Document #: P2947R0
Date: 2023-07-17
Project: Programming Language C++
Audience: SG21, SG23
Reply-to:
Andrei Zissu <andrziss@gmail.com>
Ran Regev <regev.ran@gmail.com>
Gal Zaban <galzaban@gmail.com>
Inbal Levi <sinbal2l@gmail.com>

Contracts must avoid disclosing sensitive information

Contents

Abstract

[P2811R7] proposes `comment()` and `source_location()` as properties of a `std::contracts::contract_violation` object. [P2811R7] allows for these properties to not be populated (and return empty strings).
This paper proposes that an implementation that does *not* provide an opt out for leaving these properties unpopulated would be a non-conforming implementation.

Motivation

Enabling the **`source_location`** and **`comment`** strings to be present in shipped binaries might reveal sensitive information to anyone in possession of such a binary, such as file and function names in **`source_location`** and CCA texts in **`comment`**.

It is accepted that it is impossible to prevent reverse engineers from analyzing binaries. However, there is a significant advantage to reverse engineering a binary with symbols or debug information that indicates the name and purpose of functions. This makes the reversing process faster and lowers the entry barrier for reverse engineering that binary.
Additional information on reversing C++ binaries can be found in the following talk: CppCon 2019: Behind Enemy Lines - Reverse Engineering C++ in Modern Ages (Gal Zaban)

Nowadays, major vendors invest significant efforts in product security. As part of these efforts, C++ binaries released in the final products are usually stripped, so that reversers can only find limited information. If the implementation does not require this option, it could potentially discourage the usage of contracts, at least in production builds. Vendors may choose not to use the feature if they have no way to restrict the strings added to their binaries.

Another point worth considering is the (voluntary or otherwise) adherence of many companies to various regulations and accordingly the need to pass security audits (especially in industries in which safety and security are most important). This may often include required pen testing, whereby any weakness or undue ease of debugging is likely to be listed in the test report. Case in point, this is how RTTI wound up disabled in the case of company C mentioned below. More information about this: SOC 2 Compliance: Do I need a pentest or vulnerability scanning? (Valentina Flores)

The authors have witnessed first hand the importance given to avoiding having such sensitive strings in shipped binaries. Use cases witnessed on several different occasions:

1. Company A requested the removal of log messages from production code, which resulted in a full-blown compile-time text obfuscation facility with an accompanying log decoder utility (Personal Log - Where No Init Has Gone Before (Andrei Zissu)).
2. Company B translated all log messages into IDs at compile time, producing a map-file alongside the binaries to map log-message-number into its text. The map file stays in-house while the binaries are producing numbers-only log messages (Yet another fast log (Ran Regev)).
3. Company C disabled RTTI to avoid having plain text symbol names in shipped binaries. This was a direct result of a pen tester report.

These examples illustrate a willingness on the part of companies to put in a great effort and/or to accept various limitations in order to avoid shipping sensitive embedded textual information and to protect intellectual property as first and foremost priority and no less than other safety and security considerations.

Q&A

If the opt out is taken, how do we identify which CCA was violated?

In our opinion, it is ultimately the user's choice to eliminate this valuable information, and even the fact that a contract violation happened may be useful enough.
We also expect crash dumps to be a partial mitigation of this issue, as they normally include call stacks. For example, Visual Studio allows separate inclusion of symbols in pdb files which are not typically shipped, but may be used to reconstruct full call stacks out of crash dumps received from customers.
Another mitigation may be the ability to place breakpoints in contract violation handlers and observe the call stack under a debugger.
In configurations lacking terminating contract semantics this will be a trade-off to be considered by users, which may choose to provide less revealing means of identifying their own CCAs.

How does the process of Reverse Engineering C++ change when debug information is added to the binary?

The debug information and information like the `source_location` can guide reverse engineers to the interesting parts of the binary simply by looking at references to relevant function names in the binary's strings. Similar to researching binaries compiled in 'debug' mode, this requirement would make it easier for researchers, as any function containing a contract would still reference its original name from the source.
Furthermore, per [P2811R7]:[3.10.4] the source location information passed to the contract violation handler may represent call sites, either instead or in addition to the location of the CCA itself. This would add even more information useful to reversers, as each precondition CCA may generate multiple textual references in the compiled code representing potentially all the call sites of the functions to which the CCA appertains.
There are various talks and blogs available online for more information on this subject:

- Reverse Engineering Tips - Run-Time Type Identification (Thomas Roccia)
- Reversing C++ (Paul Vincent Sabanal, Mark Vincent Yason)
- Recovery of Object Oriented Features from C++ Binaries (Kyungjin Yoo, Rajeev Barua)
- RTTI Internals in MSVC (Lukasz Lipski)

Is this only about ease of reversing?

No, it's not only about the direct effect on reverse engineering, but also about *perception* and what effects it might have on contracts adoption. Several use cases have been listed in this paper which clearly show examples of companies caring about this enough to do something about it.

The standard doesn't usually specify compiler flags, so why propose them now?

We are not. We propose mandating implementations to provide unspecified means of opting out of CCA-related textual information. In practice such means are indeed likely to come in the form of compiler flags, but we're not proposing spelling this out in the standard.

Summary of Proposed Changes

We propose adding clear stipulations to the **`contract_violation`** type introduced in [P2811R7], to the effect of requiring conforming implementations to provide means of opting out of having any meaningful information in the **`source_location`** and **`comment`** properties. It is up to the implementations what such means shall be and whether a single means will be provided for opting out of populating both properties or different means for either of them separately. In case of opt out by the user, the information that would have populated these properties will not appear anywhere in the binary code (unless it's required for another reason). At the very least such information will not appear in a way that associates it with a corresponding CCA and reveals its location and meaning.

Impact of the Changes

- Adopting this proposal does not change the essence of [P2811R7].
- Adopting this proposal puts security at a higher priority and is aligned with SG23 goals.

Wording

Edits are relative to [P2811R7], in [support.contract.cviol]

```
const char* comment() const noexcept;
Returns: Implementation-defined text describing the predicate of the
violated contract.
An implementation shall provide a way for this accessor to return no
information.

source_location location() const noexcept;
Returns: The implementation-defined source code location where this
contract violation was detected.
An implementation shall provide a way for this accessor to return no
information.
```

References

P2811R7: Contract-Violation Handlers (Joshua Berne)
(https://isocpp.org/files/papers/P2811R7.pdf)

CppCon 2019: Behind Enemy Lines - Reverse Engineering C++ in Modern Ages (Gal Zaban)
(https://www.youtube.com/watch?v=ZJpvdl_VpSM)

SOC 2 Compliance: Do I need a pentest or vulnerability scanning? (Valentina Flores)
(https://www.redsentry.com/blog/soc-2-compliance)

CppCon 2022: Personal Log - Where No Init Has Gone Before in C++ (Andrei Zissu)
(https://www.youtube.com/watch?v=0a3wjaeP6eQ)

Yet Another Fast Log (Ran Regev)
(https://github.com/regevran/yafl)