Software Assurance Tips

A product of the Software Assurance Tips Team[3]

Kevin Keen

Monday 9th August, 2021

1 Ada ~~Low~~High Integrity Profiles

Updated Friday 6th August, 2021

Ada is often touted by its supporters as the best programming language ever devised by man. And at least on a conceptual level there are some good arguments in its favor. But as with any complex programming language there are corner cases that should be understood and dealt with carefully to avoid those advantages turning into disadvantages.

Enter the Ada High Integrity Profiles. What are these mysterious profiles? With a name like "high integrity" they must be good. And if Ada itself is already the best programming language, then high integrity Ada must be the best of the best, right? Not so fast! As we will see, you may want to avoid these profiles unless you absolutely need them!

Profiles in Ada came out of the 1997 International Real-Time Ada Workshop. The `Ravenscar` Profile was intended to produce a more lightweight (and provably correct) executable with task scheduling that was more suited to real time applications. It was revised, and eventually included as part of the Ada standard in 2005.[**20210806:grm**] There are several versions of `Ravenscar` including `ravenscar-cert`, `ravenscar-zfp`, and more. Each variant of a profile has different restrictions on what is and is not allowed using that runtime system. The most restrictive are the Zero Footprint Profiles (ZFP). These profiles go to great lengths to make sure that the resulting executable does not link against the Ada runtime library. As such, they offer the least "bulk" in the binary image, but also the fewest features. One step above the ZFPs are the "cert" variants, which are intended for profiles that have undergone a separate certification.

Of particular note is the way in which exceptions are handled. Since all of these profiles are attempting to reduce memory footprint, the handling of exceptions is also reduced. The ZFPs in particular have the least exception handling. Exceptions still happen (including user defined), but they must be handled locally. No traditional exception propagation happens in ZFP. There is a "last chance handler" that will be called if an exception is not handled locally.[2, § 4.2.6] In ZFP, this handler must be implemented by the programmer, and the compiler will enforce its existence.

Beyond exceptions, there is an entire gamut of Ada features that are excluded in these profiles.[2, § 4.2.2] One that was particularly interesting is the restriction on non-library level tagged types. Tagged or managed types are of particular importance to us for code review. Tagged types will automatically have their `Initialize` procedure called when an object of that type is constructed without an initial value (think constructor but not quite).[5] Without that guaranteed initialization, more complex projects that have to be segmented in order to be analyzed by adacore wind up with tons of false positives about possibly un-initalized variables. We had heard of a company whose Ada programming standard deliberately avoided the use of tagged types. It made little sense to us for them to avoid tagged types until we learned about these profiles and their restrictions.

When Ada is written against embedded systems, it is not uncommon for us to see the use of VxWorks. There are a set of runtime systems and associated profiles for various versions of VxWorks. For this tip, the salient points of profiles in general still apply. It is worth noting that the "cert" version of the VxWorks profiles do come with a default last chance handler that by default will print the stack, raise an `APEX` exception and exit.[1, § A, Replacement of the Default Last Chance Handler]

While none of these restrictions are a security concern in and of themselves, it does beg for care to be taken. When combined with the "continue running at any cost" mentality that we see quite often one could see a scenario where exceptions effectively go unhandled. Code using `cert` or `zfp` profiles, combined with a last chance handler that swallows exceptions, could render an application unstable and vulnerable. Documentation states that all last chance handlers must effectively terminate the application,[2, § 3.1] but makes no statement that this is enforced. We often see the sin of catching top level exceptions[4, p. 159] with the mindset of "continue running at any cost." It is just as much of a concern under these conditions as it would be in C++ code.

As a bonus tip, note that even MORE risky code can be obtained with the pragmas `No_Exception_Handlers` and `No_Exceptions`. The `No_Exception_Handlers` pragma disallows all exception handlers except for the last chance handler. The `No_Exceptions` pragma disables runtime exceptions entirely![2, § 3.1]

References

[1] AdaCore. GNAT User's Guide Supplement for Cross Platforms. Aug. 5, 2021. URL: https://docs.adacore.com/gnat_ugx-docs/html/gnat_ugx/gnat_ugx.html (visited on 08/06/2021).

[2] AdaCore. GNAT User's Guide Supplement for GNAT Pro Safety-Critical and GNAT Pro High-Security. Oct. 17, 2018. URL: https://docs.adacore.com/gnathie_ug-docs/html/gnathie_ug/gnathie_ug.html (visited on 08/06/2021).

[3] Jon Hood, ed. SwATips. https://www.SwATips.com/.

[4] John Viega, Michael Howard, and David LeBlanc. 24 Deadly Sins of Software Security. Programming Flaws and McGraw-Hill Education, 2009. ISBN: 978-0-07-162675-0.

[5] Wikibooks. Ada Programming/Object Orientation — Wikibooks, The Free Textbook Project. 2021. URL: https://en.wikibooks.org/w/index.php?title=Ada_Programming/Object_Orientation&oldid=3956556#Constructors (visited on 08/06/2021).