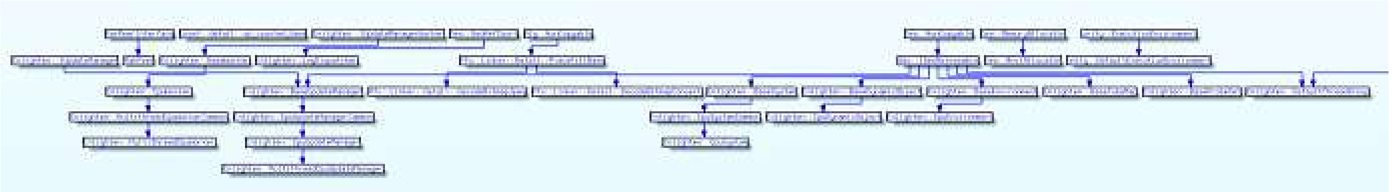
Python Class Informer: an IDAPython plugin for viewing run-time type information

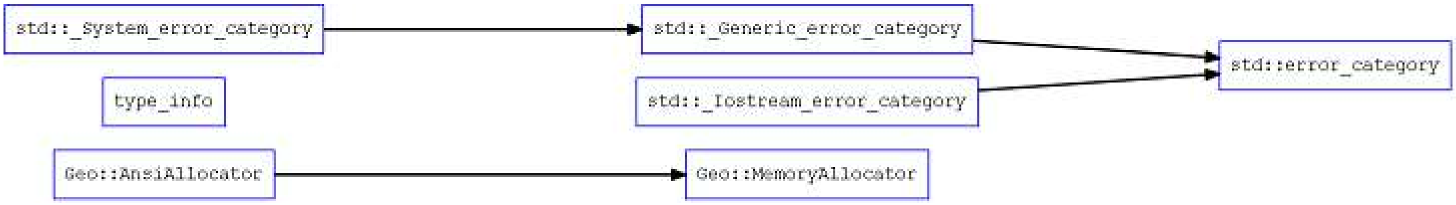
Background

Run-type type information, or RTTI, refers to class information present in compiled C++ binaries. Depending on the class hierarchy used by the programmer who wrote the original code, it may be possible to retrieve a great deal of information about the program. This can be extremely useful in the course of reverse engineering–just knowing names can be quite helpful, but knowing which classes inherit from which other classes is even more powerful. For more detailed information on RTTI, see the Wikipedia article on the topic.

There are already other IDA plugins that perform this functionality, such as the well-established Class Informer, so why make a new plugin? It can be quite useful to extract RTTI from a binary, but if the target binary contains a lot of classes with complex inheritance relationships, it can be difficult to fully grasp how they relate to each other in practice. Here's how Python Class Informer attempts to improve upon this workflow: * Class diagram within IDA * Rather than outputting a flat textual list of classes and their relationships to other classes, this plugin generates a graph within IDA showing the hierarchy of classes:



* DOT export functionality * In addition to displaying a graph using IDA's own graphing engine, the plugin supports exporting to DOT format, which allows you to produce even nicer graphs using an external tool such as GraphViz:

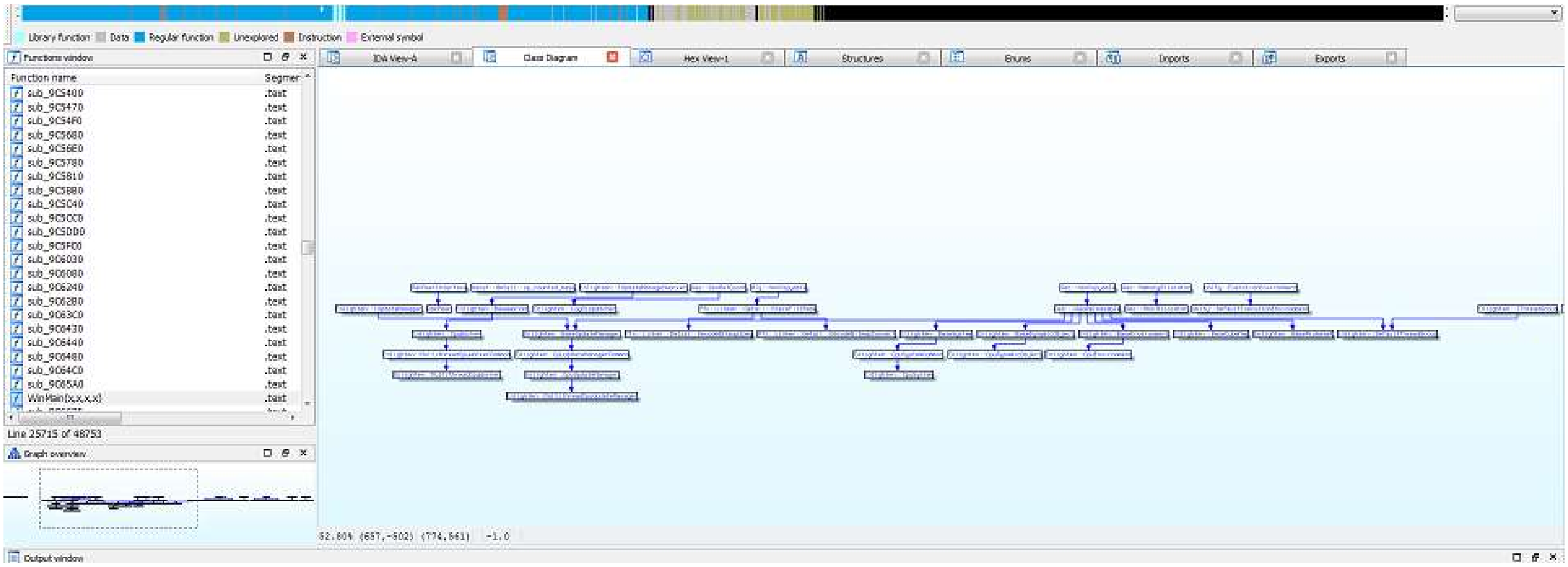

std::_System_error_category
std::_Generic_error_category
std::error_category
type_info
std::_Iostream_error_category
Geo::AnsiAllocator
Geo::MemoryAllocator

* Support for multiple compilers * RTTI formats are different for each C++ compiler. This plugin supports parsing RTTI from binaries compiled with MSVC or GCC. * Architecture- and platform-independent * This plugin is written in Python, allowing it to work on any architecture (32-bit or 64-bit IDA versions) and on any platform (Windows, Linux, Mac) without modification or recompilation.

Usage

The requirements to use the plugin are fairly simple: as long as you have a copy of IDA Pro installed, with a corresponding installation of Python. The plugin can be obtained from GitHub at the following URL: https://github.com/nccgroup/PythonClassInformer

First, open up the IDA database that you'd like to read RTTI for. Then, within IDA, go to File -> Script file… and choose "classinformer.py". (You can also press Alt-F7 to avoid having to use the File menu every time you'd like to run the plugin.) If all goes well, you should see a graph within IDA looking something like this:

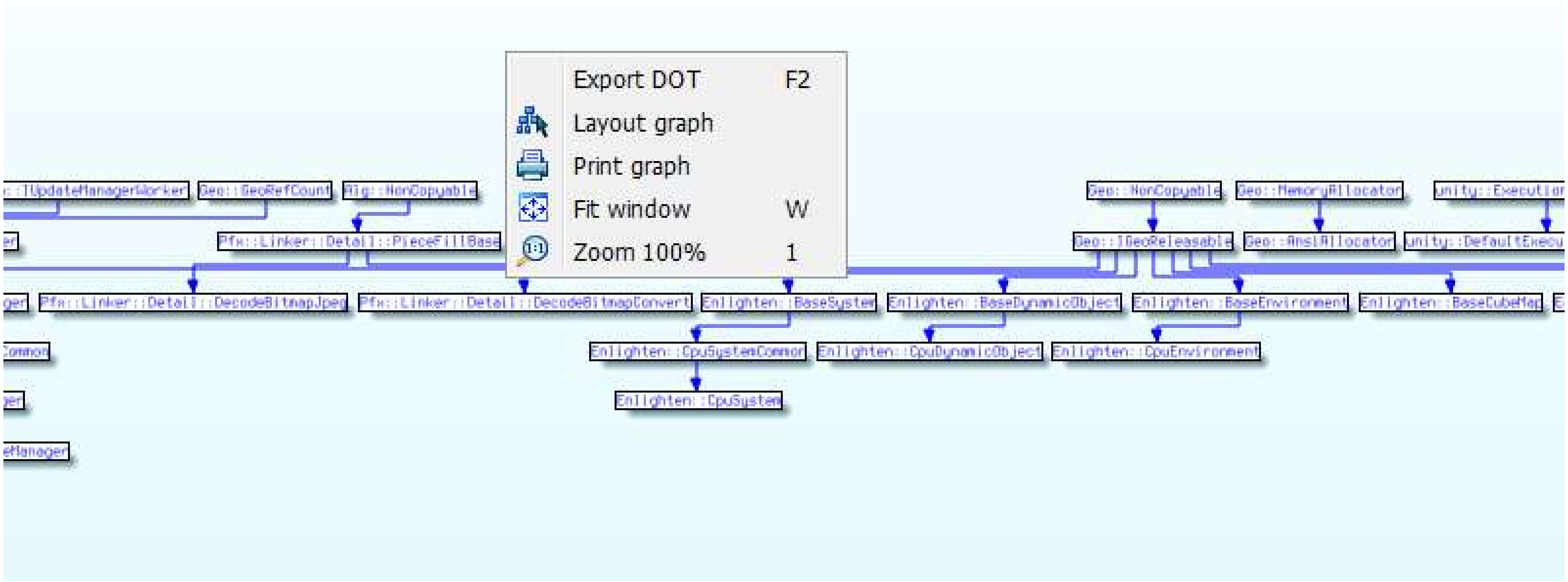


Function name	Segment
sub_9C5400	.text
sub_9C5470	.text
sub_9C54F0	.text
sub_9C5680	.text
sub_9C56E0	.text
sub_9C5780	.text
sub_9C5810	.text
sub_9C5880	.text
sub_9C5C40	.text
sub_9C5CC0	.text
sub_9C5DD0	.text
sub_9C5F00	.text
sub_9C6030	.text
sub_9C6080	.text
sub_9C6240	.text
sub_9C6280	.text
sub_9C63C0	.text
sub_9C6430	.text
sub_9C6440	.text
sub_9C6480	.text
sub_9C64C0	.text
sub_9C65A0	.text
WinMain(x,x,x,x)	.text

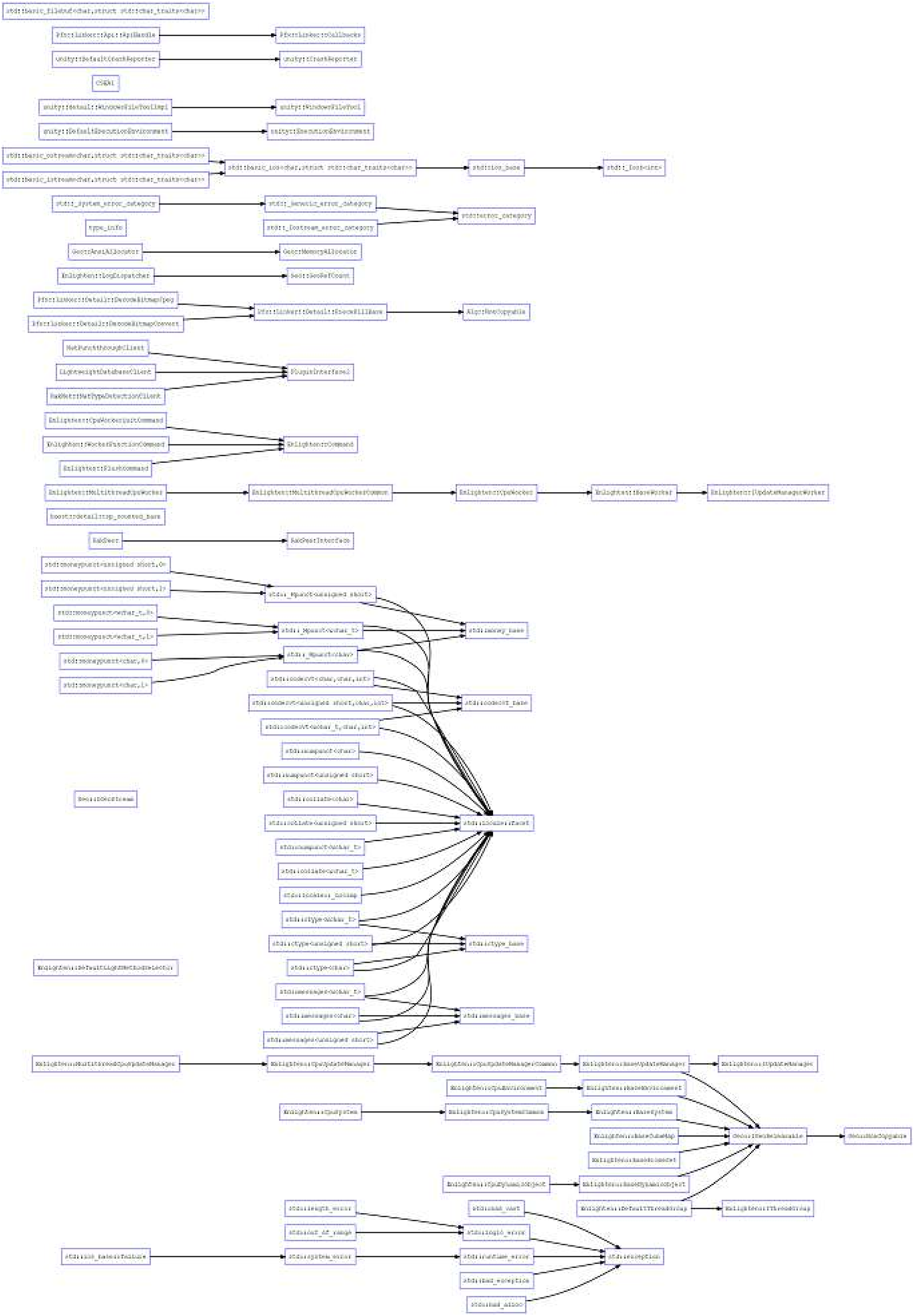
Line 25715 of 48753

From there, you can navigate around the graph and explore the class hierarchy in the exact same fashion as you would explore basic blocks in IDA.

If you'd like to have more control over how the graph is formatted, you can also export the graph to a DOT file in order to render a graph using an external tool. Just right-click on the graph and click “Export DOT”:



Then, you can use a tool such as GraphViz to render the graph:


std::basic_filebuf<char,struct std::char_traits<char> >
Pfx::Linker::Api::ApiHandle
Pfx::Linker::Callbacks
unity::DefaultCrashReporter
unity::CrashReporter
CSHA1
unity::detail::WindowsFileToolImpl
unity::WindowsFileTool
unity::DefaultExecutionEnvironment
unity::ExecutionEnvironment
std::basic_ostream<char,struct std::char_traits<char> >
std::basic_istream<char,struct std::char_traits<char> >
std::basic_ios<char,struct std::char_traits<char> >
std::ios_base
std::_Iosb<int>
std::_System_error_category
std::_Generic_error_category
std::_Iostream_error_category
std::error_category
type_info
Geo::AnsiAllocator
Geo::MemoryAllocator
Enlighten::LogDispatcher
Geo::GeoRefCount
Pfx::Linker::Detail::DecodeBitmapJpeg
Pfx::Linker::Detail::DecodeBitmapConvert
Pfx::Linker::Detail::EncodeFillBase
Alg::NonCopyable
NatPunchthroughClient
LightweightDatabaseClient
RakNet::NatTypeDetectionClient
PluginInterface2
Enlighten::CpuWorkerQuitCommand
Enlighten::WorkerFunctionCommand
Enlighten::FlushCommand
Enlighten::Command
Enlighten::MultithreadCpuWorker
Enlighten::MultithreadCpuWorkerCommon
Enlighten::CpuWorker
Enlighten::BaseWorker
Enlighten::IUpdateManagerWorker
boost::detail::sp_counted_base
RakPeer
RakPeerInterface
std::moneypunct<unsigned short,0>
std::moneypunct<unsigned short,1>
std::_Mpunct<unsigned short>
std::moneypunct<wchar_t,0>
std::moneypunct<wchar_t,1>
std::_Mpunct<wchar_t>
std::money_base
std::moneypunct<char,0>
std::moneypunct<char,1>
std::_Mpunct<char>
std::codecvt<char,char,int>
std::codecvt<unsigned short,char,int>
std::codecvt_base
std::codecvt<wchar_t,char,int>
std::numpunct<char>
std::numpunct<unsigned short>
Geo::GeoStream
std::collate<char>
std::collate<unsigned short>
std::locale::facet
std::numpunct<wchar_t>
std::collate<wchar_t>
std::locale::_Locimp
std::ctype<wchar_t>
std::ctype<unsigned short>
std::ctype_base
std::ctype<char>
Enlighten::DefaultLightMethodSelector
std::messages<wchar_t>
std::messages<char>
std::messages_base
std::messages<unsigned short>
Enlighten::MultithreadCpuUpdateManager
Enlighten::CpuUpdateManager
Enlighten::CpuUpdateManagerCommon
Enlighten::BaseUpdateManager
Enlighten::IUpdateManager
Enlighten::CpuEnvironment
Enlighten::BaseEnvironment
Enlighten::CpuSystem
Enlighten::CpuSystemCommon
Enlighten::BaseSystem
Enlighten::BaseCubeMap
Geo::IGeoReleasable
Geo::NonCopyable
Enlighten::BaseProbeSet
Enlighten::CpuDynamicObject
Enlighten::BaseDynamicObject
std::length_error
std::bad_cast
Enlighten::DefaultThreadGroup
Enlighten::IThreadGroup
std::out_of_range
std::logic_error
std::ios_base::failure
std::system_error
std::runtime_error
std::exception
std::bad_exception
std::bad_alloc

Conclusion

RTTI can be an extremely helpful way to gain insight about a C++ binary during reverse engineering, and Python Class Informer's visualization of the class hierarchies can strengthen these insights even further. We hope reverse engineers' lives will become a little easier using the visualizations produced by this plugin. Currently, there are several known issues with the parsing of RTTI data that we're planning on fixing, as well as some features we'd like to add, so stay tuned for future updates.

Published date: 05 October 2017

Written by: Tyler Colgan