Persistent Security

SafeGuard LM 5.6 Developer Guide
(25-apr-2016)

Software licensing with *SafeGuard LM* provides many benefits to you and your end-users:

- *S*ecure encryption
- *A*nti-Spoofing technology
- *F*ast deployment
- *E*ase of use
- *G*reater revenue with minimal intrusion
- *U*nmatched performance and quality
- *A*udit trail of issued licenses
- *R*obust application programming interface
- *D*ual license implementation format

SafeGuard LM Features

SafeGuard LM supports both Floating (concurrent) license management as well as node-locked licensing.

SafeGuard LM is localized in seventeen languages.

SafeGuard LM is as easy for you as it is for your customers. Your customer can install your application and run it for a pre-determined amount of time. This also allows you to distribute evaluation copies, as well as allow your customer time to "enable" your application according to your license agreement at their convenience. *SafeGuard LM* allows you to enable your application with multiple components. In simple terms, lets say you create an application called *office*, and *office* has three extensions you wish to license separately named *draw*, *paint* and *write*, you can license these components separately from the main application, each with their own enabling attributes.

SafeGuard LM licenses are "component" based. A component can be the application itself or a specific feature or extension within the application. You decide what functionality you want to license by name.

SafeGuard LM allows you to choose which IdTypes to secure your application or components. IdTypes are attributes like the mac (ethernet) address, IP address, Username, Hostname, Volume Serial Number (Windows), your own IdType, even no IdType at all that still requires a valid license to run.

SafeGuard LM allows you to issue licenses that timeout on a specific date, or never at all.

SafeGuard LM also allows you to issue licenses for specific versions of your application. For example, if you issue a license for an application with a version number of 10.5, then that application can be run as long as the version of the application is version 10.5 or less.

SafeGuard LM also allows you to choose whether to store licenses on the customers computer as a text file or imbed the licenses within your application. The license API to perform all functions on component licenses is the same for both methods. You just instruct *Safeguard LM* which method to use for the calling application. The default is an external text license file. There are pros and cons to both methods depending on your situation. For example, if you develop an application that you wish to distribute to the masses for a trial period and don't want the end user to have to deal with installing a license file or you would rather them not see a license file, yet you still want to maintain complete control over which components of your application you want them to use and for how long, an imbedded license might be best. With this method, you could imbed a "any" ID license with a specified timeout date that would run on any computer before you send them the application. Then, if at the end of the trial time they wanted to purchase your product, you could obtain the proper ID from their computer and generate a permanent license for them that would only run on that computer. A trial license can be distributed with a text license as well. It really comes down to which style you wish to use.

SafeGuardManager is the application you use to manage all your customers licenses as well as customer information and contact information. You can view each customers profile and what licenses you have generated for them over time. You can update licenses for a particular customer, edit customer information, edit contact information, update licenses for a customer and generate new licenses for a customer and save that information for further reference.

Software Structure

The *SafeGuard LM* release is organized in a hierarchical directory structure. Each platform has a similar directory tree. The top end folder is named pspro_<platform> where platform is the name of the operating system. For example, pspro_win32 (for Windows), pspro_mac64 (for Mac OS X), pspro_lin32 (for Linux) and so on. Not all platforms have the same files and folders due to differences in supported features.

Description of folder contents: (files in red should **not** to be distributed to end-users)

lib

- libpsprolib_lin.a (*) - used to build your application static
- libpsprolib_lin.so (*) - distribute only if you build your application dynamically

bin

- pshostid - obtains ID's for specific IdType's from a command prompt
- pskeycode (*) - command line license generator (do not distribute)
- pspro-ext (*) - command line external license file management tool for users
- pspro-int (*) - command line imbedded license file management tool for users
- sggendate - date generation program
- sglmd - license manager service/daemon (distribute with floating licenses)
- sglmserver (*) - license manager server (distribute with floating licenses)
- sglmutil (*) - license manager utility program (distribute with floating licenses)
- sgpad - Product Activation Server service/daemon
- sgpaserver (*) - Product Activation Server
- sgpautil (*) - Product Activation Server utility program
- control.txt - Product Activation Server control file
- create.sql - Product Activation Server table creator
- psseed - used to create your own unique encryption seed for your license generator and application(s) (do not distribute)
- sgproxyserver - Reverse proxy server for the license server and product activation server

h

- psprolib.h - header file for C/C++ applications only
- psprolib_enum.h - header file for C/C++ applications only

doc

- DeveloperGuide.pdf
- ProgrammerGuide.pdf
- javadoc
- FeaturesAndSpecifications.pdf
- SafeGuardManager.pdf
- SystemRequirements.pdf
- LicenseManagerGuide.pdf (Distribute if you sell floating licenses)
- ProductActivationServer.pdf
- PASDatabaseDescription.pdf
- PASInstallGuide.pdf
- PASFeaturesAndSpecifications.pdf
- WhatsNew5.pdf

java

- SG_IDTYPE.java - SG_IDTYPE class
- SG_ACTIVATETYPE.java - SG_ACTIVATETYPE class
- SG_LICENSETYPE.java - SG_LICENSETYPE class
- SG_ATTRTYPE.java - SG_ATTRTYPE class
- SG_LANGTYPE.java - SG_LANGTYPE class
- SG_DATECHECK.java - SG_DATECHECK class
- psprolib.java - psprolib class
- SWIGTYPE_p_int.java - SWIG function
- psprolibConstants.java - psprolibConstants class
- psprolibJNI.java - psprolibJNI class
- libpsprolibjni_lin.so (*) - psprolib java runtime library (distribute with your Java app)

python

- _psprolib.so (*) - dynamic psprolib python library (distribute with your Python app)
- psprolib.py - python source for importing psprolib (distribute with your app)
- psprolib.pyc - python compiled code for importing psprolib (distribute w/app)

examples

- c - sample “C” programs
- python - sample Python programs
- php - sample PHP definitions
- java - sample Java programs

SafeGuardManager (do not distribute this application to the end-user)

- SafeGuardManager.jar - application to manage your customer license generation
- lib - folder used by java
- pskeycode_lin (*) - license generator
- sggendate_lin - date generator
- sgisdate_lin - date validator

NOTE: You should change your private encryption seed right away after installing the developer kit. Use the *psseed* program to do this (see below) for every file above marked with an (*). Failure to choose your own private encryption seed means that anyone can generate licenses for your application(s). This only needs to be done once after you install SafeGuard LM or if there are any updates to the SafeGuard LM software.

Securing your software to a computer

The following describes available locking mechanisms *SafeGuard LM* supported on Linux.

Ethernet address:	The MAC (Media Access Control) addresses are assigned by the manufacturer of a network interface card (NIC) and are stored in its hardware, the card's read-only memory.
IP address:	This is the IP address of the current hostname.
Hostname:	This is the computer name.
Hostid:	The computers hostid
Username:	This is the current logged in user.
Any:	The “any” ID Type is not locked to anything on the computer. It does, however, still lock the component and timeout together. This is a great way to distribute evaluation software without having to contact the potential customer.
Vendor:	This is used when you want to provide your own ID retrieval method. This requires some programming on your part to implement this. *SafeGuard LM* will use your function to obtain a unique ID from whatever source you want. .
Os:	The *SafeGuard LM* Operating System identifier. The list of identifiers are: mac32, mac64, win32, win64, lin32, lin64.

Expiration Dates

SafeGuard LM licenses include a timeout attribute which controls when a particular component expires. Expiration dates are in the format “*dd-mon-yyyy*” where *dd* is the two digit day of the month. The *mon* is the three character representation of the month, values include *jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec*, and *yyyy* representing the 4-digit year. The special expiration date of “never” will not expire.

Encryption Seed

SafeGuard LM comes built with a unique encryption seed which is used in conjunction with the encryption algorithm. This, by itself, is sufficient to generate and distribute licenses to your customer. However, if you would like to create your own private encryption seed that is unique to your company and in turn would be used by the *SafeGuard LM*, you can do this easily. The benefit of this, is that licenses you generate would be unique to your company. Even if some other *SafeGuard LM* customer were to generate a license with all the same attributes as your license, the license certificate would not match and therefore would not work with your software.

To utilize your own encryption seed, you would use the “psseed” application located in the *bin* folder.
This application must be run from a shell prompt. It takes two arguments. The fist argument is the binary to imbed the seed in, and the second argument is the seed itself. The seed when stored in the binary is encrypted and cannot be seen as a text string in the binary. The seed is a character string and may contain special characters although not necessary because the seed is used as a bit pattern. If you imbed spaces or special characters, be sure to double quote the string. The seed can be up to 25 characters in length. You only have to imbed the seed once for each release of *SafeGuard LM*, so be sure to keep your seed in a safe place so you don't forget it, as it cannot be retrieved.

The seed will need to be imbedded in more than one file. So you will be required to execute the “psseed” application more than once with the same seed. For simplicity, it is best if you imbed your private encryption seed into all possible binaries even if you don't use or distribute all of them. Make sure you perform this task before you begin to use the software.

Example:

```
./psseed pskeycode RX0ZZY3A966E7

Seed "RX0ZZY3A966E7” planted.
```

Do this for each of the binaries listed in the description of folder contents above marked with (*).

Programming Language Documentation

- All developer information is located in the ProgrammerGuide.pdf

Vendor ID sample

To build a function to lock your software to your own unique Vendor Hostid like a hardware key, you need to build either a static or dynamic library containing a function that retrieves your unique ID. See the sample code that demonstrates under the examples/c folder.

Generating licenses for your customers

To generate licenses for your customers there are two ways to do this. First, you can use SafeGuardManager.jar which is a GUI based application that will generate both node-locked and floating licenses. You can save all aspects of the customer information such as contact information, company details, etc., along with the actual license file. This information is saved in two files under the “Customer” folder within the SafeGuardManager directory.

The customer information in the “Customer” folder is binary. It contains the contact and company information. There is another sub-directory SG_Data which contains two files per customer. There is a “customer.txt” and a “customer.licenses” file. They are both text files that can be printed for your records. The customer.txt file contains all the information filled out in the customer and company tabs of SafeGuardManager, plus at the end of the file is a copy of the license(s) generated.

The “customer.licenses” file, contains only the license data. This is the actual license(s) generated and can be sent to your customer as their license file.

The second way to generate a license file is to use the command line version of pskeycode found in the bin folder.

Pskeycode can generate both node-locked and floating licenses. The usage can look intimidating, but it is really simple once you understand it.

Usage: pskeycode <rc4 | sha256> <feature> <version> <pshostid> <ethernet | ipaddress | inet6 | hostname | username | vsn | vendor | hostid | macintosh | os | metered | any | country | iphone | android | blackberry> <timeout>
Usage: pskeycode <rc4 | sha256> <feature> <version> <pshostid> <ethernet | ipaddress | inet6 | hostname | username | vsn | vendor | hostid | macintosh | os | metered | any | country | iphone | android | blackberry> <timeout> <count>
Usage: pskeycode <rc4 | sha256> <feature> <version> <pshostid> <ethernet | ipaddress | inet6 | hostname | username | vsn | vendor | hostid | macintosh | os | metered | any | country | iphone | android | blackberry> <timeout> <count> <server> <port>
Usage: pskeycode <rc4 | sha256> -f <file>

There are two types of encryption for the license. RC4 or SHA256. This being the first argument to the generator. Then we have the <feature> <version> <id> <idtype> and <timeout>

This minimal number of arguments will generate a node-locked license. For example

pskeycode rc4 monitor 8.1 1e2346fe273d ethernet never

will generate a node-locked license like:

NAME=monitor VERS=8.1 ID=1e2346fe273d IDTYPE=ethernet EXPIRES=never
CERT=06aa45f32b2992d00985740ad209

You can use the >> (greater than) symbols from the command line on all operating systems to redirect the output to a file such as license.lic using two > signs will append multiple executions of pskeycode to the file. So if you are generating a license file with multiple components, you would do it with the double >> redirection symbols.

To generate a floating license file, there are additional arguments to the pskeycode application. Remember that the first line in a floating license file is the SERVER line. This gives information to the license server about what TCP port to listen on as well as other information.

Running the pskeycode application to generate a floating license file requires more information the first time it is run, then fewer arguments for subsequent generation of additional component lines.

To generate a floating license file with two components, you would run pskeycode like this:

pskeycode rc4 monitor 8.1 1e2346fe273d ethernet never 10 localhost 29750
SERVER=localhost ID=1e2346fe273d IDTYPE=ethernet PORT=29750
NAME=monitor VERS=8.1 ID=1e2346fe273d IDTYPE=ethernet COUNT=10 EXPIRES=never
CERT=06aa45e23150fcd6158a6b71dc7c1bae

Notice the information up to the timeout is identical to a node-locked license, but then we add a <count> of licenses. In this case 10 concurrent licenses along with the hostname and port the license server will use. Now for all practical purposes the hostname of the license server will always be “localhost”, as the license server will always be run on the host it is being locked to which is always “localhost”. The license server will also be locked to the ethernet address specified, and the port used is the default 29750 TCP port. You can use whatever port you want, but the software will by default look for 29750@localhost to find the license server.

Users in a networked environment will need to set an environment variable SG_LICENSE_FILE=<port>@<host> if the license server is on a different machine. See the LicenseManagerGuide.pdf for more details.

Now to add a second floating license to this floating license file, you will run the pskeycode application again without the <server> and <port> arguments such as:

pskeycode rc4 process 8.1 1e2346fe273d ethernet never 5
NAME=process VERS=8.1 ID=1e2346fe273d IDTYPE=ethernet COUNT=5 EXPIRES=never
CERT=0ea745fa24309cc11787780cd41763

The important thing to recognize here is the COUNT= in the component line. This is the distinguishing part of the component line that says it is a floating license.

The usage:

Usage: pskeycode <rc4 | sha256> -f <file>

This is so that you can generate licenses using a text file as input where you specify information about the component(s) and from the command line you instruct the license generator wither to generate with rc4 or sha256.

Here is for example a text file named “input.txt”

monitor 8.1 1e2346fe273d ethernet never
process 8.1 1e2346fe273d ethernet never

You can then generate the license file by running pskeycode like:

pskeycode rc4 -f input.txt
NAME=monitor VERS=8.1 ID=1e2346fe273d IDTYPE=ethernet EXPIRES=never
CERT=06aa45f32b2992d00985740ad209
NAME=process VERS=8.1 ID=1e2346fe273d IDTYPE=ethernet EXPIRES=never
CERT=0ea743fa24229bc916f57c6eca78

or:

pskeycode sha256 -f in.txt
NAME=monitor VERS=8.1 ID=1e2346fe273d IDTYPE=ethernet EXPIRES=never
CERT=a47e326c606868e4923b33c4c06c7df569c6c177ba004b3b63ca4522bda00145
NAME=process VERS=8.1 ID=1e2346fe273d IDTYPE=ethernet EXPIRES=never
CERT=7449baa86fb876688b99522c770acb905a552dc463e7031511c2c64b00d46a76