x

Inspect Data
Software Development Kit
For C++

Overview

The Inspect Data SDK for C++ provides tools to enhance functionality for content scanning and associated security solutions. This iteration of the toolkit allows for the ability to identify sensitive information; this identification aids in setting rules to safeguard data. The toolkit is delivered as a C++ library, a header file, and several related system libraries and data assets. This guide describes the SDK's interface needed for integration. This version necessitates a system capable of loading and running on 64-bit Linux-based shared modules.

Definitions

The system is comprised of an engine context, configuration and sessions:

Engine context – control mechanism for running scans based on configuration.

Configuration – set of rules/classifiers used by engine to identify matches.

Sessions - context for a scan of a single data set.

The system creates an *engine context* tied to a particular configuration upon instantiation of an SDK object. It is possible for multiple engine contexts, tied to different configurations, to be used at the same time by instantiating additional SDK objects (say, for a multi-tenant solution), but in normal use cases there will be only one engine context per process.

A *configuration* consists of a list of classifiers. A *classifier* is a definition of a type of sensitive information (for example, SSN) that will be identified by the engine. A classifier is comprised of multiple entities of different types (for example, regexes, pre-defined strings, dictionaries, so on and so forth.) When a classifier is identified, it is also associated with a particular *confidence score* ranging from 0 to 100, with 100 being perfect confidence.

When scanning, the *engine context* is used to create a *session* and the session should be used for subsequent scans. Sessions can be reused between different scans, but in this case the session should be reset when switching to the new scan.

The SDK calls are thread safe, but multiple threads cannot concurrently process the same document or piece of text -- for a given document, the text must be processed serially. It is fine to process documents on multiple threads as long as a different session is used for each thread.

When the SDK has detected a piece of sensitive information, it will invoke a callback to the caller's code. The calling code can deal with the information appropriately. The callback routine may also return a non-zero value to halt the scanning process.

SDK Details

The Inspect Data SDK for C++ is intended for software development teams integrating direct, raw data scanning into an existing or new application. The SDK consists of a C++ library, header file, and several system libraries. The SDK is exposed to the client application via a native class interface hierarchy, as follows:

Interface	Usage
Create SDK	`InspectData::SDK sdk;`
Create Session	`InspectData::Session* session{ sdk.newSession() };`
Scan Data	`session->scan((const char*)dataToScan, (long)size, 0, [](InspectData::MatchEvent& match) { auto [ snippet, left, right, name, confidence ] = match; return 0; });`

Using the SDK is intended to be minimally impactful on the implementor. The SDK consists of a single master object, the SDK itself, and a worker object, a session. The SDK object is always initialized once, and for each “scan”, a session object is created (or a single session object may be used.) Note that in cases of muti-threaded applications, each thread must use its own session. The implementor supplies a callback routine that will “receive” each discovered element.

A minimal usage example can be built as shown below.

```
#include <cstring>
#include <iostream>
#include "InspectDataSDK.hpp"
#include "InspectDataFileReader.hpp" // this is only for demo purposes

int main (int argc, char** argv) {
  InspectData::SDK sdk;
  for (auto i=1; i<argc; i++) {
    auto [ contents, size ] = InspectData::FileReader::read(argv[i]);
    if (contents && (size > 0)) {
      std::cout << " - new session for " << argv[i] << ", size=" << size << "\n";
      InspectData::Session* session{ sdk.newSession() };
```

```
      session->scan((const char*)contents, (long)size, 0, [](InspectData::MatchEvent&
match) {
        auto [ snippet, left, right, name, confidence ] = match;
        std::cout << " - match: snippet=" << snippet << ", left=" << left << ",
right=" << right << ", name=" << name << ", confidence=" << confidence << std::endl;
        return 0;
      });
      delete[] contents;
    }
  }
  return 0;
}
```

SDK API

The core SDK object is typically created once for the application, as follows;

```
InspectData::SDK sdk;
```

The object is fully instantiated and configured upon creation. There are two options for creating an SDK instance;

1. `InspectData::SDK sdk;` - creates a default SDK with all built in classifiers enabled
2. `InspectData::SDK sdk(configFileName)` - creates an SDK based on a custom configuration

All sessions created from an SDK instance will use the same configuration.

A session is created to perform a scan. A single session may be used to perform many scans, or a session may be created for each scan. The SDK will cleanup all sessions when the SDK object is deconstructoed, or in the case of a new session for each scan, it is possible to manually destroy the session when it is complete.

There are four important methods exposed;

1. SDK.newSession(…) - creates a new session
2. `SDK.deleteSession(…)` - r emoves the session immediately instead of waiting until the SDK object is freed
3. `Session.scan()` - performs a scan
4. `Session.reset()` - resets a session to its initial state

Technical details for each method is given below.

```
<session>.scan(
       const char* data,
       long size,
       int flags,
       MatchCallback callback
)
```

Parameter	Usage
data	UTF-8 data to scan
size	Number of UTF-8 characters (bytes)
flags	Combination of zero or more scan flags
callback	The handler for the callbacks, matching the MatchCallback signature (can be an inline, anonymous function)

The flags that can be passed in:

Flag	Purpose
CSV_FILE	Data being scanned is a CSV file. This also applies the CSV context to classifiers.
CSV_FILE_WITH_HEADER	Data being scanned is a CSV file with a header. If this is the first scan being called with this session (or after a session reset), then the first line of the buffer will be treated as a list of field names for the data to follow.
ROW_DATA	Treat the file as row data (that is, classifiers will not span lines).

The scan method performs a scan of the entire data buffer. Each matching item generates a report back to the callback routine. The callback routine must conform to the following signature;

```
int callback(InspectData::MatchEvent& match);
```

MatchEvent is a tuple containing the following;

`std::string` – the snippet that matched
`int` – the starting position of the match
`int` – the ending position of the match
`std:string` – the name of the classifier that matched the snippet

`int – a value in the range 1 (low) to 100 (high)` indicating the confidence of the match

The callback routine must return an integer value, where 0 (zero) tells the scanner to continue working, and any other value terminates the scan prematurely. If the client application terminates the scan, it must reset the session before reuse.

<session>.reset()

Failure to reset a session that was terminated may result in false matches or missed matches in the next scan for the same session object.

`<session>.reset()`

The reset method will return a session to its initial, clean state. This method should only be called when a session is truly complete. This call is primarily used when you wish to re-use a session object to process another file; it is rarely needed in other cases but may be useful in cases where the application needs to abort a scanning session in the middle and restart it with new data.

`<sdk>.deleteSession()`

The destroy method will remove a session from memory. The SDK object keeps track of all sessions it creates (via `InspectData::SDK sdk`) and will clean up all objects and memory upon deconstruction by the system. However, some applications may wish to force a session out in a more timely or controlled manner. Applications that reuse a single session object will not need to call the destroy method.

Including the SDK in a C++ project

An implementation uses the SDK by simply adding the C++ library to a project (inspectdatasdk_static.lib) and including the header file (InspectDataSDK.hpp). For reference, the cmake file for the C++ demo project;

```
project(IDSDK_Demo)

set(BASE_DIR "${CMAKE_SOURCE_DIR}")
set(SRC_DIR "${CMAKE_CURRENT_SOURCE_DIR}")
set(BIN_DIR "${CMAKE_BINARY_DIR}")

include_directories(
  ${SRC_DIR}
  ${BASE_DIR}/src
```

```
  ${BASE_DIR}/public
)

set(SOURCES "${SRC_DIR}/IDSDK_Demo.cpp")

add_executable(${PROJECT_NAME} ${SOURCES})

if(MSVC)
  target_link_libraries(${PROJECT_NAME} inspectdatasdk_static.lib)
elif(APPLE)
  target_link_libraries(${PROJECT_NAME} inspectdatasdk_static.lib)
else()
  target_link_libraries(${PROJECT_NAME} inspectdatasdk_static.lib stdc++)
endif()
```