A secure marking system
Administrators guide

Bruce Merry and Carl Hultquist

January 24, 2014

Contents

1 Introduction

The Secure Marking System is a system that automatically marks programs by considering their input and output. It is designed for computer olympiads in the style of the International Olympiad in Informatics (IOI), but has also been used for marking online contests, web-based training and practical tests at the University of Cape Town. It is best suited to environments where final marking is done off-line after the event, rather than with real-time feedback.

The system incorporates a web-based handin system, similar in features to those used at the IOI and other computer olympiads. Submissions uploaded to the server are immediately run on a sample data set, thus providing sanity checking on the submission.

It also supports typesetting of problems in LaTeX. It will automatically fill in problem names, sample input and output, time limits and so on. It can also prepare a combined document of all problems for a day, and generated an overview sheet.

This manual is one of four. The others are the environment manual (for contestants who will use the system), the problem authors guide (which describes the interface between individual problems and the SMS framework), and the developers guide (for advanced users and those wanting to hack the system).

The system consists of two parts: a server and several markers (markers are also called clients). They can either all be run on one machine, or several separate machines can be used as clients to speed up evaluation.

2 Requirements

2.1 Clients

The clients will only run on a Linux system i.e., one running the Linux kernel version 2.6. It uses a kernel module to ensure security, so it would be difficult to port. It also requires a POSIX operating environment. It has only ever been tested on GNU/Linux systems (Debian and Gentoo), so it may be dependent on behaviour specific to the GNU tools. You will need the following tools (ordered roughly from most exotic to most standard).

- `Perl`, version 5.8 or higher
- `gcc`, `tar`, `gzip`

A stable Gentoo GNU/Linux system has sufficiently recent versions of all packages, as should most current Linux distributions. The FastCGI support is optional, as the system supports plain CGI operation (although at significantly reduced speed).

Of course, you will also need compilers for the supported languages. The system is designed to make it easy to add new compilers, but by default the following are supported:

- GCC for C, C++ and Java
- Free Pascal
- Java (with the Sun or Blackdown JDK)
- Python
- GHC for Haskell

Finally, you will need some accounts and some groups on the client (explained in the next section) which the system administrator will need to set up. Once the initial setup is done, however, root access is not required for maintainence.

2.2 Server

The server could theoretically be any POSIX system, but it has only ever been tested on GNU/Linux systems and it is likely that compatibility issues would arise with other systems. The requirements are

- `pwgen`
- `sudo`
- A web server such as Apache. The web server user has access to all contest data, so untrusted users must not be given cgi-bin or PHP access except in a restricted environment. There is also potential for the user that owns the contest to run arbitrary code as the apache user.
- Optional, but recommended for speed: FastCGI capability in the web-server. For Apache this is provided by mod_fcgid; note that you should use at least version 1.08 as earlier versions were broken with respect to file uploads.
- `Perl`, version 5.8 or higher, with the following extra modules:
  - FCGI
  - TimeDate (Date::Parse and Date::Format packages)
  - Text::CSV_XS
  - MIME::Lite
- GNU `diff3`, from `diffutils`
- `gcc`, `make`, `tar`, `gzip`
- Optionally, `aspell` (with the appropriate language dictionary) to enable the spell-checker in the sanity check.

3 Setup

3.1 Client

Most setup on the client can be done by running

```
./install.sh --client-only
```

However, before you do so, you should see the next section (kernel module). The install script does the following:

- Adds users called *sms* and *sandtray*. The former is used to run the client, while the latter is a reduced-privileges account used to run user submissions.
- Adds items to /etc/sudoers to allow *sms* to run commands as *sandtray*.
- Installs utility libraries in `/opt/sms/exectime` which allow user programs to query the CPU time used, and also adds this to the Python library paths.
- Compiles and installs the kernel module.
- Creates a stub /etc/smsd.conf for you to complete. You need to supply the hostname of the server (if not the local host) and a password that matches the password chosen on the server.

NB: the communication between the client and the server is **unencrypted**. If the network is not physically secured, you are responsible for setting up a VPN, IPSec, an SSH tunnel or some other means of security!

3.1.1 Kernel module

All the contestant programs run under the same account. Although they should not be able to interfere with the marking system, they could interfere with each other (send signals, use ptrace, or fill up quotas). In addition, it is necessary to prevent certain behaviour in the boxed programs, such as executing external programs or filling up the filesystem. The security module does just that.

A recent 2.6 kernel is required (2.6.22.1 has been tested). You will need to modify your kernel configuration. In the `Security options` section, set the following options:

```
[*] Enable different security models
[*]   Socket and Networking Security Hooks
<M>   Default Linux Capabilities
```

Note that the last option *must* be a module and not compiled in. Do not compile any other security modules into the kernel, although it is okay to compile them as modules.

Once you have compiled and installed your new kernel, you may run the `install.sh` script as described above. If your distribution is not detected, it will give you a list of things to handle manually.

3.2 Server

The server machine may similarly be set up for the first time by running

```
./install.sh --server-only
```

If both the client and the server are to be run on the same machine, `install.sh` may be run without arguments. The server-side installation

- Adds a user called *sms*, which will own contest trees.
- Adds entries to `/etc/sudoers` to allow the web server to run jobs under the *sms* account.
- Installs `smsd`, the lock and queue manager that communicates with the server processes and with the clients.
- If possible, installs a sample configuration for Apache.

3.3 Accounts

The names of the accounts *sms* and *sandtray* can be overridden: run `./install.sh --help` for details. You must remember to pass the arguments every time you run `install.sh`, though. The accounts are created only if they do not exist, so you can use an existing account.

You should also have a group for all users who are permitted to directly edit files in the contest (it is recommended that one uses `sudo` instead, but this exists for historical reasons). If you do not want anyone except *sms* to have this kind of access, just create a dummy group (do not use an existing group like `users`, since that would give everybody access). This group is labelled as *controlgroup* below.

3.4 Creating a contest tree

Once system-level installation is done, you are ready to set up a contest tree. Log in as *sms* on the server, and run

`scripts-server/setup` *directory controlgroup*

This will create a skeleton contest in *directory*, with group ownership of *controlgroup*. Section 4 describes how to complete the configuration.

3.5 Creating a client tree

On each client machine, you will need a client tree. This can be created by running

`scripts-client/setup` *directory sandtray controlgroup*

The client is then executed by running `scripts/sms-client`. It produces output to standard output, so it is a good idea to run it inside `screen` for remote maintainence.

For quick setup, simply edit `config/client` and modify the examples to taste. Refer to section 4.5 for more detailed instructions, especially if the client and server are on different machines.

3.6 Web server

There are two scripts in the `cgi` directory that can be used by the web server: `index.cgi` for plain CGI and `index.fcg` for FastCGI. One of them must be set as the directory index, because internal links point to the directory. There are two Apache modules for FastCGI: `mod_fastcgi` and `mod_fcgid`. The latter has better process management and is recommended. An example of Apache configuration is shown below:

```
Alias /contest "/path/to/contest/php/"

<Directory "/path/to/contest">
    Order allow,deny
    Allow from all
    Options +FollowSymLinks +ExecCGI
    AllowOverride None
    SSLRequireSSL
    DirectoryIndex index.fcg
</Directory>

FcgidBusyTimeout 1800        # Allows script to run for 20 minutes
FcgidIOTimeout 300           # Allows script to take 5 minutes to start output
FcgidMinProcessesPerClass 1  # Reduces idle load by keeping only process alive
FcgidOutputBufferSize 1024   # Sends output to browser in smaller chunks
FcgidMaxRequestLen 268435456 # Allows large files to be uploaded
```

The last block configures options for FastCGI, and unfortunately cannot be done on a per-directory basis. Refer to the `mod-fcgid` manual for the details of what these fields mean.

Passwords are transmitted in the clear, unless a secure server (SSL) is being used. The passwords are stored as MD5 checksums on the server, but are not salted. It is highly recommended you configure the web server to deny non-SSL access to the contest tree, as shown above.

Finally, the template generator is located in `misc/`. It consists of an HTML page with a form and a PHP script that calls the backend and generates a template. It is completely separate from the rest of the system (although this may change). If it is wanted, both files should be aliased or symlinked[1] into a suitable area of the web server.

4 Configuration

All configuration of the contest (except for per-problem configuration) is done in the `config` subdirectory of the directory created in 3.4.

[1]The rest of the evaluation system must be in the correct place relative to the PHP script, so a simple copy will not work.

4.1 Problems

Problems are organised into "days". A day is simply a group of problems, and is used in the following ways:

- The results page shows sub-totals for each day.
- It is possible to restrict the visible problems to a particular day.
- If problems are typeset in LaTeX, then a single PDF is generated for each day with an overview sheet.
- There is a configuration file per day, to easily change settings on a per-day basis. Problems on the same day can still have different permissions and deadlines.

If problems are introduced singly over time (for example, in an online course), then it is better to put all the problems in a single "day". Separate days are more useful to group together problems that are introduced and marked as a set.

Create files `problems-1`, `problems-2` and so on in the `config` directory, where the suffix indicates the day. Each file contains a list of problem names, one per line. The number of days is determined by the number of sequential files that are found. A single-day contest is just one with only a `problems-1` file.

4.2 Users

The user management is split into two sets of files. `authdata` and `groupdata` are meant to be shared between contests, while `people` and `groups` describe how users and groups apply to a particular contest. To share the first group, remove the empty files that are created by the `setup` script and create symbolic links to the shared files.

Wherever possible, scripts and the web frontend should be used to modify these files, to prevent race conditions. However, not everything can be done with the frontend.

All the files consist of a sequence of lines, with blank lines and comments beginning with a # ignored.

4.2.1 `authdata`

This file maps usernames to real names (shown on mark reports) and password hashes (used to validate logins). The format is `user:realname:password`, where `password` is an MD5 checksum of the actual password. Using an asterisk for the password hash will disable the account; this is useful for pseudo-accounts that do not belong to real people.

4.2.2 `groupdata`

This file defines groups, similar to UNIX groups. The format is `group:user,user,...`. The users must be listed in `authdata`.

4.2.3 `people` and `groups`

These files describe how users and groups fit into the system. For small sets of users, the `people` file may be sufficient (in which case `groups` and `groupdata` need not exist at all), but for larger contests the group support eases management. The format of each line is `user:role,role,...` (replace `user` with `group` for `groups`). Users that are specified in both files get the union of the listed roles. Roles determine how the system treats users. The defined roles are:

contestant Users with this role have a basic set of priviledges, including testing, submission, editing of problems for which they are named editors.

guest The same as contestant, except that results do not appear on the results printouts.

admin Admins have all rights.

model Used for a set of model solutions. This is used by the backend to check that model solutions score 100%

4.3 `users`

Specifies the details of the usernames and groups. This is created by the `setup` script and should not be modified unless you really know what you are doing. FIXME: rename things more sensibly (e.g. people->users, users->unix or something).

4.4 Configuration engine

The remaining configuration files are used by the "configuration engine". This is a very versatile configuration system, which is described fully in the developers guide. FIXME: should it be an appendix instead? Most of the complexity is hidden in the default set of rules, though, so for most contests you can just use lines of one of these forms:

- `setting` — enables a boolean setting
- `!setting` — disables a boolean setting
- `setting='value'` — sets a value
- `setting=number` — sets a numeric value
- `setting=[date]` — sets a date value, in the format "1 Jan 2000 09:00:00"

Single quotes can be included in values by escaping them with a backslash. Blank lines and comments preceded by a `#` are ignored. Numbers can be suffixed with `k`, `m` or `g` to indicate kibibytes, mebibytes or gibibytes.

The files that are used are:

`config/policy` The default settings. Do not edit this file unless you are developing SMS, as it is shared between contests.

`config/general` Per-contest settings.

`config/status-day` Per-day settings

`problem/problem/options` Per-problem settings.

Later files override earlier files. The settings are prefixed with `contest.`, `day.` or `problem.` to indicate the lowest level at which they will typically be used (the `compiler.` prefix is for compilation-related settings, but they have the same scope as `problem.` settings). It is common to use settings at higher levels to set defaults, and in some circumstances they can be specified at lower levels to set special behaviour.

4.4.1 Contest-wide settings

`contest.title` The name of the contest, as it will appear on web pages.

`contest.title.latex` The name of the contest, as it will appear on the problem texts (defaults to the setting of `contest.title`).

`contest.sponsor`n Optional; sets the base filename of a logo to be displayed on problem texts. Substitute n with a value from 1 to 4. The currently defined logos are `om` Old Mutual

`cssa2` Computer Society of South African

`uct` University of Cape Town

`stdbank2` Standard Bank

`tsf` The Shuttleworth Foundation

`chpc` Center for High Performance Computing

The first two slots are best suited to crest-type logos (the first three), while the other two slots are the bottom of the page and better suited to a wide logo (such as `stdbank2`).

`contest.place` Optional; sets the location of the contest as displayed in problem texts.

`contest.day` During a contest, set this to the day of the contest. Users with the `contestant` role will only be able to see problems from this day, and will not be able to edit problems even for which they are an editor.

`contest.publicaccess` Enable this to allow people to see problems and test submissions even without logging in.

`contest.web` Enable this to allow people to read the problem texts online.

`contest.url` Specify a piece of text, generally an URL, that will appear on the cover sheet.

`contest.registration.allowed` If set, users may sign up for new accounts on the web interface.

`contest.registration.default_group` Users that register online will automatically be placed in this group.

`contest.clarifications` Link to a clarifications page, which is embedded using a iFrame.

`contest.documentation` Link to documentation page, which is embedded using a iFrame.

`contest.notinresults` Set for user/group to not include them on the results page an exclude them from updates

`contest.medals.gold` Number of gold medals allocated to top contestants, highlighted on results page

`contest.medals.silver` Number of silver medals allocated to top contestants not receiving gold

`contest.medals.bronze` Number of bronze medals allocated to top contestants not receiving any other medal

`contest.medals.group` If set, restricts the contestants eligible to win medals.

`contest.feedback.submissions` If set, enables detailed feedback. Set to a number to limit the number of detailed feedback submissions per problem.

`contest.feedback.status` Prints status for each detailed feedback test case

`contest.feedback.verbose` More verbose detailed feedback

`contest.feedback.evaluate` Run evaluation for detailed feedback

`contest.feedback.summary` Show summary of results for detailed feedback

`contest.feedback.data` View detailed feedback test cases

`conntest.loganswers.submit` Store history of all submissions in `answers/<user>/old/` directory `conntest.loganswers.testing` Store history of source for all test runs in `answers/<user>/testing/` directory

`contest.expect_ip` Regular expression of the IP range expected for contestants, mismatches highlighted on logs page

`server.session_lifetime` The lifetime of the cookies used to authenticate users, in seconds (defaults to 1 hour)

`server.post_max` The maximum size of a POST request e.g. for a file upload (defaults to 16MiB)

`server.web_queue_timeout` The maximum number of seconds to wait for progress when submitting jobs through the web interface [270].

`server.global_queue_timeout` The maximum number of seconds to wait for progress when using `scripts/global` [0 i.e., no timeout].

`server.default_queue_timeout` The maximum number of seconds to wait for progress when using the Update option on the results page [270].

4.4.2 Per-problem settings

The problem writers guide specifies a number of per-problem settings. We describe only those that are generally *not* the problem writer's responsibility.

`problem.available` A boolean, set to allow contestants to see the problem (as well as the public, is `contest.publicaccess` is set).

`problem.active` A boolean, set to allow contestants to hand in.

`problem.marked` A boolean, set to allow contestants access to their marks for a problem.

`problem.complete` A boolean, set to indicate that secret information about a problem may be made public. This enables the problem tarballs, solution tarballs (if `problem.marked` is also set), test data display, and testing with real data.

`problem.frozen` A boolean, set to prevent re-runs when the system things a re-run would normally be required. This should be set after a contest to prevent the marks fluctuating due to non-determinism in user programs.

`problem.start` The time at which the problem becomes visible (unset means always visible).

`problem.due` The problem's deadline (unset means no deadline).

`problem.date` A date to display on the problem text. It defaults to the setting of `problem.due`, if any.

`problem.notest_time` If set, represents a number of seconds before the deadline during which the testing facility is disabled.

`problem.nofeedback_time` If set, represents a number of seconds before the deadline during which the detailed feedback facility is disabled.

`problem.owner` The username of a user who may edit this problem.

`problem.priority.submit` Priority a submission is given when selecting which runs should get evaluated first (default 100).

`problem.priority.test` Priority a test run is given when selecting which runs should get evaluated first (default 200).

`problem.priority.feedback` Priority a detailed feedback run is given when selecting which runs should get evaluated first (default 200).

Note that `problem.start` and `problem.available` combine to make a problem available (i.e. `problem.available` must be set **and** `problem.start` must be unset or set in the past). Similarly, `problem.due` and `problem.active` combine to determine whether a solution can be handed in.

4.4.3 Per-day settings

The per-day setting are

`day.title` Gives a per-day subtitle on problem texts.

`day.date` Gives a date that appears on overview sheets. This setting also provides a default for `problem.date`. For a "day" that is really a single day, use this setting, while for a group of problems over time, use `problem.date`.

Settings related to problem status (like deadlines) are often placed in the status files.

4.4.4 Compilation settings

These settings are used per-problem, but will generally be set globally. Most of the settings have reasonable defaults, but all compilers are disabled by default. Each compiler has a name used in forming variable names; these are `gcc`, `gxx`, `gcj`, `fpc`, `jdk`, `python` and `ghc`. We list only the settings for Java (jdk); the others are similar but may lack either the compiler or interpreter settings.

`compiler.jdk` Boolean, set to allow the compiler

`compiler.jdk.compiler.flags` The extra flags given to the compiler to set optimisations. The defaults are for an optimised compile with no debugging symbols.

`compiler.jdk.interpreter.flags` The interpreter flags, as for the compiler flags.

`compiler.jdk.time_multiplier` A number to multiply the time limit by when using this compiler. The default is 1.0 for all languages.

`compiler.jdk.version` A regular expression that specifies a compiler version to use. The client may have multiple versions of a compiler defined, and this is used to select the first matching one.

Where multiple compilers are available for a single language (e.g., gcj and JDK for Java), you must make sure that only one compiler is enabled.

4.5 Client configuration

On the client, the only configuration that is required is the paths to the various compilers. If the compiler of the correct version is in the default path with the usual name then no configuration required. However, JDK is usually not in `/bin:/usr/bin` and so the path needs to be configured. It is also possible to have several versions of one compiler installed concurrently, in which case paths to each need to be set.

For each compiler, multiple slots may be defined. Slots are numbered sequentially from zero, and each slot represents one version. The system automatically queries the compiler for its version, so you do not need to explicitly indicate which versions are configured. If the server does not explicitly request a version, then slot 0 is used.

The available settings for slot *n* for JDK are:

`compiler.jdk.compiler.n` The path to the compiler.

`compiler.jdk.interpreter.n` The path to the interpreter.

For other compilers, simply substitute the name for "jdk", and remove one or the other line when it is inappropriate.

The client also needs to know the password to authenticate with smsd. If everything is running on one machine, the default setup will symlink `/etc/smsd.conf` to `config/smsd`, and all will be well. If not, you will need to write your own `config/smsd`, which should look like this:

`password` *password* `hostname` *host.some.where*

Optionally, you may specify a `port` directive to override the default port number.

5 Using the system

In general you should just need to call the script `scripts/global`, which will run the entire evaluation from compilation to producing reports. It is reasonably smart about not redoing things that do not need to be redone, although before awarding the medals it is worth doing a clean run.

5.1 Directory overview

Although you should not need to interfere with most of the directory structure, a brief overview follows:

scripts The various shell and Perl scripts used to implement the back end (symlink)

lib Perl modules used by the back end (symlink)

cgi The web front end (directoy of mostly symlinks)

texmf LaTeX utilities for typesetting questions (symlink)

config Global configuration files

problems All data regarding problems

answers Submitted source code

output Generated reports, compiled files, etc.

run Scratch space

log Directory containing logfiles

5.2 Scripts

The `scripts` directory contains a number of scripts that can be called to perform actions directly from the command line.

`scripts/global` This is the main script used for evaluation, although similar functionality is available through the web interface. By default, it will just re-run any necessary steps, taking into account any changes to the source code, input files, settings, compiler versions and so on. If given the `--validate` option, it will instead update the evaluation for users with the `model` role, and verify that they get 100% for all problems.

There are also options to restrict the set of users and problems that will be evaluated. Run

```
scripts/global -help
```

to see all the options.

`scripts/clean` This script removes cache files, forcing all the steps of evaluation to be redone. By default it applies to all users, and to all problems that do not have `problem.frozen` set. These settings can be overridden; run

```
scripts/clean -help
```

to see the options.

`scripts/reset` Use this script to repair the file permissions. This is particularly important if a problem has been copied in from a remote machine.

`scripts/sanity` This runs a number of internal checks on the evaluation tree, for example checking that certain files and settings exist. It can also be run with the option `-p problem` to restrict the checking to a single problem.

`scripts/adduser` Adds a user and password to the authentication file. It does not set groups for the new user. It is recommended that this script is used only to create an initial admin user, who can then add other users through the web interface.

`scripts/configtest` This is a debugging script that can extract information from the configuration engine.

`scripts/purge` This completely eliminates the output directory and cleans the evaluators. There should be no need for this script during normal use, and its use is discouraged. However, it is sometimes necessary if the output directory becomes corrupted. In general, use `scripts/clean` instead.

`scripts/passwords` Resets the passwords for a list of users. The users are passed as arguments. A CSV file of realname,user,password is output to stdout.

6 The output

All the useful files generated by the evaluation are placed in the `output` subdirectory of the evaluation, which is created if it does not exist.

Most stages of the evaluation are done conditionally, based on the results of a cache file. Cache files have the extension `.cache`. If for some reason you need to force a step to be re-run, identify the corresponding cache file and delete it. Deleting the output of a stage will not work, and in fact will cause subsequent stages to break.

The web front-end is the preferred way to get information out of the system. Most of the files that can be downloaded from it are generated on the fly by the frontend. If for some reason you need to get a file without going through the frontend, it is up to you to figure out where everything is.

7 Legal stuff