Trabalho 3 INE5611

Alexis Armin Huf

6 de junho de 2017

1 Introdução

Após os cases de sucesso de sua equipe no mercado internacional, o MSF (Ministério do Super Faturamento) entrou em contato para continuar a produção de um software de suma importância ao financiamento da campanha a deputado do atual ministro. O objetivo do ministério é fazer o cálculo de uma assinatura SHA-256 da maneira mais superfaturada possível.

Uma empresa já terceirizou e entregou um programa `signer` que faz esse cálculo, em C. Esse programa utiliza mapeamento de memória e exige que o arquivo mapeado tenha um layout específico. O código fonte desse programa é fornecido para sua equipe.

Sua equipe deve criar um programa em Java chamado `signer-client` que receba um arquivo qualquer como argumento via linha de comando e utilize o `signer` para calcular seu SHA-256. Para esse fim, vocês devem **obrigatoriamente utilizar memória mapeada e comunicação via pipes** com o processo `signer`.

2 Detalhes

O `signer-client` deve criar um arquivo temporário com o conteúdo esquematizado na Figura 1. O conteúdo do arquivo fornecido como argumento de linha de comando deve ser usado para preencher a região identificada como *payload*, sendo que na região *size* deve ser escrito um inteiro (big endian, como é o default para `MappedByteBuffer`) contendo o número de bytes do *payload*. O processo *signer* também mapeia esse arquivo em memória e escreve a assinatura SHA-256 na região identificada como *hash*. Após o `signer` indicar que terminou (`SIGN OK` em seu `stdout`), o `signer-client` deve imprimir em sua saída o assinatura codificada em base64. Para codificação da assinatura (sequência de bytes) em base64 pode ser utilizada a classe `java.util.Base64`, como mostrado na Figura 2.

Um esqueleto do `signer-client` foi fornecido. A única classe implementada é `SignerClient` que implementa o protocolo de comunicação a ser utilizado com o processo `signer`. Essa classe recebe no construtor um stream de saída (conectado ao stream de entrada do `signer`, `stdin`) e um stream de entrada, conectado ao stream de saída do `signer`, `stdout`. Um exemplo de uso da classe `SignerClient` é mostrado na Figura 3

O `pom.xml` já está configurado para gerar um über-jar com a classe `App` sendo a `mainClass`. Observe na Figura 1 que o `signer-client` deve receber dois argumentos, o primeiro sendo o caminho para o binário do programa `signer` e o segundo sendo o arquivo cuja assinatura SHA-256 deve ser calculada. Você pode incluir dependências adicionais disponíveis em `http://repo.jenkins-ci.org/`, mas não deve haver necessidade (**justifique o uso de bibliotecas adicionais em um README**).

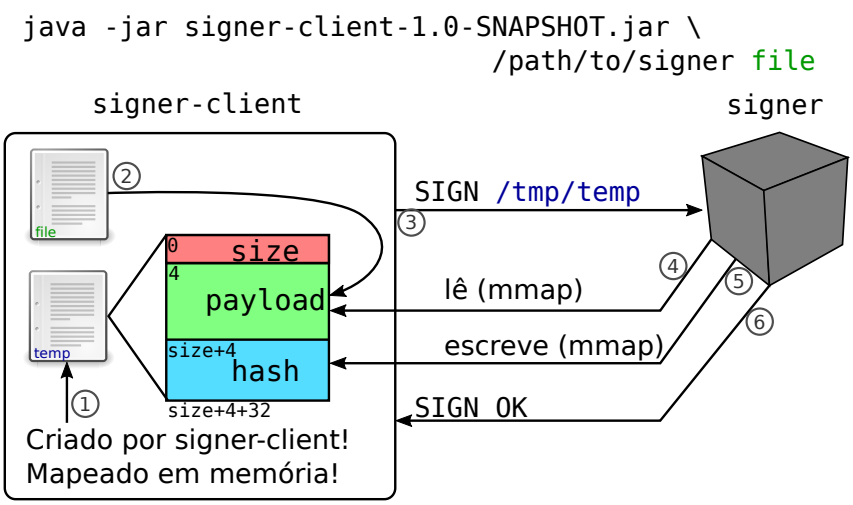

Figura 1 – Overview do hashing superfaturado.

```java
byte[] signature = new byte[32];
System.out.println(Base64.getEncoder().encodeToString(signature));
```

Figura 2 – Transformando um `byte[]` em uma string base64.

```java
SignerClient c = new SignerClient(process.getOutputStream(), process.getInputStream());
c.sign(new File(path));
```

Figura 3 – Exemplo de uso da classe `SignerClient`

2.1 Ingredientes

- *Mapeamento de memória*: **OBRIGATORIAMENTE** deve ser utilizada a classe `MappedByteBuffer`;
- *Criação de processo*: Recomenda-se `ProcessBuilder`;
- *Criação de arquivo temporário*: Recomenda-se `Files.createTempFile`;
- *Número de bytes de um arquivo*: Recomenda-se `File.length`.

2.2 Prova Real

Para confirmar que seu programa está funcionando corretamente, compare o hash lido da memória mapeada com o hash calculado sem superfaturamento. A Figura 4 mostra um método capaz de fazer isso. Lembre-se de comparar os `byte[]` utilizando o método `.equals()`.

3 Relatório e Instruções de Entrega

Entregue a implementação completa na forma de um arquivo .tar.gz ou .zip. Certifique-se de que o projeto é compilável com um `mvn clean package`. Nesse trabalho não há necessidade de relatório.

4 Avaliação

Cada componente do trabalho tem o seguinte peso:

```java
private static byte[] getExpectedSignature(File file) throws IOException {
    MessageDigest md;
    try {
        md = MessageDigest.getInstance("SHA-256");
    } catch (NoSuchAlgorithmException e) {
        throw new RuntimeException("Unexpected exception", e);
    }
    try (FileInputStream in = new FileInputStream(file)) {
        while (in.available() > 0)
            md.update((byte) in.read());
    }
    return md.digest();
}
```

Figura 4 – SHA-256 em Java sem superfaturamento.

1. Implementação: 6.0;
    - Não utilizou memória mapeada: -3.0
    - Haverá desconto de até 3.0 por más práticas já abordadas em aula ou em outros trabalhos.
2. Apresentação (nota individual): 4.0.

A nota máxima do trabalho é 10.0. O trabalho deve ser realizado em grupos de até duas pessoas. Em caso de cópia entre grupos, o componente do trabalho onde houve cópia terá a nota dividida entre todos os grupos envolvidos.

A apresentação cobrirá conceitos teóricos relacionados com o trabalho!