TAREFA 4 – ATAQUES DE SENHA

Lucas Loscheider Reis Muniz – lucaslc01@hotmail.com

INTRODUÇÃO

Neste relatório exploraremos alguns ataques em senhas e logins. Esses ataques serão força bruta, password spray, e hashings de senhas.

TERMOS IMPORTANTES

Obs:(força bruta e password spray serão explicados em seus tópicos na parte de execução).

Hash: é uma função matemática criptográfica (existem diversas delas, ex: MD5, SHA-256, etc), na qual você possui dados de entrada e, após passar pela criptografia, eles apresentam valores de saída padronizados com um tamanho fixo. Então quando um usuário digita uma senha para autenticação, o sistema calcula o valor do hash para a senha fornecida e esse valor é comparado ao hash armazenado para esse usuário. A autenticação é bem-sucedida se os dois hashes corresponderem. Hashes são unidirecionais (praticamente impossível obter o dado de entrada fazendo o caminho inverso do hash). Por esse motivo hashes pré-computados são usados em um ataque connhecido como **rainbow table**.

Rainbow table: método para descobrir senhas hash. A ideia básica é pré-computar uma longa lista de senhas, com seus respectivos hashes gerados por algum algoritmo específico, e armazenar essa lista em um arquivo, no formato de uma tabela e assim compará-los com o hash alvo.

Hydra: software utilizado para descobrir senhas de login de diversas formas e serviços.

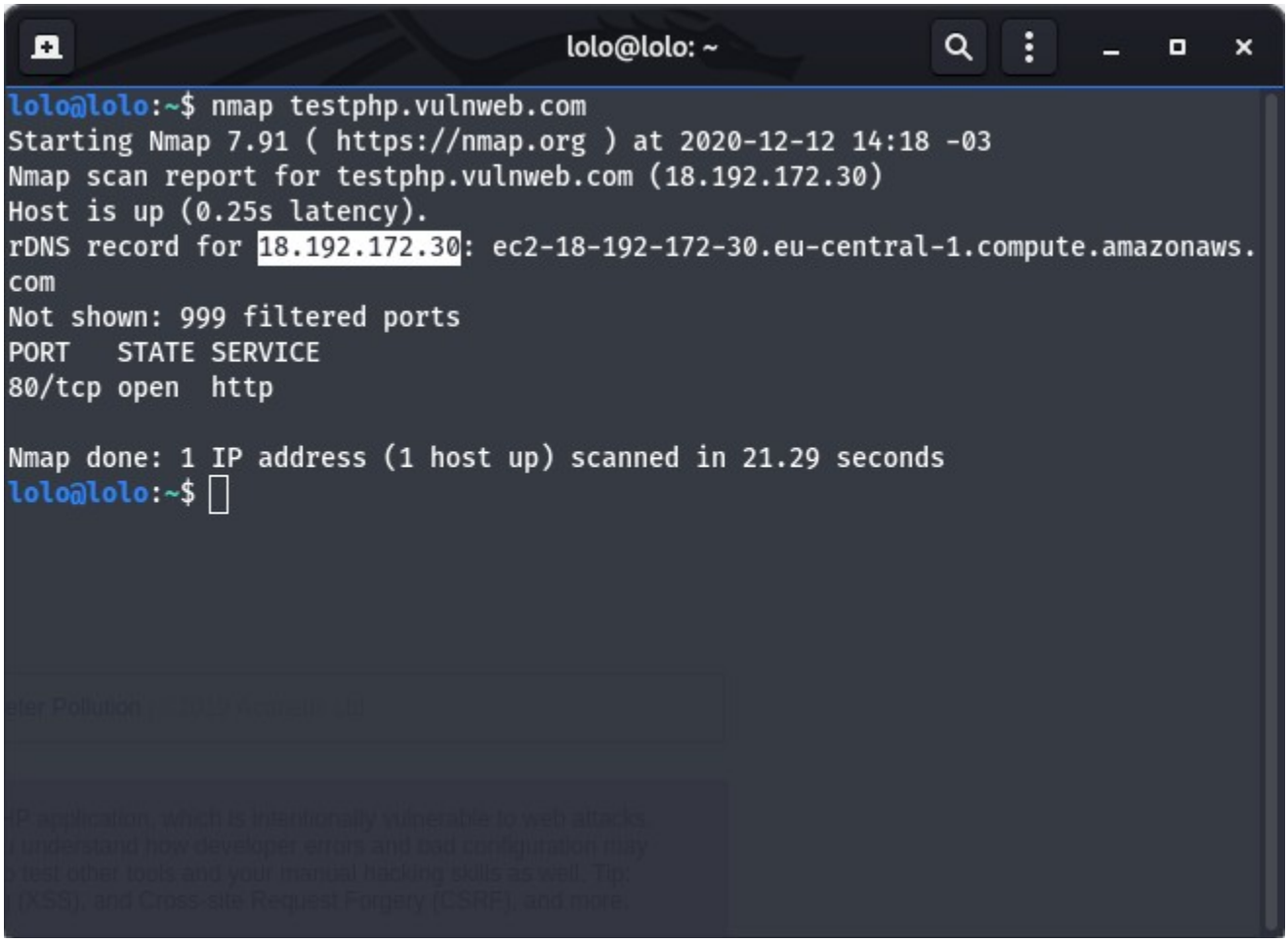
Request Body: é onde geralmente enviamos dados que queremos gravar no servidor. Geralmente utilizado em requisições do tipo POST ou PUT.

EXECUÇÃO

-Força Bruta

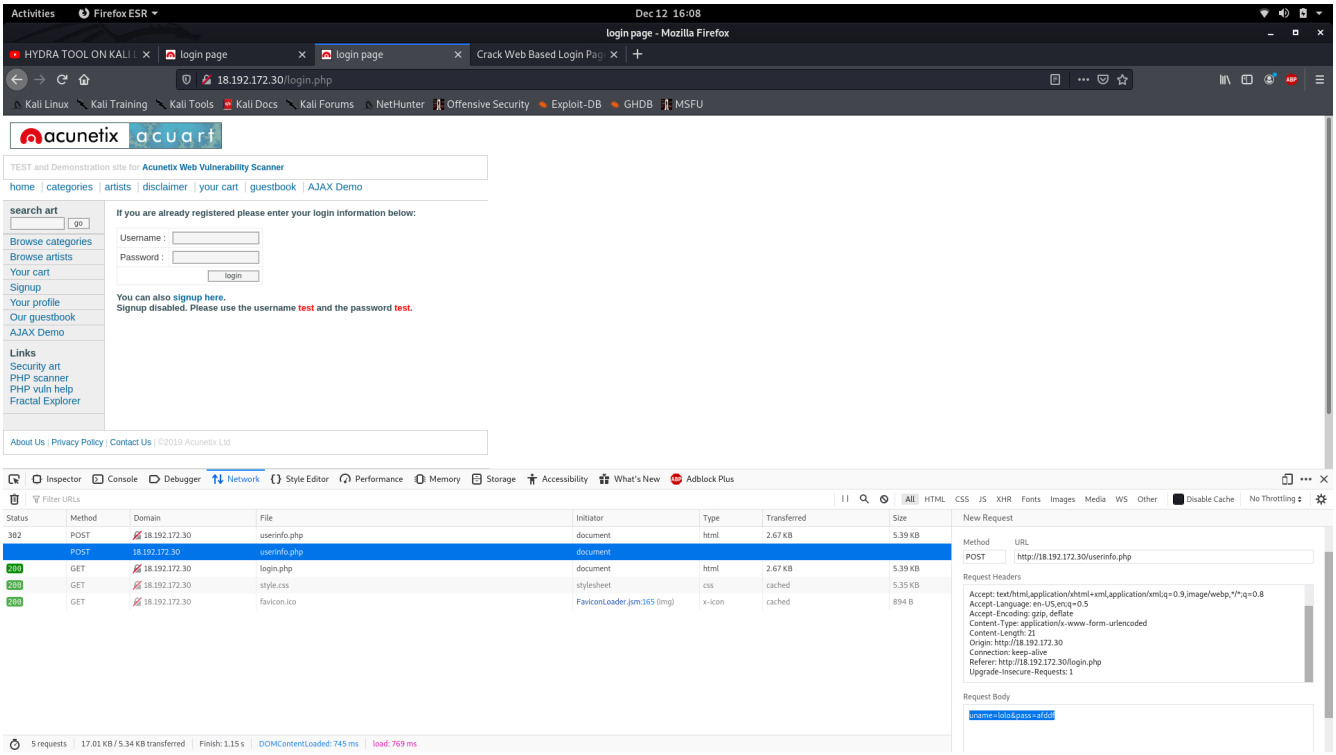
De acordo com o site owasp (https://owasp.org/www-community/controls/Blocking_Brute_Force_Attacks) ataques de força bruta são tentativas de descobrir uma senha tentando repetidamente uma combinação de números, símbolos e caracteres. A prevenção para esse tipo de ataque é limitar a quantidade de erros de acesso em uma conta durante um determinado período de tempo. Também palavras passe como perguntas de segurança ou captcha ajudam a barrar esse tipo de ataque automatizado, já que o software atacante utilizado não saberá como prosseguir. Outra solução também seria a autenticação por 2 ou mais fatores e por fim usuários utilizarem senhas não triviais.

Para iniciar o **ataque de força bruta** no site alvo ‘http://testphp.vulnweb.com/’ utilizamos a ferramenta **hydra** devido à sua facilidade no uso. Então primeiramente iniciamos o nmap para identificar o endereço ip do site:



ip: 18.192.172.30

Agora que já temos o ip, precisaremos de um login, uma lista de senhas, e precisaremos identificar como o site faz suas tentativas de acessar os logins e senhas internamente para informar ao hydra, pois a forma de ataque que utilizaremos (http post form) requer essa informação. Para identificar isso, foi acessada a página de login do site e, com o botão direito escolhido inspecionar elemento e selecionada a aba networking. Então inseri um login e senha qualquer e tentei logar. Nisso, apareceu uma opção nova en networking (identificada como POST, marcada de azul na imagem a seguir) a qual contém a parte do endereço url que utizaremos para nossas tentativas de acesso e nela temos a **request body** a qual também usaremos para o **hydra** saber onde inserir o login e senha.



Por último criei 2 documentos simples no formato txt. Um para inserir apenas um endereço de login (identificado como teste) e outro documento para inserir as possíveis senhas (coloquei 4 senhas como tentativa). Após isso, é só iniciar o software informando essas informações : ‘hydra -V -L user -P password -t 1 18.192.172.30 http-post-form “/userinfo.php:uname=^USER^&pass=^PASS^:S=logout”’

Sendo -V para nos mostrar as tentativas, -L para usarmos a lista txt de login e -P a lista de senhas, -t 1 uma tarefa, ou nesse caso tentativa de invasão, executada por vez, o endereço ip do site, o tipo de serviço a ser utilizado pelo **hydra**, a página de login do site e o **request body** do site.

```
lolo@lolo:~/Desktop$ hydra -V -L user -P password -t 1 18.192.172.30 http-post-form "/userinfo.php:uname=^USER^&pass=^PASS^:S=logout"
Hydra v9.0 (c) 2019 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal purposes.

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2020-12-12 16:19:53
[DATA] max 1 task per 1 server, overall 1 task, 5 login tries (l:1/p:5), ~5 tries per task
[DATA] attacking http-post-form://18.192.172.30:80/userinfo.php:uname=^USER^&pass=^PASS^:S=logout
[ATTEMPT] target 18.192.172.30 - login "test" - pass "1234" - 1 of 5 [child 0] (0/0)
[ATTEMPT] target 18.192.172.30 - login "test" - pass "qwert" - 2 of 5 [child 0] (0/0)
[ATTEMPT] target 18.192.172.30 - login "test" - pass "password" - 3 of 5 [child 0] (0/0)
[ATTEMPT] target 18.192.172.30 - login "test" - pass "test" - 4 of 5 [child 0] (0/0)
[80][http-post-form] host: 18.192.172.30   login: test   password: test
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2020-12-12 16:20:00
lolo@lolo:~/Desktop$
```

Pela imagem observamos que houve uma tentativa de invasão com sucesso na força bruta, nos informando que o login **test** com a senha **test** é uma forma de logar no site.

-Password Spray

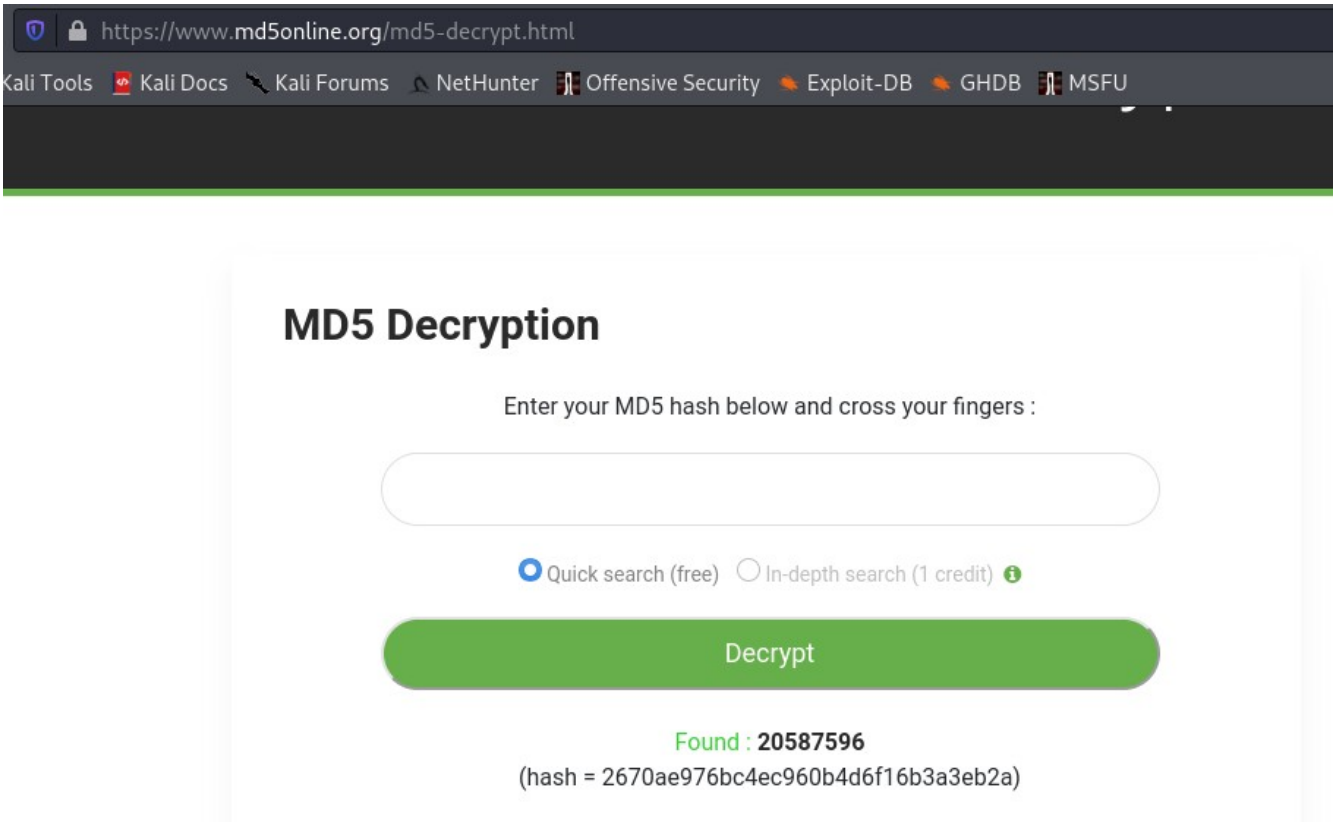
Este tipo de ataque em senha consiste em pegar uma lista de logins de diversos usuários e tentar adivinhar em qual deles uma determinada senha (ou um número bem limitado de senhas) conseguirá acesso. Para se prevenir desse tipo de ataque será preciso utilizar a autenticação por 2 ou mais fatores, utilização de senhas não triviais, utilizar captchas e perguntas passe.

Para realizar essa tentativa, basicamente utilizamos os mesmos comandos no hydra, acrescentando apenas um numero maior de de usuários no documento txt de logins e deixando apenas uma única senha no documento de passwords. Nesse caso tentamos 6 logins diferentes para a mesma senha (teste):

Observamos que mais uma vez foi um sucesso nossa tentativa nos dando o login **test** e senha **test** novamente.

-Força bruta no hash

Foi informado o seguinte **hash**: 2670ae976bc4ec960b4d6f16b3a3eb2a. Existem diversos tipos algoritmos usados, sendo MD5 e SHA-1 um dos mais simples. Então fiz uma pesquisa no google sobre algum site online para quebrar hashing e cliquei no primeiro que apareceu(md5online.org). Nele eu supuz que o hash dado era do tipo MD5 e então o inseri no site:

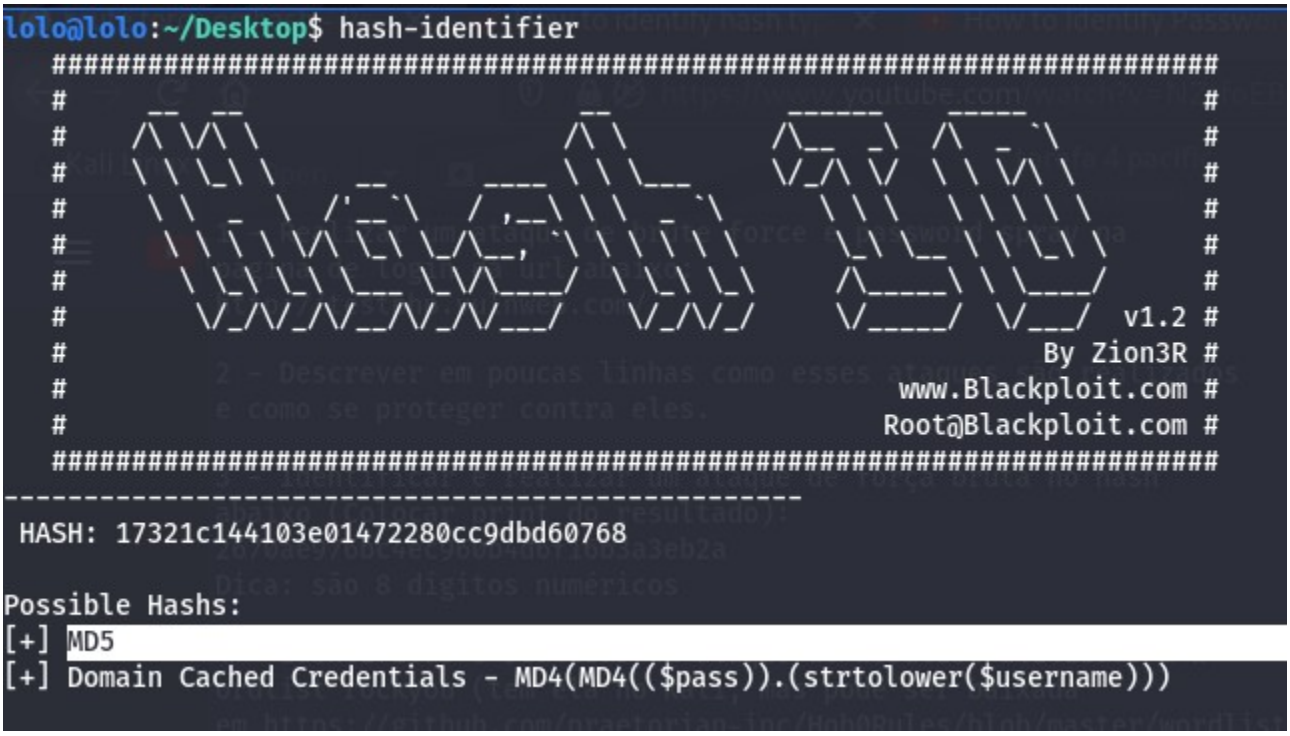


Como podemos observar ele me entregou diretamente a senha contida no hash: 20587596.

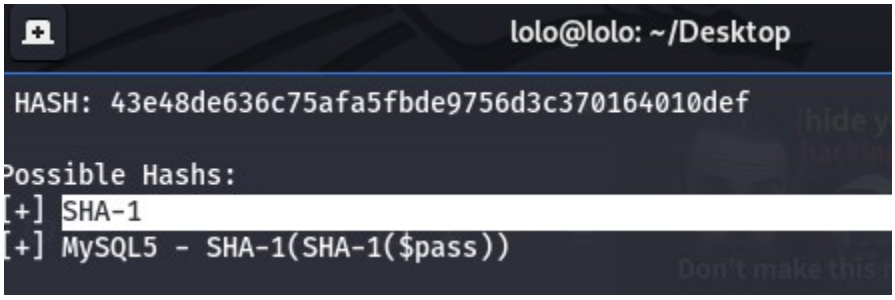
-Identificando e realizando ataque força bruta em 4 hashes

Uma vez que o atacante tenha acesso a uma das hashes de senha armazenadas em um banco de dados, ele poderá tentar descobrir essa senha sem precisar de acesso à internet (ataque offline).

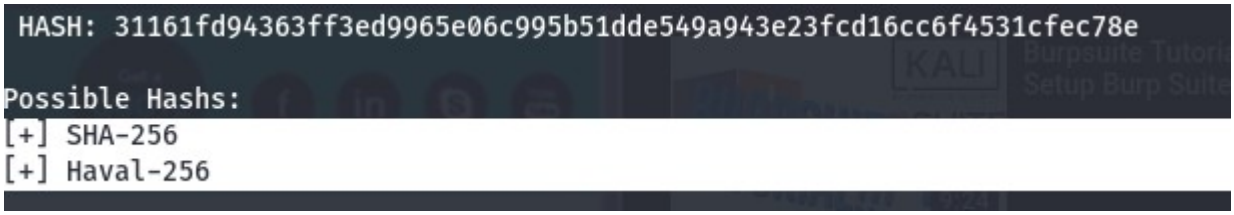
Foi nos dado 4 hashes com o objetivo de terem suas senhas descobertas. A partir disso precisaremos identificar qual função matemática pertence cada hash para iniciar o ataque de rainbow table.
A primeira hash foi nos dado sua função pertencente: 0676C65AC8CDD3FFA6CE0194C333DEE0 – **NTLM.** Nas outras três foi utilizado o software **hash identifier** para defenir quais seriam as hashes mais prováveis:



hash 2: 17321c144103e01472280cc9dbd60768 – **MD5**



hash 3: 43e48de636c75afa5fbde9756d3c370164010def – **SHA-1**



hash 4: 31161fd94363ff3ed9965e06c995b51dde549a943e23fcd16cc6f4531cfec78e – **SHA-256**

Depois de identificados, foi hora de iniciar a consulta nas rainbow tables. Utilizei o software **hashcat**. Nele foi preciso informar informar o modo -m, ou seja, qual função matemática (tipo de hash) devemos utilizar (variável de acordo com a tabela do hashcat); o modo de ataque -a, no caso 0 para straight, ou seja informar a hash alvo em um arquivo txt e compará-la com possíveis senhas antes do hash em outro arquivo txt (wordlist) e por fim as duas listas txt a serem comparadas, no caso o txt contendo o hash alvo e depois o txt rockyou padrão do kali o qual contém várias senhas sem hash algum:

hash 1: 0676C65AC8CDD3FFA6CE0194C333DEE0 – **NTLM**

```
lolo@lolo:~/Desktop$ hashcat -m 0 -a 0 hashes.txt rockyou.txt
hashcat (v6.1.1) starting...

Host memory required for this attack: 65 MB

Dictionary cache hit:
* Filename..: rockyou.txt
* Passwords.: 14344385
* Bytes.....: 139921507
* Keyspace..: 14344385

17321c144103e01472280cc9dbd60768:me211233*

Session..........: hashcat
Status...........: Cracked
Hash.Name........: MD5
Hash.Target......: 17321c144103e01472280cc9dbd60768
Time.Started.....: Sat Dec 12 17:49:10 2020 (2 secs)
Time.Estimated...: Sat Dec 12 17:49:12 2020 (0 secs)
Guess.Base.......: File (rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  2953.7 kH/s (0.43ms) @ Accel:1024 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests
Progress.........: 5672960/14344385 (39.55%)
Rejected.........: 0/5672960 (0.00%)
Restore.Point....: 5668864/14344385 (39.52%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#1....: me2xyz -> mdjmdj2

Started: Sat Dec 12 17:48:47 2020
Stopped: Sat Dec 12 17:49:14 2020
```

senha: me211233*

hash 2: 17321c144103e01472280cc9dbd60768 – **MD5**

```
lolo@lolo:~/Desktop$ hashcat -m 1000 -a 0 hashes.txt rockyou.txt
hashcat (v6.1.1) starting...
```

```
Dictionary cache built:
* Filename..: rockyou.txt
* Passwords.: 14344392
* Bytes.....: 139921507
* Keyspace..: 14344385
* Runtime...: 2 secs

0676c65ac8cdd3ffa6ce0194c333dee0:me211233*

Session..........: hashcat
Status...........: Cracked
Hash.Name........: NTLM
Hash.Target......: 0676c65ac8cdd3ffa6ce0194c333dee0
Time.Started.....: Sat Dec 12 17:42:31 2020 (2 secs)
Time.Estimated...: Sat Dec 12 17:42:33 2020 (0 secs)
Guess.Base.......: File (rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  3386.1 kH/s (0.36ms) @ Accel:1024 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests
Progress.........: 5672960/14344385 (39.55%)
Rejected.........: 0/5672960 (0.00%)
Restore.Point....: 5668864/14344385 (39.52%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#1....: me2xyz -> mdjmdj2

Started: Sat Dec 12 17:41:47 2020
Stopped: Sat Dec 12 17:42:34 2020
lolo@lolo:~/Desktop$
```

senha: me211233*

hash 3: 43e48de636c75afa5fbde9756d3c370164010def – **SHA-1**

```
lolo@lolo:~/Desktop$ hashcat -m 100 -a 0 hashes.txt rockyou.txt
hashcat (v6.1.1) starting...
```

```
Dictionary cache hit:
* Filename..: rockyou.txt
* Passwords.: 14344385
* Bytes.....: 139921507
* Keyspace..: 14344385

43e48de636c75afa5fbde9756d3c370164010def:me211233*

Session..........: hashcat
Status...........: Cracked
Hash.Name........: SHA1
Hash.Target......: 43e48de636c75afa5fbde9756d3c370164010def
Time.Started.....: Sat Dec 12 17:53:16 2020 (2 secs)
Time.Estimated...: Sat Dec 12 17:53:18 2020 (0 secs)
Guess.Base.......: File (rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  3434.6 kH/s (0.51ms) @ Accel:1024 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests
Progress.........: 5672960/14344385 (39.55%)
Rejected.........: 0/5672960 (0.00%)
Restore.Point....: 5668864/14344385 (39.52%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#1....: me2xyz -> mdjmdj2

Started: Sat Dec 12 17:52:53 2020
Stopped: Sat Dec 12 17:53:19 2020
```

senha: me211233*

hash 4: 31161fd94363ff3ed9965e06c995b51dde549a943e23fcd16cc6f4531cfec78e – **SHA-256**

```
lolo@lolo:~/Desktop$ hashcat -m 1400 -a 0 hashes.txt rockyou.txt
hashcat (v6.1.1) starting...
```

```
Dictionary cache hit:
* Filename..: rockyou.txt
* Passwords.: 14344385
* Bytes.....: 139921507
* Keyspace..: 14344385

31161fd94363ff3ed9965e06c995b51dde549a943e23fcd16cc6f4531cfec78e:me211233*

Session..........: hashcat
Status...........: Cracked
Hash.Name........: SHA2-256
Hash.Target......: 31161fd94363ff3ed9965e06c995b51dde549a943e23fcd16cc...fec78e
Time.Started.....: Sat Dec 12 17:55:49 2020 (3 secs)
Time.Estimated...: Sat Dec 12 17:55:52 2020 (0 secs)
Guess.Base.......: File (rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  2119.8 kH/s (0.96ms) @ Accel:1024 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests
Progress.........: 5672960/14344385 (39.55%)
Rejected.........: 0/5672960 (0.00%)
Restore.Point....: 5668864/14344385 (39.52%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#1....: me2xyz -> mdjmdj2

Started: Sat Dec 12 17:55:25 2020
Stopped: Sat Dec 12 17:55:53 2020
```

senha: me211233*

Dentre os 4 tipos de hash apresentados o mais recomendado seria o hash **SHA-256** por ser mais atual e possuir um tamanho de hash de 256 bits, contribuindo para diminuir ainda mais a chance de colisao, ou seja, a chance de haver duas senhas diferentes com o mesmo hash.

-Ataque de força bruta customizado

Foi dada a hash 885701fadda5534fd569dfe0699ae68b e sabemos que essa hash contém uma senha de 8 dígitos = Letra maiúscula/minúscula/números nos 3 primeiros dígitos. Símbolo no 4° digito e números nos 4 dígitos finais. Tendo isso em vista, o software **hashcat** é capaz de realizar uma tentativa de quebra de senha também só que utilizando a forma de ataque 3 brute-force na opção -a, a qual é capaz de selecionar opções individuais para cada dígito, já que sabemos o tamanho da senha e o leque de opções contito em cada posição dos dígitos.

Consultando a tabela do **hashcat** com o comando no terminal ‘hashcat --help’:

```
  ?  | Charset
 ===+=========
  l  | abcdefghijklmnopqrstuvwxyz
  u  | ABCDEFGHIJKLMNOPQRSTUVWXYZ
  d  | 0123456789
  h  | 0123456789abcdef
  H  | 0123456789ABCDEF
  s  |  !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
  a  | ?l?u?d?s
  b  | 0x00 - 0xff
```

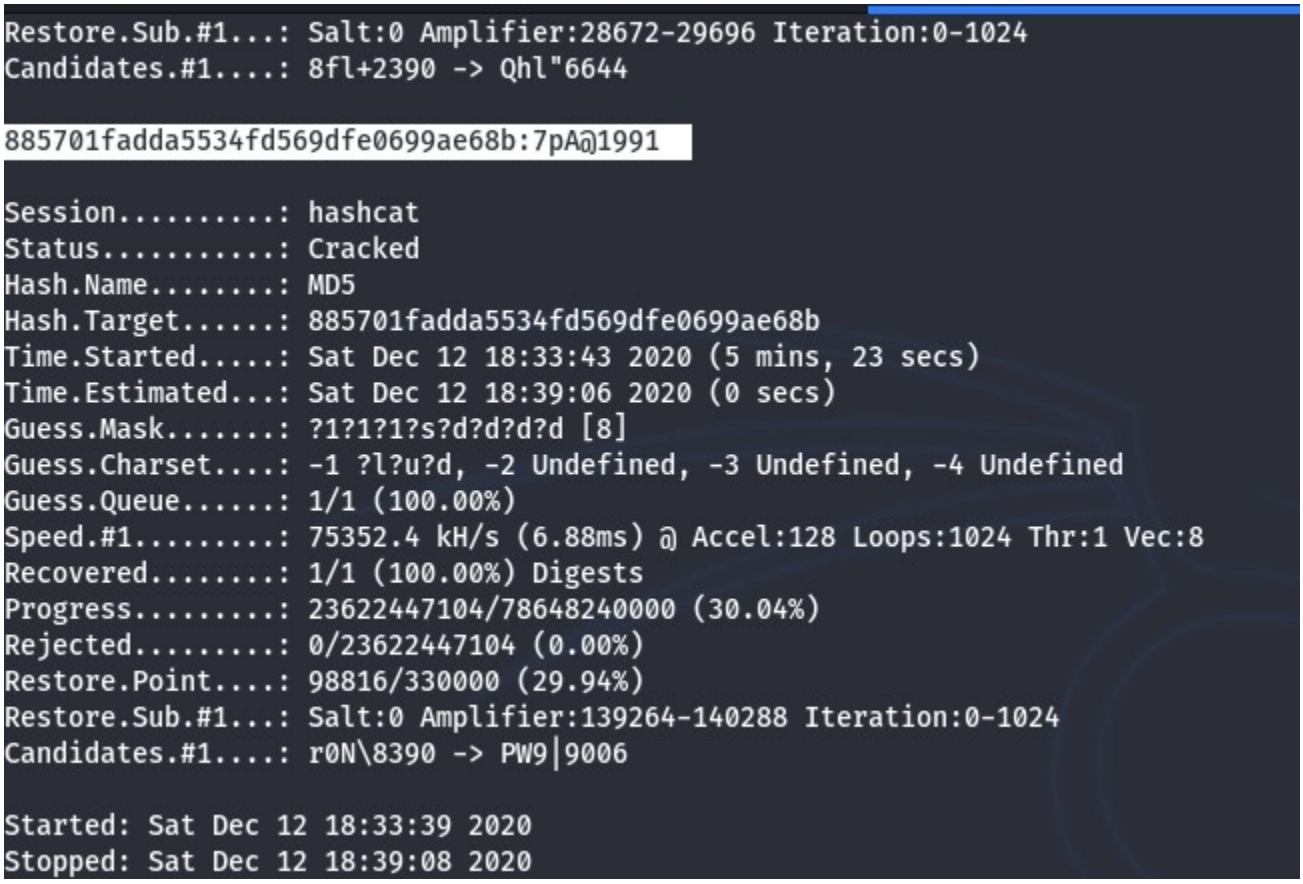
Saberemos qual opção usar para cada um dos dígitos. Também identificamos a provável hash usada (novamente com o hash-identifier):

```
lolo@lolo:~/Desktop$ hash-identifier
   #########################################################################
   #     __  __                     __           ______   _____           #
   #    /\ \/\ \                   /\ \         /\__  _\ /\  _ `\         #
   #    \ \ \_\ \     __      ____ \ \ \___     \/_/\ \/ \ \ \/\ \        #
   #     \ \  _  \  /'__`\   / ,__\ \ \  _ `\      \ \ \  \ \ \ \ \ \     #
   #      \ \ \ \ \/\ \_\ \_/\__, `\ \ \ \ \ \      \_\ \__ \ \_\ \_\ \    #
   #       \ \_\ \_\ \___ \_\/\____/  \ \_\ \_\     /\_____\ \ \____/     #
   #        \/_/\/_/\/__/\/_/\/___/    \/_/\/_/     \/_____/  \/___/  v1.2 #
   #                                                             By Zion3R #
   #                                                    www.Blackploit.com #
   #                                                   Root@Blackploit.com #
   #########################################################################
--------------------------------------------------
 HASH: 885701fadda5534fd569dfe0699ae68b

Possible Hashs:
[+] MD5
[+] Domain Cached Credentials - MD4(MD4(($pass)).(strtolower($username)))

Least Possible Hashs:
[+] RAdmin v2.x
[+] NTLM
```

Apontando para o MD5. Por isso informamos o seguinte para **hashcat** no terminal:
“hashcat -m 0 -a 3 hash.txt -1 ?l?u?d ?1?1?1?s?d?d?d?d” ondea hash alvo se encontra no arquivo hash.txt; a parte -1 ?l?u?d indica que onde estiver 1, o dígito pode der tanto l-letra minúscula, u-letra maiúscula, d-dígito; e a parte ?1?1?1?s?d?d?d?d a senha de oito dígitos sendo letra maiúscula/minúscula/números nos 3 primeiros dígitos. Símbolo no 4° digito e números nos 4 dígitos finais.



E então depois de decorridos +- 5 minutos descobrimos a senha do hash: **7pA@1991**.

- Reforço na segurança: Salt

Uma vez que o atacante tenha uma rainbow table prévia para usar como consulta em diversos serviços de criptografia, basta ter acesso a um banco de dados para ter uma chance de descobrir alguma senha, já que nem todo usuário faz uso de senhas fortes (grande diversidade de caracteres e combinações não triviais). Por esse motivo criou-se o **Salt:** um dado aleatório criado para ser fixado em uma senha antes que a mesma seja hasheada. Isso impede que hashes pré-computadas (as **rainbowtables**) sejam utilizadas em novos ataques, pois cria hashes diferentes sempre, para uma mesma senha.

Usarei de exemplo a **hash 4:**
31161fd94363ff3ed9965e06c995b51dde549a943e23fcd16cc6f4531cfec78e – **SHA-256.** Sabemos que a senha é **me211233*.** Em qualquer site de consulta de **SHA-256** o hash será o mesmo. Porém caso acrescentemos uma senha não trivial **!#45ac** (o banco de dados também a armazena juntamente com o hash para o processo de verificação de senha) ao final da senha, teremos: **me211233*!#45ac.** Assim criptografamos com o **SHA-256** e teremos a nova hash:
5c250135238f0955b91cd4c6b3db4f95802d7b9497ec1700e4a046a12ce046cd.