Faculty of Computers, Informatics and Microelectronics

Technical University of Moldova

SI

Laboratory work # 1

Title

Author:
Petru Negrei

Supervisor:
A. Railean

September 2014

1 Introduction

1.1 Objective

Understand the purpose of hashing algorithms and create a tool that solves a problem using one of them.

1.2 Generic requirements

1.2.1 Directory integrity checker

This program keeps an eye on the contents of a directory, notifying you when something inside it has changed. It is ran at regular intervals by a scheduler, comparing the current state of the system with a previous snapshot, reporting differences it found. Thus, if your system was hacked and malware was planted into the file system, or if the existing files were modified to include malicious code - you'll know right away.

1.2.2 Dedupe

A duplicate finder, which analyzes a given directory and prints a list of identical files that have different names or paths.

2 Implementation

The following code implements all the requirements and also some bonus features.

2.1 Main program

```
#!/usr/bin/env ruby
require 'gli'
begin # XXX: Remove this begin/rescue before distributing your app
require 'lab1'
require 'digest'
require "sqlite3"
require 'rufus-scheduler'
PATH = "/home/peter/test"
MEGABYTE = 1024*1024
MAXSIZE = 50*MEGABYTE

include GLI::App

program_desc 'First laboratory work at the information security'

version Lab1::VERSION

subcommand_option_handling :normal
arguments :strict

# Global options
desc 'skip files above a certain size given in bytes'
default_value MAXSIZE
flag [:max_size]

desc 'skip paths that match a pattern'
default_value ""
flag [:exclude]

desc 'this command line argument ensures that nothing is printed on the screen if no differences
    were found'
default_value true
switch [:silent]

# dircheck command
desc 'This program keeps an eye on the contents of a directory, notifying you when something
    inside it has changed.'
arg_name 'path name'
```

```
    command :dircheck do |c|
      c.action do |global_options,options,args|
        path = args.first || PATH
        @scheduler.every("10s") do
          checker = DirCheck.new path: path , max_size: global_options[:max_size], exclude:
              global_options[:exclude], db: @db
          puts checker.show_result
        end

        @scheduler.join
      end
    end

    # dedupe command
    desc 'A duplicate finder, which analyzes a given directory and prints a list of identical files
        that have different names or paths.'
    arg_name 'path name'
    command :dedupe do |c|
      c.action do |global_options,options,args|
        path = args.first || PATH
        checker = DupFile.new path: path , max_size: global_options[:max_size], exclude:
            global_options[:exclude]
        puts checker.list_same_files
      end
    end

    pre do |global,command,options,args|
      # create a scheduler
      @scheduler = Rufus::Scheduler.new
      # Open a database
      @db = SQLite3::Database.new "check.db"
      begin
         rows = @db.execute <<-SQL
          create table hash_table (
            path varchar(30),
            hash varchar(30),
            size int
          );
         SQL
      rescue SQLite3::SQLException => e
         "Table already exists"
      end
    end

    post do |global,command,options,args|
    end

    on_error do |exception|
      true
    end

    exit run(ARGV)
```

```
peter  lab1  master  bundle exec bin/lab1 --help
NAME
    lab1 - First laboratory work at the information security

SYNOPSIS
    lab1 [global options] command [command options] [arguments...]

VERSION
    0.0.1

GLOBAL OPTIONS
    --exclude=arg  - skip paths that match a pattern (default: )
    --help         - Show this message
    --max_size=arg - skip files above a certain size given in bytes (default: 52428800)
    --[no-]silent  - this command line argument ensures that nothing is printed on the screen if no differences were found (default: enabled)
    --version      - Display the program version

COMMANDS
    dedupe   - A duplicate finder, which analyzes a given directory and prints a list of identical files that have different names or paths.
    dircheck - This program keeps an eye on the contents of a directory, notifying you when something inside it has changed.
    help     - Shows a list of commands or help for one command
```

2.2 Duplication example

```
# Reopen File Class
   class File
    def each_chunk(chunk_size=MEGABYTE)
     yield read(chunk_size) until eof?
    end
   end

   class DupFile
    attr_reader :hash

    def initialize args
     @path = args[:path]
     @max_size = args[:max_size]
     @exclude = args[:exclude]
     @hash = {}

     calculate_size
    end

    def list_all_files
     hash.inject([]) do |res, (path, info)|
       res << "#{path} : [#{info[:size]} bytes] [#{info[:hash]}]"
     end.join "\n"
    end

    def list_same_files
     same_files.inject([]) do |res, (hash, paths)|
       res << "#{hash}"
       paths.each {|path| res << path }
       res
     end.join "\n"
    end

    def same_files
     hash_contents.group_by{|_, info| info[:hash] }
           .each {|_, v| v.map! {|h| h.first } }
           .select {|_, v| v.size > 1 }
    end

    private

    def calculate_size
     Dir.glob("#{@path}/**/*")
          .reject { |file| File.directory? file }
          .reject { |file| unwanted_paterns(file) }
          .each { |file| @hash[file] = {size: File.size?(file) } }
    end

    def filter_size
     hash.select {|key, value| value[:size] < @max_size }
    end

    def hash_contents
     filter_size.each { |path, info| @hash[path][:hash] = digest_file path }
    end

    def unwanted_paterns str
     @exclude.gsub("*", "\w*").split(" ").any? { |regx| str =~ /#{regx}/ }
    end

    def digest_file filename
     # Digest::SHA256.file(filename).hexdigest
     open(filename, "rb") do |f|
       sha256 = Digest::SHA256.new
       f.each_chunk() {|chunk| sha256 << chunk }
       sha256.hexdigest
     end
```

```
    end

  end
```

```
peter lab1 master bundle exec bin/lab1 dedupe
8b5b9db0c13db24256c829aa364aa90c6d2eba318b9232a4ab9313b954d3555f
/home/peter/test/3.txt
/home/peter/test/nest/3.txt
7692c3ad3540bb803c020b3aee66cd8887123234ea0c6e7143c0add73ff431ed
/home/peter/test/nest/1.txt
/home/peter/test/nest/4.txt
/home/peter/test/1.txt
/home/peter/test/4.txt
```

2.3 Modified in system

```
class DirCheck
  attr_reader :hash

  def initialize args
    @path = args[:path]
    @max_size = args[:max_size]
    @exclude = args[:exclude]
    @db = args[:db]
    @rows = @db.execute( "select * from hash_table" )
    @hash = {}
    @result ={created: [], deleted: [], modified: [] }
    initiate_hash
  end

  # initiate hash form {path => size}
  def initiate_hash
    Dir.glob("#{@path}/**/*")
        .reject { |file| File.directory? file }
        .reject { |file| unwanted_paterns(file) }
        .each { |file| @hash[file] = {size: File.size?(file) } }
  end

  def check_contents
    @rows.size > 1 ? search_change : fill_table
  end

  def show_result
    check_contents
    result = @result.inject([]) do |res, (status, paths)|
      paths.each { |path| res << "#{path} (#{status})" }
      res
    end.join "\n"
    refresh_table
    result
  end
  def refresh_table
    @db.execute('DELETE FROM hash_table')
    fill_table
  end

  def search_change
    check_new_files
    check_modified_files
  end

  def check_new_files
    current, old = @hash.keys, @rows.map(&:first)
    @result[:deleted] = old - current
    @result[:created] = current - old
  end

  def check_modified_files
```

```ruby
      @rows.each do |db_path, db_hash, db_size|
        if @hash.has_key?(db_path)
          @result[:modified] << db_path if digest_file(db_path) != db_hash
        end
      end
    end

    # generate new content table
    def fill_table
      hash_contents.each do |path, info|
        @db.execute("insert into hash_table values ( ?, ?, ?)", [path, info[:hash], info[:size]])
      end
    end

    def filter_size
      hash.select {|key, value| value[:size] < @max_size }
    end

    # hash all files
    def hash_contents
      filter_size.each { |path, info| @hash[path][:hash] = digest_file(path) }
    end

    def unwanted_paterns str
     @exclude.gsub("*", "\w*").split(" ").any? { |regx| str =~ /#{regx}/ }
    end

    def digest_file filename
      # Digest::SHA256.file(filename).hexdigest
      open(filename, "rb") do |f|
        sha256 = Digest::SHA256.new
        f.each_chunk() {|chunk| sha256 << chunk }
        sha256.hexdigest
      end
    end

  end
```

```
peter lab1 master bundle exec bin/lab1 dircheck
/home/peter/test/4.txt (created)
/home/peter/test/5.txt (deleted)
/home/peter/test/1.txt (modified)
```

a. Start the scheduler

b. Create the class responsilbe for logic

c. Initiate a dictionary with path and size of file

d. Filter by given options

e. Check the database if is empty introduce the data to it

f. If database is not empty check each file wtih current state.

g. And create a result based on hash comparision of them.

3 Conclussion

After making this laboratory work I learn about different purposes of hashing algorithms. Hash functions are related to (and often confused with) checksums, check digits, fingerprints, randomization functions, error-correcting codes, and ciphers, usually applied to detect duplicated records in a large files (similar DNA sequence) and allows one to easily verify that some input data matches a stored hash value. If we compare some of hash algorithms like MD5, SHA1 or SHA512 they will not improve the security of the construction so much. Computing a SHA256 or SHA512 hash is very fast. An attacker with common hardware could still try tens of millions of hashes per second. Good password hashing functions include a work factor to slow down attackers.