

گزارش

معرفی و بررسی آسیب پذیری:

ذخیره کردن هش پسورد بدون Salt یک ریسک امنیتی جدی است زیرا مهاجمین با استفاده از حملات دیکشنری می‌توانند کلمات دیکشنری (شامل پسوردهای لو رفته، اطلاعاتی درباره کاربران در صورت وجود، کلماتی از دیکشنری زبان‌های مختلف و ...) یا ترکیبات رایج پسوردها را امتحان کرده و آنها را با هش‌های سرقت شده مقایسه کنند. اگر مطابقت پیدا شود، آنها به رمز عبور دست می‌یابند. این کار بدون استفاده از Salt با سرعت بالاتری انجام می‌شود و افزودن Salt رندوم و خوب می‌تواند تا حد زیادی سرعت حمله را کاهش دهد. همچنین مهاجمین می‌توانند از جداول از پیش محاسبه شده‌ای که پسوردهای رایج را به مقادیر هش شده آنها نگاشت می‌کند (Rainbow Tables) استفاده کنند. اگر از Salt استفاده نشود، مهاجم به سادگی می‌تواند هش سرقت شده را با جدول رنگین مقایسه کند تا پسورد اصلی را پیدا کند که در این آزمایش به بررسی این موضوع پرداخته شده است.

نحوه پیاده سازی نقض تعریف شده:

برای پیاده سازی این آزمایش، یک صفحه ثبت نام کاربر ساده در نظر گرفته شده است که از طریق آن کاربر، نام کاربری و پسورد خود را وارد می‌کند. هش پسورد با استفاده از salt و pepper در دیتابیس s_users.txt و بدون استفاده از آن‌ها در دیتابیس users.txt به همراه نام کاربری و (salt در دیتابیس دیگر) ذخیره می‌شود. در این دیتابیس‌های فرضی، به منظور مقایسه پسورد رمزگشایی شده با پسورد اصلی، خود پسورد نیز نگهداری می‌شود. (تصویر 1)

انجام حمله به واسطه نقض تعریف شده:

در این قسمت، جدول Rainbow با استفاده از کتابخانه rainbowtables برای پسوردهای 4 رقمی متشکل از حروف کوچک انگلیسی و ارقام، پیاده سازی شده است. (پیاده سازی شده در فایل rainbow_tables.py) فرض بر این است که مهاجم به هش پسوردهای دیتابیس دسترسی پیدا کرده است و پسورد آن‌ها را با استفاده از جدول Rainbow جستجو می‌کند. در جدول Rainbow مکانیزم کلی به این صورت است که لیستی از پسوردهای پرکاربرد و رایج P در نظر گرفته می‌شود که روی هر یک از این پسوردها برای ایجاد زنجیره‌ها (chain) توابع هش و کاهش (reduction) صدا زده می‌شود. توابع کاهش می‌توانند به طرق مختلف نوشته شوند و الگوریتم یکتایی ندارد و به اطلاعات مهاجم از جایی که می‌خواهد حمله کند، برمی‌گردد، اما به طور کلی،

```

1  from django.shortcuts import render
2      import hashlib
3  import os
4
5      pepper = 'scrt'
6
7  def index(request):
8      return render(request, 'index.html')
9
10 def submit_data(request):
11     if request.method == 'POST':
12         username = request.POST.get('username')
13         password = request.POST.get('password')
14         hashed_passwords('users.txt', username, password)
15         salt_pepper_hashed_passwords('s_users.txt', username, password)
16     return render(request, 'welcome.html')
17
18
19 def salt_pepper_hashed_passwords(filename, username, password):
20     with open(filename, 'a') as file:
21         salt = os.urandom(16)
22         hash_value = hashlib.md5(salt + password.encode() + pepper.encode()).hexdigest()
23         file.write(f"{username},{password},{hash_value}, {salt}\n")
24
25
26 def hashed_passwords(filename, username, password):
27     with open(filename, 'a') as file:
28         hash_value = hashlib.md5(password.encode()).hexdigest()
29         file.write(f"{username},{password},{hash_value}\n")
30
31

```

تصویر 1

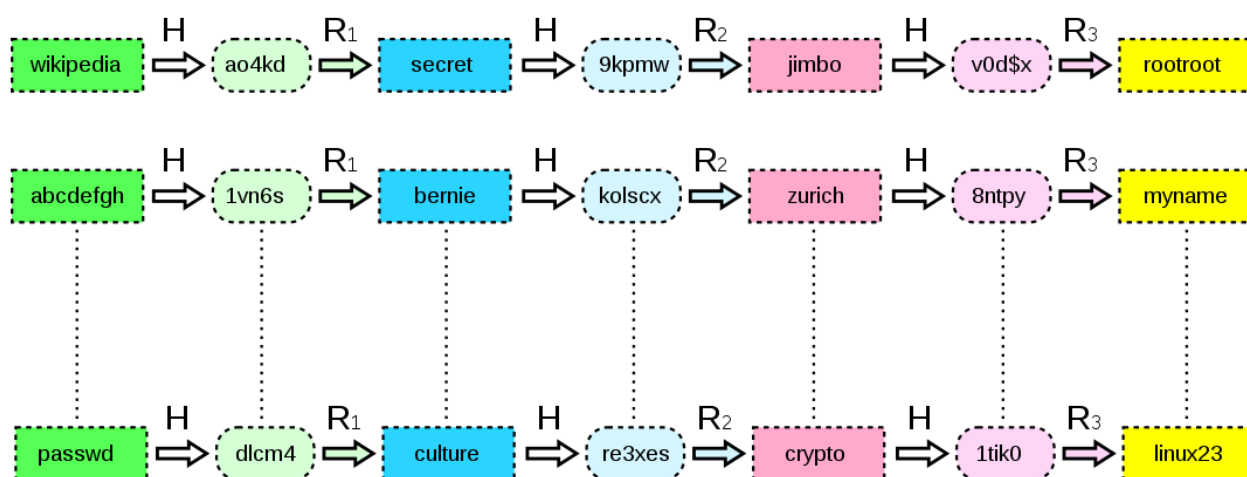
این توابع طوری نوشته می شوند که هش ها را به پسوردی جدید از P تبدیل می کنند. برای ایجاد جدول تعدادی از پسوردهای لیست P به صورت رندوم انتخاب می شوند و برای هر یک زنجیره ای به طول ثابت k ایجاد می شود و تنها اولین و آخرین پسورد زنجیره در جدول ذخیره می شود. برای مثال اگر P لیستی از پسوردهای 6 کارکتری باشد و تابع هش، هش های 32 بیتی تولید کند، خواهیم داشت:

aaaaaa \xrightarrow{H} 281DAF40 \xrightarrow{R} sgfnyd \xrightarrow{H} 920ECF10 \xrightarrow{R} kiebgt

مثال 1

و در جدول مقادیر (aaaaa, kiebgt) قرار می گیرند.

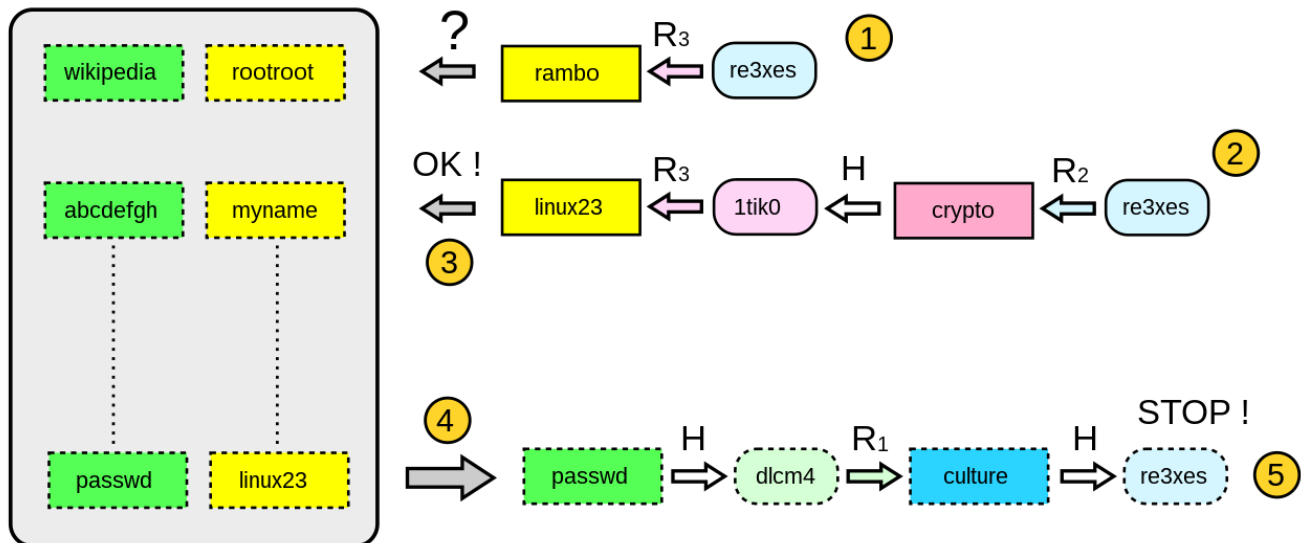
مثال زیر، مثالی دیگر از نحوه ایجاد جدول Rainbow با سه تابع کاهش می باشد.



مثال 2

برای برگرداندن هش re3xes و پیدا کردن پسورد مربوطه با استفاده از این جدول ، تابع کاهش سوم را برای re3xes صدا میزنیم و مقدار حاصل را در ستون آخر جدول جستجو می کنیم. اگر این مقدار پیدا نشد، تابع کاهش دوم را روی re3xes صدا زده، حاصل را هش می کنیم و سپس تابع کاهش سوم را روی هش حاصل مطابق تصویر زیر صدا می زنیم و مجددا مقدار حاصل را در ستون آخر جدول جستجو می کنیم.

اگر جستجو بی نتیجه بود، باید آنقدر تابع کاهش را صدا بزنییم تا مقدار حاصل در جدول مطابقت داده شود. اگر نتیجه جستجو مثبت بود، از مقدار ستون اول شروع کرده، آن را آنقدر هش می کنیم و سپس به ترتیب مثال 2 تابع کاهش را صدا می زنیم تا به مقداری برسیم که با هش مورد نظر ما یکسان است. یک مرحله قبل از، همان پسورد مورد نظر برای آن هش است که در این مثال پسورد مورد نظر culture می باشد.

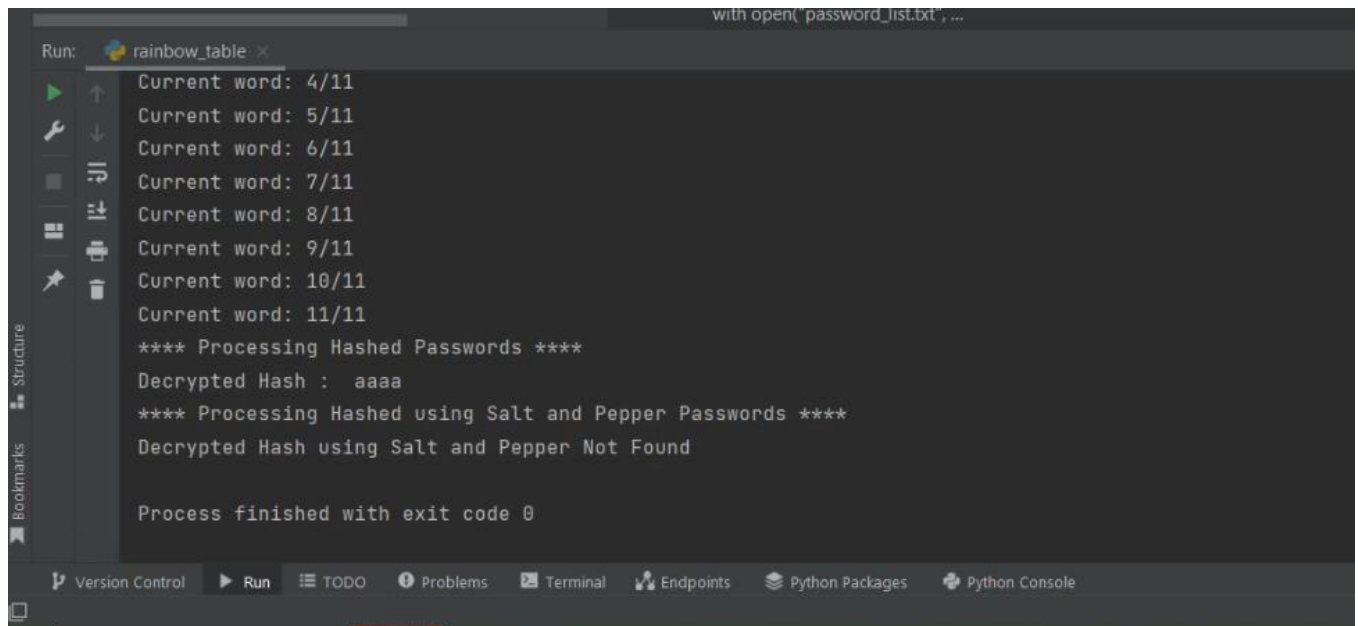


راه حل :

یکی از راه های مرسوم مقابله با حملات از طریق جدول Rainbow ، استفاده از salt های بزرگ و رندوم است. salt قبل از هش شدن پسورد هر کاربر به آن اضافه می شود و سپس هش می شود. در این صورت مهاجم به ازای هر salt به یک جدول Rainbow جدید نیاز دارد که اگر salt به اندازه کافی بزرگ باشد، باعث پیچیده شدن پسوردها می شود و کار مهاجم را سخت می کند همچنین این موضوع که هر کاربر از یک salt منحصر به فرد استفاده میکند، باعث می شود روند حمله به شدت کند شود. علاوه بر آن ممکن است برخی کاربران از پسوردهای یکسان و رایج مثل ("1234", "Password123") استفاده کنند، در صورت استفاده از salt ، هش پسوردهای آن ها یکسان نخواهد بود و پسورد های آن ها از امنیت بالاتری برخوردار خواهد بود.

راه حل دیگر استفاده از Pepper علاوه بر salt می باشد. Pepper نیز مانند salt یک مقدار رندوم می باشد، با این تفاوت که salt یک مقدار منحصر به فرد برای هر کاربر است اما Pepper برای همه کاربران موجود در دیتابیس یکسان است. این مقدار در دیتابیس ذخیره نمی شود و مخفی است.

همانطور که مشاهده می شود برای user1 با پسورد aaaa ، پسورد دارای salt و Pepper رمزگشایی نشد اما پسوردی که به تنهایی هش شده بود، رمزگشایی شد.



```
with open('password_list.txt', ...  
Run: rainbow_table x  
Current word: 4/11  
Current word: 5/11  
Current word: 6/11  
Current word: 7/11  
Current word: 8/11  
Current word: 9/11  
Current word: 10/11  
Current word: 11/11  
**** Processing Hashed Passwords ****  
Decrypted Hash : aaaa  
**** Processing Hashed using Salt and Pepper Passwords ****  
Decrypted Hash using Salt and Pepper Not Found  
  
Process finished with exit code 0  
Version Control Run TODO Problems Terminal Endpoints Python Packages Python Console
```

تصویر 2