THM-Hashing-Crypto 101(哈希-密码)-学习

本文相关的TryHackMe实验房间链接： https://tryhackme.com/room/hashingcrypto101

H2 基础关键词

H3 理论

Plaintext：明文

加密或进行哈希计算之前的数据，通常是文本，但不总是，因为它可以是一张照片或其他文件。

Encoding：编码

这不是一种加密形式，只是一种数据表示形式，如 base64或十六进制。

Hash：哈希

hash意译是散列，音译是哈希，哈希值是哈希（散列）函数的输出。哈希也可以用作动词，“ to hash”，意思是生成某些数据的 hash（散列） 值。

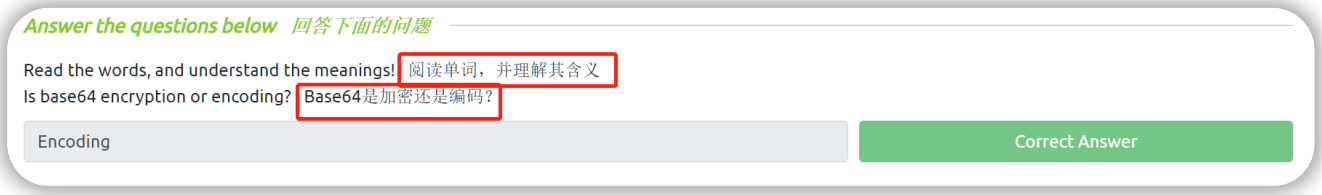
Brute force：暴力破解

通过尝试每个不同的密码或每个不同的密钥来破解加密

Cryptanalysis：密码分析

通过发现基数的弱点来攻击 "密码加密技术"

H3 答题



H2 哈希（散列）函数是什么？

H3 理论

哈希函数介绍：

hash函数与普通加密有很大不同，hash函数是没有key的，这意味着很难或者不可能从hash函数的输出返回到哈希函数的输入（不可逆）。

hash函数接受任意大小的输入数据，并创建该数据的digest(“摘要”)，而它的输出是固定大小的；很难预测任何输入的输出是什么，反之亦然；好的哈希算法计算起来(相对)快，反转起来(从输出开始并确定输入)慢；输入数据的任何小变化(即使是一个位)都会导致输出的大变化。

哈希函数的输出通常是原始字节，然后可以对这些字节进行编码，通常是以64为基数进行编码（base64编码）或以十六进制形式进行编码，解码这些数据不会给你任何有用的信息。

哈希在网络安全中经常使用。当你登录 TryHackMe 时，它使用哈希来验证你的密码；当你登录到您的计算机，这也会使用哈希值验证你的密码；你与哈希的间接交互比你想象中要多。

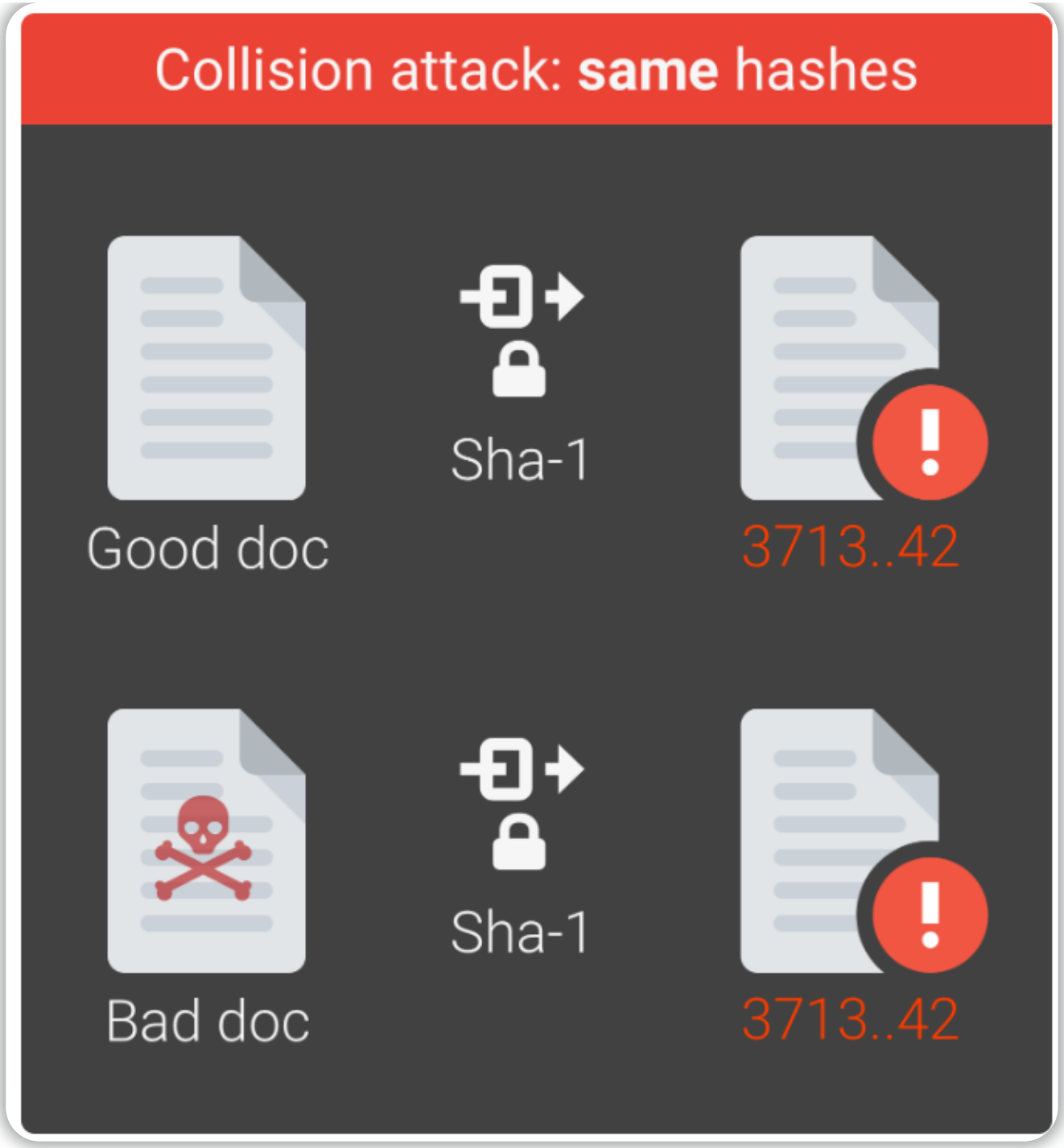
hash碰撞：

哈希碰撞是指两个不同的输入给出相同的输出。哈希函数的设计要尽可能避免这种情况，特别要尽量避免能够制造(或者有意制造)碰撞，但是碰撞本身又是难以完全避免的：由于输入多于输出，一些输入必须给出相同的输出。

MD5和 SHA1已经受到攻击，并且由于其能够制造哈希碰撞而导致技术不安全。但是，还没有攻击能够在两种哈希算法中同时给出碰撞，所以如果你使用 MD5散列和 SHA1散列进行比较，你会发现它们是不同的。

MD5碰撞的例子可以从 https://www.mscs.dal.ca/~selinger/md5collision/ 中找到；有关 SHA1碰撞的详情，请浏览 https://shattered.io/ 。

由于hash碰撞的原因，你不应该完全信任 关于计算密码或数据哈希值的算法。



H3 答题

Answer the questions below *回答下面的问题*

What is the output size in bytes of the MD5 hash function?MD5散列函数的输出大小(以字节为单位)是多少？

16 Correct Answer 正确答案

Can you avoid hash collisions? (Yea/Nay)

你能避免散列冲突吗? (是/否)

Nay Correct Answer 正确答案

If you have an 8 bit hash output, how many possible hashes are there?

如果您有一个8位散列输出，有多少可能的散列？

256 Correct Answer 正确答案

使用搜索引擎查找相关资料

H2 哈希的用途

H3 理论

哈希用于网络安全的两个主要目的：1.验证数据的完整性；2.验证密码。

使用hash进行密码验证

大多数网络应用程序在某些时候都需要验证用户的密码，如果将这些密码以纯文本形式明文存储 那是相当不安全的。你可能已经看到了有关公司数据库泄露的新闻报道。我们能够猜想到，有一些人在很多登陆上可能都会使用相同的密码，包括登陆他们的银行账户，所以泄露这些明文密码会非常糟糕。

不少数据泄露都是因为泄露了明文密码，你可能很熟悉 Kali 上的“ rockyou.txt”密码字典，这个字典来自于一家为 MySpace （社交网站）制作小工具的公司，他们用明文存储密码，最终导致 MySpace公司的数据遭到了泄露。这个文本文件包含超过1400万个密码(尽管有些不太可能是用户密码)，按长度进行排序。

Adobe 有一个值得注意的数据泄露，但情况略有不同。密码是加密的，而不是经过哈希计算的，所使用的加密也是不安全的，这意味着可以相对快速地检索明文。

Linkedin 也有一个数据漏洞， Linkedin 使用 SHA1进行密码验证，而使用 GPU 能很快计算出来密码的SHA1值。

你不能对密码进行加密，因为加密所用的密钥必须存储在某个地方，如果有人拿到了这个密钥，他们就可以直接破解密码。

这时候就需要哈希了，如果不存储密码，而是存储密码的哈希值，会怎么样？这意味着你永远不必存储用户的密码，如果你的数据库被泄露，那么攻击者将不得不破解每个密码，以找出密码是什么。

但是有一个问题，如果两个用户有相同的密码怎么办？由于哈希函数总是将相同的输入转换为相同的输出，因此你将为每个密码相同的用户存储相同的密码哈希值。也就是说，如果有人破解了一个和其他人相同的密码------他们就会进入不止一个账户。这也意味着有人可以创建一个“彩虹表”来破解哈希值。

彩虹表是一个针对明文密码的哈希查找表，你可以通过哈希值快速查找用户的密码。彩虹表是用空间换取破解哈希的时间，这里有一个简单的例子，你可以试着理解他们是什么样的。

Hash	Password
02c75fb22c75b23dc963c7eb91a062cc	zxcvbnm
b0baee9d279d34fa1dfd71aadb908c3f	11111
c44a471bd78cc6c2fea32b9fe028d30a	asdfghjkl
d0199f51d2728db6011945145a1b607a	basketball
dcddb75469b4b4875094e14561e573d8	000000
e10adc3949ba59abbe56e057f20f883e	123456
e19d5cd5af0378da05f63f891c7467af	abcd1234
e99a18c428cb38d5f260853678922e03	abc123
fcea920f7412b5da7be0cf42b8c93759	1234567

像 Crackstation（哈希破解网站） 这样的网站内部会使用巨大的彩虹表为无盐哈希（散列）提供快速的密码破解。在已排序的哈希列表中进行查找确实非常快，比尝试直接破解哈希要快得多。

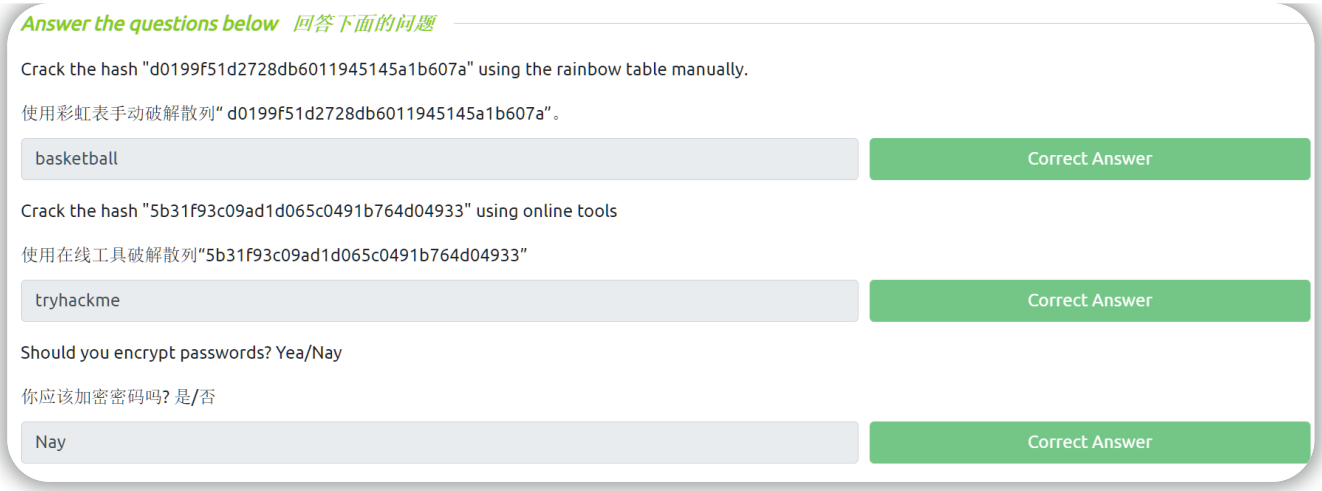
防范彩虹表

为了应对彩虹表破解hash，我们可以在密码中添加一个 Salt（加盐操作）。Salt 是随机生成并存储在数据库中的值，对于每个用户都是独一无二的。理论上，你可以对所有用户使用相同的 salt，但这意味着重复的密码仍然具有相同的 hash，并且仍然可以使用该 salt 创建特定的密码。

Salt 被添加到密码的开头或结尾，然后进行哈希计算，这意味着即使每个用户都有相同的密码，他们也会有不同的密码哈希值。像 bcrypt 和 sha512crypt 这样的哈希（散列）函数会自动进行加盐，盐不需要保密。

> 在线hash破解：
>
> https://hashes.com/en/decrypt/hash
>
> https://crackstation.net/

H3 答题



使用上面给出的 简单彩虹表和在线hash破解网站

H2 识别密码的哈希值

H3 理论

有一些工具能够自动识别密码的哈希值，比如 https://pypi.org/project/hashID/ ，但它们往往只能对具有前缀的哈希值进行识别，对于其他格式的哈希值可能并不一定能识别准确，所以我们要将工具与密码的哈希值所在的环境结合起来，如果你在 Web 应用程序数据库中找到哈希值，那么它更有可能是 md5而不是 NTLM，自动哈希识别工具经常会把这些哈希值的类型混在一起。

Unix 风格的密码的哈希值很容易识别，因为它们有一个前缀，前缀会告诉你用于生成哈希值的哈希算法。标准格式是：$format$rounds$salt$hash 。

Windows 密码使用 NTLM 进行哈希计算，NTLM 是 md4的变体。NTLM在视觉上与 md4和 md5哈希（散列）相同，因此结合哈希值所在的上下文来判断哈希（散列）类型非常重要。

在 Linux 上，密码的哈希值存储在/etc/shadow 中，此文件通常只能由 root 用户读取，它们过去存储在/etc/passwd 中，每个人都可以读取。

在 Windows 上，密码哈希值存储在 SAM 中，Windows 会试图阻止普通用户转储它们，但是有 Mimikatz 这样的工具可以做到转储密码哈希值。SAM中的哈希值分为 NT 哈希（散列）和 LM 哈希（散列）。

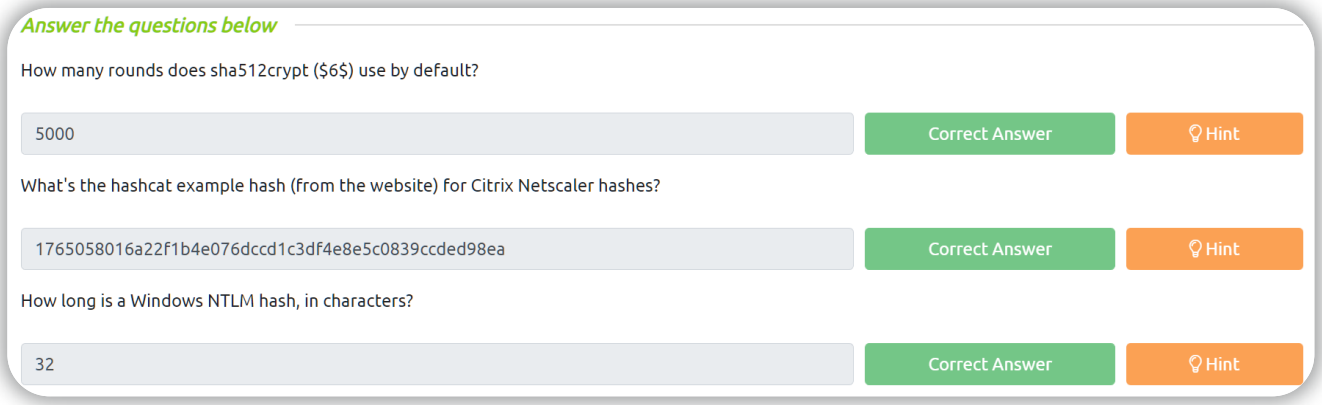
下面你将看到关于大多数 Unix 样式的密码的哈希值前缀的快速表格：

前缀	算法
$1 $	Md5crypt(用于思科和旧的 Linux/Unix 系统)
2，$2a$，$2b$，$2x$，$2y$	Bcrypt (流行于 Web 应用程序)
$6 $	Sha512crypt (大多数 Linux/Unix 系统的默认值)

能够帮助你找到更多哈希类型和密码前缀的哈希示例网站是： https://hashcat.net/wiki/doku.php?id=example_hashes

关于其他哈希类型的识别，通常需要根据长度判断、根据编码判断或对生成它们的应用程序进行分析才能得出结论。

H3 答题



> 第一题通过搜索引擎查找得出答案
>
> 第二题和第三题通过在线网站可以找到答案： https://hashcat.net/wiki/doku.php?id=example_hashes

H2 密码hash值破解

H3 理论

我们已经提到了彩虹表，它是一种破解无盐哈希的方法，但是如果涉及到加盐哈希呢?

你不能直接“解密”密码哈希值，因为它不是普通的加密。你必须通过对大量不同的输入（比如rockyou.txt）进行哈希计算 以此来破解哈希值，如果有salt的话，还要添加salt，最后将哈希计算结果与目标哈希值进行比较，哈希计算结果一旦匹配到了目标哈希值，你就能知道密码了。

常见的破解hash的工具有：Hashcat、John the Ripper等

在GPU上运行hash算法来破解hash

GPU有成千上万个核心，尽管它们不能完成 CPU 所能完成的工作，但是它们非常擅长进行一些哈希函数中所涉及的数学运算。这意味着你可以使用显卡更快地破解大多数哈希类型。一些哈希算法，特别是bcrypt，被设计成在 GPU 上进行哈希计算和在 CPU 上进行哈希计算速度差不多，这在一定程度上 有助于它们抵抗破解。

在虚拟机上运行hash算法来破解hash

虚拟机通常不能访问主机的显卡(你可以设置这一点，但需要做大量工作)。

如果你想运行 hashcat，最好在你的主机上运行(hashcat网站上有 Windows 版本，下载后 可以在powershell 上运行)，你可以让 Hashcat 在 VM 中使用 OpenCL运行，但是破解速度可能会比在主机上破解要糟糕得多。

John the Ripper（开膛手约翰）默认使用 CPU运行,在虚拟机里开箱即用，但是将 John the Ripper运行在主机操作系统上，破解速度可能更快，因为它将有更多的线程，而且也没有在 VM 中运行所需的开销。

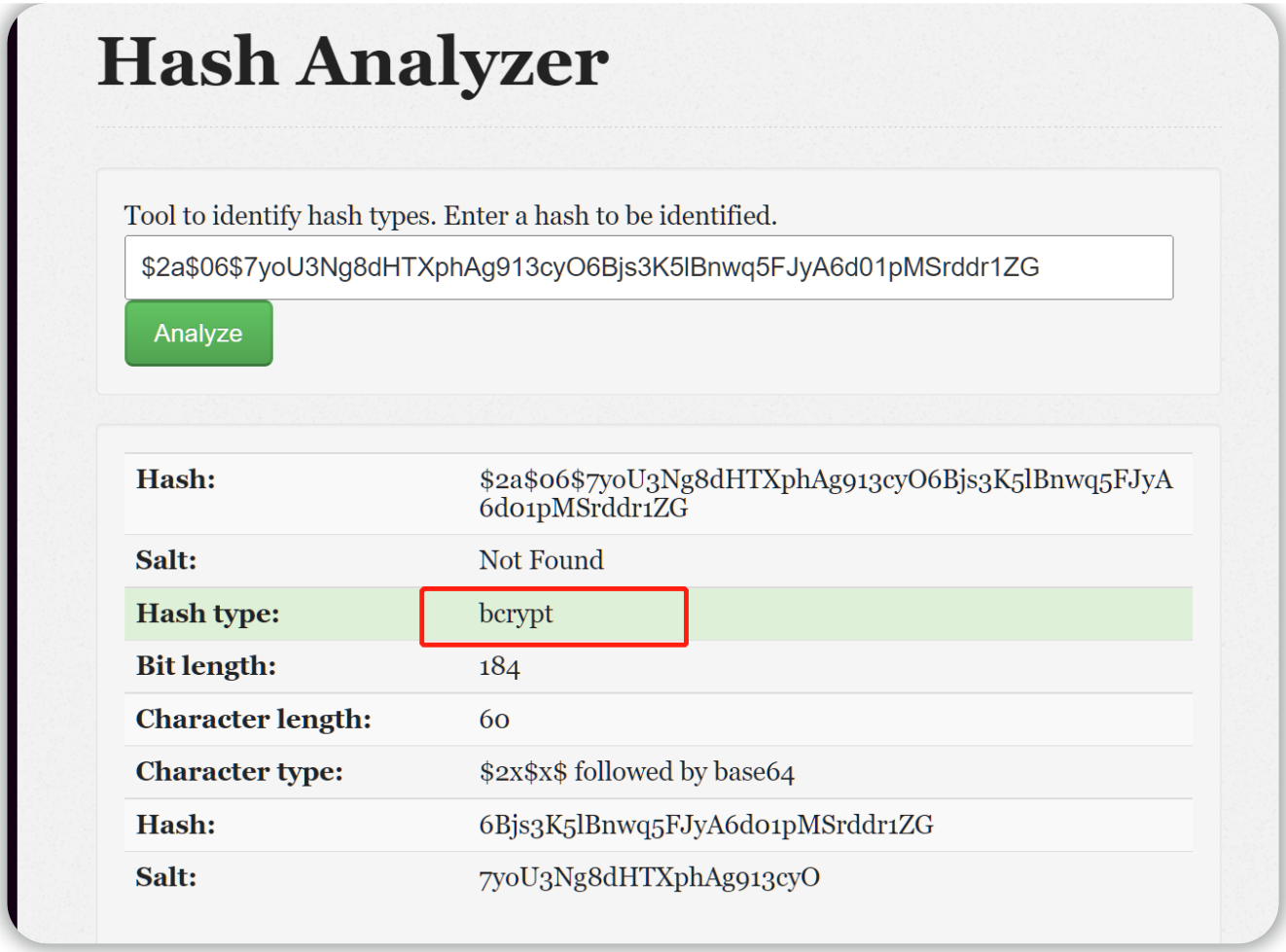
谨慎使用hashcat的--force参数，它可能给出一个错误的密码，或者跳过正确的hash值。

补充：从kali2020.2开始，hashcat 6.0将支持在CPU上运行，在虚拟机的kali环境下 不需要再使用--force参数，但是仍然建议使用主机操作系统来进行hash破解，因为如果你用 GPU来运行hash计算，那么破解速度就会快得多。

> 常用的hash破解工具：hashcat、John the Ripper
>
> hash类型在线分析网站： https://www.tunnelsup.com/hash-analyzer/

H3 答题

首先使用在线网站，分析hash的类型，第一题给出hash的类型是：bcrypt



在kali里面使用hashcat工具，先查bcrypt在hashcat中的哈希模式编号（也可以利用网站查找 https://hashcat.net/wiki/doku.php?id=example_hashes ）

```
hashcat --help|grep bcrypt
```

```
┌──(root㉿hekeats)-[/home/hekeats]
└─# hashcat --help|grep bcrypt
  3200 | bcrypt $2*$, Blowfish (Unix)                 | Operating System
 [illegible] | bcrypt(md5($pass)) / bcryptmd5          | Forums, CMS, E-Commerce
 25800 | bcrypt(sha1($pass)) / bcryptsha1              | Forums, CMS, E-Commerce
```

结合给出的哈希值前缀$2a$，可知该值在hashcat中的哈希模式编号为3200，将该hash值保存到一个文件中，然后进行hash破解操作（先cd到要使用的字典的路径下）：

```
hashcat -m 3200 hash.txt rockyou.txt
```

```
┌──(root㉿hekeats)-[/usr/share/wordlists]
└─# nano hash.txt

┌──(root㉿hekeats)-[/usr/share/wordlists]
└─# cat hash.txt
$2a$06$7yoU3Ng8dHTXphAg913cyO6Bjs3K5lBnwq5FJyA6d01pMSrddr1ZG
```

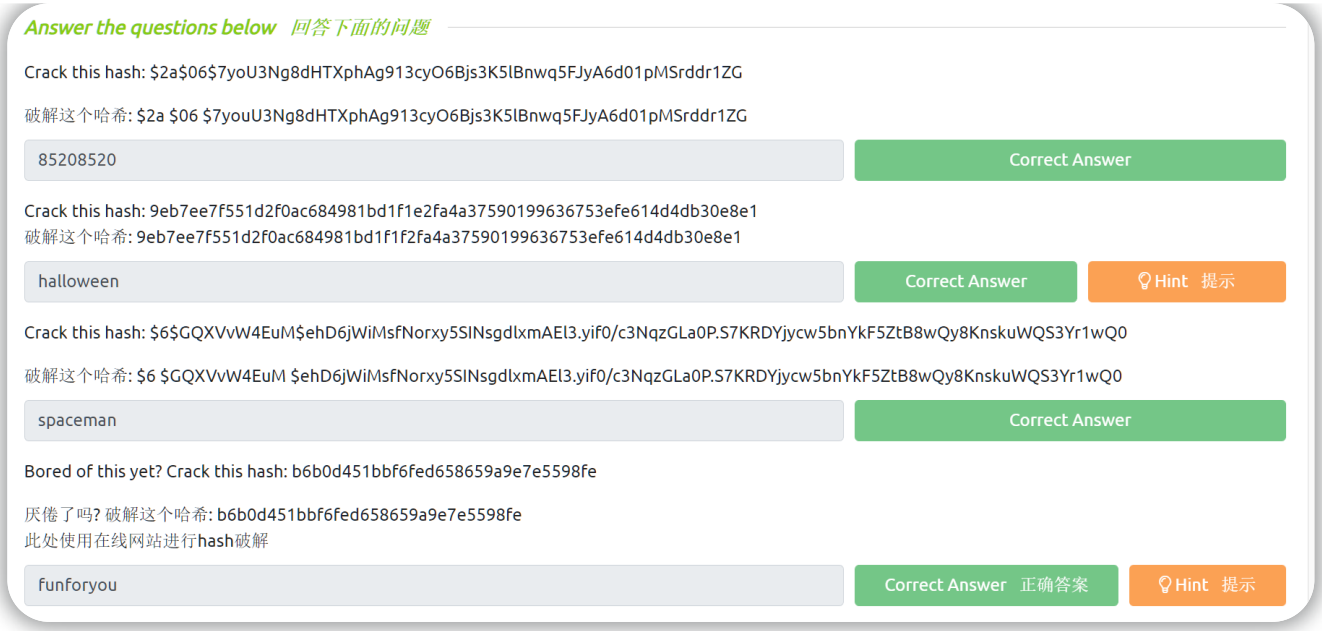
```
┌──(root㉿hekeats)-[/usr/share/wordlists]
└─# hashcat -m 3200 hash.txt rockyou.txt
hashcat (v6.2.5) starting

OpenCL API (OpenCL 3.0 PoCL 3.0+debian  Linux, None+Asserts, RELOC, LLVM 13.0.1, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
====================================================================================================================================
* Device #1: pthread-AMD Ryzen 7 6800H with Radeon Graphics, 1428/2921 MB (512 MB allocatable), 8MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 72

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
* Zero-Byte
* Single-Hash
* Single-Salt

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 0 MB

Dictionary cache built:
* Filename..: rockyou.txt
* Passwords.: 14344392
* Bytes.....: 139921507
* Keyspace..: 14344385
* Runtime...: 1 sec

Cracking performance lower than expected?

* Append -w 3 to the commandline.
  This can cause your screen to lag.

* Append -S to the commandline.
  This has a drastic speed impact but can be better for specific attacks.
  Typical scenarios are a small wordlist but a large ruleset.

* Update your backend API runtime / driver the right way:
  https://hashcat.net/faq/wrongdriver

* Create more work items to make use of your parallelization power:
  https://hashcat.net/faq/morework

$2a$06$7yoU3Ng8dHTXphAg913cyO6Bjs3K5lBnwq5FJyA6d01pMSrddr1ZG:85208520
```

由上图可知，第一题的破解结果为：85208520

第二、三题使用同样的步骤得出破解结果即可（分析hash类型，使用hashcat找到hash类型对应的哈希模式编号，使用hashcat结合字典破解hash）

第三题给出的hash值的hash类型可通过前缀识别，结合上一节中的快速表格可知：6前缀的是sha512crypt类型

第四题使用工具破解失败，使用在线网站破解hash值可得出结果



H2 利用"hash计算"进行完整性检查

H3 理论

完整性检查

哈希可用于检查文件是否已更改。如果你输入相同的数据进行哈希计算，你总是能得到相同的哈希值，即使只有一个位置的数据发生变化，计算得出的哈希值也会发生很大的变化。

这意味着你可以使用hash来检查文件是否被修改，或者确保它们已经被正确地下载。你也可以使用hash查找重复的文件，如果两个图片有相同的hash值，那么他们可能就是相同的图片。

HMACs

HMAC 是一种使用加密哈希函数来验证数据的真实性和完整性的方法，TryHackMe 的VPN 使用 HMAC-SHA512进行消息身份验证，你可以在终端输出中看到这一点。

HMAC 可以用来确保创建 HMAC 的人是本人(真实性)，并且消息没有被修改或损坏(完整性)。HMAC使用一个密钥和一个哈希算法来产生一个哈希值。

H3 答题

第一题 要求找到amd64 Kali 2019.4 ISO 的 SHA1值的和，访问网站下载对应文件，查看即可： http://old.kali.org/kali-images/kali-2019.4/

File Name ↓	File Size ↓
Parent directory/	-
SHA1SUMS	669 B
SHA1SUMS.gpg	833 B
SHA256SUMS	885 B
SHA256SUMS.gpg	833 B
kali-linux-2019.4-amd64.iso	2.6 GiB
kali-linux-2019.4-amd64.iso.torrent	206.5 KiB
kali-linux-2019.4-amd64.iso.txt.sha256sum	94 B
kali-linux-2019.4-gnome-amd64.iso	2.7 GiB
kali-linux-2019.4-gnome-amd64.iso.torrent	218.4 KiB
kali-linux-2019.4-gnome-amd64.iso.txt.sha256sum	100 B
kali-linux-2019.4-i386.iso	2.6 GiB
kali-linux-2019.4-i386.iso.torrent	204.9 KiB
kali-linux-2019.4-i386.iso.txt.sha256sum	93 B
kali-linux-2019.4-kde-amd64.iso	3.1 GiB
kali-linux-2019.4-kde-amd64.iso.torrent	246.0 KiB
kali-linux-2019.4-kde-amd64.iso.txt.sha256sum	98 B
kali-linux-2019.4-light-amd64.iso	1.1 GiB
kali-linux-2019.4-light-amd64.iso.torrent	89.6 KiB
kali-linux-2019.4-light-amd64.iso.txt.sha256sum	100 B
kali-linux-2019.4-light-i386.iso	1.1 GiB
kali-linux-2019.4-light-i386.iso.torrent	90.2 KiB
kali-linux-2019.4-light-i386.iso.txt.sha256sum	99 B

> 该值为：186c5227e24ceb60deb711f1bdc34ad9f4718ff9

第二题要求回答HMAC-SHA512 (key = $pass) 在hashcat中的哈希模式编号，使用命令查看即可：

```
┌──(root㉿hekeats)-[/home/hekeats]
└─# hashcat --help|grep HMAC-SHA512
  1750 | HMAC-SHA512 (key = $pass)                    | Raw Hash authenticated
  1760 | HMAC-SHA512 (key = $salt)                    | Raw Hash authenticated
 12100 | PBKDF2-HMAC-SHA512                           | Generic KDF
 27300 | SNMPv3 HMAC-SHA512-384                       | Network Protocol
 24500 | Telegram Desktop >= v2.1.14 (PBKDF2-HMAC-SHA512) | Instant Messaging Service

┌──(root㉿hekeats)-[/home/hekeats]
└─#
```

> 对应的模式编号为：1750