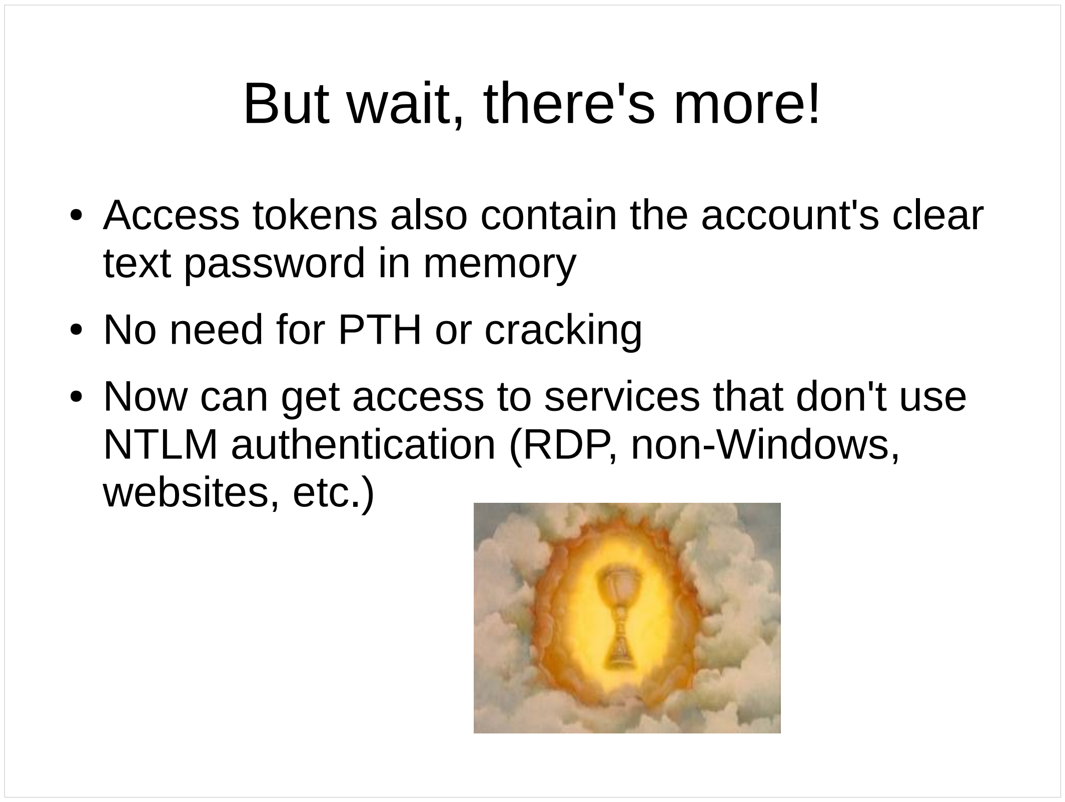
Show of hands

How many people here have a domain admin account?

Have you ever logged on to a system and then didn't reboot it afterwards?

A user's system?

All this for $9.99? Can you believe it?



But wait, there's more!

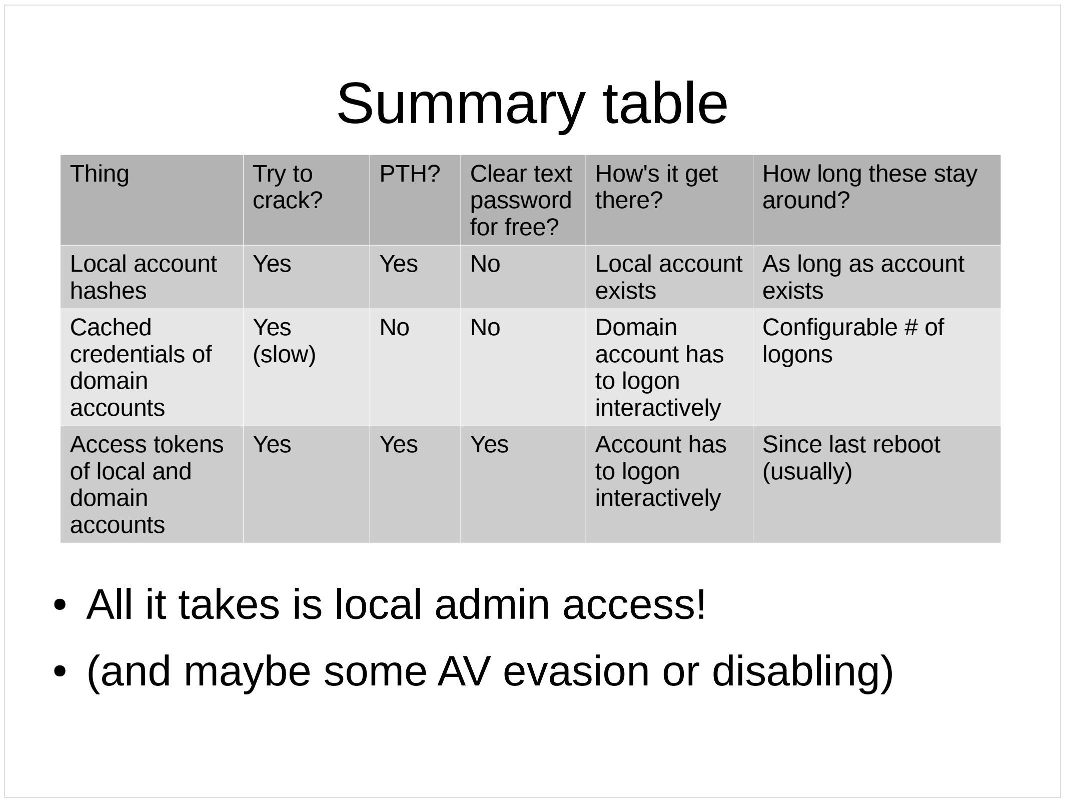
- Access tokens also contain the account's clear text password in memory
- No need for PTH or cracking
- Now can get access to services that don't use NTLM authentication (RDP, non-Windows, websites, etc.)

TRANSITION

This has only become widely known in the past couple years. The first tool to recover these passwords from memory was mimikatz, but WCE and others (Metasploit, etc.) do it too.

How many folks already knew this?

This should have been obvious, since features exist to allow SSO to non-NTLM auth services (HTTP Digest, etc.), which would necessitate the clear text password. Also, needed to renew Kerberos ticket.



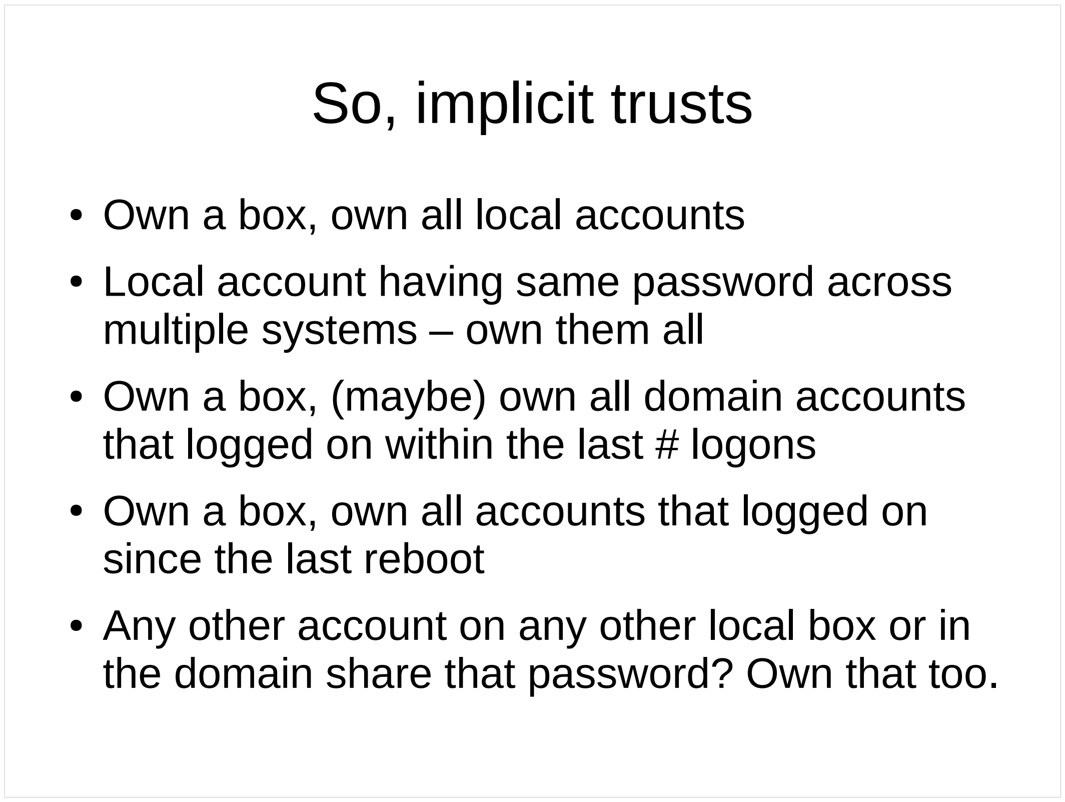
Summary table

Thing	Try to crack?	PTH?	Clear text password for free?	How's it get there?	How long these stay around?
Local account hashes	Yes	Yes	No	Local account exists	As long as account exists
Cached credentials of domain accounts	Yes (slow)	No	No	Domain account has to logon interactively	Configurable # of logons
Access tokens of local and domain accounts	Yes	Yes	Yes	Account has to logon interactively	Since last reboot (usually)

- All it takes is local admin access!
- (and maybe some AV evasion or disabling)

The tools that are used to actually obtain any of this information (hashes/tokens/passwords) and assume the identity of others tend to be considered “hacking tools” by most AV products and so will be detected and/or stopped. But you're already a local admin, so usually you can disable, reconfigure or evade AV.

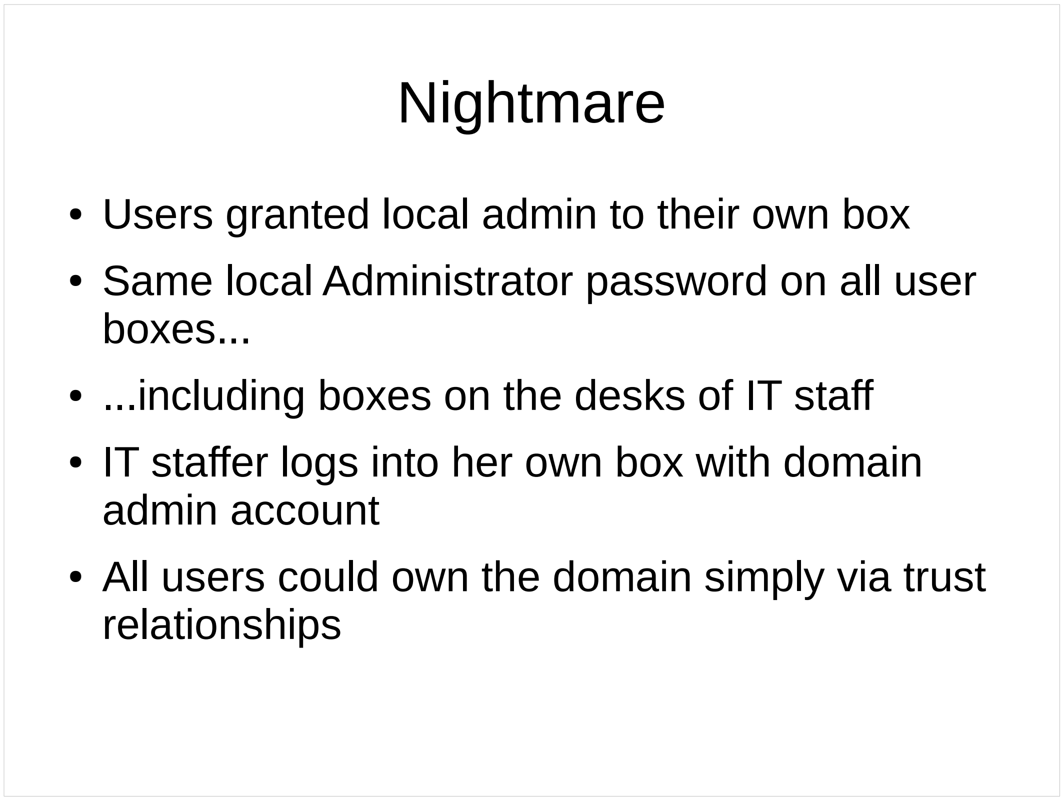
AV is not a sufficient protection against these attacks.



So, implicit trusts

- Own a box, own all local accounts
- Local account having same password across multiple systems – own them all
- Own a box, (maybe) own all domain accounts that logged on within the last # logons
- Own a box, own all accounts that logged on since the last reboot
- Any other account on any other local box or in the domain share that password? Own that too.

TRANSITION



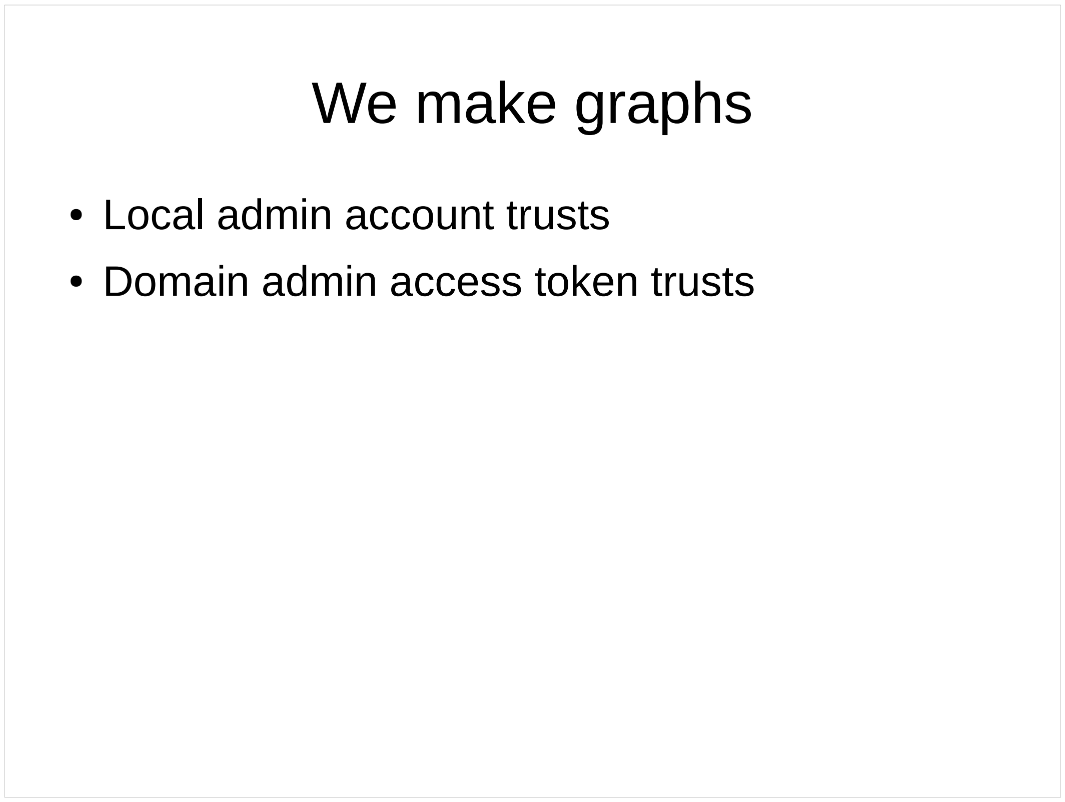
Nightmare

- Users granted local admin to their own box
- Same local Administrator password on all user boxes...
- ...including boxes on the desks of IT staff
- IT staffer logs into her own box with domain admin account
- All users could own the domain simply via trust relationships

TRANSITION

It's also bad when you have servers where lots of users, including privileged ones, logon interactively.

Although not completely tested or studied, we have seen most domain users' access tokens on an Exchange server in at least one environment. This implies that at least in some situations, Outlook's connection to the Exchange server constitutes an interactive logon and creates an access token there.

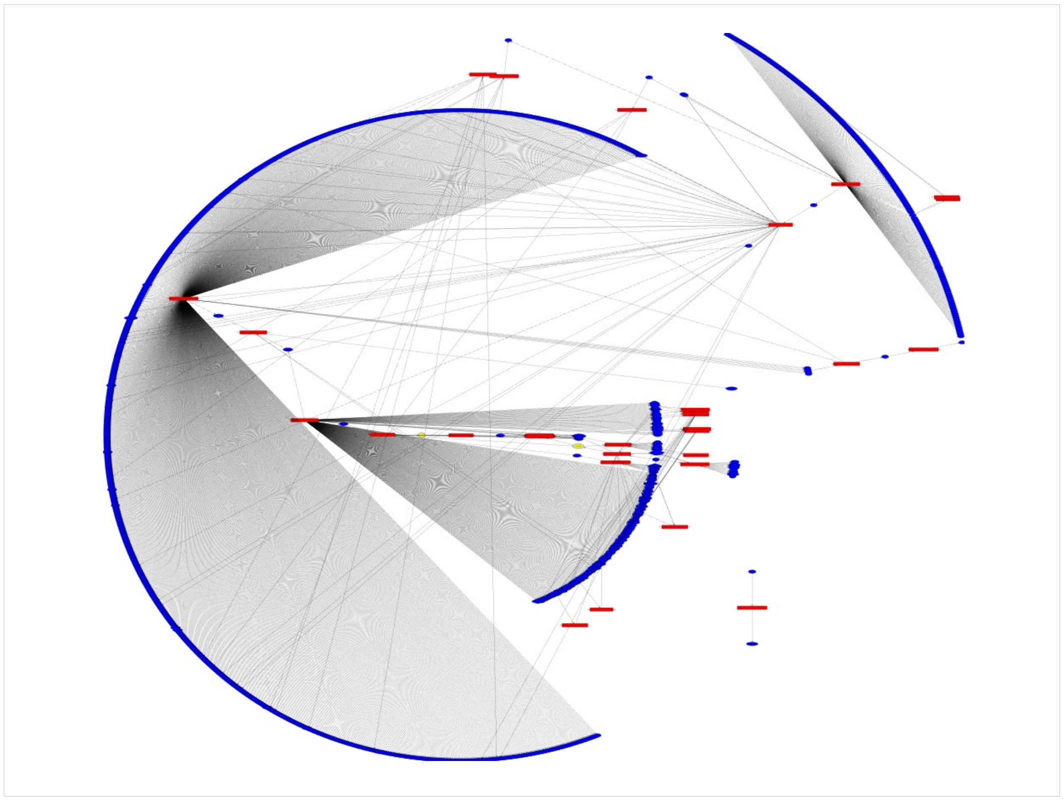


We make graphs

- Local admin account trusts
- Domain admin access token trusts

These graphs are from a recent internal assessment. Network had around 1,700 Windows boxes, we sampled about 1,500 of them to get the data for these graphs.

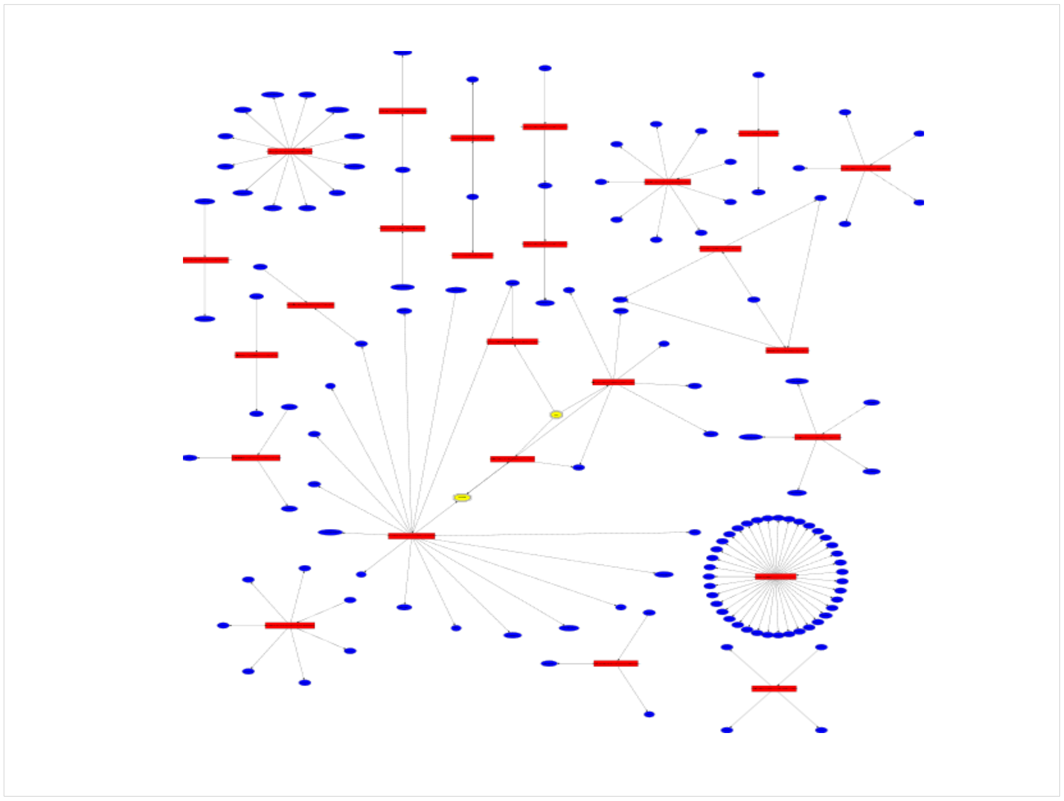
You can't read the labels on anything in the graphs on purpose.



Blue ovals are hosts, red boxes are credentials (username/password combination), and yellow spots are the domains (domain controllers, to be specific).

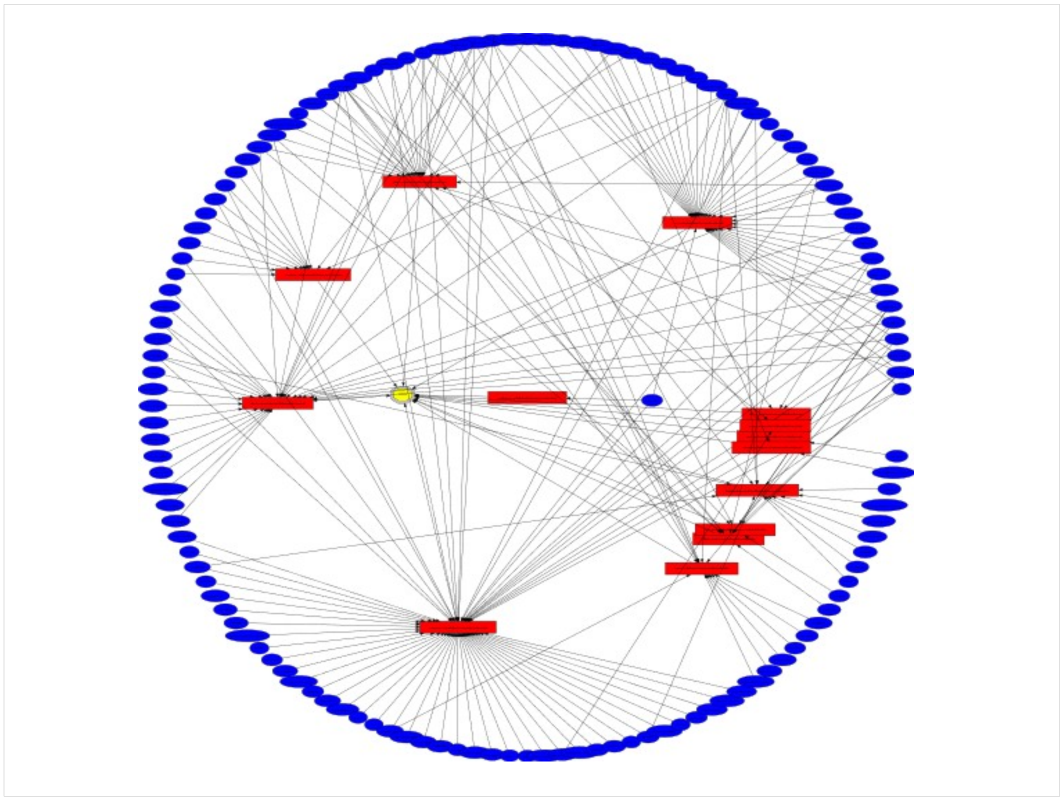
The “credentials” in this one are local administrative accounts, so this represents local account trusts for administrative level users (admin on hosts and/or the domain controllers).

~1,400 hosts involved in trusts with at least one other, many with many others, including the domain.



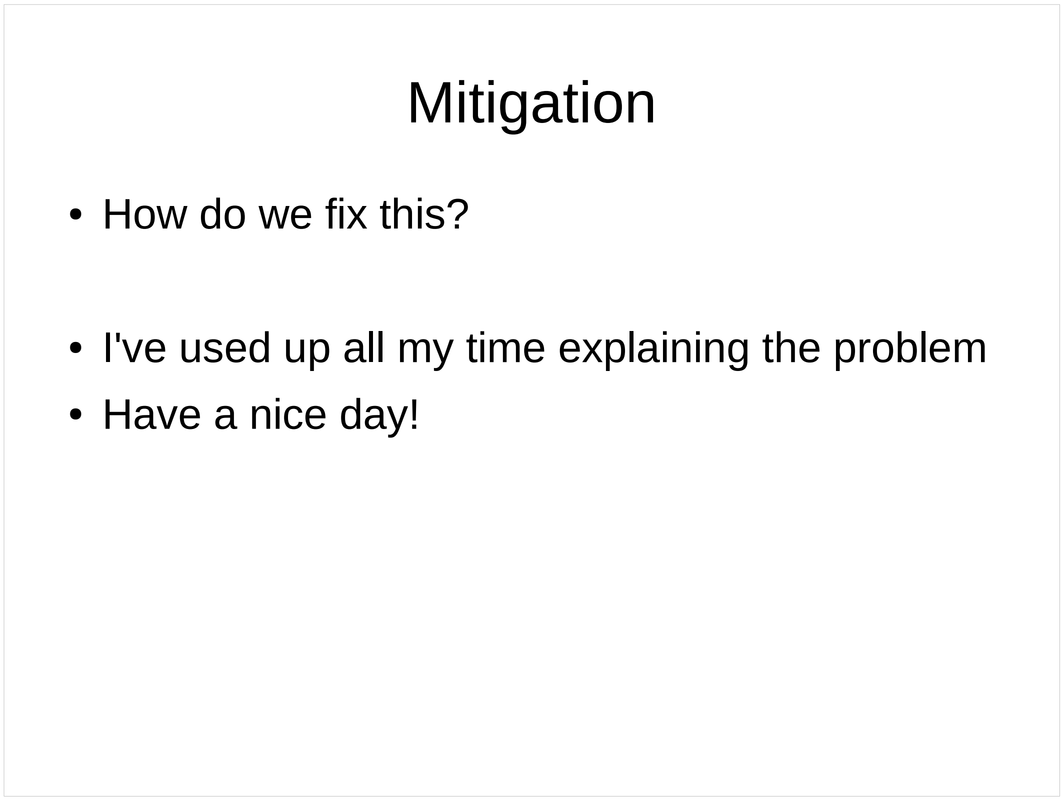
Blue ovals are hosts, red boxes are credentials (username/password combination), and yellow spots are the domains (domain controllers, to be specific).

The “credentials” in this one are local admin accounts, so this shows local account trusts for admin level users (admin on hosts and/or the domain controllers) EXCEPT that this time, the actual “Administrator” accounts are excluded. In other words, it's the same as the previous graph, if they were to fix just all of the local “Administrator” accounts. Only 129 hosts now involved in local account trust relationships. So by fixing the local “Administrator” account on all their boxes, they can achieve an order of magnitude improvement in # hosts involved.



Blue ovals are hosts, red boxes are credentials (username/password combination), and yellow spots are the domains (domain controllers, to be specific).

The “credentials” in this one are domain admin access tokens, so this represents access token trusts for domain admins only.

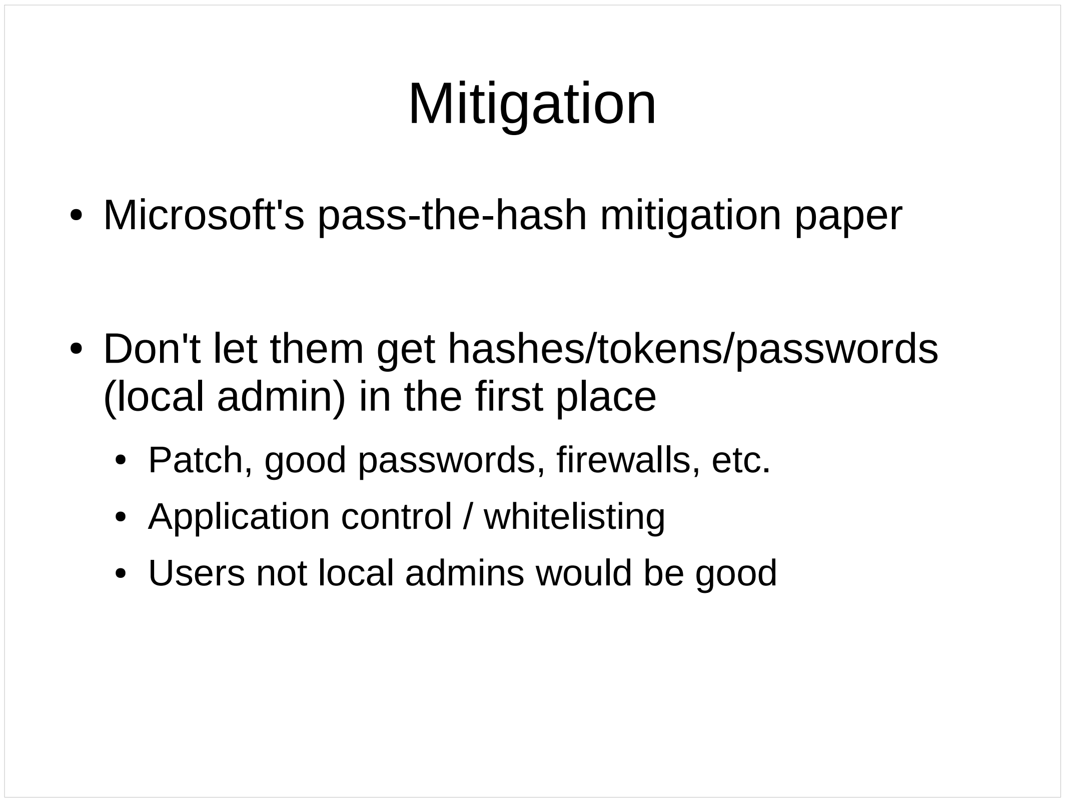


Mitigation

- How do we fix this?

- I've used up all my time explaining the problem
- Have a nice day!

Just kidding, I hope.



Mitigation

- Microsoft's pass-the-hash mitigation paper
- Don't let them get hashes/tokens/passwords (local admin) in the first place
  - Patch, good passwords, firewalls, etc.
  - Application control / whitelisting
  - Users not local admins would be good

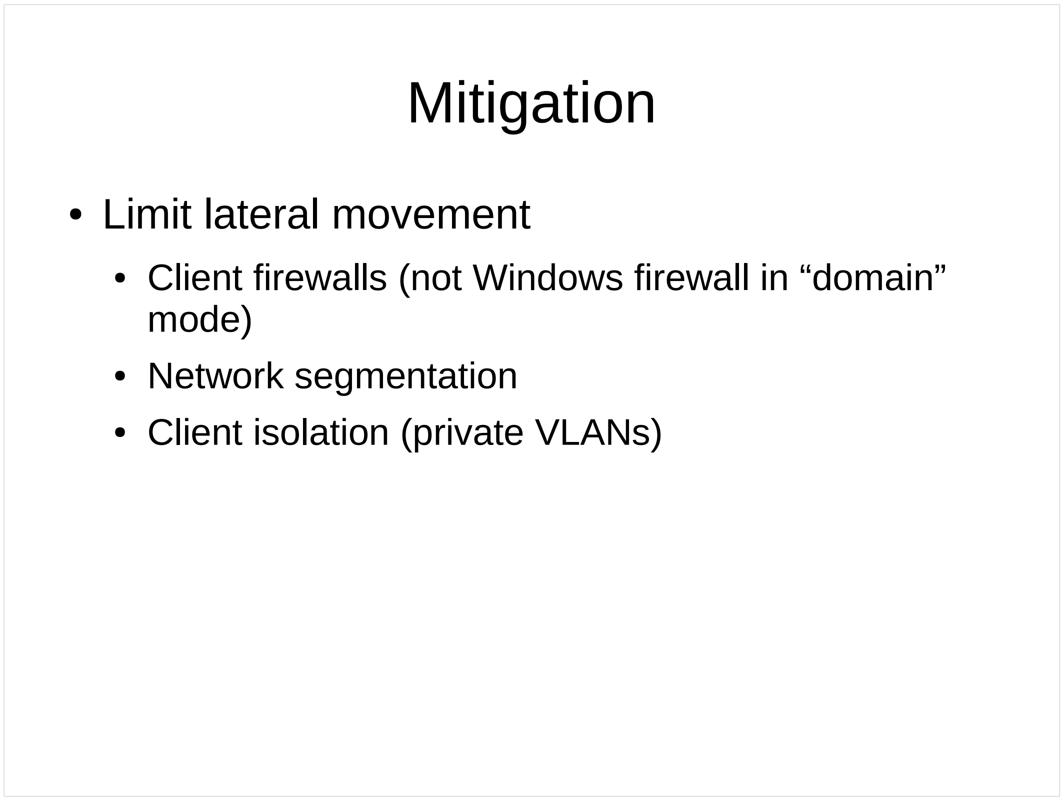
Does two factor auth (smart cards, biometrics, etc.) fix this? Haven't tested, but probably not, due to the nature of the problem.

Mitigation

- Minimize the number of hashes/tokens/passwords they can get
  - Limit cached credentials
  - Reduce number of local accounts, especially administrative ones
  - Limit number of interactive logons
  - Reboot frequently?

Mitigation

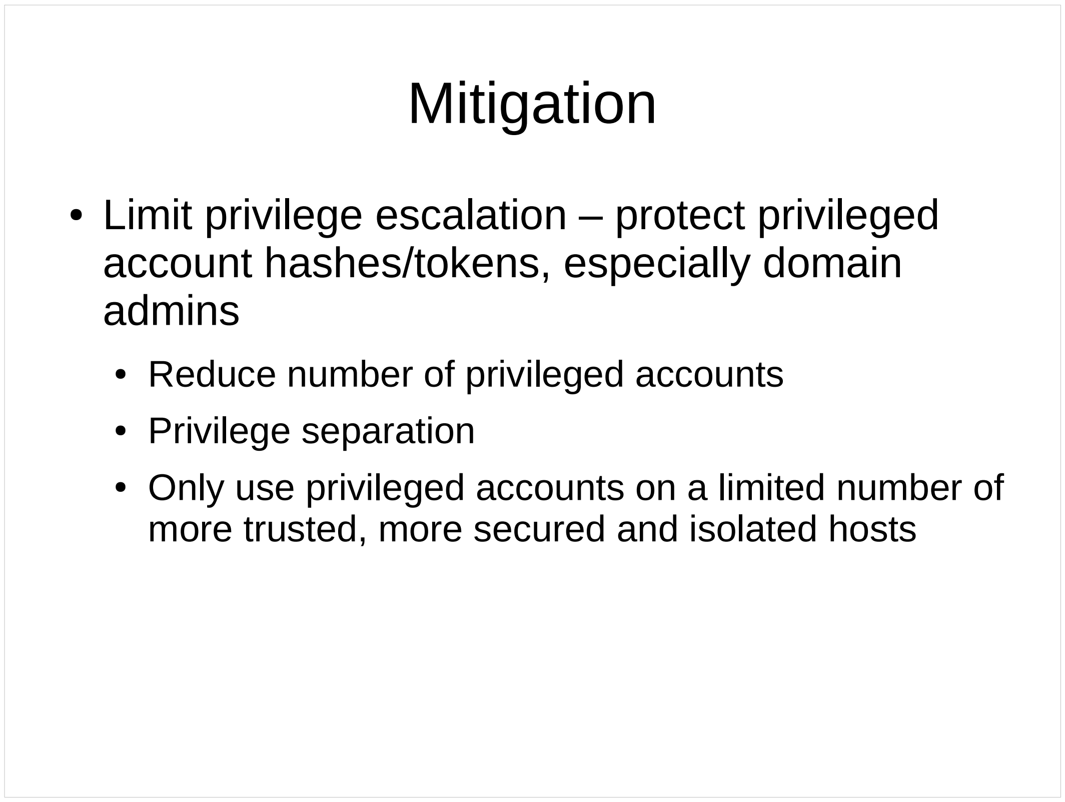
- If they do get hashes/tokens/passwords, make them useless to move around with
  - No shared passwords
  - Disable local admin accounts
  - Turn off network access to unnecessary accounts (network and RDP)



Mitigation

- Limit lateral movement
  - Client firewalls (not Windows firewall in “domain” mode)
  - Network segmentation
  - Client isolation (private VLANs)

For these types of attacks, we're talking about Windows networking ports (135-139, 445) for the most part.



Mitigation

- Limit privilege escalation – protect privileged account hashes/tokens, especially domain admins
  - Reduce number of privileged accounts
  - Privilege separation
  - Only use privileged accounts on a limited number of more trusted, more secured and isolated hosts

LAST SLIDE

References and links

- Microsoft "Mitigating Pass-the-Hash (PtH) Attacks and Other Credential Theft Techniques" paper: http://www.microsoft.com/en-us/download/details.aspx?id=36036
- SANS info on incident responding safety: http://computer-forensics.sans.org/blog/2012/02/21/protecting-privileged-domain-account-safeguarding-password-hashes
- More good SANS info: http://computer-forensics.sans.org/blog/2012/03/21/protecting-privileged-domain-accounts-access-tokens
- All the fgdump, Medusa, Praeda, OWNR and other awesomeness you could ever hope for: http://www.foofus.net
- Windows Credential Editor (formerly Pass The Hash Toolkit): http://www.ampliasecurity.com/research/wcefaq.html
- Mimikatz: http://blog.gentilkiwi.com/mimikatz
- Skip Duckwell's PTH blog: http://passing-the-hash.blogspot.com/

The end

foofus.net

The Danger Is Real

Special thanks to the fine folks at foofus.net!

None of the information presented here is original work, it's all stuff other people have figured out. Too many to thank.