

John The Ripper| tryhackme walkthrough



amshi · Follow

10 min read · Jan 31, 2024



Listen



Share



More

N.B: *You would have to first identify what type of hash it is then insert the type in the format part of the command & to identify the hash make sure to use tools like hash-identifier.*

John the Ripper is one of the most well known, well-loved and versatile hash cracking tools out there. It combines a fast cracking speed, with an extraordinary range of compatible hash types.



John the Ripper — An extremely powerful and adaptable hash cracking tool

What are Hashes?

A hash is a way of taking a piece of data of any length and representing it in another form that is a fixed length. This masks the original value of the data. This is done by

running the original data through a hashing algorithm. There are many popular hashing algorithms, such as MD4, MD5, SHA1 and NTLM.

What makes Hashes secure?

Hashing algorithms are designed so that they only operate one way. This means that a calculated hash cannot be reversed using just the output given.

Where John Comes in...

Even though the algorithm itself is not feasibly reversible. That doesn't mean that cracking the hashes is impossible. If you have the hashed version of a password, for example- and you know the hashing algorithm- you can use that hashing algorithm to hash a large number of words, called a dictionary. You can then compare these hashes to the one you're trying to crack, to see if any of them match. If they do, you now know what word corresponds to that hash- you've cracked it!

This process is called a **dictionary attack** and John the Ripper, or John as it's commonly shortened to, is a tool to allow you to conduct fast brute force attacks on a large array of different hash types.

Task 2: Setting up John the Ripper

Q: What is the most popular extended version of John the Ripper?

A: Jumbo John

Task 3: Wordlists

Wordlists

As we explained in the first task, in order to dictionary attack hashes, you need a list of words that you can hash and compare, unsurprisingly this is called a wordlist. There are many different wordlists out there, a good collection to use can be found in the [SecLists](#) repository.

Q: What website was the rockyou.txt wordlist created from a breach on?

A: rockyou.com

Task 4: Cracking Basic Hashes

John Basic Syntax

The basic syntax of John the Ripper commands is as follows.

```
john [options] [path to file]
```

john - Invokes the John the Ripper program

[path to file] - The file containing the hash you're trying to crack, if it's in the same directory you won't need to name a path, just the file.

Automatic Cracking

John has built-in features to detect what type of hash it's being given, and to select appropriate rules and formats to crack it for you, this isn't always the best idea as it can be unreliable- but if you can't identify what hash type you're working with and just want to try cracking it, it can be a good option! To do this we use the following syntax:

```
john --wordlist=[path to wordlist] [path to file]
```

--wordlist= - Specifies using wordlist mode, reading from the file that you supply in the following path...

[path to wordlist] - The path to the wordlist you're using, as described in the previous task.

Example Usage:

```
john --wordlist=/usr/share/wordlists/rockyou.txt hash_to_crack.txt
```

Identifying Hashes

Sometimes John won't play nicely with automatically recognising and loading hashes, that's okay! We're able to use other tools to identify the hash, and then set john to use a specific format. There are multiple ways to do this, such as using an online hash identifier like [this](#) one. I like to use a tool called [hash-identifier](#), a Python tool that is super easy to use and will tell you what different types of hashes the one you enter is likely to be, giving you more options if the first one fails.

To use hash-identifier, you can just pull the python file from gitlab using: `wget https://gitlab.com/kalilinux/packages/hash-identifier/-/raw/kali/master/hash-id.py`.

Then simply launch it with `python3 hash-id.py` and then enter the hash you're trying to identify- and it will give you possible formats!

Format-Specific Cracking

Once you have identified the hash that you're dealing with, you can tell john to use it while cracking the provided hash using the following syntax:

```
john --format=[format] --wordlist=[path to wordlist] [path to file]
```

`--format=` - This is the flag to tell John that you're giving it a hash of a specific format, and to use the following format to crack it

`[format]` - The format that the hash is in

Example Usage:

```
john --format=raw-md5 --wordlist=/usr/share/wordlists/rockyou.txt  
hash_to_crack.txt
```

A Note on Formats:

When you are telling john to use formats, if you're dealing with a standard hash type, e.g. md5 as in the example above, you have to prefix it with `raw-` to tell john you're just dealing with a standard hash type, though this doesn't always apply. To check if you need to add the prefix or not, you can list all of John's formats using `john --list=formats` and either check manually, or `grep` for your hash type using something like `john --list=formats | grep -iF "md5"`.

Q:What type of hash is hash1.txt?

A: md5

Q:What is the cracked value of hash1.txt?

A: biscuit

Q:What type of hash is hash2.txt?

A: sha1

Q:What is the cracked value of hash2.txt

A: kangaroo

Q:What type of hash is hash3.txt?

A: sha256

Q:What is the cracked value of hash3.txt

A: microphone

Q:What type of hash is hash4.txt?

A: whirlpool

Q:What is the cracked value of hash4.txt

A: colossal

Task 5: Cracking Windows Authentication Hashes

NTHash / NTLM

NThash is the hash format that modern Windows Operating System machines will store user and service passwords in. It's also commonly referred to as "NTLM" which references the previous version of Windows format for hashing passwords known as "LM", thus "NT/LM".

Q:What do we need to set the "format" flag to, in order to crack this?

A: nt

Q:What is the cracked value of this password?

A: mushroom

Task 6: Cracking /etc/shadow Hashes

Cracking Hashes from /etc/shadow

The /etc/shadow file is the file on Linux machines where password hashes are stored. It also stores other information, such as the date of last password change and password expiration information. It contains one entry per line for each user or user account of the system. This file is usually only accessible by the root user- so in order to get your hands on the hashes you must have sufficient privileges, but if you do- there is a chance that you will be able to crack some of the hashes.

Unshadowing

John can be very particular about the formats it needs data in to be able to work with it, for this reason- in order to crack /etc/shadow passwords, you must combine it with the /etc/passwd file in order for John to understand the data it's being given. To do this, we use a tool built into the John suite of tools called unshadow. The basic syntax of unshadow is as follows:

```
unshadow [path to passwd] [path to shadow]
```

unshadow - Invokes the unshadow tool

[path to passwd] - The file that contains the copy of the /etc/passwd file you've taken from the target machine

[path to shadow] - The file that contains the copy of the /etc/shadow file you've taken from the target machine

Q:What is the root password?

A: 1234

Task 7: Single Crack Mode

Word Mangling

The best way to show what Single Crack mode is, and what word mangling is, is to actually go through an example:

If we take the username: Markus

Some possible passwords could be:

- Markus1, Markus2, Markus3 (etc.)
- MArkus, MARkus, MARKus (etc.)
- Markus!, Markus\$, Markus* (etc.)

This technique is called word mangling. John is building it's own dictionary based on the information that it has been fed and uses a set of rules called "mangling rules" which define how it can mutate the word it started with to generate a wordlist based off of relevant factors for the target you're trying to crack. This is exploiting

how poor passwords can be based off of information about the username, or the service they're logging into.

Using Single Crack Mode

To use single crack mode, we use roughly the same syntax that we've used to so far, for example if we wanted to crack the password of the user named "Mike", using single mode, we'd use:

```
john --single --format=[format] [path to file]
```

--single - This flag lets john know you want to use the single hash cracking mode.

Example Usage:

```
john --single --format=raw-sha256 hashes.txt
```

A Note on File Formats in Single Crack Mode:

If you're cracking hashes in single crack mode, you need to change the file format that you're feeding john for it to understand what data to create a wordlist from. You do this by prepending the hash with the username that the hash belongs to, so according to the above example- we would change the file hashes.txt

From:

1efee03cdcb96d90ad48ccc7b8666033

To

mike:1efee03cdcb96d90ad48ccc7b8666033

Q:What is Joker's password?

A: Jok3r

Task 8: Custom Rules

Common Custom Rules

Many organisations will require a certain level of password complexity to try and combat dictionary attacks, meaning that if you create an account somewhere, go to create a password and enter:

polopassword

You may receive a prompt telling you that passwords have to contain at least one of the following:

- Capital letter
- Number
- Symbol

This is good! However, we can exploit the fact that most users will be predictable in the location of these symbols. For the above criteria, many users will use something like the following:

Polopassword1!

A password with the capital letter first, and a number followed by a symbol at the end. This pattern of the familiar password, appended and prepended by modifiers (such as the capital letter or symbols) is a memorable pattern that people will use, and reuse when they create passwords. This pattern can let us exploit password complexity predictability.

Now this does meet the password complexity requirements, however as an attacker we can exploit the fact we know the likely position of these added elements to create dynamic passwords from our wordlists.

How to create Custom Rules

Custom rules are defined in the `john.conf` file, usually located in `/etc/john/john.conf` if you have installed John using a package manager or built from source with `make` and in `/opt/john/john.conf` on the TryHackMe Attackbox.

Let's go over the syntax of these custom rules, using the example above as our target pattern. Note that there is a massive level of granular control that you can define in these rules, I would suggest taking a look at the wiki [here](#) in order to get a full view of the types of modifier you can use, as well as more examples of rule implementation.

The first line:

`[List.Rules:THMRules]` - Is used to define the name of your rule, this is what you will use to call your custom rule as a John argument.

We then use a regex style pattern match to define where in the word will be modified, again- we will only cover the basic and most common modifiers here:

`Az` - Takes the word and appends it with the characters you define

`A0` - Takes the word and prepends it with the characters you define

`c` - Capitalises the character positionally

These can be used in combination to define where and what in the word you want to modify.

Lastly, we then need to define what characters should be appended, prepended or otherwise included, we do this by adding character sets in square brackets `[]` in the order they should be used. These directly follow the modifier patterns inside of double quotes `" "`. Here are some common examples:

`[0-9]` - Will include numbers 0-9

`[0]` - Will include only the number 0

`[A-z]` - Will include both upper and lowercase

`[A-Z]` - Will include only uppercase letters

`[a-z]` - Will include only lowercase letters

`[a]` - Will include only a

`[!£$%@]` - Will include the symbols `!£$%@`

Q:What do custom rules allow us to exploit?

A: Password complexity predictability

Q:What rule would we use to add all capital letters to the end of the word?

A: `Az"[A-Z]"`

Q:What flag would we use to call a custom rule called "THMRules"

A: — rule=THMRules

Task 9: Cracking Password Protected Zip Files

Q:What is the password for the secure.zip file?

A: pass123

Q:What is the contents of the flag inside the zip file?

A: THM{w3ll_d0n3_h4sh_r0y4l}

Task 10: Cracking Password Protected RAR Archives

Q:What is the password for the secure.rar file?

A: password

Q:What is the contents of the flag inside the zip file?

A: THM{r4r_4rch1ve5_th15_t1m3}

Task 11: Cracking SSH Keys with John

Using John to crack the SSH private key password of id_rsa files. Unless configured otherwise, you authenticate your SSH login using a password. However, you can configure key-based authentication, which lets you use your private key, id_rsa, as an authentication key to login to a remote machine over SSH. However, doing so will often require a password- here we will be using John to crack this password to allow authentication over SSH using the key.

SSH2John

Who could have guessed it, another conversion tool? Well, that's what working with John is all about. As the name suggests ssh2john converts the id_rsa private key that you use to login to the SSH session into hash format that john can work with. Jokes aside, it's another beautiful example of John's versatility. The syntax is about what you'd expect. Note that if you don't have ssh2john installed, you can use ssh2john.py, which is located in the /opt/john/ssh2john.py. If you're doing this, replace the

ssh2john **command with** python3 /opt/ssh2john.py **or on Kali,** python /usr/share/john/ssh2john.py .

```
ssh2john [id_rsa private key file] > [output file]
```

ssh2john — Invokes the ssh2john tool

[id_rsa private key file] - The path to the id_rsa file you wish to get the hash of

> - This is the output director, we're using this to send the output from this file to the...

[output file] - This is the file that will store the output from

Example Usage

```
ssh2john id_rsa > id_rsa_hash.txt
```

Cracking

For the final time, we're feeding the file we output from ssh2john, which in our example use case is called "id_rsa_hash.txt" and, as we did with rar2john we can use this seamlessly with John:

```
john --wordlist=/usr/share/wordlists/rockyou.txt id_rsa_hash.txt
```

Q: What is the SSH private key password?

A: mango

#end

Tryhackme Walkthrough

Tryhackme

Writeup

Ctf

Cybersecurity



Follow

Written by amshi

11 Followers · 4 Following

No responses yet



What are your thoughts?

Respond

More from amshi



amshi

Nmap| tryhackme walkthrough

An in depth look at scanning with Nmap, a powerful network scanning tool.

Dec 30, 2023 🖱 4



```
Return-Path: <reback-a3970-837890-838253-c8b776d9=952622232=8@ant.anki-tech.com>
X-Originating-Ip: [43.255.56.161]
Received-SPF: pass (domain of ant.anki-tech.com designates 43.255.56.161 as permitted sender)
Authentication-Results: atlas206.free.mail.ne1.yahoo.com;
  dkim=pass header.i=@ant.anki-tech.com header.s=default;
  spf=pass smtp.mailfrom=ant.anki-tech.com;
  dmarc=pass(p=NONE) header.from=ant.anki-tech.com;
X-Apparently-To: alexa@yahoo.com; Mon, 21 Jun 2021 15:36:02 +0000
X-YMailISG: iU.RbH8WLDuS8PKXbPwmeJ5ksCZTcrgGSzQNPLV2GG3TS1LJ
  tLtofC8wExjmLmWTFhcEr1guoWTIy09uPLSlg2sv9ZNXf366atDDf8yKQApo
  rfdxFKAerJalk4hzdsHGAinSPoQR6AZmaFo83Hso0emdB0z7hjSYwHAjfpZn
  G9EYqjGm8Krb5Wf9RVTqVUh_xam0JSRA7Srl1b3d73aea31ilE0blddfzZ_W
  Wl37yrp9kU6_dIWfGR.1pABp95cRj_mDJUvpJnSpMfer0r8Jj70BJ03VAdnx
  DNqWFFnIsacy_4uofvHG_Bk7r.Q6FA2Kr1fnyhS_o.ZHpkgjE4eggUHG2b3J
  gSzYSw57V_QMOP7vW6MMkQiAVAiN7H_z.548QaUg7pzS0g0a4aLuJm5FjfwT
  FMgAS1tZVU9qfjPkbFxDxL8AnHLCw3BtZaMhipp7XiTb3PZcaQDvNq3PRyyr
  QtzrJl9GnAd7D_CF9RA.HCQm6V.pT6I_z0rJEIpY33Ip8.S5vkDWlrEl_h6g
  UiigoHtg4WZbNwyyKiypPtdSv6X5WA_Pzwjfy0fT5_GaATPCqPdXoNcWukUN
  1pvdU3RK_74JlDv0MqUVWhk58jgmaJXEeJb0I54D4xka6ssN1ierLAjAS9Su
```

 amshi

Phishing Analysis Fundamentals| tryhackme walkthrough

Learn all the components that make up an email.

Feb 21, 2024 🖱 2

 amshi

Burp Suite: The Basics| tryhackme walkthrough

Burp Suite is a framework written in Java that aims to provide a one-stop-shop for web application penetration testing.

Dec 21, 2023 🖱 4



```
Filter
tcp.analysis.flags && !tcp.analysis.window_update
hsrp.state != 8 && hsrp.state != 16
ge stp.type == 0x80
ospf.msg != 1
icmp.type eq 3 || icmp.type eq 4 || icmp.type eq 5 || icmp.type eq 11 || icmpv6.type eq 1 || icmpv6.type eq 2 || icmpv6.type eq 3 || icmpv6
arp
icmp || icmpv6
tcp.flags.reset eq 1
sctp.chunk_type eq ABORT
(! ip.dst == 224.0.0.0/4 && ip.ttl < 5 && !pim && !ospf) || (ip.dst == 224.0.0.0/24 && ip.dst != 224.0.0.251 && ip.ttl != 1 && !(vrrp || car
eth.fcs.status=="Bad" || ip.checksum.status=="Bad" || tcp.checksum.status=="Bad" || udp.checksum.status=="Bad" || sctp.checksum.s
smb || nbss || nbns || netbios
http || tcp.port == 80 || http2
dcerpc
hsrp || eigrp || ospf || bgp || cdp || vrrp || carp || gvrp || igmp || ismp
tcp.flags & 0x02 || tcp.flags.fin == 1
tcp
udp
eth[0] & 1
systemd_journal || sysdig
```



amshi

Wireshark 101 | tryhackme walkthrough

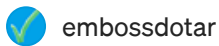
Wireshark, a tool used for creating and analyzing PCAPs (network packet capture files), is commonly used as one of the best packet analysis...

Jan 23, 2024 🖱 53 💬 1



See all from amshi

Recommended from Medium



embossdotar

TryHackMe—Hashing Basics—Writeup

Key points: Hashing | Hashing functions | File integrity checking | Hashcat | John the Ripper | Rainbow table | Password cracking. Hashing...

★ Oct 23, 2024 🖱 2



In T3CH by Axoloth

TryHackMe | Training Impact on Teams | WriteUp

Discover the impact of training on teams and organisations

★ Nov 5, 2024 🖱 60



Lists



Tech & Tools

22 stories · 377 saves



Medium's Huge List of Publications Accepting Submissions

377 stories · 4298 saves



Staff picks

791 stories · 1541 saves



Natural Language Processing

1881 stories · 1516 saves

Cyber Security 101 > Cryptography > Cryptography Basics

Cryptography Basics

Learn the basics of cryptography and symmetric encryption.

🔒 Easy ⌚ 45 min

👉 Share your achievement

Help ▾

🔖 Save Room

👍 143 🗨

⚙ Options ▾

Room completed (100%)

- Task 1 ✓ Introduction
- Task 2 ✓ Importance of Cryptography
- Task 3 ✓ Plaintext to Ciphertext
- Task 4 ✓ Historical Ciphers
- Task 5 ✓ Types of Encryption
- Task 6 ✓ Basic Math



Jawstar

Cryptography Basics | Tryhackme Write Up | By jawstar

CYBER SECURITY 101

★ Oct 29, 2024 🖱 2



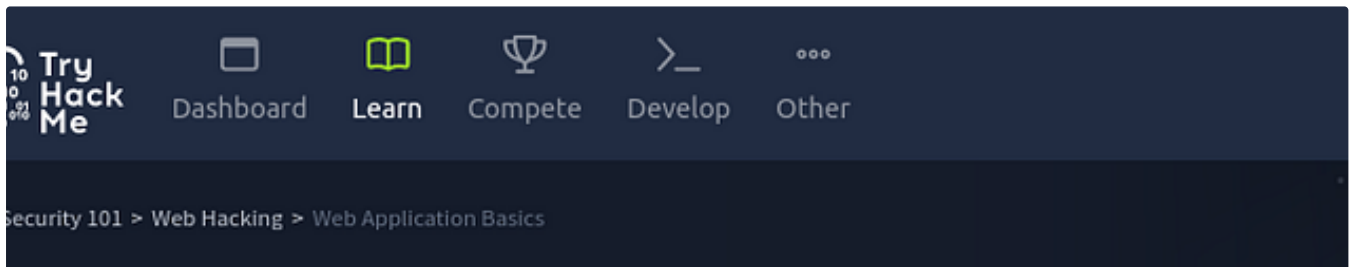


rutbar

TryHackMe—JavaScript Essentials | Cyber Security 101 (THM)

Essential Concepts

★ Oct 26, 2024 🖐️ 9



Open in app ↗

Medium



Share your achievement

Show Split View

Help ▼

Save Room

👍 48



C




Jawstar

Web Application Basics | TryHackMe

Overview: Welcome to Web Application Basics! In this room, we'll walk through the key elements of a web application, such as URLs, HTTP...

★ Oct 23, 2024 🖱 5

 Z3pH7

TryHackMe—Hashing Basics | Cyber Security 101 (THM)

Hey everyone! TryHackMe just announced the NEW Cyber Security 101 learning path, and there are tons of giveaways this time! This article...

Oct 29, 2024 🖱 101 💬 1

[See more recommendations](#)