May 10, 2018

The Art of Detecting Kerberoast Attacks

Written by Ben Mauch

Share

As a former defender, there is a sense of “happiness” when I can put defenses in place that allow you to detect attacks and potential indicators of compromise (IoC). It's like those old spy toys you would get as a kid that had the "laser" light and would make a sound if the light beam was tripped. I felt so powerful because I had an early warning system when someone entered my room. In the realm of defensive controls, early warning detections are key. If you can gain insight into a potential IoC or active attack, you can engage incident response (IR) procedures in a proactive state, instead of a reactive state. Often, this means isolation of the affected user account or system which reduces exposure and eliminates the threat. Unfortunately, for some of the latest attack[SKIP TO MAIN CONTENT] more difficult. In 2016, several blog posts and

articles were published around polling Service Principal Name (SPN) accounts and the associated tickets. This attack was named "Kerberoasting". If an attacker had a single valid user account and password, they could pull down the SPN tickets and attempt to crack them offline. The real issue here was that the defense against it was extremely limited. What makes Kerberoasting great for the attacker is that the technique isn't breaking anything and technically it is not exploiting any part of the Kerberos process. The technique is using Kerberos exactly the way it was designed to be used. What made this tough for defenders was that the detections were difficult to identify among normal Kerberos events. We recommended (and still recommend) that any SPN account have a password with a minimum of 25 characters. This will reduce the chance the attacker is able to crack the password offline, if they are successful in pulling the SPN tickets. At the time of the release of the Kerberoast attack, the detection was riddled with false positives and was determined to not be effective. I decided to do more research into the Kerberos events and identified a unique indicator in them, which allowed me to build a reliable detection. Let's look at the Kerberos event titled 4769. [caption id="attachment_14169" align="aligncenter" width="589"]

Fig. 1 - Event 4769 - 'A Kerberos service ticket was requested.'[/caption] If you have ever looked at the 4769 and 4768 events, you probably realized it was so much noise that any reliable detection was probably futile. I have worked with clients that specifically ignore these events because the amount of storage space it would take to capture them all from their systems would likely double their storage requirements. I understand you must balance cost of detection with risk of missing an early IoC. With the success of the Kerberoast attack, the 4769 event is your only detection into this attack. There are ways to reduce the number of events you need to capture. I am going to show you the limiters to put into your forwarders, which should reduce the amount of additional storage space, while gaining early insight into this attack. To set up this blog, I used *setspn* to register a *sqlsvc* account as an SPN, then used *GetUserSPNs.py* from Impacket and "GetU

with the "sqlsvc" account and password to pull the

SPN tickets. This allowed me to compare normal Kerberos events with the Kerberoast attack. [caption id="attachment_14170" align="aligncenter" width="974"]

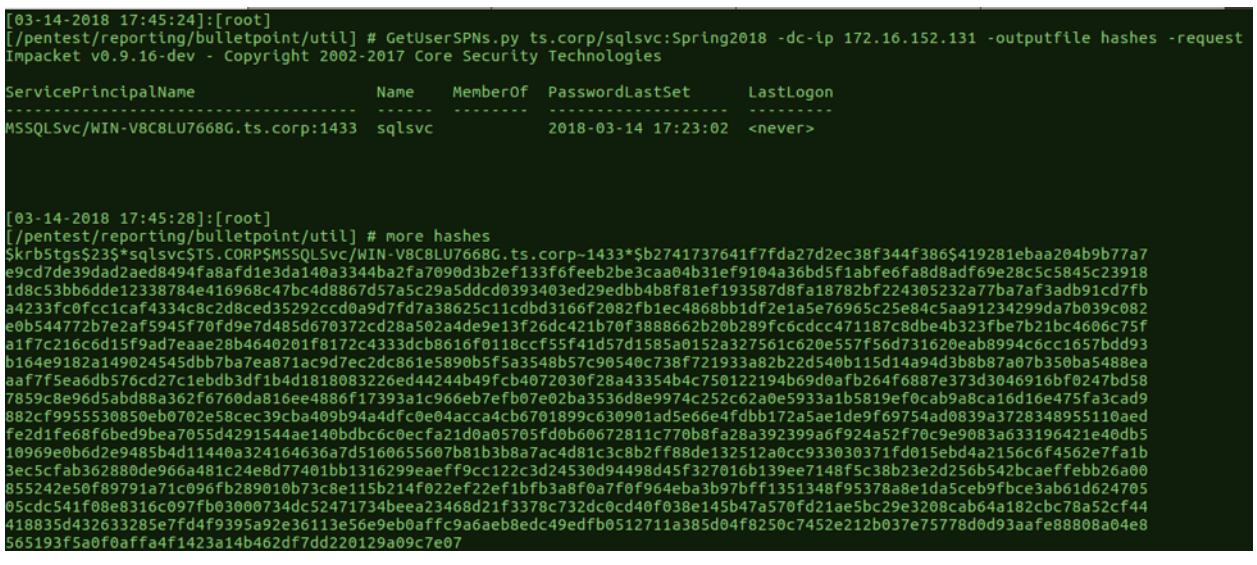


Fig. 2 - Request SPN Tickets with GetUserSPNs.py[/caption] [caption id="attachment_14171" align="aligncenter" width="974"]

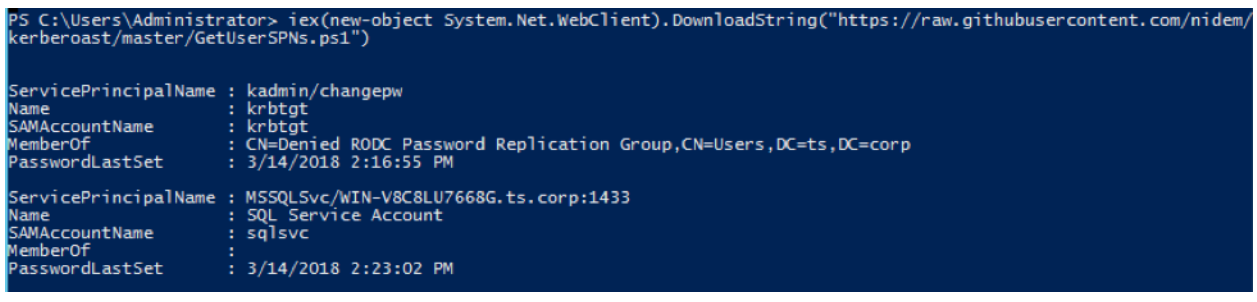


Fig. 3 - Request SPN Tickets with GetUserSPNs.ps1[/caption] With any event I investigate, I use PowerShell to help look at some parts of each event which may be unique to one another. I use the "Get-EventLog" Cmdlet and then use some functions which allow me to see parts of the event and compare them to other events with the same ID. I started by grabbing all the 4769 event IDs for the last 24 hours.

```
$kerb_tickets = Get-EventLog -LogName Security -InstanceId 4769 -After "03/14/
```

Note: Set the -After parameter to yesterday's date. This will give you 24 hours of events matching 4769. Otherwise you may get way more events than you need. The first thing I compared was the Service Information section. When I compared normal Kerberos traffic to my Kerberoast attacks, I noticed the "Service Name" for normal events typically ended with a $ or was "krbtgt". My Kerberoast attacks had the user name of the account I used to request the SPN tickets.

```
$kerb_events | % { Write-Output $_.Message.split("`n")[8]
```

Note: This code will take the Message segment and split each line into a collection. By referencing the [8] index, which is the 9th line of the Message, I can compare each Service Name. [caption id="attachment_14172" align="aligncenter" width="712"]

```
PS C:\Users\Administrator> $kerb_events | % { Write-Output $_.Message.split("`n")[8] }
        Service Name:           WIN-V8C8LU7668G$
        Service Name:           krbtgt
        Service Name:           WIN-V8C8LU7668G$
        Service Name:           WIN-V8C8LU7668G$
        Service Name:           WIN-V8C8LU7668G$
        Service Name:           WIN-V8C8LU7668G$
        Service Name:           WIN-V8C8LU7668G$
        Service Name:           WIN-V8C8LU7668G$
        Service Name:           WIN-V8C8LU7668G$
        Service Name:           WIN-V8C8LU7668G$
        Service Name:           krbtgt
        Service Name:           WIN-V8C8LU7668G$
        Service Name:           WIN-V8C8LU7668G$
        Service Name:           WIN-V8C8LU7668G$
        Service Name:           sqlsvc
        Service Name:           krbtgt
        Service Name:           WIN-V8C8LU7668G$
        Service Name:           WIN-V8C8LU7668G$
        Service Name:           WIN-V8C8LU7668G$
        Service Name:           WIN-V8C8LU7668G$
        Service Name:           WIN-V8C8LU7668G$
        Service Name:           WIN-V8C8LU7668G$
        Service Name:           WIN-V8C8LU7668G$
        Service Name:           WIN-V8C8LU7668G$
        Service Name:           WIN-V8C8LU7668G$
        Service Name:           WIN-V8C8LU7668G$
        Service Name:           krbtgt
        Service Name:           WIN-V8C8LU7668G$
        Service Name:           WIN-V8C8LU7668G$
        Service Name:           WIN-V8C8LU7668G$
```

Fig. 4 - Pulling the Service Name from every 4769 event.[/caption] Our first limiter becomes "Service Name is not equal to "krbtgt" and Service Name does not end with a dollar sign ($)." However, when a user maps a drive, this limiter by itself creates a false-positive. We need a few more limiters to isolate the Kerberoast attack from normal Kerberos events. The next thing I looked at was the Account Name. I noticed that most, but not all, Kerberos requests specified the account name as "<MachineName>$@<DOMAIN>". There were some requests that had Administrator@<DOMAIN> so this limiter by itself was also not enough to reduce the false-positives. We are getting closer though!

```
$kerb_events | % { Write-Output $_.Message.split("`n")[3] }
```

[caption id="attachment_14173" align="aligncenter" width="641"]

```
PS C:\Users\Administrator> $kerb_events | % { Write-Output $_.Message.split("`n")[3] }
        Account Name:           Administrator@TS.CORP
        Account Name:           WIN-V8C8LU7668G$@TS.CORP
        Account Name:           WIN-V8C8LU7668G$@TS.CORP
        Account Name:           WIN-V8C8LU7668G$@TS.CORP
        Account Name:           Administrator@TS.CORP
        Account Name:           WIN-V8C8LU7668G$@TS.CORP
        Account Name:           WIN-V8C8LU7668G$@TS.CORP
        Account Name:           WIN-V8C8LU7668G$@TS.CORP
        Account Name:           WIN-V8C8LU7668G$@TS.CORP
        Account Name:           WIN-V8C8LU7668G$@TS.CORP
        Account Name:           WIN-V8C8LU7668G$@TS.CORP
        Account Name:           Administrator@TS.CORP
        Account Name:           sqlsvc@TS.CORP
        Account Name:           WIN-V8C8LU7668G$@TS.CORP
        Account Name:           WIN-V8C8LU7668G$@TS.CORP
        Account Name:           WIN-V8C8LU7668G$@TS.CORP
        Account Name:           WIN-V8C8LU7668G$@TS.CORP
        Account Name:           Administrator@TS.CORP
        Account Name:           WIN-V8C8LU7668G$@TS.CORP
        Account Name:           WIN-V8C8LU7668G$@TS.CORP
        Account Name:           WIN-V8C8LU7668G$@TS.CORP
        Account Name:           WIN-V8C8LU7668G$@TS.CORP
        Account Name:           WIN-V8C8LU7668G$@TS.CORP
        Account Name:           WIN-V8C8LU7668G$@TS.CORP
        Account Name:           WIN-V8C8LU7668G$@TS.CORP
        Account Name:           WIN-V8C8LU7668G$@TS.CORP
        Account Name:           WIN-V8C8LU7668G$@TS.CORP
```

Fig. 5 - Pulling the Account Name from every 4769 event.[/caption] I then looked at the "Additional Ticket Information" section of the event. I realized that 4769 shows both "success" and failure" with the Failure Code. There is an entire list of failure codes, but we are only concerned about the success code of "0x0". We are not concerned if someone failed to get the SPN tickets.

```
$kerb_events | % { Write-Output $_.Message.split("`n")[18] }
```

[caption id="attachment_14174" align="aligncenter" width="583"]

```
PS C:\Users\Administrator> $kerb_events | % { Write-Output $_.Message.split("`n")[18] }
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
        Failure Code:           0x0
```

Fig. 6 - Pulling the Failure Code from every 4769 event.[/caption] Finally, I looked at the Ticket Encryption Type. There was very limited information about this, but the event did state this was based on RFC 4120. I went through the RFC and identified the table which describes each of these codes. [caption id="attachment_14175" align="aligncenter" width="566"]

7.5.7. Kerberos Message Types

Message Type	Value	Meaning
KRB_AS_REQ	10	Request for initial authentication
KRB_AS_REP	11	Response to KRB_AS_REQ request
KRB_TGS_REQ	12	Request for authentication based on TGT
KRB_TGS_REP	13	Response to KRB_TGS_REQ request
KRB_AP_REQ	14	Application request to server
KRB_AP_REP	15	Response to KRB_AP_REQ_MUTUAL
KRB_RESERVED16	16	Reserved for user-to-user krb_tgt_request
KRB_RESERVED17	17	Reserved for user-to-user krb_tgt_reply
KRB_SAFE	20	Safe (checksummed) application message
KRB_PRIV	21	Private (encrypted) application message
KRB_CRED	22	Private (encrypted) message to forward credentials
KRB_ERROR	30	Error response

Fig. 7 - Ticket Encryption Type information from RFC 4120.[/caption] When I compared the Kerberoast event Ticket Encryption Type with most of the other Encryption Types, it was very easy to see which event was the Kerberoast and which was normal Kerberos traffic. My Kerberoast was 0x17 "user-to-user krb_tgt_reply" whereas the normal Kerberos traffic was 0x12 "Request for authentication based on TGT".

```
$kerb_events | % { Write-Output $_.Message.split("`n")[17] }
```

[caption id="attachment_14182" align="aligncenter" width="681"]

SKIP TO MAIN CONTENT

```
PS C:\Users\Administrator> $kerb_events | % { Write-Output $_.Message.split("`n")[17] }
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x17
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
        Ticket Encryption Type: 0x12
```

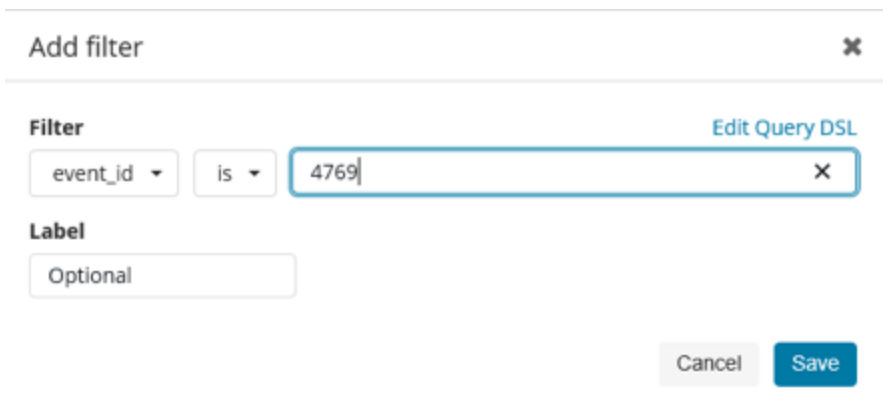
Fig. 8 - Pulling Ticket Encryption Type from every 4769 event.[/caption] We now have our limiters! Let's review:

1. Event ID 4769
2. Service Name not equal to 'krbtgt'
3. Service Name does not end with '$'
4. Account Name does not match '<MachineName>$@<Domain>'
5. Failure Code is '0x0'
6. Ticket Encryption Type is '0x17'

Using these limiters, we can create specific search queries in our SIEM or event aggregator system to identify when someone is requesting SPN tickets. While I am demonstrating this in an ELK Stack (Elasticsearch, Logstash, Kibana), you can translate this to Splunk or other query languages. In ELK, you will need to create 6 filters. The first 4 are straightforward:

1. event_id "is" 4769
2. Status "is" 0x0
3. Ticket_Encryption_Type "is" 0x17
4. Service_Name "is not" krbtgt

To add a filter, click the button and select the field in the drop down. Then choose "is" or "is not" and enter the value. Click Save to save the filter.

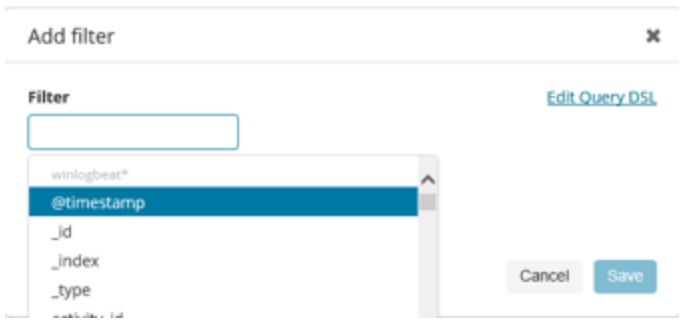


The last 2 require "NOT" with a wildcard search. We will create the wildcard filters first and then change them to "NOT". The two queries we will add are:

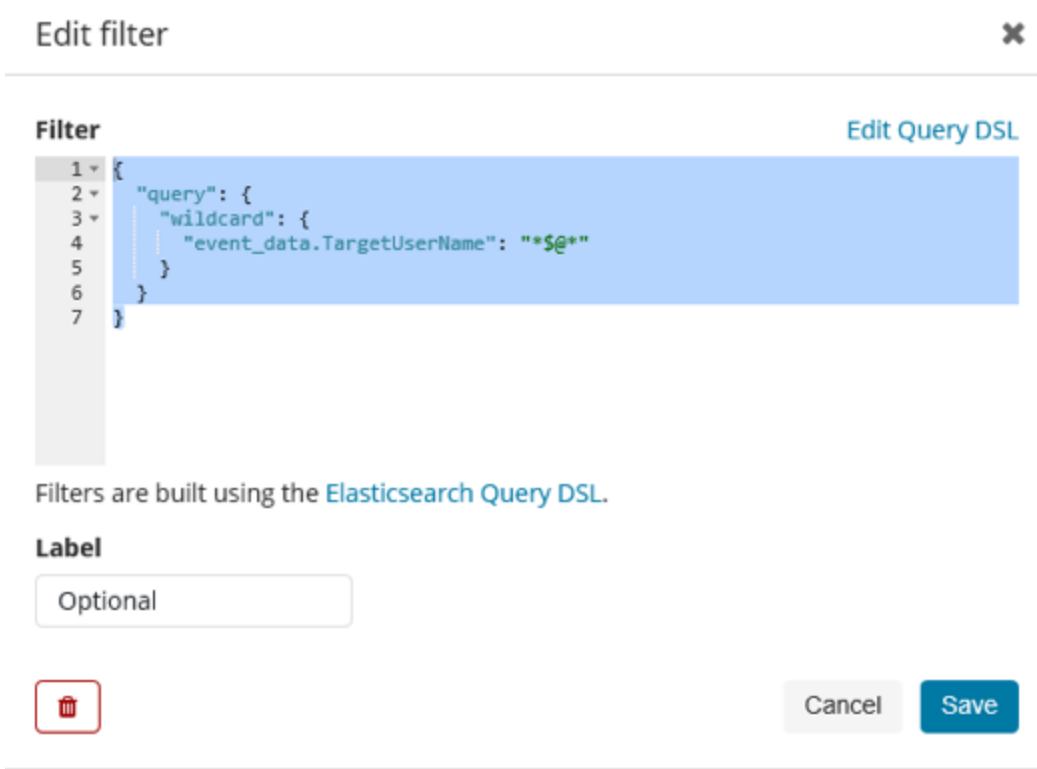
1. {"query": {"wildcard": {"event_data.ServiceName": "*$"}}}
2. {"query": {"wildcard": {"event_data.TargetUserName": "*$@*"}}}

To addo the following steps for both queries above:

1. Add a filter and click Edit Query DSL



2. Enter the query in the filter window



3. Click Save

4. Hover over the filter and click the magnifying glass to change the query to "NOT"

Once we are done, we should have a list of only our Kerberoast SPN ticket requests! I can see the request from yesterday with "GetUserSPNs.py" and the one from today where I used the PowerShell module "GetUserSPNs.ps1". [caption id="attachment_14183" align="aligncenter" width="974"]

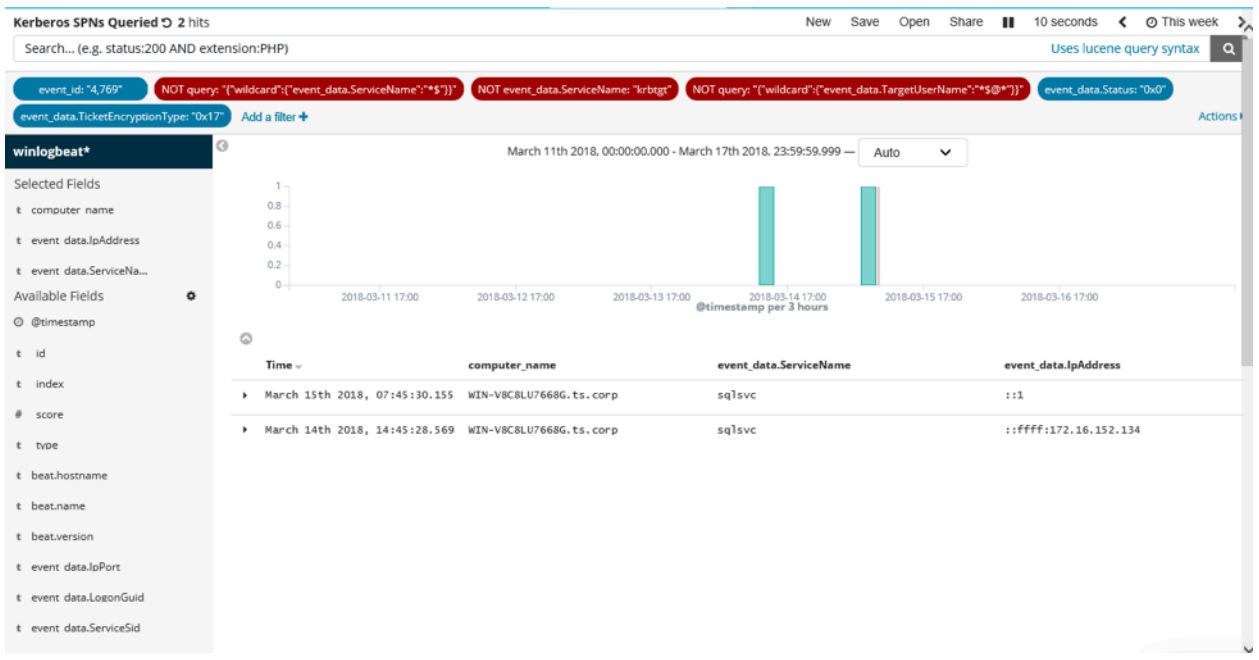


Fig. 9 - Kerberos SPN Queried Detection in ELK[/caption] We now have a reliable way to detect when someone pulls the SPN tickets. While this does not stop the attack, it gives us insight into the early indicators and allows us to react accordingly. We would recommend using Managed Service Accounts which takes care of the password manag . However, if SPN accounts are going to be

managed manually, we recommend having the SPN accounts set up with a minimum of 25 characters for the password. As it is right now, the hash becomes too large for most hash cracking software and prevents the attack from successfully cracking the password. While the cracking software could be updated to handle larger hashes, this detection gives you the knowledge of the attack. And, in the immortal words of my childhood cartoon show, "... and knowing is half the battle!"