Gabriel Diniz Gisoldo **RA: 22214007-1**

Relatório atividade 4 - Projeto de curso

Introdução

Esse relatório é referente ao projeto de curso, onde foi solicitado a conversão de uma base de logins e senhas em *plaintext* para uma base, no mesmo formato, mas com as senhas em *cyphertext*.

O projeto foi realizado em Python 3.8.5, usando Cython para compilar o código em C, com o objetivo de executar as funções em um melhor tempo.

Foram usados três (3) algoritmos para geração do *cyphertext*, sendo eles:

1. SHA256
2. SCrypt
3. Base64 Encode/Decode

Todo o programa, assim como as instruções de uso, está disponibilizado no GitHub pelo link: https://github.com/huine/seguranca-informacao-fei

Tempo gasto para converter a base fornecida

Para realizar a conversão da base para o *cyphertext* foram necessários 1280,2291215 segundos ou 21,20 minutos.

A conversão dos trezentos mil itens foi dividida entre 16 *threads* para converter de forma concorrente cada um dos itens. Essa divisão foi feita com o intuito de diminuir o tempo para a conversão, distribuindo a carga entre várias *threads* do processador ao invés de uma única *thread*.

Aumento no tempo média para fazer a autenticação de um usuário

Para o teste da validação, foi criada uma função que valida a base inteira comparando o *plaintext* da base inicial com o *cyphertext* gerado pela conversão.

A validação da base também foi dividida em 16 *threads* para verificar cada um dos itens. No total, foram 1283,908427202 segundos ou 21,23 minutos de execução.

Foi criada uma outra função para realizar a validação de um único usuário, primeiro com a senha em *plaintext* e depois com a senha em *cyphertext*. Através dessa função pode-se observar um pequeno aumento no tempo médio

para a validação de um usuário, tendo um aumento de 38,19%, indo de 0,1327226500 segundos em *plaintext* para 0,1834028933 segundos em *cyphertext*.

Detalhamento dos algoritmos e lógicas utilizadas para construir as funções disponibilizadas

Foram construídos 3 módulos para o programa.

O *orquestrador.py* é responsável pela interface de linha de comando, onde é possível passar os parâmetros e escolher o modo de execução.

O c*onversor.pyx* é o módulo responsável pela criação das *threads*, por ler e escrever no disco e calcular os tempos de execução.

O *encoder.pyx* é responsável por gerar o *cyphertext* e por validar se a senha é válida.

Gerar o *cyphertext*:

1. Obter a senha em plaintext
2. Gerar um *hash* a partir da senha usando o algoritmo *SHA256*, limitando o tamanho da senha em 256 bits para evitar problemas na criptografia e no armazenamento.
3. Convertendo a *hash* de binário para base 64 para ter uma *string*.
4. Geramos um *salt* aleatório de 16 bytes.
5. Passamos a *hash* em base 64 para o algoritmo *SCrypt*, que vai derivar uma chave de 64 bytes a partir da *hash*, passando também o *salt*.
6. Convertemos essa chave gerada pelo *SCrypt* para base 64
7. Formatamos a saída no formato *{n}${r}${p}${salt}${chave}*, onde n, r e p são variáveis de configuração do *SCrypt*. (Foram usados os valores recomendados em RFC7914).

Validar uma senha:

1. Recebemos a senha em *plaintext* e o *cyphertext*.
2. Quebramos o *cyphertext*, usando o símbolo $ como marcador, para conseguir a configuração do *SCrypt* usada na geração do *cyphertext*.
3. Aplicamos o mesmo processo de geração do *cyphertext* usando a configuração extraída no passo anterior e o *plaintext* recebido.
4. Comparamos a saída do passo anterior com o *cyphertext* recuperado da base. Essa comparação é realizada *bit* a *bit*, sem interromper mesmo que exista diferença, para evitar ataques de tempo no processo de validação.

Converter a base:

1. Abrimos o arquivo e transformamos em uma lista.
2. Para cada item da lista a linha é dividida, usando | como marcador.

3. Criamos uma estrutura de dados chamada *User*, com 3 atributos: *login*, *pwd* e *hash*.
4. Transformamos a lista de linha em uma lista de objetos *User*.
5. Dividimos a lista em n partes iguais, onde n é a quantidade de *threads* disponíveis para o processo.
6. Cada parte da lista é entregue para uma *thread* diferente.
7. Cada *thread* executa um *loop* onde é gerado o *cyphertext* para cada item na parte da lista recebida.
8. Ao final é retornado essa lista com o *cyphertext*
9. Quando todas as *threads* finalizam o seu processo, o *output* é escrito em disco.

Validar a base:

1. Abrimos o arquivo de output gerado com o parâmetro *-p* do *orquestrador*
2. Transformamos o arquivo em uma lista
3. Para cada item da lista a linha é dividida, usando | como marcador.
4. Criamos uma estrutura de dados chamada *User*, com 3 atributos: *login*, *pwd* e *hash*.
5. Transformamos a lista de linha em uma lista de objetos *User*.
6. Dividimos a lista em n partes iguais, onde n é a quantidade de *threads* disponíveis para o processo.
7. Cada parte da lista é entregue para uma thread diferente.
8. Cada *thread* executa um *loop* onde é realizado o processo de validar uma senha.
9. Se, em qualquer momento, uma validação falhar, o programa levantará uma exceção e para a execução.

Para a criptografia foi usado o algoritmo *SCrypt* pois ele é altamente resistente a ataques de força bruta, uma vez que você pode configurar o esforço de CPU e memória necessários para derivar a chave. Outro motivo para o uso desse algoritmo é que ele tem uma implementação nativa na linguagem, sem necessidade de instalar bibliotecas adicionais.

Os parâmetros usados na configuração do SCrypt foram: N = 16384; R = 8; P = 1; *dklen* = 64; Onde N indica o custo de CPU/memória; R indica o tamanho dos blocos; P indica o paralelismo e *dklen* indica o tamanho da chave em bytes.

Usar o SHA256 não seria o suficiente?

Não, infelizmente não seria. Embora o *SHA256* seja muito mais rápido para calcular, essa velocidade causa problemas quando se trata do armazenamento de credenciais. Uma vez que o algoritmo pode ser implementado diretamente em hardware, ele pode ser atacado por força bruta e como não existe um método para alterar o esforço do cálculo da hash nem é possível alterar o número de iterações, isso faz com que ele seja vulnerável a ataques de dicionário, como a *Rainbow Table*.

Link para a base convertida

A base convertida, assim como todo o programa, pode ser encontrada no GitHub através do link: https://github.com/huine/seguranca-informacao-fei

O link direto para a base convertido é: https://raw.githubusercontent.com/huine/seguranca-informacao-fei/main/base-output.txt

Referencias:

Tarsnap - https://www.tarsnap.com/scrypt.html

Stronger Key Derivation Via Sequential Memory-Hard Functions (Colin Percival) - https://www.tarsnap.com/scrypt/scrypt.pdf

RFC7914 - https://datatracker.ietf.org/doc/html/rfc7914

Practical Cryptography for Developers (Svetlin Nakov, PhD) - https://cryptobook.nakov.com/mac-and-key-derivation/scrypt

Python Docs - https://docs.python.org/3.8/library/hashlib.html