DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Relatório do trabalho de Segurança de Aplicações e Dados

Aluno:
1210510 Rafael Magalhães

Docente(s)/Orientador(es):
Marcelo Santos,
Vladir
Unidade Curricular
Segurança de Aplicações e Dados

[25 de Novembro, 2023]

isep Instituto Superior de Engenharia do Porto

Breaking Enigma

O programa consiste em uma implementação modificada da cifra de Caesar. Ele utiliza uma técnica de bruteforce combinada com métodos de criptoanálise para decrifar a senha.

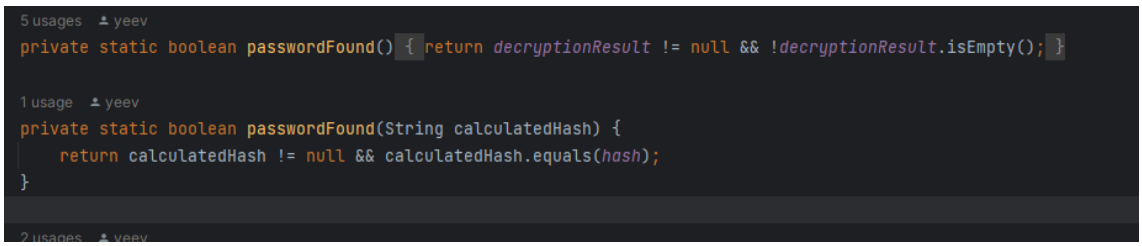
O programa busca criar e testar várias possibilidades de palavras e "salts" com base na configuração do "plugboard" e nos parâmetros fornecidos, tentando encontrar a combinação correta que resultará no hash específico da senha fornecida.

```
public class BreakingEnigma {

    4 usages
    private static String hash;
    2 usages
    private static final Map<String, String> plugboard = new HashMap<>();
    3 usages
    private static String decryptionResult;
    4 usages
    private static final String enigmaAlphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

    no usages  Yevgraf +1
    public static void main(String[] args) throws IOException {
        if (args.length < 3) {
            System.out.println("Usage: java -jar BreakingEnigma.jar <hash> <plugboard_config> <wordlist_filepath>");
            System.out.println("Example: java -jar BreakingEnigma.jar 6e8c4b3a71543af80890dd501a70e030f0a1f867631175f7440a4599041d52e3 \"{'N': 'Q', 'S': '
            return;
        }

        hash = args[0];
        parsePlugboardConfig(args[1]);
        String wordlistFilePath = args[2];
        List<String> wordlist = readWordlist(wordlistFilePath);
```

O programa recebe 3 argumentos na linha de comandos a hash,plugboard e a wordlist.

O parsePlugboardConfig() vai analisar e configurar o plugboard fornecido nos argumentos e armazena a informação num map.

readWordlist() função lê a wordlist, elimina potenciais espaços.



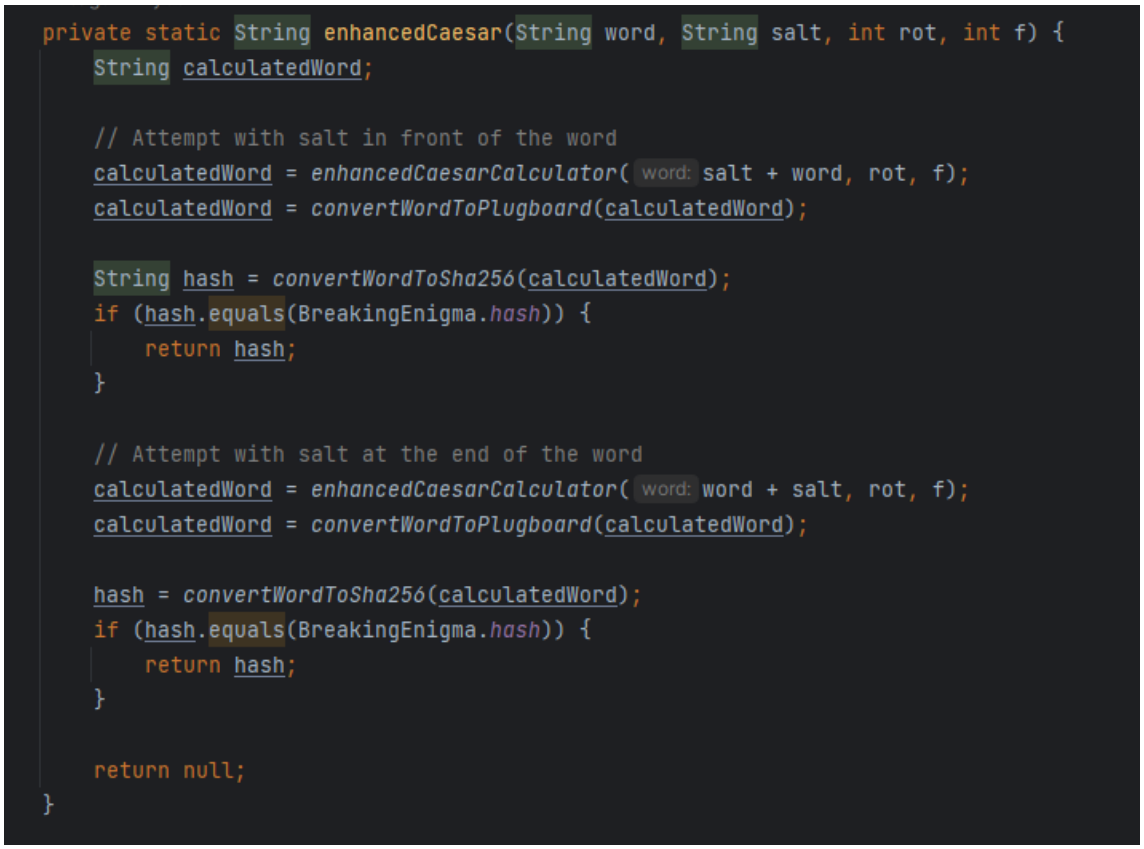
passwordFound() x2 são funções auxiliares para verificarem se senha foi encontrada e verica se decryptionResult não é nulo.

```java
private static String convertWordToSha256(String word) {
    try {
        java.security.MessageDigest digest = java.security.MessageDigest.getInstance( algorithm: "SHA-256");
        byte[] hashBytes = digest.digest(word.getBytes());
        StringBuilder hexString = new StringBuilder();
        for (byte hashByte : hashBytes) {
            String hex = Integer.toHexString( i: 0xff & hashByte);
            if (hex.length() == 1) hexString.append('0');
            hexString.append(hex);
        }
        return hexString.toString();
    } catch (java.security.NoSuchAlgorithmException e) {
        e.printStackTrace();
    }
    return null;
}
```

ConvertWordToSha() Aplica a função de hash a uma palavra criada pelo pela substituição em convertWordToPlugboard()

```java
4 usages  Yevgraf +1
private static String convertWordToPlugboard(String word) {
    StringBuilder plugWord = new StringBuilder();
    for (char w : word.toCharArray()) {
        String plugboardChar = plugboard.getOrDefault(String.valueOf(w), String.valueOf(w));
        plugWord.append(plugboardChar);
    }
    return plugWord.toString();
}
```

convertWordToPlugboard() é reponsavel por aplicar a configuração do plugboard a uma palavra



```
private static String enhancedCaesar(String word, String salt, int rot, int f) {
    String calculatedWord;

    // Attempt with salt in front of the word
    calculatedWord = enhancedCaesarCalculator( word: salt + word, rot, f);
    calculatedWord = convertWordToPlugboard(calculatedWord);

    String hash = convertWordToSha256(calculatedWord);
    if (hash.equals(BreakingEnigma.hash)) {
        return hash;
    }

    // Attempt with salt at the end of the word
    calculatedWord = enhancedCaesarCalculator( word: word + salt, rot, f);
    calculatedWord = convertWordToPlugboard(calculatedWord);

    hash = convertWordToSha256(calculatedWord);
    if (hash.equals(BreakingEnigma.hash)) {
        return hash;
    }

    return null;
}
```

enhancedCaesar() utiliza o algoritmo caesar modificado com variações da rotação e um salt. A função chama enhancedCaesarCalculator() duas vez pois o salt pode ser no inicio ou final da palavra. Após calcular a palavra cifrada com essas variações, converte a palavra usando o plugboard e gera o seu hash256. Compara o hash criado ao hash fornecido e retorna o hash correspondente.

```java
private static String enhancedCaesarCalculator(String word, int rot, int f) {
    StringBuilder calculatedWord = new StringBuilder();

    for (int i = 0; i < word.length(); i++) {
        char currChar = word.charAt(i);
        int currIndex = enigmaAlphabet.indexOf(currChar);

        if (currIndex == -1) {
            calculatedWord.append(currChar);
            continue;
        }

        int newIndex = (currIndex + rot + (i * f)) % enigmaAlphabet.length();
        if (newIndex < 0) {
            newIndex += enigmaAlphabet.length();
        }

        char newChar = enigmaAlphabet.charAt(newIndex);
        calculatedWord.append(newChar);
    }

    return calculatedWord.toString();
}
```

enhancedCaesarCalculator() função implementa a lógica central da Caesar modificada.

Percorre cada letra na palavra fornecida aplicando a cifra caesar modificada com base na rotação e no índice do caractere multiplicando por um fator (f), criando variações na criptografia para cada letra do alfabeto. O objetivo é gerar possíveis senhas a serem verificadas e relação ao hash fornecido

```java
int rot = -1;
while (!passwordFound()) {
    rot++;
    for (String word : wordlist) {
        String plugWord = convertWordToPlugboard(word);

        if (passwordFound()) break;

        for (char firstSaltChar : "!#%&*+-:;=?@".toCharArray()) {
            for (char secondSaltChar : "!#%&*+-:;=?@".toCharArray()) {
                String salt = convertWordToPlugboard(String.valueOf(firstSaltChar) + secondSaltChar);
                System.out.println("Salt: " + salt + "    Rotation: " + rot + "    Word: " + word);

                for (int f = 0; f < plugWord.length() ; f++) {
                    String passwordHash = enhancedCaesar(plugWord, salt, rot, f);

                    if (passwordFound(passwordHash)) {
                        decryptionResult = word;
                        System.out.println("Password Found: " + word + "    Salt: " + salt + "    Rotation: " + rot + "    Hash: " + passwordHash);
                        break;
                    }
                }

                if (passwordFound()) break;
            }

            if (passwordFound()) break;
        }

        if (passwordFound()) break;
    }
}
```

MainLoop

É responsável por tentar decifrar a password. Utiliza nested loops para iterar a lista de palavras e aplica todas as combinações possíveis de 2 caracteres do alfabeto de salt.

A rotação começa a -1. O programa corre até encontrar uma senha e incrementa o valor da rotação quando começa a repetir a lista de novo.

```
Password Found: MARIKO    Salt: -!    Rotation: 2
```