libswid Documentation

Release 0

OpenSCAP Team

Jun 08, 2018

Contents:

libswid is a library that aims to help developers to deal with SWID tags.

Quote from the website:

> Software identification tags (SWID tags) record unique information about an installed software application, including its name, edition, version, whether it is part of a bundle and more. SWID tags support software inventory and asset management initiatives. The structure of SWID tags is specified in international standard ISO/IEC 19770-2:2015.

libswid aims to provide a base, that tool developers will use to build support tools for their target audience:

- Software developers: Developers of any software should make sure that they distribute SWID metadata to their users, or even to redistributors.
- System administrators: Sysadmins need tools to deduce or validate system state from a set of SWID tags.

CHAPTER 1

Contents

1.1 User guide

libswid is able to transfer SWID tag files to memory structures, and vice versa.

It supports a subset of the SWID standard:

- SoftwareIdentity
  - Entity
  - Link

libswid uses XML backends. You can choose an available backend, recommended one is `xerces`.

1.1.1 Usage

Include the `libswid` header. It includes the `SWIDStruct` data structure definition, and backend loader.

1.1.2 Example

```
#include <iostream>

#include <libswid>

using std::cout;


int main(int argc, const char ** argv) {
        if (argc != 2) {
                cout << "No input file has been specified.\n";
                cout << "Specify exactly one SWID tag filename.\n";
                return 1;
```

(continues on next page)

(continued from previous page)

```
        }
        SWIDStruct tag_data;

        const char * xml_backend = "xerces";
        auto io = SWIDTagIO();
        try {
                io.setBackend(xml_backend);
        } catch (const std::runtime_error & e) {
                cout << "Error getting XML backend '" << xml_backend
                        << "': '" << e.what()
                        << "', try another one.\n";
                return 1;
        }
        try {
                tag_data = io.load(argv[1]);
        } catch (XMLIOError & exc) {
                cout << exc.what() << "\n";
                return 1;
        }

        bool creator_not_found = true;
        for (auto it = tag_data.entities.begin(); it != tag_data.entities.end();␣
↪it++) {
                if (it->role & SWID_ROLE_SOFTWARE_CREATOR) {
                        creator_not_found = false;
                        cout << "Creator of " << tag_data.name << " software: ";
                        cout << it->name << " [" << it->regid << "]\n";
                        break;
                }
        }
        if (creator_not_found) {
                cout << "Creator of " << tag_data.name << " software has not been␣
↪found.\n";
        }
        return 0;
}
```

1.1.3 Bindings

You can use `libswid` from other languages besides `C++`.

Python

You need SWIG and Python3 to build the Python bindings. If you want to build them, make sure you have

- Python3 development files, and
- SWIG>=3

During the `cmake` pass, you should see the `Building Python 3 bindings: ON` message. In that case, the build directory will contain the `swid` package and `setup.py` file.

You can install that package as any other Python package. The compiled binary extension will be shipped as package data.

```python
#!/usr/bin/env python3

import sys

import swid


if __name__ == "__main__":
    if len(sys.argv) != 2:
        msg = (
            "No input file has been specified.\n"
            "Specify exactly one SWID tag filename."
        )
        print(msg, file=sys.stderr)
        sys.exit(1)

    tag_data = swid.SWIDStruct()
    xml_backend = "xerces"

    io = swid.SWIDTagIO()
    io.setBackend(xml_backend)
    if not io:
        msg = ("Error getting XML backend '{}', try another one."
               .format(xml_backend))
        print(msg, file=sys.stderr)
        sys.exit(1)
    try:
        tag_data = io.load(sys.argv[1])
    except swid.XMLIOError as exc:
        print(str(exc), file=sys.stderr)
        sys.exit(1)

    creator_not_found = True
    for ent in tag_data.entities:
        if ent.role & swid.SWID_ROLE_SOFTWARE_CREATOR:
            creator_not_found = False
            msg = ("Creator of {} software: {} [{}]"
                   .format(tag_data.name, ent.name, ent.regid))
            print(msg)
            break
    if creator_not_found:
        print("Creator of {} software has not been found.".format(tag_data.name))
```

C

Just include `<libswid.h>`. The `C` interface also mimicks the `C++` interface.

- There are quasi-getters and quasi-setters for individual elements, e.g. `swid_get_root_name`, `swid_set_entity_name`, generally `swid_(get|set)_<structure name>_property`.
- You create new structures by calling `swid_create_<structure name>` and destroy them by `swid_destroy_<structure name>`.
- Whatever you create, you must destroy to avoid memory leaks. When you e.g. create an `Enrity` and append it to list of entities to the root of the SWID tag, the tag root data structure uses a copy of your created entity. When you obtain something using a quasi-getter, you obtain a reference, therefore, don't destroy it.

```c
#include <stdio.h>

#include <libswid.h>


int main(int argc, const char ** argv) {
        if (argc != 2) {
                printf("No input file has been specified.\nSpecify exactly one SWID tag filename.\n");
                return 1;
        }

        SWIDHandle swid = swid_create_root();

        const char * xml_backend = "xerces";
        const char * swid_fname = argv[1];
        SWIDIOHandle io = swid_create_io(xml_backend);

        if (io == NULL) {
                printf("Error getting XML backend '%s', try another one.\n", xml_backend);
                return 1;
        }
        int rc;
        rc = swid_load_root(io, argv[1], swid);
        swid_destroy_io(io);
        if (rc != 0) {
                printf("Error loading SWID '%s'\n", swid_fname);
                swid_destroy_root(swid);
                return 1;
        }


        char creator_not_found = 1;
        SWIDEntityHandle entity;
        for (int i = 0; (entity = swid_root_get_entity(swid, i)); i++) {
                if (swid_entity_get_role(entity) & SWID_ROLE_SOFTWARE_CREATOR) {
                        creator_not_found = 0;
                        printf("Creator of %s software: %s [%s]\n", swid_root_get_name(swid), swid_entity_get_name(entity), swid_entity_get_regid(entity));
                        break;
                }
        }

        if (creator_not_found) {
                printf("Creator of %s software has not been found.\n", swid_root_get_name(swid));
        }
        swid_destroy_root(swid);
        return 0;
}
```

1.2 Developer guide

As of Q2/2018, the project is being actively develped, not being interface-stable. Generate doxygen documentation by running `make doxygen` in the `doc` subdirectory of the project.

1.2.1 Building libswid

Project is built using `cmake`. It has the following prerequisities: * C++11 compiler. * Xerces XML parsing library. * TinyXML XML parsing library.

Optional dependencies: * *lcov* for coverage reports.

Typically, you will want to build, test, and install `libswid`. In order to do so, execute the following commands in the project root:

```
$ mkdir build
$ cmake -DCMAKE_INSTALL_PREFIX=$HOME/.local
$ make && make test && make install
```

If everything went fine, congratulations!

CHAPTER 2

Indices and tables

- genindex
- modindex
- search