

Curso de

## Licenciatura em Engenharia Informática

Cadeira de Computação Distribuída

Projecto de avaliação 2010/2011.

Desenvolve uma aplicação distribuída, de criptografia, para a descoberta de chaves SHA-1 duplicadas para textos diferentes.

*Secure Hash Algorithm 1* é um algoritmo de dispersão\*. Ele recebe como argumento uma sequência de bytes (habitualmente um ficheiro ou um texto) e devolve uma sequência de 20 bytes (160 bits), habitualmente representado como uma sequência de 40 caracteres hexadecimais. Este número (o valor de dispersão), calculado a partir do ficheiro/texto que lhe foi passado, “representa” esses dados, no sentido em que mesmo mínimas modificações a esses dados levam a um valor de dispersão completamente diferente.

Esta técnica é usada para detecção de modificações em ficheiros e (logo) assinaturas digitais, guardar senhas de forma segura e outras funções criptográficas.

Uma inevitabilidade do sistema SHA-1 e outros semelhantes (MD5, etc.) é que se ele gera um valor de dispersão de 160 bits, então só consegue dar valores de dispersão diferentes a  $2^{160}$  combinações diferentes de dados. Inevitavelmente, alguns dados diferentes irão gerar o mesmo valor de dispersão. A falha de detecção de modificações em ficheiros nesses casos é uma limitação aceite nestes algoritmos. Eles são no entanto construídos de forma a que:

1. Não seja possível prever qual a modificação que leva a um valor de dispersão idêntico, e
2. As modificações necessárias sejam muito grandes, tornando a fraude por falsificação inviável.

Devido a estas considerações, só é possível encontrar exemplos de 2 sequências de dados que geram o mesmo valor SHA-1 por força bruta, ou seja, experimentando sequencialmente (ou aleatoriamente) sequências diferentes de dados até que uma gere um valor de dispersão idêntico. Como a quantidade de combinações possível é muito grande ( $2^{160} = 1,46 \times 10^{48}$ ), esta exploração irá demorar muito tempo pelo que deve ser feita em paralelo por vários processadores e computadores.

Este é o objectivo deste projecto.

---

\* Ou algoritmo de *hashing*.

Encontras código para calcular o SHA-1 com licença GPL que podes usar no teu projecto, em:  
<http://polarssl.org/trac/browser/trunk/library/sha1.c>

Cria uma aplicação distribuída que descobre chaves SHA-1 duplicadas para um qualquer ficheiro de entrada. Esta aplicação deve ser dividida em dois módulos com interface de consola:

1. Módulo de interface com o utilizador, que recebe na linha de comandos o nome do ficheiro a usar. Deve dividir o problema da pesquisa por tantos computadores quantos estiverem disponíveis, lançar as explorações em cada computador e receber relatórios de progresso de cada um para exibir ao utilizador.
2. Módulo(s) de exploração (parcial) da gama de dados. Estas aplicações devem estar dormentes até serem “acordadas” pela aplicação de interface. O pedido de exploração deve incluir o ficheiro em causa e o “ponto” de onde deve começar (e até onde ir) a exploração. A ligação é mantida aberta e a aplicação deve regularmente indicar o progresso a quem fez o pedido.

A exibição de progresso deve ser feita como “N/M, X%”, onde M é o total de passos que esse módulo irá executar, N é o número de passos que já executou e X é precisamente a fração N/M. Esta exigência prende-se com o facto de se conseguir ver alguma evolução num curto espaço de tempo.

Não é necessária a implementação da descoberta automática dos módulos de exploração por parte da interface. Se o módulo de interface usar um ficheiro estático com a lista dos IPs ou FQDNs dos módulos de exploração, essa solução é aceite.

Mas como fazer a exploração?

Uma solução simples é ver o ficheiro como um número grande. Repara como funciona um odómetro (conta-quilómetros): 00001, 00002, ..., 00009 e agora muda (também) o 2º dígito a contar da direita: 00010. Mas em seguida voltamos ao dígito mais à direita: 00011, 00012, etc.

Pensa em cada byte do ficheiro como “um dígito”. E para ser fácil trabalhar com o ficheiro, vamos trabalhar nesses bytes da esquerda para a direita.

Começamos com uma sequência de zeros, com o mesmo tamanho do ficheiro original. Fazemos o SHA-1 desses dados e vemos se é o mesmo do ficheiro original. Se sim, verificamos também se esses dados são iguais ao ficheiro original. Se não, encontrámos!! Em todos os outros casos continuamos a exploração.

E como se continua a exploração? Somo 1 ao 1º byte. Se ele “deu a volta” e resultou em zero, somo também 1 ao 2º byte (que também verifico se “deu a

volta e assim por diante). Terminei a exploração se todos os bytes dos dados “deram a volta”.

Há problemas que não estão resolvidos. Como divido este processo de exploração por  $K$  módulos de exploração? Se cada módulo de exploração demora (hipoteticamente) 3 meses a fazer o seu cálculo, como evito ter de esperar de novo 3 meses se um dos módulos falhar no final da sua exploração? Como acho a sequencia de dados mais parecida possível ao ficheiro original (já que podem existir várias que geram o mesmo valor de difusão)? Estes problemas (e outros) ficam para tu resolveres.

Critérios de avaliação (1,5 valores por cada, excepto o  $K$  que vale 5):

- A.** A aplicação consegue ter o módulo de interface a comunicar com (pelo menos) um módulo de exploração e arrancar a exploração.
- B.** A aplicação consegue lidar com uma quantidade infinita de módulos de exploração (apenas limitada pelos recursos e limites dos sistemas operativos subjacentes).
- C.** O módulo de interface exhibe o progresso (e estado: a funcionar / em baixo) de todos os módulos de exploração que arrancou.
- D.** A aplicação detecta quando um dos módulos de exploração falha (deixa de responder), e lança novo pedido ou deixa em fila de espera a exploração daquela sequência para um dos módulos a funcionar.
- E.** Os módulos que estão a fazer uma exploração detectam quando a interface falha ou desliga a ligação e desistem da exploração.
- F.** A distribuição da exploração pelos módulos de exploração é feita em vários pedidos (em vez de ser 1 grande pedido por módulo) de forma a lidar com falhas de forma mais resistente.
- G.** O algoritmo que distribui (e executa) a exploração, é mais inteligente do que a exploração sequencial explicada, no sentido em que não vai alterando os dados da esquerda para a direita, evoluindo muito lentamente para a direita conforme explora todas as combinações à esquerda, mas em vez disso vai distribuindo a localização das alterações por todos os dados.
- H.** O algoritmo descrito no ponto anterior, não repete combinações durante a exploração (e.g.: não é um algoritmo de alterações aleatórias).
- I.** O algoritmo descrito nos 2 pontos anteriores, encontra uma sequência parecida ao ficheiro original.
- J.** Capacidade de resolução do tema e qualidade do código.
- K.** Deve ser produzido e entregue um relatório que explica os conceitos do projecto de acordo com os conceitos teóricos dados nas aulas. Pretende-se diagramas de comunicação, explicação do protocolo (tipo de mensagens) trocadas, formas de resolução dos problemas de divisão da exploração, menção a características de fiabilidade e sua importância, etc.. O relatório deve ter 5 páginas e é penalizado se for demasiado extenso.

Critérios de avaliação individual:

- À-vontade em relação ao próprio trabalho, durante a apresentação oral do mesmo.
- Saber resposta a perguntas teóricas colocadas pelo professor, em relação com o trabalho.

Existe um factor de desconto (subjectivo e aplicável caso a caso) para projectos que tenham sinais de cópia ou execução por terceiros.

O projecto deve ser realizado e apresentado em grupos de 3 a 5 pessoas. A nota do projecto pode ser diferente para cada elemento do grupo. Um elemento do grupo que não apareça ao mesmo tempo que os restantes para a discussão terá nota zero para a apresentação oral do mesmo, e nota reduzida para os restantes critérios.

Boa Sorte!

Pedro Freire