Universidade Federal
de São João del-Rei

CURSO DE CIÊNCIA DA COMPUTAÇÃO

Relatório Trabalho Prático

Carlos Mito, Giovanni Filho, Marcos Fonseca

Trabalho da disciplina de Introdução à Criptografia com o objetivo de aplicar os conceitos e técnicas aprendidos na disciplina, aperfeiçoando os conhecimentos apresentados em aula.

SÃO JOÃO DEL-REI

JULHO DE 2023

Sumário

1 Introdução

Este trabalho prático tem como objetivo realizar um ataque de força bruta em um arquivo de senhas de usuários e encontrar o maior número de senhas. O arquivo contém informações sobre cada usuário, incluindo o seu nome, um valor de sal aleatório (em hexadecimal) e o hash da combinação do sal com a senha (também em hexadecimal). A função de hash utilizada foi a SHA256. Portanto, é necessário gerar as senhas candidatas para cada usuário, calcular os hashes correspondentes e então compará-los com aqueles fornecidos no arquivo.

As senhas dos usuários possuem limitações, elas podem ter no máximo 16 caracteres, podendo incluir letras maiúsculas e minúsculas, dígitos e os caracteres especiais @, *, $ e %. Dessa forma, o conjunto válido de caracteres tem um total de 66 caracteres, e o número total de possibilidades será: $\sum_{n=1}^{16}(66^n)$. Abaixo está uma tabela com o número de caracteres e a quantidade de possibilidades:

Quantidade de caracteres	Número total de possibilidades
1	66
2	4422
3	291918
4	19266654
5	1271599230
6	83925549246
7	5539086250302
8	365579692519998
9	24128259706319934
10	1592465140617115710
11	105102699280729636926
12	6936778152528156037182
13	457827358066858298454078
14	30216605632412647697969214
15	1994295971739234748065968190
16	131623534134789493372353900606

A quantidade de caracteres cresce exponencialmente, dessa forma não é possível testar todas as possibilidades em tempo hábil. Por isso, é necessário encontrar maneiras eficientes de calcular e comparar hashes assim como testar as senhas mais prováveis de serem utilizadas.

Na descrição do problema, também foram vazadas informações sobre as senhas de alguns dos usuários: A senha do usuário Admin é 12345678; Somente as senhas dos usuários user2, user4, user5, userB e userD utilizam caracteres especiais; A menor senha, do usuário userD, tem apenas 4 caracteres.

2 Metodologia

Dado o grande número de possibilidades para o problema, foram utilizadas wordlists com prováveis senhas disponibilizadas pelo site (Weakpass, 2023). Foram baixados ao total cerca de 300GB de wordlists disponíveis no site, sendo as principais: All-in-One-P, weakpass_3a e kaonashi. A quantidade de senhas nas wordlists ultrapassa os 20 bilhões. Um problema ao utilizar as wordlists é que as senhas disponibilizadas não necessariamente são válidas para o problema em questão.

Dado o grande número de senhas para se testar, foi utilizada a GPU por meio da API CUDA para obter uma melhor performance para os testes das senhas de cada usuário. Para a execução do código foi utilizada uma placa de vídeo NVIDIA GeForce GTX 1050 com 4GB de memória GDDR5 e um clock de 1354 MHz.

3 Resultados

Para ler as senhas dos arquivos das wordlists foi criado um código em C que lê em pedaços de 1.5GB e realiza um pequeno processamento para resolver o problema dos caracteres indesejados. Dessa forma, ao encontrar um caractere que não pertence ao conjunto de caracteres válidos um aleatório é sorteado para a substituição.

Além disso, as senhas não foram testadas sequencialmente e separadamente. Isto é, na leitura do arquivo as quebras de linhas são removidas fazendo com que as senhas estejam contíguas na memória. A Figura 1 tem um exemplo de como as senhas admin, 12345678 e pass são organizadas na memória. Além disso, essa estrutura de armazenamento reduz a quantidade de memória tendo em vista que as quebras de linha não são armazenadas, aumenta a velocidade de acesso dentro das threads tendo em vista que é um acesso contíguo e também possibilita que as senhas sejam testadas de forma contínua o que possibilita um melhor uso das wordlists.

A Figura 1 mostra a estratégia de execução onde as senhas são armazenadas de forma contígua e são utilizadas `sliding windows` para a execução em cada thread. Pelo threadId é possível saber a posição de início da memória que será utilizada por cada thread e daí são realizados os testes iniciando com um tamanho $min_p = 4$ e terminando em $max_p = 16$, tendo em vista que a menor senha possui 4 caracteres. Assim, em cada execução do kernel CUDA cada thread executa 12 checagens de possíveis senhas para apenas um usuário. Isso porque, ao executar o kernel o número de blocos CUDA são definidos de acordo com o número de senhas que ainda não foram quebradas, e a identificação de qual usuário será testado por thread é identificado pelo blockId.

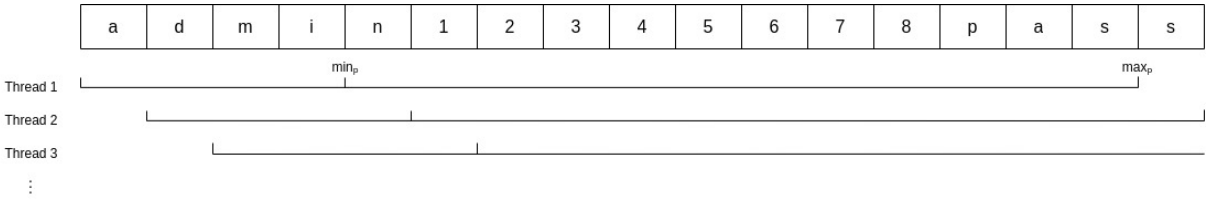

Figura 1: Estrutura das senhas na memória.

O código CUDA foi baseado em (tanmayb123, 2020), porém algumas otimizações foram feitas. Uma delas foi na remoção de códigos de checagem para a criação dos blocos do algoritmo SHA256, isso porque os blocos são criados a partir de 64 bytes de dados. Porém, sabe-se que o sal tem um tamanho fixo de 16 bytes e o tamanho máximo de cada senha é de 16 bytes. Sendo assim, todas as possibilidades serão menores do que o tamanho necessário para a criação de um novo bloco, dessa forma, as checagens que são feitas a cada atualização do estado interno foram removidas. Além disso, para uma melhor performance também houve a redução de uso dos registradores dentro da função do kernel, pois é um fator limitante para o número de threads que executam ao mesmo tempo. Para isso, variáveis desnecessárias foram removidas e foi feito o `loop unrolling` dos loops que tinham limites constantes.

Tabela 1: Relação de usuários e senhas encontradas.

Usuário	Senha encontrada
Admin	12345678
user0	abcDef
user1	-
user2	1@2@3@4@
user3	passworD
user4	p@ssw0rd
user5	-
user6	12081786
user7	mysecretPassword
user8	00000000
user9	AbCdEfGh
userA	HarryP0tter
userB	M3t@llic@
userC	hgh356
userD	6%Fg

Para o cálculo do tempo de execução foi utilizado um contador para o total de senhas testadas que foi dividido pelo tempo total, com as otimizações feitas chegou-se a uma performance acima de 500MH/s. Para processar os 105GB do arquivo de senhas weakpass_3a, foram gastas aproximadamente 1 hora e meia. Durante a criação do programa, algumas senhas foram quebradas durante alguns testes, por isso não tem um registro exato do tempo gasto para cada senha separadamente. Na Tabela 1 está a relação de usuários e as senhas que foram encontradas, 2 senhas não foram encontradas, a do user1 e do user5.

4 Conclusão

Durante o ataque de força bruta, conseguimos quebrar 13 das 15 senhas propostas com sucesso. Esse resultado foi alcançado através do uso de wordlists e da computação paralela da GPU. Enviamos o código implementado juntamente com as senhas que foram decifradas.

Uma possível melhoria na performance do código que não foi testada consiste em realizar múltiplas checagens dos usuários por cada thread, permitindo a reutilização do valor de hash calculado. Atualmente, cada thread verifica apenas a senha de um usuário por vez, o que pode ser otimizado com essa abordagem de múltiplas checagens por thread.

Referências

[tanmayb123, 2020]TANMAYB123. *SHA256 CUDA implementation*. 2020. Disponível em: <https://github.com/tanmayb123/redesigned-train/blob/master/sha256.cuh>.

[Weakpass, 2023]WEAKPASS. 2023. Disponível em: <https://weakpass.com/>.