Password-based Attacks

Valid Accounts (I)

- Adversaries may obtain and abuse credentials of existing accounts as a means of gaining:
  - **Initial Access**
  - **Persistence**
    - may grant persistent access to remote systems and externally available services, such as VPNs, Outlook Web Access, network devices, and remote desktop.
  - ...

Valid Accounts (II)

- Adversaries may obtain and abuse credentials of existing accounts as a means of gaining:
  - **Initial Access**
  - **Persistence**
  - **Privilege Escalation**
    - may grant **increased privilege to specific systems** or **access to restricted areas** of the network.
  - **Defense Evasion**
    - may be used to **bypass access controls** placed on various resources on systems within the network

  (and **Lateral Movement** through Remote Services)

- **Not only Initial Access!**

Valid Accounts (III)

- **Very attractive to Attackers**
  - Exploits may not be available
  - Exploits may not be reliable
  - Exploits may have to be used as little as possible (to not risk **disclosing** their existence)

  - Detecting **legitimate access** with **malicious purposes** is **more difficult** than detecting malware, attack tools, exploits

Obtaining Passwords

- **MANY** techniques in **multiple** tactics

- **Reconnaissance**
  - Phishing for Information (fake login pages)
  - Gather Victim Identity Information (data breaches, purchase)
- **Credential Access**
  - **17** techniques (with many sub-techniques)

> ## Credential Access
>
> The adversary is trying to steal account names and passwords.
>
> Credential Access consists of techniques for stealing credentials like account names and passwords.

Credential Access in a nutshell

- **Online guessing**
  - Systematically **try to guess** PWD on the service
- **Stealing**
  - **Cleartext**
    - PWD stolen
  - **Not** in Cleartext
    - **Non-invertible function** of PWD stolen
    - **Not** ready for immediate use: further **offline** attack techniques necessary for **attempting** to obtain PWD

Online Guessing

Brute Force

Brute Force (4)

Password Guessing

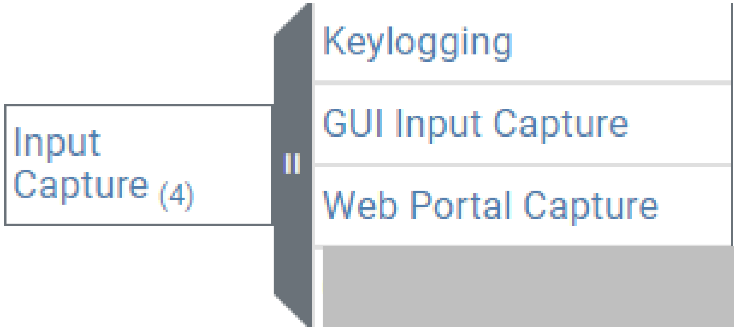
[redacted]

Password Spraying

Credential Stuffing

Adversaries may use brute force techniques to gain access to accounts when passwords are unknown [redacted] Without knowledge of the password for an account or set of accounts, an adversary may systematically guess the password using a repetitive or iterative mechanism. Brute forcing passwords can take place via interaction with a service that will check the validity of those credentials [redacted]

Stealing: Cleartext (I)



During normal system usage, users often provide credentials to various different locations, such as login pages/portals or system dialog boxes.

Input capture mechanisms may be **transparent** to the user or **rely on deceiving** the user into providing input into **what they believe to be a genuine service**.

("you have a malware where you are inserting credentials")

Stealing: Cleartext (II)

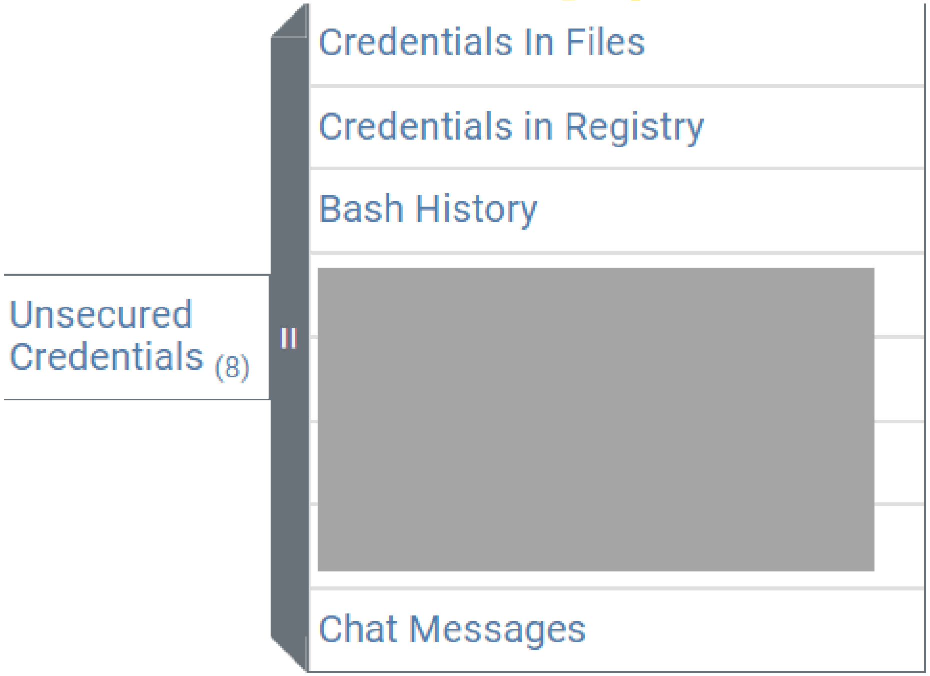
Network Sniffing

Adversaries may **sniff network traffic** to capture information about an environment

An adversary may place a network interface into promiscuous mode to passively access data in transit over the network, or use span ports to capture a larger amount of data.

Data captured via this technique may include **user credentials**, especially those sent over an **insecure, unencrypted protocol**.

Stealing: Cleartext (III)

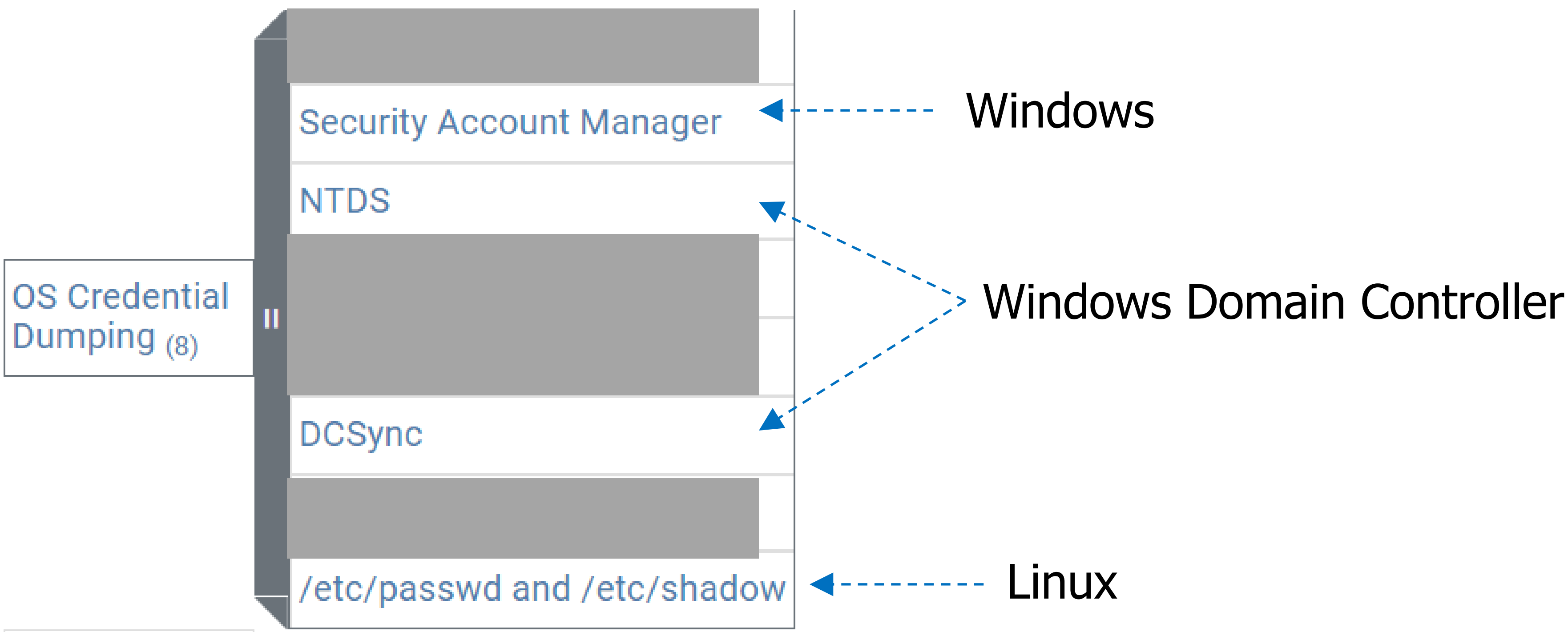


Adversaries may search compromised systems to find and obtain **insecurely stored credentials** (e.g., plaintext files).

These credentials can be stored and/or misplaced in many locations on a system

Stealing: Not in Cleartext

- Stealing **password storage** of the **o.s.**
- **Many** accounts of the same system



Nice Windows property

- **Stealing**
  - **Not** in Cleartext
    - **Non-invertible function** of PWD stolen
    - **Not** ready for immediate use: further **offline** attack techniques necessary for **attempting** to obtain PWD

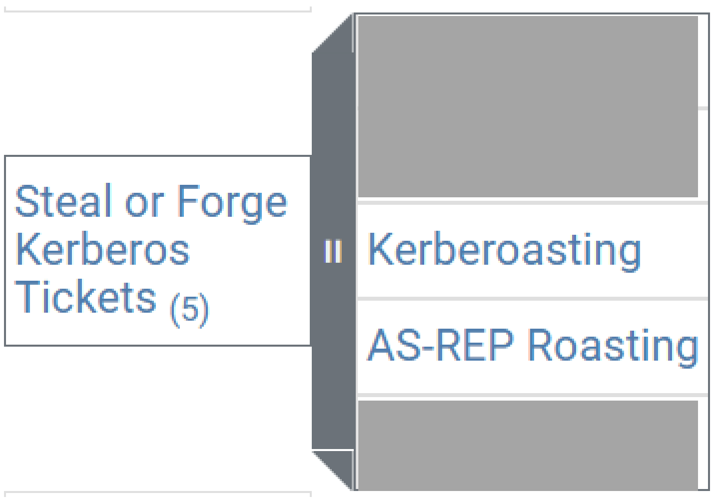
- **Windows** systems:
  - You can use the **non-invertible function** of PWD (!)
  - You do **not** need a further offline attack for **attempting** to reconstruct PWD

- Stealing password storage $\Rightarrow$ Catastrophe

Credential Access in nutshell **One more bit**

- **Stealing**
  - **Not** in Cleartext
    - Data **encrypted** with PWD
    - **Not** ready for immediate use: further **offline** attack techniques necessary for **attempting** to obtain PWD

- Very common (and very dangerous) in Windows Active Directory



Online Guessing

Online Guessing (I)

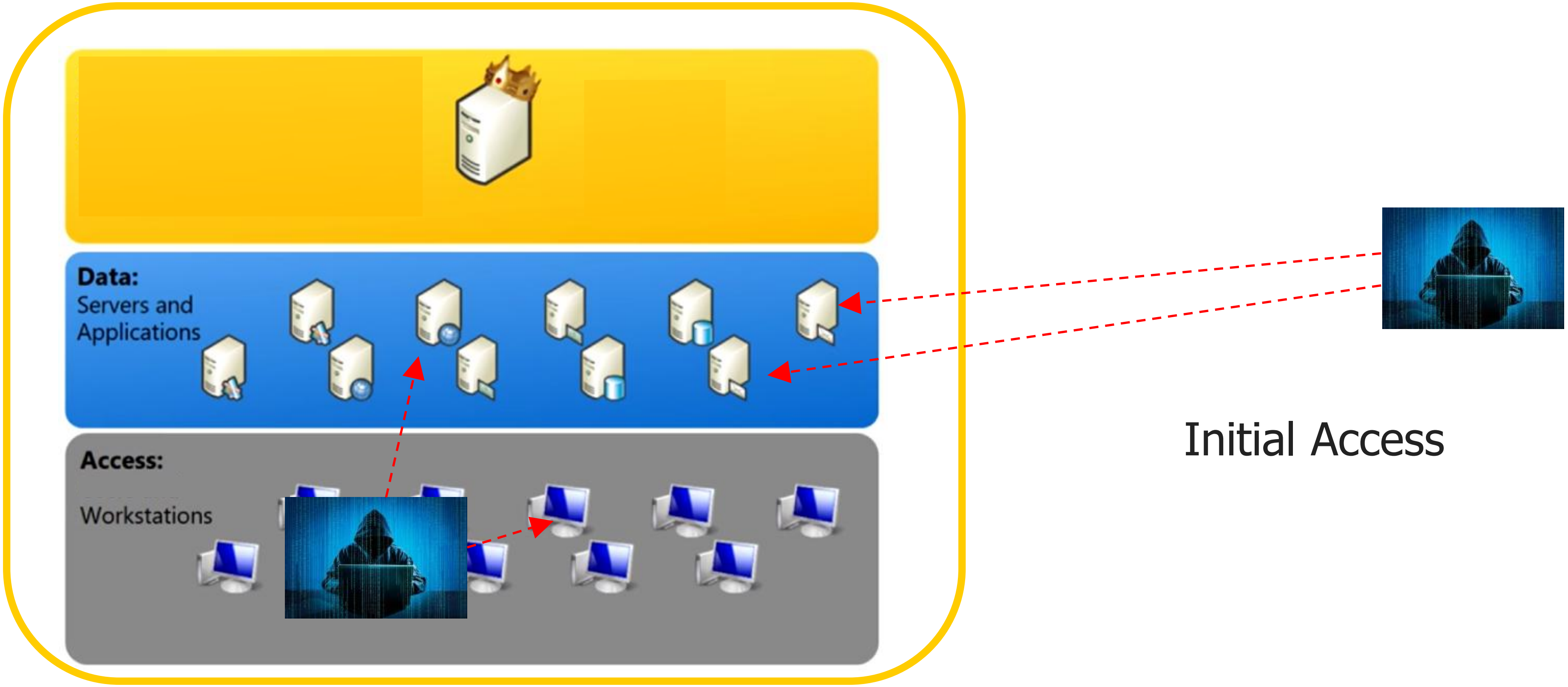
Brute Force

Brute Force (4)

- Password Guessing
- [redacted]
- Password Spraying
- Credential Stuffing

Adversaries may use brute force techniques to gain access to accounts when passwords are unknown [redacted] Without knowledge of the password for an account or set of accounts, an adversary may systematically guess the password using a repetitive or iterative mechanism. Brute forcing passwords can take place via interaction with a service that will check the validity of those credentials [redacted]

Online Guessing (II-a)



Post-compromise

Online Guessing (II-b)

Often done by
large botnets
automatically

Which Services?

- SSH (22/TCP)
- Telnet (23/TCP)
- FTP (21/TCP)
- NetBIOS / SMB / Samba (139/TCP & 445/TCP)
- LDAP (389/TCP)
- Kerberos (88/TCP)
- RDP / Terminal Services (3389/TCP)
- HTTP/HTTP Management Services (80/TCP & 443/TCP)
- MSSQL (1433/TCP)
- Oracle (1521/TCP)
- MySQL (3306/TCP)
- VNC (5900/TCP)
- SNMP (161/UDP and 162/TCP/UDP)

And email, cloud, office 365, wi-fi, routers,...

Not Targeted

- Guessing attacks are usually **not targeted**
- Password of **any account** is enough
- Usually <> Always

Sub-Technique: Password Guessing

- An adversary may guess login credentials without prior knowledge of system or environment passwords during an operation by using a list of common passwords.

- **"Foreach username, try a few thousand passwords"**
- Construct target-set
- foreach u ∈ target-set
  - foreach p ∈ candidate-password-set
    - try (u, p)

Common Passwords (I)

The 2018 **Worst Passwords of the Year** list was determined after SplashData evaluated **over 5 million passwords** that have **leaked** online in the **last year**.

1. **123456** (Unchanged)
2. **password** (Unchanged)
3. **123456789** (Up 3)
4. **12345678** (Down 1)
5. **12345** (Unchanged)
6. **111111** (New)

Common Passwords (II)

- "github default password list"

Apache-Tomcat-Default-Passwords.mdown	List with Default Apache Tomcat Credentials
IPMI-Default-Password-List.mdown	Update IPMI-Default-Password-List.mdown
Oracle-Default-Password-List.mdown	Create Oracle Default Password List
PostgreSQL-Default-Password-List.md	Create PostgreSQL-Default-Password-List.md
README.md	Update README.md
VoIP-Default-Password-List.mdown	Update VoIP-Default-Password-List.mdown
Windows-Default-Password-List.mdown	Update Windows-Default-Password-List.mdown

Common Passwords (III)

- "default password list"
- "default database password list"
- "default oracle password list"
- "administrator default password"
- "default router password list"
- ...

3COM	SuperStack II Switch	2200	Telnet	debug	synnet
3COM	SuperStack II Switch	1100/3300	Telnet	monitor	monitor
3COM	SuperStack II Switch	1100/3300	Telnet	security	security

Username	Password
SYSTEM[Foot 1]	MANAGER
SYS[Foot 2]	CHANGE_ON_INSTALL[Foot 3]
ANONYMOUS	ANONYMOUS
CTXSYS	CTXSYS
DBSNMP	DBSNMP
LBACSYS	LBACSYS
MDSYS	MDSYS
OLAPSYS	MANAGER
ORDPLUGINS	ORDPLUGINS
ORDSYS	ORDSYS
OUTLN	OUTLN
SCOTT	TIGER
WKSYS	WKSYS
WMSYS	WMSYS
XDB	CHANGE_ON_INSTALL

How many guesses?

- Hard to tell

- It depends on:
  - Detection / reaction capabilities of the target
  - Existence of alternative techniques
  - Importance of that specific target

- Maybe "no more than a few thousands per account"
- ...but maybe much less than that
  - Wannacry/Mirai propagated with very short lists

Sub-Technique: Password Spraying

- Adversaries may use a **single** or **small list** of commonly used passwords against **many different accounts**

- Construct target-set
- foreach p ∈ candidate-password-set   // **swap** loops
  - foreach u ∈ target-set
    - try (u, p)

- Usually **more efficient** and **harder to detect**

Sub-Technique: Credential Stuffing

- Adversaries may use credentials obtained from breach dumps of unrelated accounts to gain access to target accounts through credential overlap
  (same username across different organizations)

- The information may be useful to an adversary attempting to compromise accounts by taking advantage of the tendency for users to use the same passwords across personal and business accounts.

KEEP IN MIND

- Not in any dictionary
  - Not common
  - Not default
  - Not reused from a breached site
    (never use the same password on multiple sites!)

MUCH more important than

- "7 digits, 3 special symbols, 2 uppercase, ..."

Remark: Threat model

- Stealing passwords / Cleartext
  - Attacker is able to **run software on victim machine** (privilege level determines impact of techniques)

- Online Guessing
  - Attacker is able to **execute authentication protocol** with victim machine

- Online Guessing requires much smaller capabilities!

Online Guessing: Defense

Defense: Detection

1. **Detection**
   - Guessing / Stuffing
     - **Many** failed attempts at a **single** account
     - Common and "easy"
   - Spraying
     - **Few** failed attempts at a **single** account
     - ...but many failed attempts at sets of accounts
     - Not common and "difficult"

2. **Action**

Defense: Action (I)

- **Automatic account lockout**
  - Extremely dangerous: trivial avenue for denial of service!
- **Blacklist guessing IP address**
  - Easily circumvented (Botnet, TOR)
  - Potential false positives (NAT)
- **Alert toward targeted account**
  - Not feasible for Spraying
  - What to do next?

Defense: Action (II)

- Automatic username lockout
- Blacklist guessing IP address
- Alert toward targeted username coinvolti

- **Time throttling (progressive increase of response delay)**
  - Very effective
  - ...but what if your service does not have it???

  - In practice, many kinds of server software exposed on the Internet do **not** have it

Microsoft Data

- Around **0.5%** of all accounts get compromised **each month**
  - In January 2020 was about **1.2 million**

- In most cases, ... simplistic attacks
- **40% guessing/stuffing**
- **40% spraying**
  - In January 2020 was about 480.000 + 480.000

- Nearly all hacked accounts are on **legacy protocols**
  - SMTP, POP, IMAP,...

Remark

- Around **0.5%** of all accounts get compromised **each month**

- UniTS: 3000 people ⇒ 15 accounts
  - Merely by stuffing e spraying

- Threat model for organizations:
  - "Bad actors outside / Trusted zone inside"
    Completely unrealistic
  - **"Assume breach"**

Keep in mind

1. **Detection**
2. Action

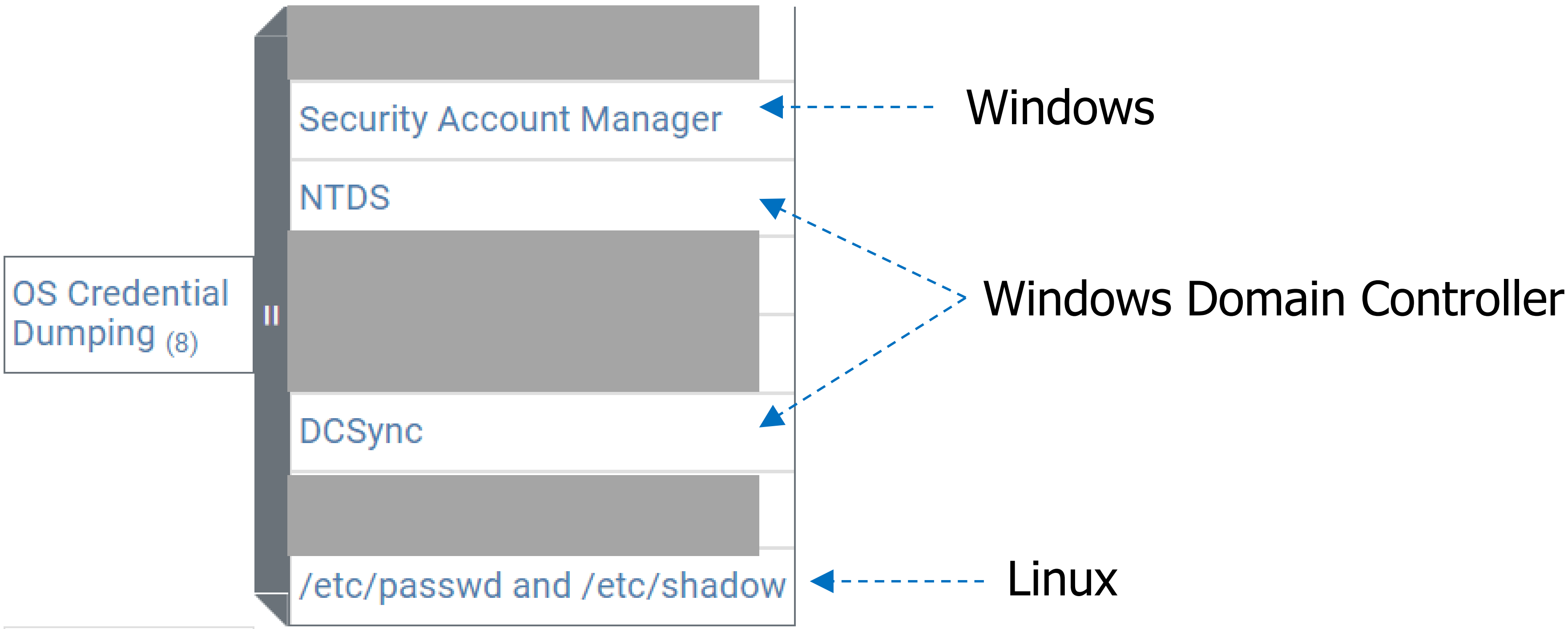
- **Absence of any "not trivial" detection logic is no longer acceptable**

- Many thousands of failed attempts to a certain account...
- ...not even one single alert???

Secure Password Storage

Stealing Password Storage

Stealing Password Storage



Full system

(many accounts of the same system)

Remark: VERY CONCRETE RISK

May 2016

111 pwned websites

977,283,532 pwned accounts

September 2016

129 pwned websites

1,388,845,883 pwned accounts

Top 10 breaches

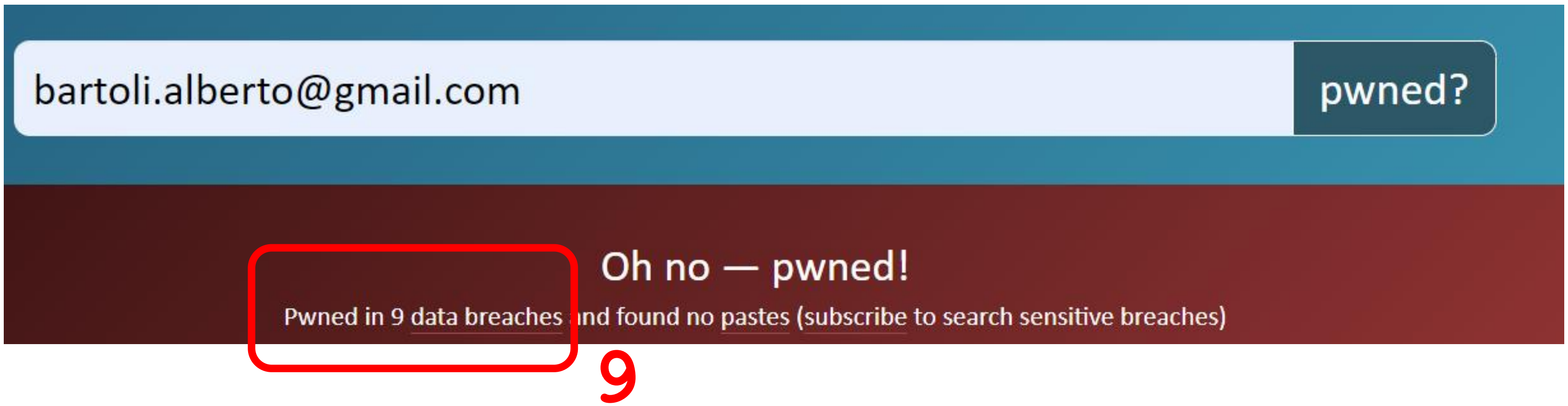
359,420,698	MySpace accounts
164,611,595	LinkedIn accounts
152,445,165	Adobe accounts
65,469,298	tumblr accounts
40,767,652	Fling accounts
30,811,934	Ashley Madison accounts
27,393,015	Mate1.com accounts
13,545,468	000webhost accounts
13,186,088	R2Games accounts
8,243,604	Gamigo accounts

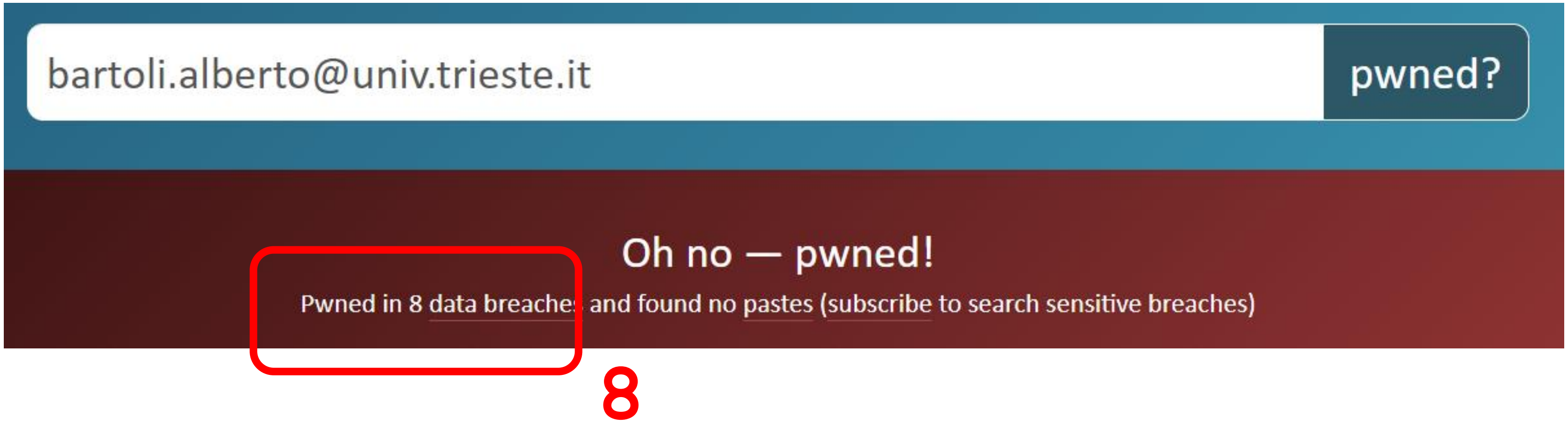
Sensitive breach, not publicly searchable

359,420,698	MySpace accounts
164,611,595	LinkedIn accounts
152,445,165	Adobe accounts
112,005,531	Badoo accounts
93,338,602	VK accounts
68,648,009	Dropbox accounts
65,469,298	tumblr accounts
49,467,477	iMesh accounts
40,767,652	Fling accounts
30,811,934	Ashley Madison accounts

https://haveibeenpwned.com/

This is me (as of January 29-th 2021)





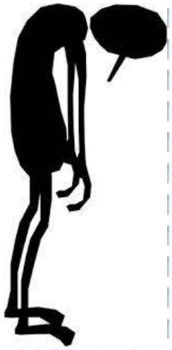
Keep in mind

- Never ever deploy a server with passwords stored in **cleartext**

- Attacker that manages to steal password storage

- Catastrophe

Bad Example Server (Stolen)



id	presidenza	pwd	name	note	file	news	admin	key	eventi	keywords	sections	emailaddress
3355	0	1234										
3355	0	andrea1										
3355	0	password41										
3355	0	password53										
		password55										
		password10										
		15crs001										
		PASSWORD05										
		password03										
		password34										
		password58										
		Antonietta31										
		ducati996										
		893ZqYy99R										
		1234										
3355	0	attilio										
3355	0	1810										
3355	0	crsbd										
3355	0	password30										
3355	0	password46										
3355	0	studio53										

https://bartoli-alberto.blogspot.com/2018/11/perche-la-password-deve-essere.html

Bad Example: Tomcat web server DEFAULT

```xml
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
    <role rolename="tomcat"/>
    <role rolename="role1"/>
    <user username="tomcat" password="tomcat" roles="tomcat"/>
    <user username="both" password="tomcat" roles="tomcat,role1"/>
    <user username="role1" password="tomcat" roles="role1"/>
</tomcat-users>
```

Common scenario for developer:

- Creates web site with protected portion
- Stores credentials in a database associated with the web server
- Defines format of database table with passwords

- ...Storing passwords "as they are" is the easiest thing to do

Secure Password Storage

Secure Password Storage

- Secure Password Storage: does **not** contain passwords in cleartext

- We need to understand:
  - How passwords are **represented** in the storage

  - How the storage is **used** in **normal** operation
    - The service receives the account password
    - How to check its validity?

  - How attackers use **stolen** storage

Hashed (I-a)

AuthDB

```
...
alberto       1e69e0a615e8cb797d75d4f08bdc2f56
eric          6163aabee5d4f08bdc615e8cb797d72b
...
```

- < account, Credentials > for each authorized account
- Credentials may be:
  1. **Hashed** < acc-x, H(pwd-acc-x) >
  2. ...

- H() is a **non reversible** function (**hash**)

Hashed (I-b)

```
                    AuthDB
...
alberto       1e69e0a615e8cb797d75d4f08bdc2f56
eric          6163aabee5d4f08bdc615e8cb797d72b
...
```

- Service:
  1. Receive acc-x, pwd-acc-x
  2. Compute H(pwd-acc-x)
  3. Check H(pwd-acc-x) $\in$ AuthDB

Key Advantage

```
                    AuthDB
...
alberto       1e69e0a615e8cb797d75d4f08bdc2f56
eric          6163aabee5d4f08bdc615e8cb797d72b
...
```

- Adversary that manages to **steal** secure password storage does **not** have the **password**
- Some **additional** and **different** attack is needed

Defense in Depth

- **Fundamental** principle
- **Multiple** and **independent** defensive layers
- Of course, without increasing defensive cost linearly in the number of layers
- Good defensive tool:
  - DefenseCost-Increase **<<** AttackCost-Increase

Hashed (II)

```
                    AuthDB
...
alberto       1e69e0a615e8cb797d75d4f08bdc2f56
eric          6163aabee5d4f08bdc615e8cb797d72b
...
```

- **Requirements** on function H():
  1. H(x) is **constant** length (128-256 bit)
  2. H(x) does **not** provide **any** information on x ("cryptographically secure hash function")
  3. *...later...*

Hashed (III)

- Common for:
  - **Operating systems**
    - Local accounts, stored in every machine
  - **Large organizations**
    - Domain accounts, stored in domain controller
  - **Application servers**
    - Stored in database table

Example: Windows SAM (local accounts)

Username : Security identifier : HASH_1(PWD) (no longer used) : **HASH_2 (PWD)**

```
Administrator:500:aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f71d913c245d35b50b:::
alberto:1019:aad3b435b51404eeaad3b435b51404ee:7a21990fcd3d759941e45c490f143d5f:::
anakin_skywalker:1011:aad3b435b51404eeaad3b435b51404ee:c706f83a7b17a0230e55cde2f3de94fa:::
artoo_detoo:1007:aad3b435b51404eeaad3b435b51404ee:fac6aada8b7afc418b3afea63b7577b4:::
ben_kenobi:1009:aad3b435b51404eeaad3b435b51404ee:4fb77d816bce7aeee80d7c2e5e55c859:::
boba_fett:1014:aad3b435b51404eeaad3b435b51404ee:d60f9a4859da4feadaf160e97d200dc9:::
chewbacca:1017:aad3b435b51404eeaad3b435b51404ee:e7200536327ee731c7fe136af4575ed8:::
c_three_pio:1008:aad3b435b51404eeaad3b435b51404ee:0fd2eb40c4aa690171ba066c037397ee:::
darth_vader:1010:aad3b435b51404eeaad3b435b51404ee:b73a851f8ecff7acafbaa4a806aea3e0:::
greedo:1016:aad3b435b51404eeaad3b435b51404ee:ce269c6b7d9e2f1522b44686b49082db:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
han_solo:1006:aad3b435b51404eeaad3b435b51404ee:33ed98c5969d05a7c15c25c99e3ef951:::
jabba_hutt:1015:aad3b435b51404eeaad3b435b51404ee:93ec4eaa63d63565f37fe7f28d99ce76:::
jarjar_binks:1012:aad3b435b51404eeaad3b435b51404ee:ec1dcd52077e75aef4a1930b0917c4d4:::
kylo_ren:1018:aad3b435b51404eeaad3b435b51404ee:74c0a3dd06613d3240331e94ae18b001:::
lando_calrissian:1013:aad3b435b51404eeaad3b435b51404ee:62708455898f2d7db11cfb670042a53f:::
leia_organa:1004:aad3b435b51404eeaad3b435b51404ee:8ae6a810ce203621cf9cfa6f21f14028:::
luke_skywalker:1005:aad3b435b51404eeaad3b435b51404ee:481e6150bde6998ed22b0e9bac82005a:::
sshd:1001:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
sshd_server:1002:aad3b435b51404eeaad3b435b51404ee:8d0a16cfc061c3359db455d00ec27035:::
vagrant:1000:aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f71d913c245d35b50b:::
```

Example: Application Server (Stolen)

password	userid	username
e9c17a2a22d7d8243c6eea17c8b192cc		
36ca45f981dc4643324744c3a0ea8ca8		
bc7ca5f24d00e681cb76b3e33dbae286		
56ef309697ca53a9c2d1b9cfdbd3d441		
4bf60591e1aaabc68e3ed1cde7c80bba		
97c14c073cc07c382b2597d80430afb8		
822dd494b3e14a82aa76bd455e6b6f4b		
037c70dbc1c812f6b2091688804d7b17		
91d78e7024a53864566c4ffdb423758f		
2b35ee7be359ca322583e8a66368d953		

https://bartoli-alberto.blogspot.com/2018/11/perche-la-password-deve-essere.html

Hashed and Salted (I-a)

```
                         AuthDB

...
alberto     1e69e0a615e8cb797d75d4f08bdc2f56  6623780aa33b
eric        6163aabee5d4f08bdc615e8cb797d72b  77635aabbce0
...
```

- < account, Credentials > for each authorized account
- Credentials may be:
  1. **Hashed** < acc-x, H(pwd-acc-x) >
  2. **Hashed and Salted** < acc-x, salt-x, H(concat(pwd-acc-x,salt-x) >
  3. ...

- H() is a **non reversible** function (**hash**)
- salt-x is a random number (need **not** be secret)

Hashed and Salted (I-b)

```
                         AuthDB
...
alberto       1e69e0a615e8cb797d75d4f08bdc2f56
eric          6163aabee5d4f08bdc615e8cb797d72b
...
```

- Service:
  1. Receive acc-x, pwd-acc-x
  2. Extract salt-x
  3. Compute concat(pwd-acc-x,salt-x)
  4. Compute H(concat(pwd-acc-x,salt-x))
  5. Check H(concat(pwd-acc-x,salt-x)) $\in$ AuthDB

Hashed and Salted (II)

- For reasons analyzed **later**:
  - **More secure** than Hashed
    (in case stolen by Attacker)
  - **Cannot** be used in Windows O.S.
    (thus not even in Domain Controllers!)

- **In practice:**
  - Local Accounts Windows: Hashed
  - Local Accounts Other O.S.: Hashed and Salted
  - Domain Accounts: Hashed

  - Application Servers: Can use either

Example: Linux `/etc/shadow`

`linuxize:$6$zHvrJMa5Y690smbQ$z5zdL...:18009:0:120:7:14::`

account

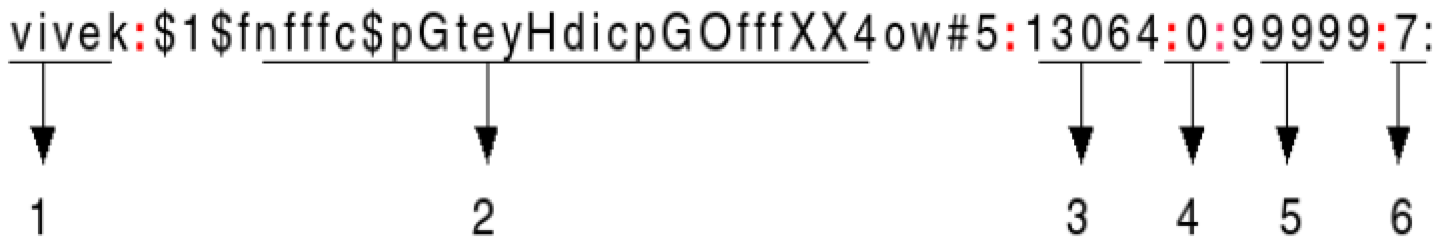
algorithm id

`salt`

`F(concat(pwd-acc-x,salt-x)`

truncated for readability

Confusing Terminology



2. **Password** : Your encrypted password is in hash format.

It is **not** "encrypted"!

There is no "decryption key" capable of recovering the plaintext

Secure Storage

Secure Storage

- For **equality check**
  - Password storage
  - How: Hashing (with or without Salting)

- For **content recall**
  - Password managers
  - Encrypted file systems (e.g., smartphones)
  - Stores of cryptographic keys
  - …
  - **How: ?**

Secure Storage for Recall (outline)

- Storage **creation**
  - Choose `P, salt`
  - Store `H(concat(P,salt)), salt`
- Storage **access**
  - Ask password `P'` to operator
  - IF `H(concat(P',salt)) =`
  - THEN encrypt/decrypt with `K = KeyDerivation(H(P),salt)`
  - ELSE abort
- Adversary with storage access:
  - Cannot compute `K`

Very common need

- Process starts unattended (**no human operator**)
- Process needs to **authenticate** against another machine
(it needs to prove knowledge of a certain `PWD-P`)

- **Very common scenario**
  - Process associated with a Computer or Service account
  - Process **unattended** must execute a **network** logon
  (by proving knowledge of `PWD-P`)

Hhmmm...

- Process starts unattended (**no human operator**)
- Process needs to **authenticate** against another machine (it needs to prove knowledge of a certain `PWD-P`)

- Process must read `PWD-P` somewhere
  - Can you store `PWD-P` on a text file?
  - Can you embed `PWD-P` in the process executable?

  - Do you need **secure storage**? for equality check or for recall?
  - Can you use the previous mechanism?
  - If not, where and how do you store the service password?

Offline Guessing (Password cracking)

Remark: Windows

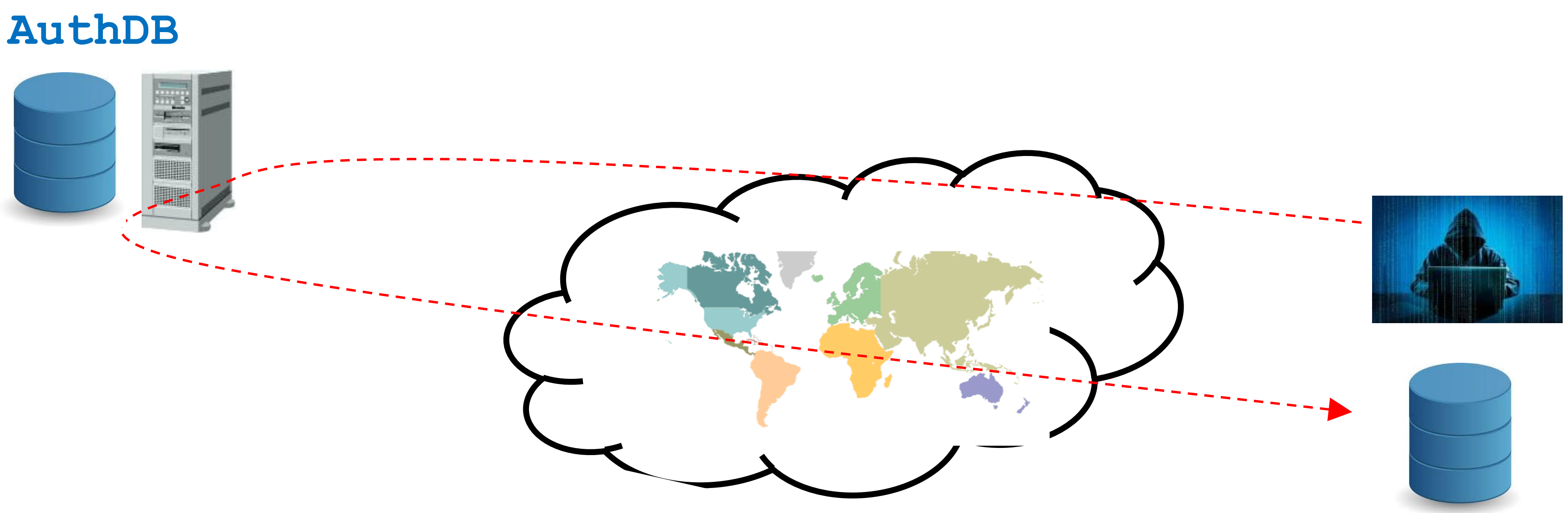
- Knowledge of H(pwd-U) suffices to impersonate U (details later)

↓

- Stealing of SAM / NTDS is a complete catastrophe

- Offline guessing required only if cleartext password was required for some reason
  - Example: access to esse3 / eduroam
    (without changing password from other Windows services)

Offline Guessing (Password cracking) (I)



Offline Guessing (Password cracking)

Brute Force

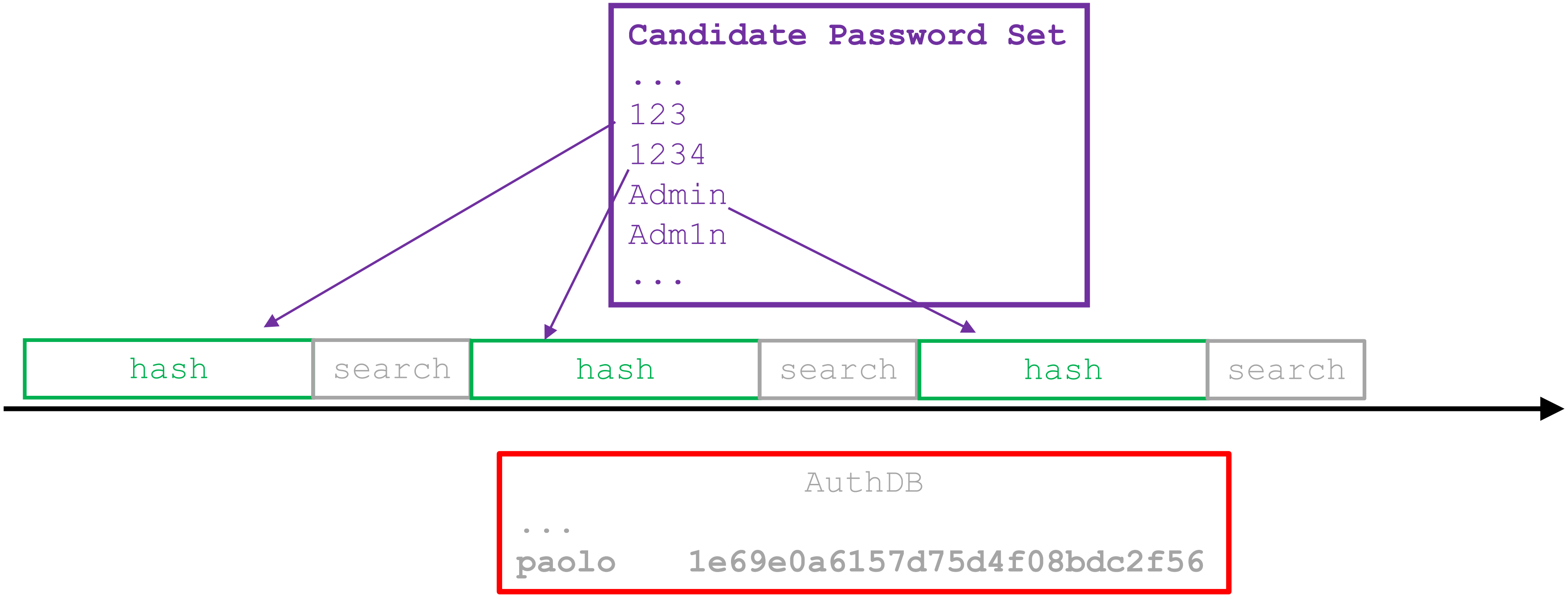
Brute Force (4)

- Password Guessing
- Password Cracking
- Password Spraying
- Credential Stuffing

Adversaries may use brute force techniques to gain access to accounts when passwords are unknown or when password hashes are obtained. Without knowledge of the password for an account or set of accounts, an adversary may systematically guess the password using a repetitive or iterative mechanism. Brute forcing passwords can take place via interaction with a service that will check the validity of those credentials or offline against previously acquired credential data, such as password hashes.

Offline Guessing (Password cracking) (II)

- foreach p ∈ candidate-password-set
  - x := H(p)
  - searchAndAddFound(x,AuthDB)



Cracking tools

AuthDB
User **H(..)**

list of
User pwd-User

CRACKING TOOL

```
candidate-password-set
...
```

- foreach p ∈ candidate-password-set
  - **x** := H(p)
  - searchAndAddFound(x,AuthDB)

```
google "John the Ripper"
google "Hashcat"
```

Hash function Requirements

- Requirements on function H():
  1. H(x) is **constant** length (128-256 bit)
  2. H(x) does **not** provide **any** informaton on x ("cryptographically secure hash function")
  3. H(x) is computationally **very heavyweight**

- State of the art: **PBKDF2** or **bcrypt**
- Computational weight can be **parameterized**
  - PBKDF2: "iterations"
  - bcrypt: "rounds"

How many guesses? (I)

- `#guesses =   #guesses/sec * time_invested`

- `#guesses/sec` depends on:
  - Available hardware
  - Hash function complexity

- `time_invested` depends on:
  - Available resources (time)
  - Fixation on that specific target

How many guesses? (II)

- `#guesses/sec` depends on:
  - Available hardware ?
  - Hash function complexity Known

- `time_invested` depends on:
  - Available resources (time) ?
  - Fixation on that specific target ?

- **Very hard to answer** (too many "unknowns")
- Realistic estimates later

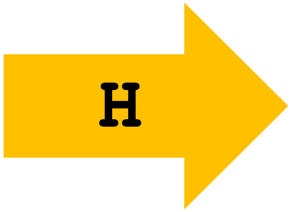
Salting

Lookup table (I)

- `foreach p ∈ candidate-password-set`
  - `x := H(p)`
  - `searchAndAddFound(x,AuthDB)`

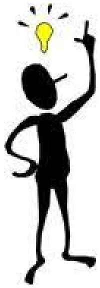
- H(p) is **computed while iterating**
- Why computing `H(p)` **each time** someone wants to try `p`?
- We could compute `H(p)` for every `p` **in advance:** iteration would be **much faster**

```
...
123
1234
Admin
Adm1n
...
```



Lookup Table

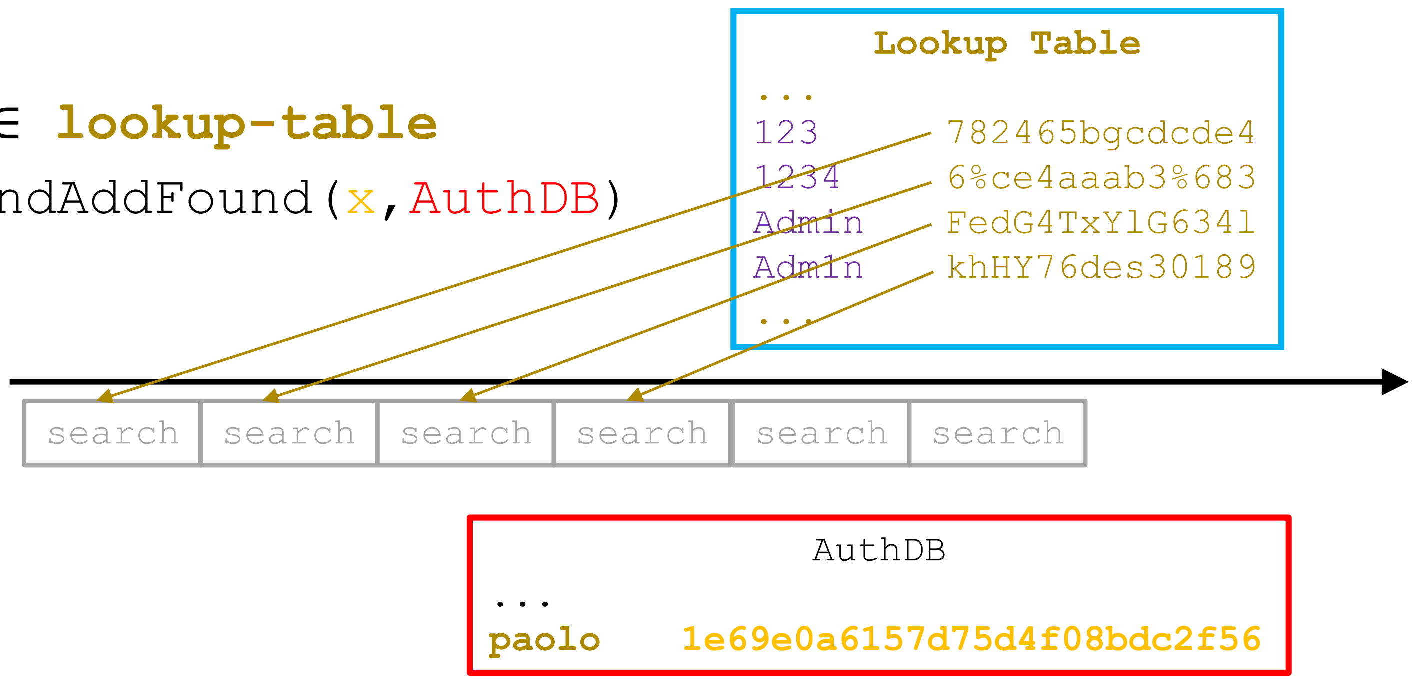
```
...
123      782465bgcdcde4
1234     6%ce4aaab3%683
Admin    FedG4TxY1G6341
Adm1n    khHY76des30189
...
```



Lookup table (II)

- **Lookup Table** ("Password Hash Dictionary")
  - Computed **in advance**
  - Valid for **any** AuthDB (that uses the corresponding hash algorithm)
  - Widely available on the Internet

- `foreach` **x** ∈ **lookup-table**
  - `searchAndAddFound(x,AuthDB)`



Hashed Format: Weakness 1

```
...
pippo                    H(12345)
```

```
...
carlo                    H(12345)
...
mario                    H(12345)
```

Hashes
can be computed
in advance,
once and for all

```
   Lookup Table
...
123        H(123)
1234       H(1234)
12345      H(12345)
Admin      H(Admin)
Adm1n      H(Adm1n)
...
```

- Offline guessing is merely a set of **searches**
- **No computation at all**

Hashed Format: Weakness 2

Hash count in given (stolen) AuthDB

HashedPasswd	occurrences
0xC31AC605793F580B386C0FB53F1B9775	223
0xB081DBE85E1EC3FFC3D4E7D0227400CD	220
0x6E9B3A7620AAF77F362775150977EEB8	212
0xC8DE1FC2BEEC2D3BFF33A75C2A317604	201
0x0F037584C99E7FD4F4F8C59550F8F507	198
0x19A2854144B63A8F7617A6F225019B12	194
0xF940A336C133D116F954DC32376D5D86	193
0x43DA75B94F8F4560177AF75F42D784EB	191
0x29AC25660E3078E87E3097D3822E50D7	184
0xCE0BFD15059B68D67688884D7A3D3E8C	184
0xCE0E51D7856FD8BF50145848EE0CD973	1
0xCE0EDFA980805551FB0981D649A3DF8F	1

- **Password Frequency Analysys possible**

- Frequent hashes = Common passwords
- Probably those users are not good at security...
- Probably those passwords are in a dictionary

- Better focus on those hashes first

Hashed and Salted (REMIND)

```
                         AuthDB

...
alberto      1e69e0a615e8cb797d75d4f08bdc2f56  6623780aa33b
eric         6163aabee5d4f08bdc615e8cb797d72b  77635aabbce0
...
```

- < account, Credentials > for each authorized account
- Credentials may be:
  1. **Hashed** < acc-x, H(pwd-acc-x) >
  2. **Hashed and Salted** < acc-x, salt-x, H(concat(pwd-acc-x,salt-x) >
  3. ...

- H() is a **non reversible** function (**hash**)
- salt-x is a random number (need **not** be secret)

Weakness 1: Solved

```
...
pippo    7685491327459    H(123457685491327459)
```

```
...
carlo    1332409100226    H(123451332409100226)
...
mario    2242330090650    H(123452242330090650)
```

- **More secure** than Hashed (in case stolen by Attacker)

- Every account has a **random** salt:
  ⇒ hashes **cannot** be computed in advance

- Offline guessing requires **computation**
- With a given amount of resources, Attacker can make **less guesses**

Weakness 2: Solved

HashedPasswd	occurrences
0xC31AC605793F580B386C0FB53F1B9775	223
0xB081DBE85E1EC3FFC3D4E7D0227400CD	220
0x6E9B3A7620AAF77F362775150977EEB8	212
0xC8DE1FC2BEEC2D3BFF33A75C2A317604	201
0x0F037584C99E7FD4F4F8C59550F8F507	198
0x19A2854144B63A8F7617A6F225019B12	194
0xF940A336C133D116F954DC32376D5D86	193
0x43DA75B94F8F4560177AF75F42D784EB	191
0x29AC25660E3078E87E3097D3822E50D7	184
0xCE0BFD15059B68D67688884D7A3D3E8C	184
0xCE0E51D7856FD8BF50145848EE0CD973	1
0xCE0EDFA980805551FB0981D649A3DF8F	1

- Every account has a **random** salt:
  ⇒ Password frequency analysis is **not** possible

Offline guessing Hashed and Salted

```
foreach user ∈ AuthDB
    foreach p ∈ candidate-password-set
        x := H(concat(p,user.salt))
        IF x == user.H THEN p is ok
```

CRACKING TOOL

- One could swap user/password loops...
- ...but hashes have to be computed anyway

- With the same amount of resources, much less guesses

Keep in mind

```
foreach p ∈ candidate-password-set
    x := H(p)
    searchAndAddFound(x,AuthDB)
```

- **Search** on **all** the accounts

```
foreach user ∈ AuthDB
    foreach p ∈ candidate-password-set
        x := H(concat(p,user.salt))
        IF x == user.H THEN p is ok
```

- **Compute** hash for **one** account

Guessing Attacks Today

Offline guessing: John the Ripper (I)

- "hundreds" of hash formats
- Usually it detects the correct one automatically

- `john --wordlist=`**`candidate_pwd_list`**` hash_list`

```
┌──(kali㉿kali)-[~]
└─$ john --wordlist=/usr/share/wordlists/rockyou.txt mysql-wpusers-hashes.txt
```

Offline guessing: John the Ripper (II)

- "hundreds" of hash formats
- Usually it detects the correct one automatically

- john --wordlist=**candidate_pwd_list** hash_list

```
┌──(kali㉿kali)-[~]
└─$ john --wordlist=/usr/share/wordlists/rockyou.txt mysql-wpusers-hashes.txt
Using default input encoding: UTF-8
Loaded 4 password hashes with 4 different salts (phpass [phpass ($P$ or $H$) 128/128 SSE2 4×3
])
Cost 1 (iteration count) is 8192 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
manager          (manager)
vagrant          (vagrant)
2g 0:00:16:11 28.58% (ETA: 06:34:19) 0.002059g/s 4392p/s 8908c/s 8908C/s richh..richfield1
Use the "--show --format=phpass" options to display all of the cracked passwords reliably
Session aborted

┌──(kali㉿kali)-[~]
└─$ 
```

Guessing Attacks Today (I-a)

Three phases:

1. **Dictionaries**
   - Default / Common / Predictable patterns
   - **Previous breaches at other sites**
   - Word lists

2. ...
3. ...

Guessing Attacks Today (I-b)

- They start by taking the >500M passwords which have been disclosed...Think of this as “**every password anyone has ever thought of, ever.**”

- ...build a list of all popular phrases, song lyrics, news headlines, whatever they can think of to pick up from search engines, Wikipedia, popular articles, etc.

- **These are available pre-canned** in the hash-breaker communities.

Microsoft

Rough Indications (I)

- They start by taking the >500M passwords which have been disclosed...Think of this as "**every password anyone has ever thought of, ever.**"
- This will break **>70%** of user passwords

- ...build a list of all popular phrases, song lyrics, news headlines, whatever they can think of to pick up from search engines, Wikipedia, popular articles, etc.
- This may pick up another **5-7%** of user passwords

Microsoft

Rough Indications (II)

- They start by taking the >500M passwords which have been disclosed...Think of this as “**every password anyone has ever thought of, ever.**”
- This will break **>70%** of user passwords

- State of the art HW 2020:
  - **Full dictionary** against 200 accounts / sec

⇒ Most accounts will fall **instantly**

Microsoft

Remark 1

SecLists / Passwords / Common-Credentials / 10-million-password-list-top-1000.txt

```
21    qwertyuiop
24    1234567890
95    987654321
120   q1w2e3r4t5
477   789456123
518   minecraft
557   metallica
```

- **Long <> hard to guess**
- Long passwords can be in dictionaries

Guessing Attacks Today (II)

Three phases:

1. **Dictionaries**
   - Default / Common / Predictable patterns
   - **Previous breaches at other sites**
   - Word lists

2. **Mangling rules** (applied to dictionary)
   - Append (and/or prepend) a special char
   - Replace o (and/or O) with 0
   - ...

3. ...

Guessing Attacks Today (III)

Three phases:

1. Dictionaries
2. Mangling rules
3. All **permutations** required by target password policy:
   - Symbol-set S1, Password length P1, P2, P3,...,PN
     Example: digits 0-9
   - Symbol-set S2, Password length P1, P2, P3,...,PN
     Example: digits 0-9 Lowercase/Uppercase letters
   - ...

Guessing Attacks Today (IV)

1. Dictionaries
2. Mangling rules — **Rarely** used
3. Permutations of symbols — **Even more rarely** used

- Either phase 1 is enough
  or
- Adversary changes attack **technique / target**

- Of course, not all Adversary have the same behavior...

KEEP IN MIND (REMIND)

- Not in any dictionary
  - Not common
  - Not default
  - Not reused from a breached site
    (never use the same password on multiple sites!)

MUCH more important than

- "7 digits, 3 special symbols, 2 uppercase, ..."

That's why...(I)

- **Do not use complexity requirements**

- It is a **poor defence** against guessing attacks.
- It places an extra burden on users, many of whom will use predictable patterns (such as replacing the letter ‘o’ with a zero) to meet the required 'complexity' criteria. Attackers are familiar with these strategies and **use this knowledge to optimise** their attacks.

- You should specify a minimum password length, to prevent very short passwords from being used.

That's why...(II)

COMPUTER SECURITY

NIST
National Institute of Standards and Technology
U.S. Department of Commerce

NIST Special Publication 800-63B

- Memorized secrets should be at least 8 characters in length...
- If the verifier **disallows** a chosen memorized secret **based on its appearance on a blacklist of compromised values**, the subscriber should be required to choose a different memorized secret.
- **No other complexity requirements** for memorized secrets should be imposed.

Remark 2 (I)

- Webapp analyzes inserted password and:
  1. Determines symbol-set
  2. Determines length
  3. Computes #permutations
  4. Estimate time for trying them all

Remark 2 (II)

"Great, I can use that password on all sites"

- Password strength meters are often **misleading!**

- Adversaries start from dictionaries: **not** from permutations!
- A password in a dictionary will be found immediately, irrespective of its length/complexity

How many guesses?

- `#guesses =   #guesses/sec * time_invested`
- **Very hard to answer** (too many "unknowns")
- Our path:
  1. Realistic `#guesses/sec`
     (state of the art password cracking hw 2019)
  2. Realistic passwords that probably resist
     (estimates..later)

Guessing speed

- Realistic `#guesses/sec`
  (state of the art password cracking hw 2019)

	#hash/sec		Relative		
NTLM	715.6	G	1.4	M	Windows SAM/NTDS
MD5	391.2	G	0.8	M	Common reference
NetNTLMv2	27795	M	53.9	K	Windows NTLM protocol
Kerberos 5, etype 23, AS-REQ Pre-Auth	7191.1	M	13.9	K	
LastPass + LastPass sniffed (Iterations: 499)	39651.7	K	76.9		Password manager
WPA-EAPOL-PBKDF2 (Iterations: 4095)	6120.2	K	11.9		Enterprise WIi-Fi (eduroam)
bcrypt $2*$, Blowfish (Unix) (Iterations: 32)	515.7	K	1.0		Hashed and Salted Linux storage

Passwords that probably resist (I)

- Passwords that will "certainly" resist:
  - **Not in any dictionary**
  - Symbol-set easily typable chars (96)
  - Longer than 10 characters

- NB: cannot be proved

Passwords that probably resist (II)

- Look at "Hive systems passwords table" in companion website

- How long as a function of password length
  - Not in any dictionary: MD5 (similar to Windows SAM)
  - Not in any dictionary: bcrypt
  - In some dictionary

Passwords that probably resist (III)

- State of the art HW 2020

- Assuming 96 easily typable characters:
  - Any **8 characters** password can be broken in **a few days**
  - Any additional character multiplies number of attempts by 96
    ⇒ **9 characters require a few months** (probably practical limit)

  - 8 chars→2^52
  - 9 chars→2^60
  - 10 chars→2^66

Microsoft

Hacking Lab: Metasploitable3 Demo 3